DEPARTMENT OF MEASUREMENTS AND INFORMATION SYSTEMS

DEVELOPING AN EFFICIENT APPLICATION FOR THE DISTRIBUTION OF LARGE FILES

B.Sc. Thesis

László András Gárdonyi

Advisor: Zoltán Micskei

Budapest, 2008

Table of Contents

1. Introduction 3
1.1. The Goal of the Thesis 3
1.2. Existing Solution 3
1.2.1. UFTP Details 4
1.3. Problems with the Existing Solution 4
1.4. High-Level Requirements 5
1.5. Alternative Solutions 5
1.5.1. Advanced Framework for the Current Solution 5
1.5.2. Active Directory Software Deployment 5
1.5.3. Commercial Software Management Systems 6
1.5.4. BitTorrent Based Solution 6
1.6. Development Method 7
2. Designing the Application 8
2.1. Requirement Analysis 8
2.1.1. Use Case Analysis 8
2.1.2. Non-Functional Requirements 11
2.2. Architecture 11
2.2.1. Technology 11
2.2.2. Layers 14
2.2.3. Security Concerns 15
2.2.4. User Interface 16
2.3. Development Environment 16
2.3.1. Version Control System 16
2.3.2. Issue Tracking System 17
2.3.3. Other Development Tools 18
2.4. Testing Principles 18
2.4.1. Unit Testing 19
2.4.2. Integration Testing 19
2.4.3. System Testing 19
2.4.4. User Acceptance Testing 19
2.4.5. Code Analysis 19
3. Implementation of the Application 21
3.1. Service Component 21
3.2. Client Component 26

3.3. UFTP Daemon 27
3.4. Logging Component 28
3.5. Test Implementations 29
3.5.1. Features to Be Tested 29
3.5.2. Features *Not* to Be Tested 29
3.5.3. Unit Testing Framework 29
3.5.4. Integration Testing Framework 31
3.5.5. Test Scripts 32
3.5.6. System Testing 36
4. Testing Results 37
4.1. Fixed Issues 37
4.2. Code Analysis Results 38
4.3. Testing in the Target Environment 39
5. Installation Instructions 40
5.1. Service Component 40
5.1.1. Requirements 40
5.1.2. Installation 40
5.1.3. Settings 40
5.1.4. Operation 43
5.2. Client Component 45
5.2.1. Requirements 45
5.2.2. Installation 45
5.2.3. Settings 45
5.2.4. Operation 46
5.3. UFTP Daemon 47
5.3.1. Requirements 47
5.3.2. Installation 47
5.4. Group Policy Settings 47
6. User Manual 48
7. Conclusions 50
7.1. Evaluation of the Solution 50
7.2. Lessons Learned 51
7.3. Future Plans 51
References 52

1. Introduction

The following chapter discusses the initiative information for the thesis. It starts by laying down the goals and explaining the existing scenario, finally going over the alternative solutions and describing the method of the development.

1.1. The Goal of the Thesis

Recently in the department's laboratories virtual machines are being used to supply the special software environments required for the different courses, e.g. high availability clusters, J2EE application servers or system management applications.

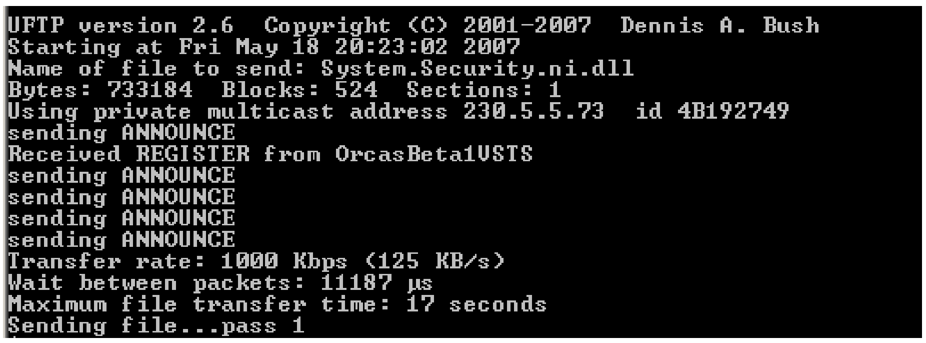
Before the lab sessions the teaching assistants have to arrange the quick distribution of the virtual machines. These virtual machines are usually large in size, at least 5 GB, but there are plenty above 10 GB, thus copying them separately to each machine would take a long time and would put a severe load on the server containing the VMs, as well as the network infrastructure.

The goal of the thesis is to find an efficient solution for distributing these large files.

1.2. Existing Solution

To solve the problem, the department staff found a utility capable of efficiently distributing large files, but didn’t require the installation of a large scale configuration management or software distribution system.

The choice fell upon the utility called UFTP (1). It's a simple command line application that uses multicast packets for copying to multiple machines. The application has two components, a daemon (the client) and a server. The daemon has to be running on each of the clients; it listens on a specific UDP port for the copy announcements. The server has to be started with a filename as a parameter. The server first sends out announce messages to the network and waits for the clients that answer, then it starts to send the file in smaller blocks. Each block has to be acknowledged by the clients to ensure reliable transmission.



```
UFTP version 2.6  Copyright (C) 2001-2007  Dennis A. Bush
Starting at Fri May 18 20:23:02 2007
Name of file to send: System.Security.ni.dll
Bytes: 733184  Blocks: 524  Sections: 1
Using private multicast address 230.5.5.73  id 4B192749
sending ANNOUNCE
Received REGISTER from OrcasBeta1VSTS
sending ANNOUNCE
sending ANNOUNCE
sending ANNOUNCE
sending ANNOUNCE
Transfer rate: 1000 Kbps (125 KB/s)
Wait between packets: 11187 µs
Maximum file transfer time: 17 seconds
Sending file...pass 1
```

Figure 1: The Output of the UFTP utility.

The copy process is currently controlled by a simple PowerShell script, which maintains the tasks that have to be performed before and after the actual copying (e.g. starting/stopping clients, creating destination directories, moving files to final destination from temporary folders on clients, setting file permissions, etc.).

1.2.1. UFTP Details

Following is the detailed description of the protocol used by UFTP according to the description on the tool's homepage (1).

UFTP's transfer protocol consists of three phases: Announce/Register, Transfer/NAK phase and the Completion/Confirmation phase.

In the first phase the server announces the transfer on a public multicast address that the clients are listening on. Depending on the parameters, the server either specifies the hosts that are allowed to participate in the transfer (closed group membership) or allows any host to participate (open group membership). The announcement also contains information such as the file name, file size, number of blocks and the private multicast address that will be used for the actual file transfer. The participating client sends a registration which the server confirms in reply.

In the next phase the server divides the transferred file into sections and blocks and starts sending them continuously to the network via UDP. Since the UDP protocol does not guarantee that the packets arrive in order, each packet is numbered, so that the clients can properly reassemble the file. If a section is complete, the server sends a "Done" message to the clients to which the clients reply with the list of NAKs (negative acknowledgements) of the packets that have not arrived successfully. Once all the sections have been transferred, the server runs additional passes by only sending the missing packets, until it receives no further NAKs from the clients.

When a client received all packets of the file, it sends a "Completion" status to the server which replies with a "Confirmation" status message. After receiving the confirmation, the client stops listening to the private multicast address.

1.3. Problems with the Existing Solution

The copying is handled efficiently by UFTP, but there's still need for manual preparation:

- The user has to log in manually on each client and start the UFTP daemon,
- The user has to check free space manually on each client machine,
- The user has to log in on the server machine and start the PowerShell script. The main problem is not the manual procedure, but the fact that user needs to be a Domain Administrator to ensure that the script can map the client's local drives. Some lab sessions are coordinated by non-administrators, thus they can't arrange these preparations,
- The user has to manually check whether the copy was successful on each client machine.

1.4. High-Level Requirements

Since the current solution has some drawbacks, the task is to develop a software system which features the following capabilities and enhancements:

- Efficient and reliable file distribution in a like or similar fashion to the UFTP tool,
- Seamless authorization and authentication for users with non-administrator accounts,
- Simple, intuitive user interface.

1.5. Alternative Solutions

This section walks through the solution alternatives to the requirements specified above.

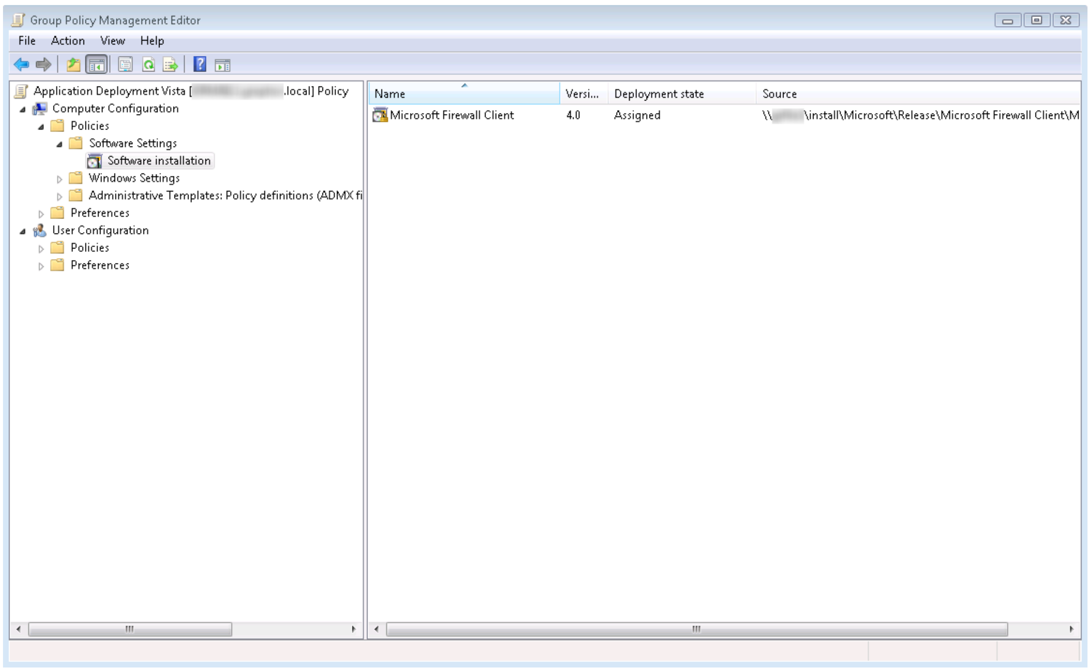
1.5.1. Advanced Framework for the Current Solution

The obvious solution, which was actually accomplished, was to take the current script and implement its features in an advanced framework. This required the time for development and access to the laboratory for testing. It was also a convenient project for a homework assignment and a project laboratory task, not to mention this thesis project.

Nevertheless it is worth looking at the pros and cons of other solutions.

1.5.2. Active Directory Software Deployment

The machines in the department laboratory are part of an Active Directory domain. Active Directory can deploy MSI installer packages using Group Policy Objects (GPOs). After deployment, packages are installed the next time the clients log into the domain. The packages have to be accessible via a shared folder on the server.



Name	Versi...	Deployment state	Source
Microsoft Firewall Client	4.0	Assigned	\\[redacted]\install\Microsoft\Release\Microsoft Firewall Client\M

Figure 2: Deploying a package via Group Policy.

Each image folder would have to be built into an MSI package that's configured to extract its content to the appropriate destination.

By default only the Domain Administrator has rights to create and edit GPOs, but access rights can be granted to other users or groups for a certain GPO, although they allow access to additional adjustments beside deployment, which could be considered as a security risk.

The actual deployment takes place when the machines in the group log in to the domain and update their group policies. The clients are not able to log on to their accounts while the installer packages are acquired from the shared folders and being executed.

This alternative would produce a higher load on the network during distribution and would also severely delay the log on time on the clients.

1.5.3. Commercial Software Management Systems

There are several software management systems available on the market, two large scale products in the category are the *IBM Tivoli Provisioning Manager* (2) and the *Microsoft System Center Configuration Manager* (3) (formerly *System Management Server*). They are both intended for managing the IT infrastructure of large enterprises, including operating system and software deployment, patch management, and also maintaining the software and hardware inventory.

Their methods of the actual software deployment is similar to that of the Active Directory solution, thus they have the same drawbacks, not to mention that we would only require a small set of the available features.

1.5.4. BitTorrent Based Solution

BitTorrent is a popular, public domain, peer-to-peer protocol designed particularly for the distribution of large files.

The distribution process consists of the following elements:

- The tracker, which keeps track of clients connecting to a certain distribution process,
- Peers, who are users downloading the distributed data using a BitTorrent client,
- Seeders, who are peers in possession of all the data being distributed,
- A static metainfo file containing details of the distribution.

The client communicates with the tracker via HTTP; communication with other peers is done directly via TCP/IP. Commands and statuses are encoded using a text based encoding called *bencode*. The basic task of the tracker is to circulate the addresses of all peers, so that they know about each other's existence.

Distributed files are divided into pieces and blocks. One piece is transferred at a time between two peers, block by block. The pieces and blocks are numbered so that the original file can be properly reassembled. Each piece has a corresponding SHA1 hash (4) that is used to validate finished pieces. If the piece fails the hash test, it is redownloaded.

What makes the protocol efficient is that although not all peers have the whole amount of distributed data, all of them are uploading the pieces that they already have. This means that if there is only a single initial seeder and a number of peers it is possible to distribute data to all the peers with the seeder having to upload each piece only once. This method dramatically decreases the network load for peers and seeders locally.

The static metainfo files contain information about the filenames and the hash of every piece, and also the addresses of trackers, all encoded in the *bencode* format. They can be recognized by their `.torrent` extension and are traditionally downloadable from thematic web sites and search engines.

A more detailed description can be found in the official BitTorrent protocol specification (5).

There are numerous clients available; some even contain a built-in tracker. A detailed list and comparison can be found here (6). There is also an open source, platform independent library available for custom implementations called *libtorrent* (7).

Choosing BitTorrent as an alternative would replace UFTP as the efficient distribution method, but would still require a proprietary framework for the administrative tasks.

1.6. Development Method

Development of the application was done in an iterative fashion similar to extreme programming. The course of the whole process can be split among the following phases:

- **Phase 1:** The script version was created and served as a basis for the design of an advanced application.
- **Phase 2:** A prototype was created as a homework project with the then new *Windows Communication Foundation* (WCF) (8) architecture.
- **Phase 3:** The prototype was extended with the implementation of the main features during the Project laboratory course.
- **Phase 4:** The thesis project added missing features and restructured the implementation to allow systematic testing.

Each phase consisted roughly of the following cyclical steps:

- Designing of a new function,
- Coding the new function,
- Testing the new function in the development environment,
- Testing the new function in the target environment,
- Feedback to the next cycle.

The rest of the thesis describes the currently last phase of the development process.

2. Designing the Application

The following chapter discusses the steps of designing the application. It starts by analyzing requirements and designing a preliminary user interface, which is followed by architecture planning, development environment setup and last but not least, the designation of testing principles.

2.1. Requirement Analysis

As the first step of the designing process the high-level requirements must be expanded into detail by examining the real life situations of the tasks in question.

2.1.1. Use Case Analysis

The system consists of one actor and several use cases. The following diagram shows the Image Distributer System on a use case diagram. The packages were introduced to enhance readability and to categorize the use cases.

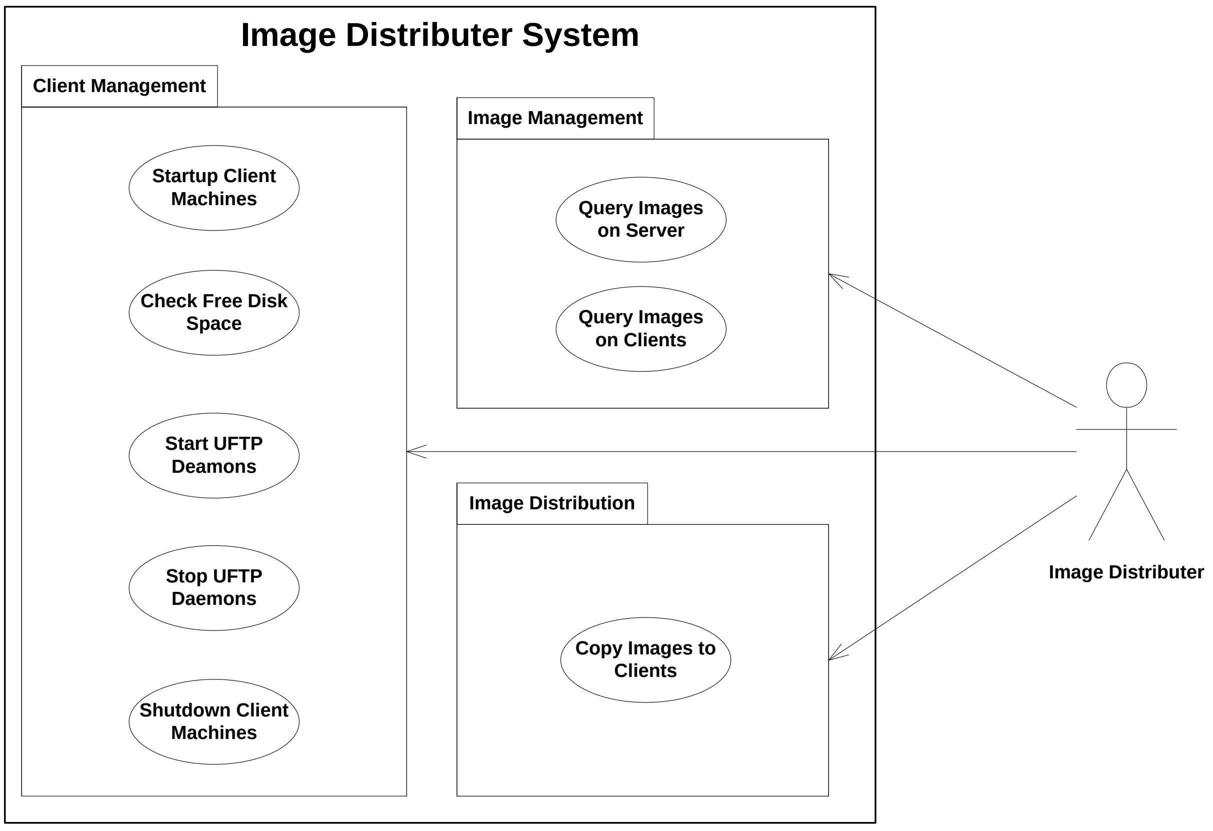


Figure 3: The Image Distributer System Use Case diagram.

Actors

There is only one actor in the diagram called **Image Distributer**. It represents the user who distributes virtual machine files to the lab computers.

Use Cases

The following is the detailed specification of use cases on the diagram:

Use Case	*Startup Client Machines*
Category	Client Management
Actor	Image Distributer
Description	Start up client machines so that they can receive distributed files.

Use Case	*Check Free Disk Space*
Category	Client Management
Actor	Image Distributer
Description	Check free disk space on client machines to verify that they have the capacity to receive the distributed files.

Use Case	*Start UFTP Daemons*
Category	Client Management
Actor	Image Distributer
Description	Start UFTP daemons on client machines before distribution.

Use Case	*Stop UFTP Daemons*
Category	Client Management
Actor	Image Distributer
Description	Stop UFTP daemons on client machines when the distribution is finished.

Use Case	*Shutdown Client Machines*
Category	Client Management
Actor	Image Distributer
Description	Shutdown client machines when the distribution is finished.

Use Case	*Query Images on Server*
Category	Image Management
Actor	Image Distributer
Description	Query the distributable images on the server.

Use Case	*Query Images on Clients*
Category	Image Management
Actor	Image Distributer
Description	Query images on the clients to check whether the images already exist on them or if the distribution has been successful.

Use Case	*Copy Images to Clients*
Category	Image Distribution
Actor	Image Distributer
Description	Copy selected images to the target client machines.

The following sequence diagram shows a typical course of the image distribution process:

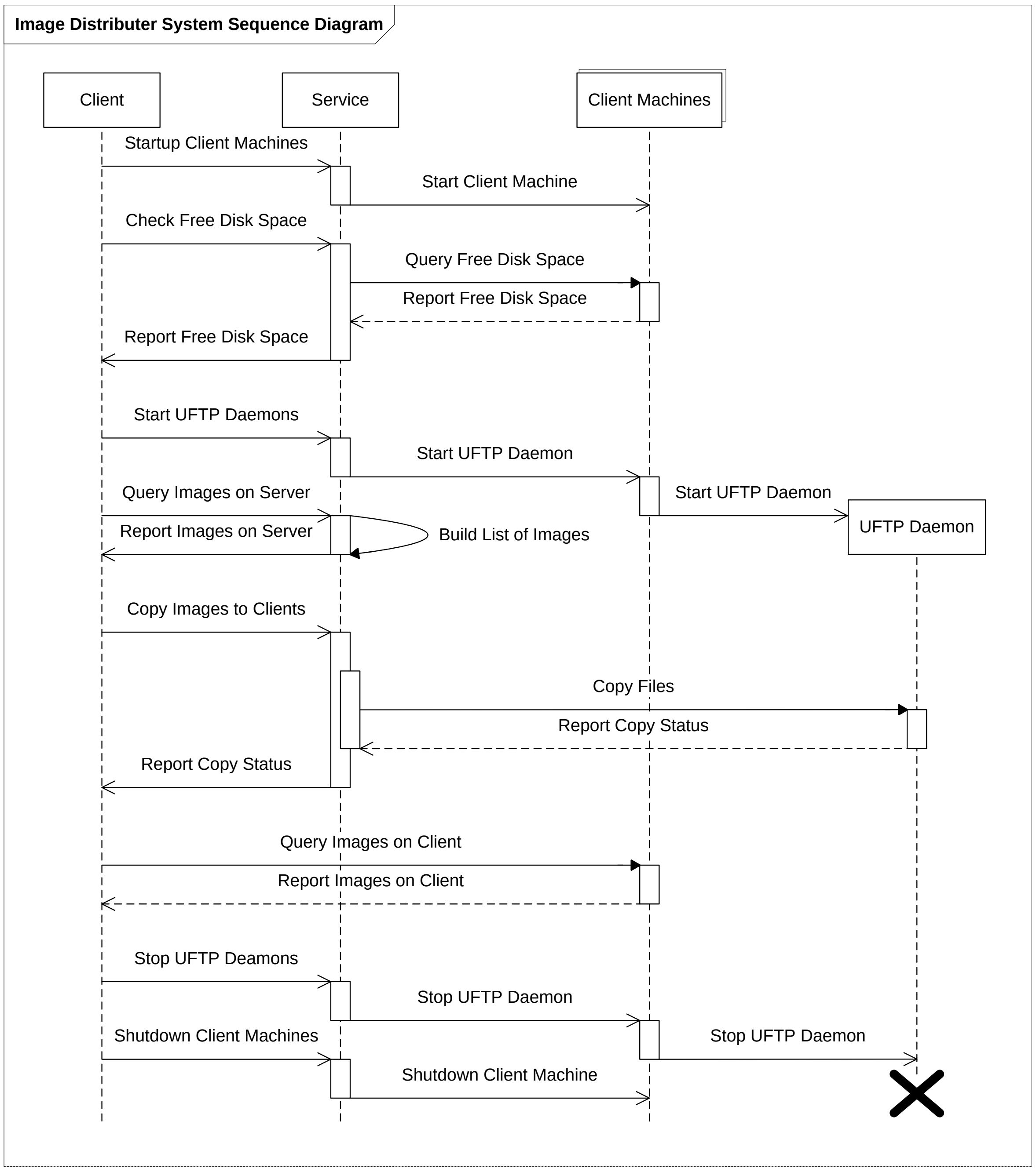


Figure 4: Sequence diagram of a typical distribution process.

2.1.2. Non-Functional Requirements

The following non-functional requirements are important to the success of the application, in order to achieve acceptance from the end users:

- **Fast Copying:** The complexity of the system should not degrade the speed of the copy process achieved by using UFTP alone.
- **Central Manageability:** The user shouldn't have to log in on each machine; the whole distribution process should be managed from a single operation central.
- **Asynchronous Operation:** Starting up and shutting down the client machines, as well as the multicast copying are very long operations, therefore they have to be performed asynchronously and the user interface has to stay responsive, otherwise the users might think that the system has crashed or feel uncomfortable using it all together.
- **Security:** Not every user should have the rights to use the system, but the rightful should not need to have Administrator privileges. By filling the hard drives of the clients or deleting existing virtual machines one can do considerable harm, therefore unauthorized users should not be able to change or forge messages in the system.

2.2. Architecture

The analysis of the requirements comes down to the following principals in the light of architecture:

- There has to be a central component, i.e. the business logic (BL), running with high privileges to gain access to all the various system resources affected in the distribution process. The user interface running under lower privileges then connects to this central component,
- The user interface and the business logic have to be loosely and asynchronously connected,
- There has to be a component on the clients that runs the UFTP daemon in the background when necessary.

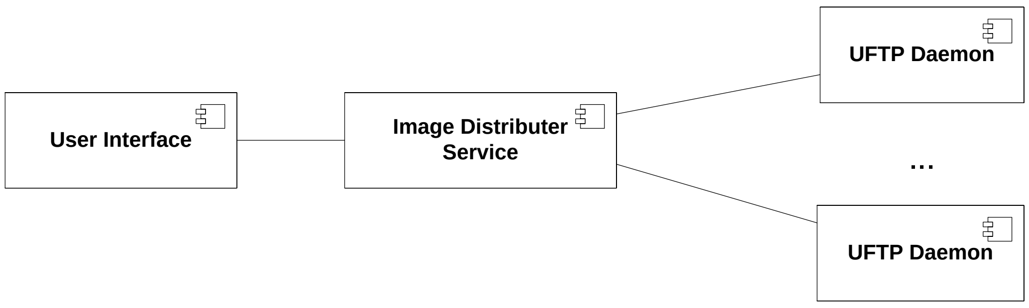


Figure 5: The diagram shows the basic architecture of the logical components.

2.2.1. Technology

The choice of technology depended on the existing environment, not to mention the preferences and curiosity of the developers:

- Since the multicast copying is already solved by UFTP it was obvious to use the tool itself internally for the actual file transfers. It is also open source, thus it is possible to re-implement in a form other than the original command line executable.

- The fundamental technology for the development is the *Microsoft .NET Framework 3.5* (9). The reason behind the choice is that both the clients and the server in the department's laboratory run Microsoft Windows operating systems and the framework has good support for the built-in facilities (service management, authentication and authorization, platform invocation). It was also interesting to explore the possibilities of the new version of the framework.
- The communication between the components is crucial to the whole architecture. The choice fell on the *Windows Communication Foundation* (WCF) (8). This relatively new technology does not mean a certain communication protocol, but a framework for service-oriented architecture (SOA) (10), largely simplifying the creation of loosely coupled components.
- Because the business logic is running separated from the user interface, it should be decided whether the latter is implemented as a thin or a thick client that is whether it should be a web interface or a separate executable. It is also a question whether to host the service in a web server or in a Windows service[1] executable.
- The main technology for turning on machines over the network is *Wake-On-Lan* (11). All of the machines' network adapters in the laboratory support it, and there are numerous tools available online, it is also easy to implement.
- *Windows Management Instrumentation* (WMI) supports initiating shutdown remotely by users with sufficient rights (12). Fortunately the .NET Framework has extensive support for WMI.

Windows Communication Foundation

Because WCF is an important factor in deciding several architectural questions, here is short summary about its features. The main objective in the designing of WCF was to provide a standard framework for the existing Microsoft communication technologies. While developing WCF applications only abstract terms are used, the actual carrier protocols are chosen later, possibly even in the installation phase.

The elements of a WCF application:

- **Service:** An independent unit, providing methods to be called remotely through one or more endpoints.
- **Client:** A unit using one or more services. As in other similar technologies, a client uses proxies to communicate with the services. The proxies are automatically generated by the framework and hide the details of the network communication process. The client only sees a method call as if it were on a local object.
- **Endpoint:** Services are provided and consumed through so called endpoints. An endpoint consists of the following elements:
    - **Contract:** Describes the operations provided by the service, as well as the data types used as their parameters and return values. It serves as an abstract description of the calling interface, e.g. when hosting web services in WCF, the contract is mapped to a WSDL (Web Service Description Language) file.

[1] Since WCF uses the same term for its components as the Windows operating system for its background processes, from here I will refer to the latter as a Windows service.

- **Binding:** Describes the protocols and encodings for the endpoint. It is possible to specify non-functional protocols, e.g. for authentication and encryption.
- **Address:** Describes the place where the endpoint is found, it is actually a URI.

- **Message:** Calls to service operations are always translated to messages that are transferred between the endpoints. According to the binding configuration, they can become e.g. binary packages or XML formatted SOAP messages.

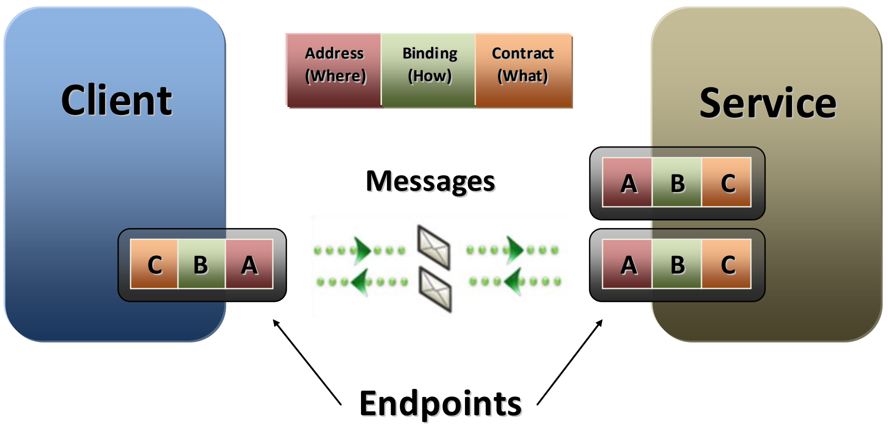


Figure 6: Basic architecture of WCF applications (13).

These terms and their basic usage are well described in this article (14). The WCF documentation (15) describes all the functions and detailed steps to using the technology.

The following is an excerpt of the main steps:

- Defining the service contract. This means writing a .NET interface annotated with specific attributes. There are three types of operations:
  - **Request/Reply:** default message transfer pattern. The call blocks the client until the operation finishes.
  - **One-way:** the client does not wait for a reply, in a fire-and-forget kind of pattern. The client is not notified of exceptions.
  - **Duplex:** two-way communication. There is a callback interface that the service can use to call operations on the client.
- Specify how the clients communicate with the service. The choices are:
  - Asynchronous or synchronous method,
  - Enabling session handling for related messages or not.
- The configuration of the service can be done by coding in the initialization phase, but it is more flexible to use configuration files. The *system.serviceModel* element serves as the container for WCF configuration. It contains the description of the service and the endpoints.
- Create a host for the service. A detailed description can be found in article (16). The basic choices for service hosts are the following:
  - Windows Forms or Console application. Since the business logic has to run indefinitely in the background, this is only acceptable for testing purposes during the development,
  - Windows service,
  - ASP.NET application, running in IIS.

2.2.2. Layers

After deciding on technologies, we must proceed to the layers of the architecture.

Business Logic

The business logic will offer its services through the WCF framework. It will need Domain Administrator privileges to acquire and modify information on the client machines; therefore security is a significant factor. Considering this, the host will be a Windows service rather than a web application hosted in IIS (although it will require minimal changes in the code if we were to change our minds later).

Some of the operations are long running and need to be run asynchronously and some also need to report progress before the end of the whole operation, e.g. while copying the client should be notified of finishing large files. This calls for implementation according to the Duplex pattern.

Client

At the very beginning of development WCF was still in beta stage and it seemed risky to try an implementation with a web interface, as it turned out to have various defects in this perspective. Thus the Client was initially developed as a Windows Forms application. This seemed suitable, because all potential users have Windows accounts in the laboratory environment and the application is genuinely intended for inside use only.

The communication protocols between the client and the service are configurable even at installation time, but the Duplex communication scheme limits the possible bindings to *WSDualHttpBinding*, *NetTcpBinding*, *NetNamedPipeBinding* and *NetPeerTcpBinding*. The *NetNamedPipeBinding* is only for use within one computer, the *NetPeerTcpBinding* is more suitable for peer-to-peer applications. The other two both provide reliable message sending and options for secure transmission, but since there is no need for interoperability among different platforms the *NetTcpBinding* is an appropriate choice, not to mention a faster one because its binary encoding.

UFTP Daemon

The existing solution already incorporates a Windows service that is a wrapper around the command line UFTP daemon. It starts and stops the daemon when needed and takes its configuration (command line arguments) from the registry.

The following diagram describes the connection between the logical components of the system:

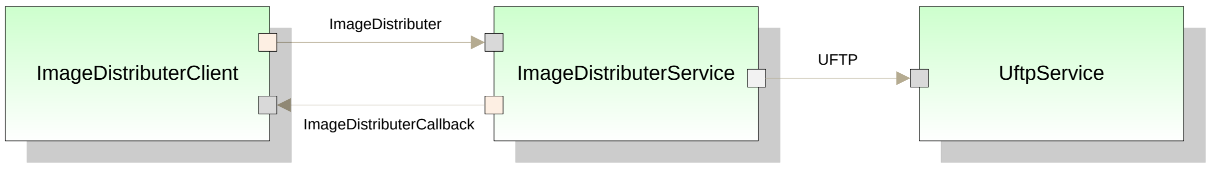


Figure 7: Logical diagram of the components.

2.2.3. Security Concerns

As stated in the high-level requirements the system should be accessible by non-administrators, but only to a selected group of users. With Active Directory it is possible to achieve role based authentication by defining user groups for access verification. As discussed earlier, the .NET Framework has built-in support for Windows based authentication.

The business logic needs Domain Administrator rights to operate, but the users should not have direct access to these privileges. It must verify access for users according to their group membership. The client should also warn the user if trying to connect from an account that is not member of the required group.

The system should not let the users decide whichever machines in the network they want to tamper with, thus the business logic must manage a list of allowed machines in the laboratory to which all operation are restricted. If the client tries to perform operations on a machine that is not on the list it should be notified that it tried accessing a disallowed machine.

UFTP in its current implementation doesn’t support any form of authentication, thus daemons should not be running at all times, because anyone in the subnet could initiate a file transfer who knows which port the daemons are listening on. This risk can be reduced by only running the UFTP daemons when distribution is in progress.

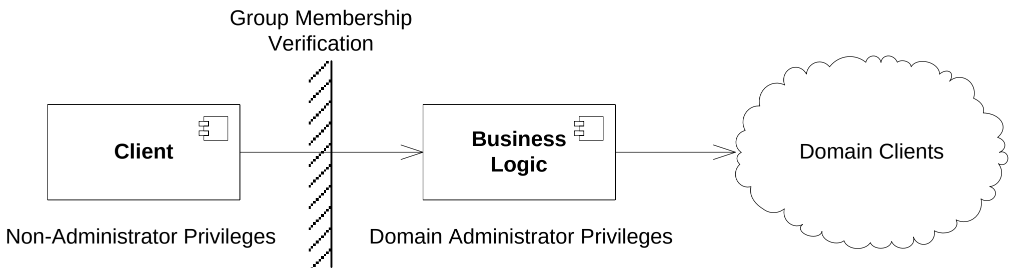
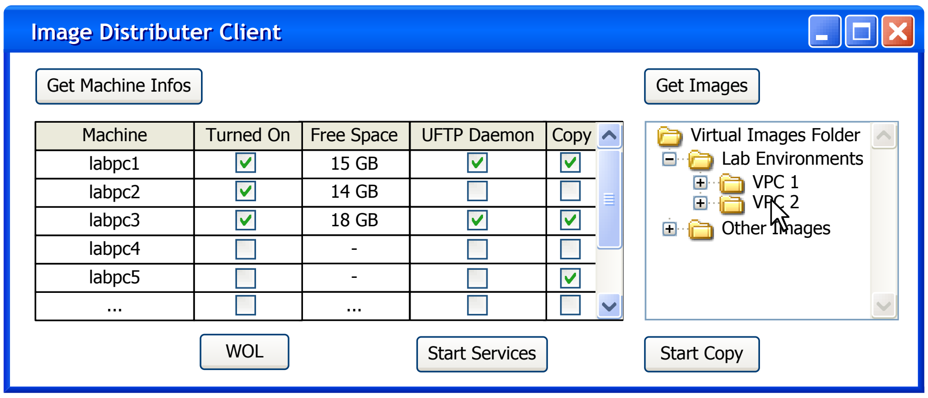

Group Membership Verification
Client
Business Logic
Domain Clients
Non-Administrator Privileges
Domain Administrator Privileges

Figure 8: Security diagram of the system components.

2.2.4. User Interface

In a large scale project it is essential to provide a preliminary concept of all user interfaces to the customers, who can provide feedback on whether they like or do not like what they see. It also helps review the functions and features planned and may point out the need for new features not yet thought of. However the final implementation might completely differ from the one below.



Machine	Turned On	Free Space	UFTP Daemon	Copy
labpc1	✓	15 GB	✓	✓
labpc2	✓	14 GB	☐	☐
labpc3	✓	18 GB	✓	✓
labpc4	☐	-	☐	☐
labpc5	☐	-	☐	✓
...	☐	...	☐	☐

Figure 9: User interface design.

The lab machines are listed in a grid view displaying necessary information for each. The list of available images will be displayed in a tree view control. There will be buttons to retrieve all information from the service and also to send commands concerning the selected machines and image folder.

2.3. Development Environment

The mainstream development environment for .NET Framework applications is *Microsoft Visual Studio*. The *Microsoft Visual Studio 2008 Team System Edition* is available for download via the *Microsoft Developer Network Academic Alliance* (MSDNAA) program intended for purely academic usage (17).

2.3.1. Version Control System

Version control systems store source files in the so called “repository”. They keep track of changes, and all changes are revertible to previous versions. Developers can also create “branches” off the main “trunk”, by viewing the repository via the popular tree analogy. They can create new features in branches and test them separately while leaving the main trunk untouched. If the branch is stable enough, it can be merged with the main trunk.

In a project with multiple participants version control is essential to store and keep track of code edited by several people. Therefore changes are labeled with the user’s name that made them.

For version control we used the *Subversion* (SVN) (18) server hosted on the department server, available at the following address:

https://svn.inf.mit.bme.hu/student/onlab/trunk/ImageDistributer/

The top folder structure of the repository is the following:

- **dist:** files to be installed on server and clients (Client, Service, UftpService),
- **doc:** documentation and related material,
- **src:** source code and project files.
- **test:** testing related subprojects.

The Subversion server uses name and password for authentication.

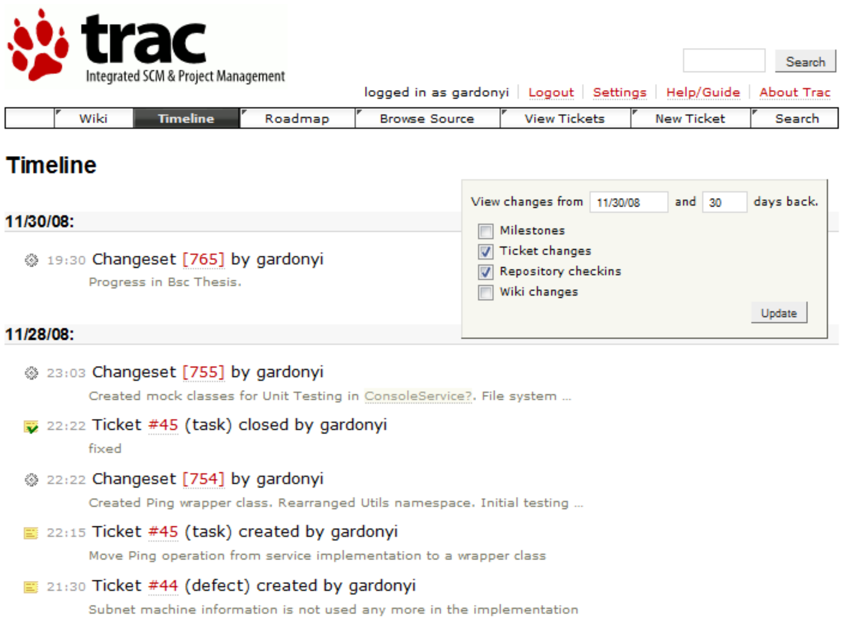
The thesis comes attached with a CD-ROM containing the most current condition of the repository.

2.3.2. Issue Tracking System

We set up a web-based issue tracking system named *Trac* (19) on the department server.

Trac keeps track of notes, bugs, ideas made by project members. Tickets can be issued in the defect, enhancement, and task categories. Milestones can be created (e.g. FirstRelease, NextVersion) to signify the goals of the project. Tickets can be assigned to certain users or can be left open for anyone to resolve. Tickets can be resolved as fixed, duplicate, worksome, wontfix, invalid.

Trac can search and filter tickets and also display events on a timeline. It is linked to the SVN server and has its own web-based repository browser. It can be configured to send email notifications to the users involved in the project. It also features a wiki style documentation system.


trac
Integrated SCM & Project Management
Search
logged in as gardonyi | Logout | Settings | Help/Guide | About Trac
Wiki
Timeline
Roadmap
Browse Source
View Tickets
New Ticket
Search
Update

Timeline

View changes from 11/30/08 and 30 days back.
Milestones
[✓ Ticket changes]
[✓ Repository checkins]
Wiki changes

11/30/08:

19:30 Changeset [765] by gardonyi
Progress in Bsc Thesis.

11/28/08:

23:03 Changeset [755] by gardonyi
Created mock classes for Unit Testing in ConsoleService?. File system ...

22:22 Ticket #45 (task) closed by gardonyi
fixed

22:22 Changeset [754] by gardonyi
Created Ping wrapper class. Rearranged Utils namespace. Initial testing ...

22:15 Ticket #45 (task) created by gardonyi
Move Ping operation from service implementation to a wrapper class

21:30 Ticket #44 (defect) created by gardonyi
Subnet machine information is not used any more in the implementation

Figure 10: Timeline view in the Trac Issue Tracking System.

Our Trac setup is available at the following address:

https://trac.inf.mit.bme.hu/ImageDistributer

A username and password must be entered for authentication.

2.3.3. Other Development Tools

Other tools that proved useful during development:

- **Service Configuration Editor:** Graphical UI for editing the XML configuration file for WCF services. It is part of the *Windows SDK* (20).
- **Service Trace Viewer:** Viewer for WCF messages passed between the client and the service, if tracing is enabled in the configuration file. Also part of the *Windows SDK* (20).
- **MDbg:** While testing in a live environment with no IDE available, this lightweight debugger came useful e.g. in catching certain authentication exceptions. It requires no installation and no framework to be installed. It can be downloaded from here (21), and more information can be found here (22).
- **psgetsid:** While testing in a Windows Domain environment we came across a problem that it is not easy to acquire a name for a certain SID using purely managed APIs, thus making debugging quite cumbersome. The tool that helped in these situations is the *psgetsid.exe*. It is available as part of the *SysInternals* (23) package.
- **TortoiseSVN:** A Subversion client that integrates itself into the Windows shell. It can check out the repository to a selected folder and the versioning services can be managed via the context menus, while file statuses are indicated with icon overlays. It also features a sophisticated diff viewer (24).

2.4. Testing Principles

Systematic testing is the key component of quality assurance in all branches of engineering, particularly in the software variety and should take up a significant amount of time during the development. In a worst case scenario without systematic testing, a major bug might only be discovered only after the final deployment of the system. The situation can possibly cause financial damages, or in the least, set back the release of the product.

In multi contributor projects, testing is crucial to the efficiency of teamwork. First of all, a defect in one component can compromise the functionality of others, thus giving a hard time for other contributors debugging their own modules. Therefore it is important to test new builds before committing changes to the source code repository. This approach is sometimes called “smoke testing”.

Since systematic testing was introduced rather late in this project, we have selected only the few methods we found suitable for a “small” venture like this, hence this is a rather minimal testing scenario.

On the long term, test driven development should be the key to quality software engineering. A significant example I came across recently is the Chromium project. This article describes (25) the chief test principles the project is based on.

2.4.1. Unit Testing

As its name suggests, unit testing focuses on individual parts of the source code, such as functions and procedures or perhaps complete classes.

The point is to test only the relevant unit ("Unit-Under-Testing" or "System-Under-Testing"), thus external dependencies have to be emulated with so called "test doubles". This detailed article describes the nature of these objects (26).

Unit tests generally include the following steps:

1. Setup input parameters (initiate test doubles),
2. Execute Unit-Under-Testing,
3. Verify that the results are as expected,
4. Clean up, if necessary.

Unit tests can be custom built, but there is a wide range of unit testing frameworks available (27), most notably there is one built into the Team System Edition of Visual Studio.

2.4.2. Integration Testing

Unit testing verifies the quality of individual components. These components communicate with each other through well specified interfaces. Integration testing focuses on these interfaces. It should verify the functional composition of the system.

Modules can be tested in smaller groups or all at once, it depends on the nature of the project.

2.4.3. System Testing

System testing is a type of black-box testing which examines the various aspects of the system as a whole. It should verify how the system satisfies the specified requirements and tolerates extreme situations, such as high loads or invalid inputs.

2.4.4. User Acceptance Testing

User acceptance testing is similar to system testing, but its goal is to verify that the end users accept the system and are ready to start using it. This test should be performed regularly when a new feature is implemented or when the development arrives to a stage where new features are likely to be added and the end users' feedback is required. The feedback results sometimes lead to decisions of redesigning certain parts of the software.

2.4.5. Code Analysis

So far we have discussed dynamic testing, which involves executing code. It is also possible to do static testing by analyzing source code with purely mathematical methods. Code analysis can be used to identify typical coding errors and security vulnerabilities, and also to enforce best practices. There are numerous tools and metrics available on the subject (28).

The majority of code analysis tools analyze the source files directly. For managed .NET applications, there is a source code analyzer built into the Visual Studio Team System Edition, but it is also available separately from Microsoft (29). This tool analyses compiled .NET assemblies instead of the original source code as .NET is considered “language independent”. .NET assemblies do not contain native machine code, but an intermediate language (IL) code that can be converted back to any of the source languages supported by the framework, but the IL code is directly suitable for code analysis.

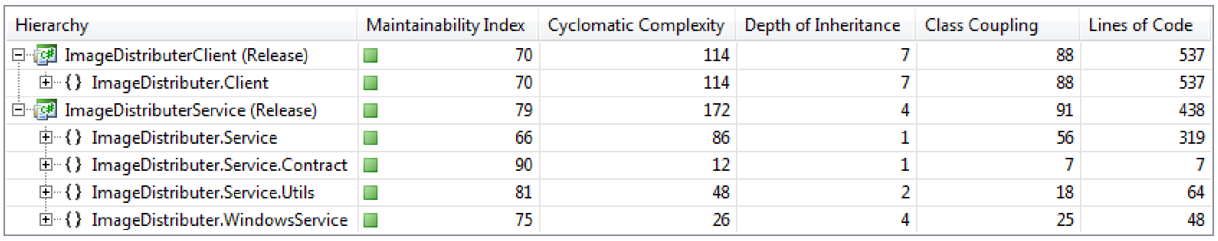
In Visual Studio projects, there’s also an option to automatically run code analysis after a successful build. The code analyzer generates its output as compiler warnings.

3. Implementation of the Application

The following chapter discusses the implementation of the system specified so far, divided by each component.

All coding was done in C#, the programming language most native to the .NET Framework.

The following table contains the code metrics generated by Visual Studio for the service and the client component projects with their underlying namespaces:



Hierarchy	Maintainability Index	Cyclomatic Complexity	Depth of Inheritance	Class Coupling	Lines of Code
ImageDistributerClient (Release)	🟩 70	114	7	88	537
{ } ImageDistributer.Client	🟩 70	114	7	88	537
ImageDistributerService (Release)	🟩 79	172	4	91	438
{ } ImageDistributer.Service	🟩 66	86	1	56	319
{ } ImageDistributer.Service.Contract	🟩 90	12	1	7	7
{ } ImageDistributer.Service.Utils	🟩 81	48	2	18	64
{ } ImageDistributer.WindowsService	🟩 75	26	4	25	48

Figure 11: Code metrics generated by Visual Studio.

The Maintainability Index should provide a general idea of how complex the code is. Both projects have reached rather high scores on the scale of 0 to 100 which means the code is highly maintainable (30).

3.1. Service Component

The service component is implemented as a Windows service, the related classes can be found in the *WindowsService* namespace. The *Program* class contains the *Main* method that creates an *ImageDistributerWindowsService* class which contains the initialization and finalization codes for Windows service. The *ProjectInstaller* class contains metadata (display name, description, startup options) that the service installer utility uses to generate the registry entries for the Windows service.

The *Contract* namespace contains the interfaces that define the operations and data types used in the WCF service.

The *Service* namespace contains the actual implementation of the service component, the *ImageDistributerService* class which implements the *IImageDistributerService* interface.

The implementation class relies on three helper classes in the *Utils* namespace:

- The *Ping* class determines whether a certain machine is turned on using ICMP ping messages.
- The *WOL* class generates Wake-On-LAN packets and broadcasts them on the network.
- The *UftpServer* class is a wrapper for the native DLL implementation of the UFTP server tool.

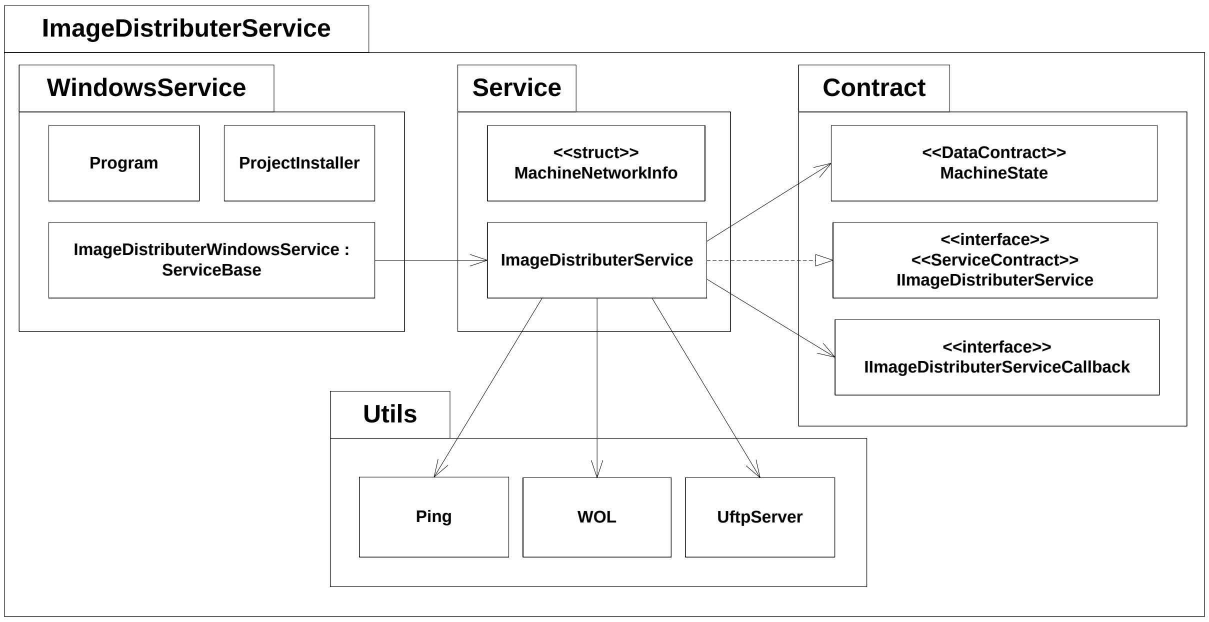

ImageDistributerService
WindowsService
Program
ProjectInstaller
ImageDistributerWindowsService : ServiceBase
Service
<<struct>> MachineNetworkInfo
ImageDistributerService
Contract
<<DataContract>> MachineState
<<interface>> <<ServiceContract>> IImageDistributerService
<<interface>> IImageDistributerServiceCallback
Utils
Ping
WOL
UftpServer

Figure 12: Composition of the service component's classes.

The following diagram contains the methods of the implementation classes and its interfaces:

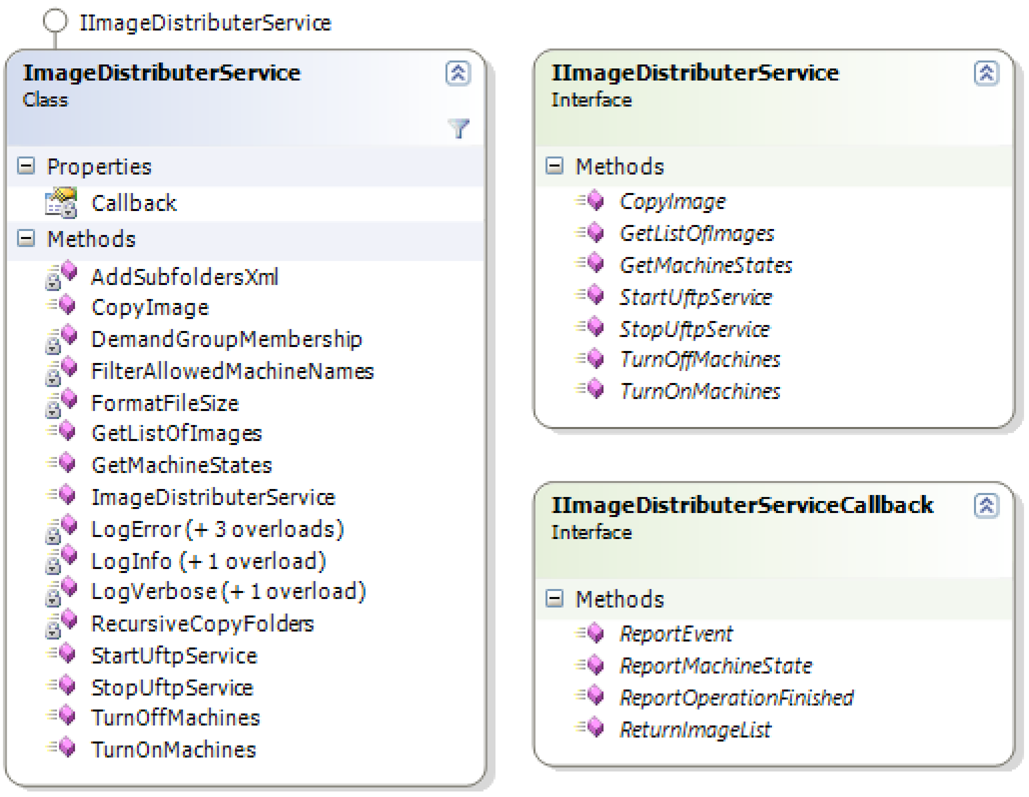


Figure 13: Class diagram of the service component.

The following is the list of signatures and descriptions of significant methods in the *ImageDistributerService* class:

Method Signature	`ImageDistributerService()`
Description	The constructor of the class. Verifies and loads machine information stored in the configuration file.

Method Signature	`long AddSubfoldersXml(XmlDocument document, XmlNode node, string path)`
Description	Recursively adds the subfolders to the supplied *XmlDocument* starting from the root path specified in the parameter. The return value is the size of all files and folders under the path specified. This function is used by the *GetListOfImages* method.

Method Signature	`void CopyImage(string imagePath, string[] machineNames)`

Description	Copies the image folder specified in the *imagePath* parameter to the machines specified in the second parameter.
	Internally the *RecursiveCopyFolders* method is called to do the actual copying.
	Pseudo code:
	
	`DemandGroupMembership()`
	`FilterAllowedMachineNames(machineNames)`
	`if imagePath does not exists then fail`
	`DeleteRemoteTempContents(machineNames)`
	`RecursiveCopyFolders(imagePath, machineNames)`
	
	Actual parameters of methods called may differ.
	This method realizes the “Copy Images to Clients” use case.

Method Signature	`void DemandGroupMembership()`
Description	Verifies that the WCF client is member of the group specified in the configuration file. If the verification fails an exception is thrown and the calling method fails, thus the WCF channel state becomes “faulted”, and further communication is blocked with the client.

Method Signature	`void FilterAllowedMachineNames(string[] machineNames, ref List<string> allowed, ref List<string> disallowed)`
Description	Given the array of machine names in the first parameter, the method returns the list of allowed machines in the second parameter according to the list of machines in the configuration file. The machines that are not on the allowed list will be returned in the third parameter.

Method Signature	`void GetListOfImages()`
Description	Returns the listing of images on the server to the client. It calls the *ReturnImageList* callback to send the results back to the client. The method uses the *AddSubfoldersXml* function to parse the folders and returns the resulting XML in a string. This method realizes the “Query Images on Server” use case.

Method Signature	`void GetMachineStates(string[] machineNames)`

Description	Queries the required information from the machines listed in the *machineNames* parameter and returns them one-by-one to the client using the *ReturnMachineState* callback method.
	Pseudo code:
	
	`DemandGroupMembership()`
	`FilterAllowedMachineNames(machineNames)`
	`foreach allowedMachine`
	`        QueryInformation`
	`        ReturnMachineState(info, allowed)`
	`end foreach`
	`foreach disallowedMachine`
	`        ReturnMachineState(emptyinfo, disallowed)`
	`end foreach`
	`ReportOperationFinished("images")`
	
	Actual parameters of methods called may differ.
	This method eventually realizes the “Check Free Disk Space” use case.

Method Signature	`bool RecursiveCopyFolders(string imagePath, FileSystemAccessRule folderRule, FileSystemAccessRule fileRule, List<string> machineNames, UftpServer uftpServer)`

Description	Recursively distributes files and folders under the root path specified in the *imagePath* parameter. It is called internally by the `CopyImage` method.
	Pseudo code:
	
	`RetrieveDirectoryInformation`
	`foreach large file`
	`        MulticastCopy(filename)`
	`end foreach`
	`foreach file`
	`        CreateRemoteDirectories(imagePath)`
	`        if small file then UnicastCopy(filename)`
	`        SetFileAccessRules(file)`
	`end foreach`
	`for each subdirectory`
	`        RecursiveCopyFolders(subdirectory, ...)`
	`end foreach`
	
	Actual parameters of methods called may differ.

Method Signature	`void StartUftpService(string[] machineNames)`
Description	Starts the UFTP daemons on the client machines specified in the *machineNames* parameter. It calls the *DemandGroupMembership* method to verify user authorization. This method eventually realizes the “Start UFTP Daemons” use case.

Method Signature	`void StopUftpService(string[] machineNames)`
Description	Stops the UFTP daemons on the client machines specified in the *machineNames* parameter. It calls the *DemandGroupMembership* method to verify user authorization. This method eventually realizes the “Stop UFTP Daemons” use case.

Method Signature	`void TurnOffMachines(string[] machineNames)`
Description	Turns off the client machines specified in the *machineNames* parameter. It calls the *DemandGroupMembership* method to verify user authorization. This method eventually realizes the “Shutdown Client Machines” use case.

Method Signature	`void TurnOnMachines(string[] machineNames)`
Description	Turns on the client machines specified in the *machineNames* parameter. It calls the *DemandGroupMembership* method to verify user authorization. This method eventually realizes the “Startup Client Machines” use case.

The following is the list of method signatures and descriptions in the *IImageDistributerServiceCallback* interface:

Method Signature	`void ReportMachineState(MachineState state, bool allowed)`
Description	Receives a *MachineState* object and a boolean value indicating whether the returned information belongs to an allowed or a disallowed machine.

Method Signature	`void ReturnImageList(string folderListXml);`
Description	Receives a string containing an XML structure that describes the folder hierarchy of the image folder on the server. The schema of the XML should be the following:

```
<xs:schema>
  <xs:element name="folders">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="folder" type="xs:element"/>
      <xs:attribute name="size" type="xs:integer"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="folder">
    <xs:complexType>
      <xs:attribute name="name" type="xs:string"/>
      <xs:attribute name="size" type="xs:integer"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Example XML:

```
<folders size="150">
  <folder name="subfolder1" size="100">
    <folder name="subfolder2" size="50" />
    <folder name="subfolder3" size="50" />
  </folder>
  <folder name="subfolder4" size="50" />
</folders>
```

Method Signature	`void ReportEvent(string message, bool error);`
Description	Receives a status messages, and a Boolean value that indicates if the message describes an error.

Method Signature	`void ReportOperationFinished(string operation);`
Description	Reports the finish of a long running operation that could send multiple status notifications during its process. The string parameter contains the name of the operation, such as "states", "images", or "copy".

3.2. Client Component

The client component is implemented as a Windows Forms application. The *Program* class contains the *Main* method which first checks whether the user trying to start the client has the appropriate credentials stated in the configuration file. If the credentials are correct it starts the application.

The *ImageDistributerClientForm* class represents the user interface and also contains the client logic, according to the development scheme of a Windows Forms application. It uses the service reference generated by the development environment as the proxy to communicate with the service component. Although not shown on the diagram, the *ImageDistributerClientForm* class implements the *IImageDistributerServiceCallback* interface (which is also generated in the service reference namespace).

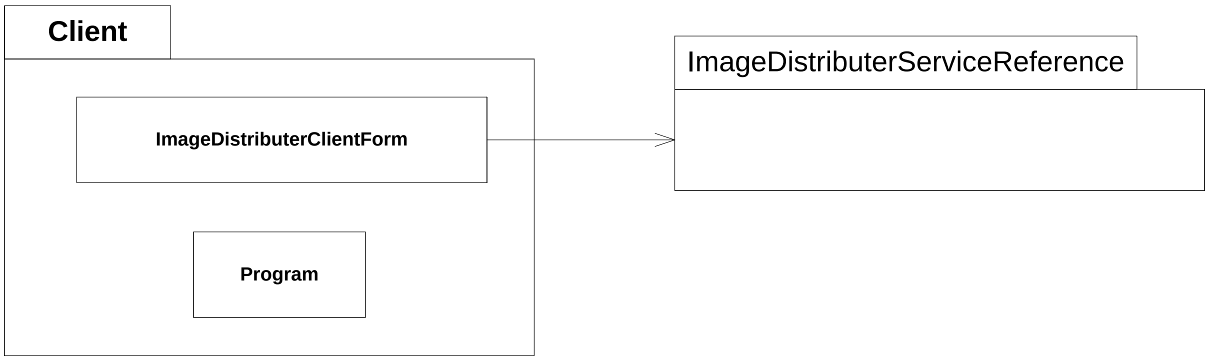
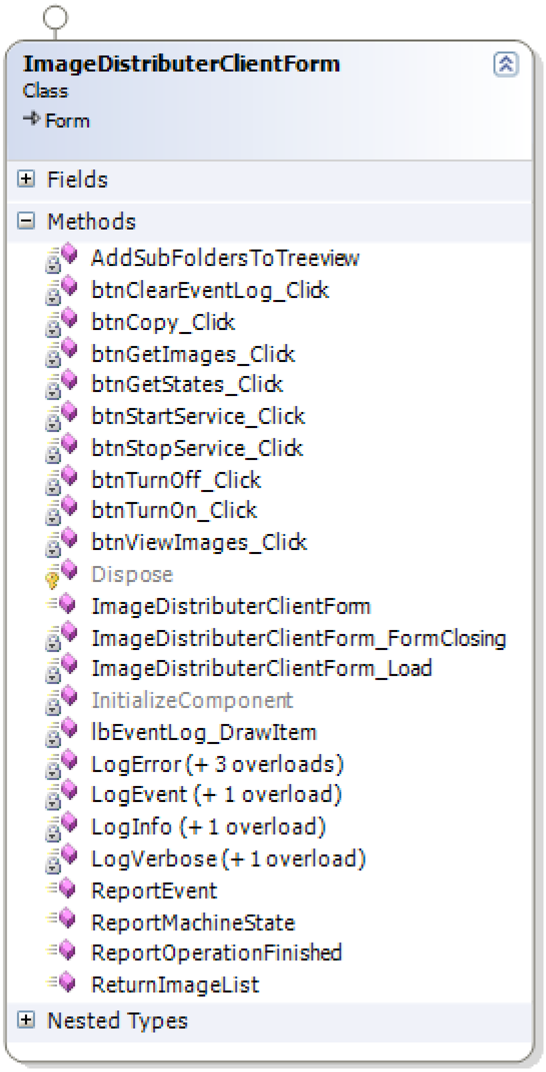


Figure 14: Composition of the client component's classes.

The following diagram contains the methods of the *ImageDistributerClientForm* class:



ImageDistributerClientForm
Class
→ Form

⊞ Fields
⊟ Methods
- AddSubFoldersToTreeview
- btnClearEventLog_Click
- btnCopy_Click
- btnGetImages_Click
- btnGetStates_Click
- btnStartService_Click
- btnStopService_Click
- btnTurnOff_Click
- btnTurnOn_Click
- btnViewImages_Click
- Dispose
- ImageDistributerClientForm
- ImageDistributerClientForm_FormClosing
- ImageDistributerClientForm_Load
- InitializeComponent
- lbEventLog_DrawItem
- LogError (+ 3 overloads)
- LogEvent (+ 1 overload)
- LogInfo (+ 1 overload)
- LogVerbose (+ 1 overload)
- ReportEvent
- ReportMachineState
- ReportOperationFinished
- ReturnImageList

⊞ Nested Types

Figure 15: Class diagram of the client component.

3.3. UFTP Daemon

The UFTP daemon is run by a Windows service that starts in the background with the parameters stored in the Registry. This Windows service was created by *András Kövi* independently of this project and has been installed on all machines in the laboratory, for it was used in the previous solution with the PowerShell scripts.

An alternative tool exists with the same functionality called *Applications as Services Utility* (`srvany.exe`) which can be found in the *Windows Server Resource Kit Tools* package (31).

3.4. Logging Component

Since the service component has no direct user interface, we needed a way to keep track of internal events without a debugging environment. Initially this was done by sending these messages to the Application Event Log, but this solution wasn't efficient on the long run. As a final solution we have decided to use a configurable logging framework for this task. Our choice fell upon the *Logging Application Block* from the *Microsoft Enterprise Library* (32).

The Enterprise Library is a set of "application blocks" which contain best practice code blocks for enterprise development. The Logging Application Block is a more flexible, extended version of the .NET Framework's built-in Debug and Trace features. It can be configured in the applications configuration file similarly to WCF services. Logging can be filtered by category or severity and sent to the appropriate outputs accordingly, such as plain text, XML files or even to the Windows Event Log.

For instance, the following piece of configuration is responsible for filtering out incoming log events other than errors:

```
<logFilters>
  <!-- Set enable to false to disable logging -->
  <add enabled="true"
      type="Microsoft.Practices.EnterpriseLibrary.Logging.Filters.
      LogEnabledFilter, Microsoft.Practices.EnterpriseLibrary.Logging,
      Version=4.1.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35"
      name="LogEnabled Filter" />
  <!-- Comment out the following item to enable verbose logging -->
  <add categoryFilterMode="DenyAllExceptAllowed"
      type="Microsoft.Practices.EnterpriseLibrary.Logging.Filters.
      CategoryFilter, Microsoft.Practices.EnterpriseLibrary.Logging,
      Version=4.1.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35"
      name="Category Filter">
    <categoryFilters>
      <add name="Error" />
    </categoryFilters>
  </add>
</logFilters>
```

This way we could configure multiple levels of logging, e.g. verbose logging is disabled by default, but it can be enabled to aid debugging sessions.

Logging was included in the client component as well yet again to assist in debugging.

More detailed information can be found in the documentation of the *Enterprise Library Logging Application Block* (33).

3.5. Test Implementations

The following section discusses the implementation of the test principles discussed in the Design chapter.

When the project was initially developed, it wasn't anticipated that it would evolve into a thesis project; therefore the design was not focused on testing. Since we decided not to restart the project with clean code, these test plans had to be designed in retrospect. Nevertheless the process was still a valuable experience for the future.

3.5.1. Features to Be Tested

The most important part of the system is the service component, specifically the *ImageDistributerService* class, which holds the implementation of the all the main use cases discussed in the design section. These operations are tested with unit testing.

The service provides its operations via the WCF Framework. Integration testing focuses on accessing the service component via this interface.

The client component is a typical Windows Forms application, where the code structure is mostly generated by the development environment. Since the operations cannot be separated from the user interface, testing of this element is done during manual system testing.

3.5.2. Features *Not* to Be Tested

The implementation was initially designed without testable components in mind; therefore it turned out that it would be quite hard to create unit tests for all the operations of the service.

When unit testing the external dependencies need to be emulated with test doubles (26). In our scenario, the service component relies on multiple external dependencies such as:

- The Windows service controller,
- Windows Communication Foundation static objects,
- Networking services for pinging,
- File system services.

The first three can be overridden and emulated with stubs, by removing the reference to their respective assemblies, but the file system features are wired into the *System* namespace, which cannot by unreferenced in .NET applications, thus unit tests cannot be made for operations that access the file system directly. It could only be achieved by major redesign with adding an extra layer of abstraction between the implementation and the file system functions, but that would also add unnecessary complexity at this stage.

3.5.3. Unit Testing Framework

The testing framework that comes with Visual Studio would've come handy, but as it turns out, it references the assembly directly from the tested project, thus eliminating the chances of creating stubs for originally referenced classes.

After seeing this we decided to create our own command line testing interface for the service component. The interface takes commands from the standard input and writes events and results to the standard output. The project is set up to copy the source files from the tested project into its own directory prior to each build. The client stub implements the callback channel and outputs all reported results to the standard output.

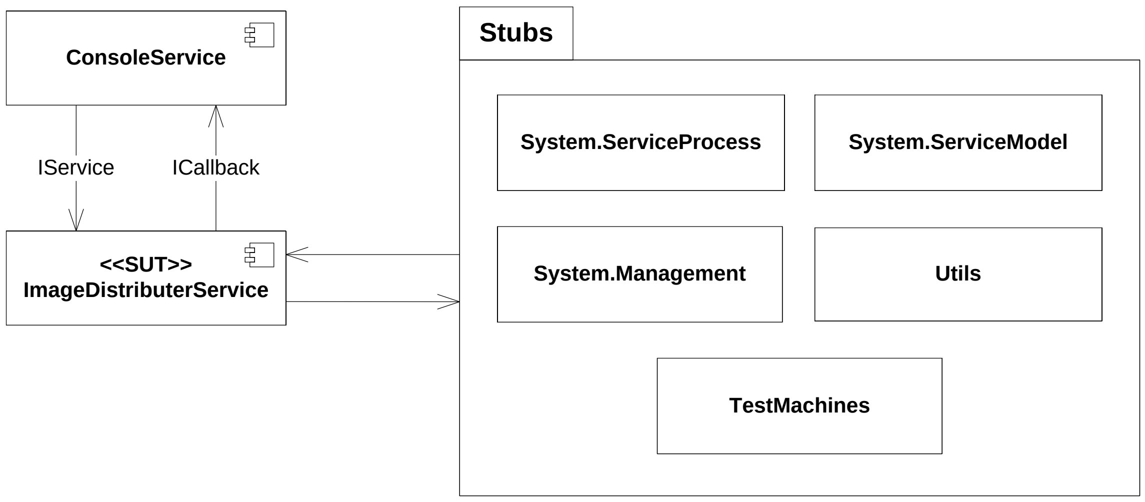


Figure 16: The architecture of the unit testing framework. SUT stands for System-Under-Testing.

The format of the commands is the following:

```
CommandName parameter1,parameter2,...
```

Lines starting with a hash mark (“#”) are treated as comments.

The list of available commands is the following:

- `GetMachineStates`
- `TurnOnMachines`
- `TurnOffMachines`
- `StartUftpService`
- `StopUftpService`

The commands resemble the names of the invoked operations and take a list of machine names as parameters.

The following is the list of namespaces that are created as stubs in order to emulate the external dependencies of the service component:

- *System.Management*, for emulating WMI related classes,
- *System.ServiceProcess*, for emulating the *ServiceController* class,
- *System.ServiceModel*, for emulating the WCF *OperationContext* class,
- *Utils* namespace for emulating the *UftpServer*, *WOL* and *Ping* utility classes,
- An extended *MachineState* object that implements the WCF contract specifications, but also adds additional features to aid the testing process.

These stubs act as their original counterparts, but the modifications are only reflected internally on a list of *MachineState* objects, that are initially loaded from an XML file named `machines.xml` (generated by serialization).

There is an additional command named *DumpMachines*, which requires no parameters, and outputs the list of internal machines, to verify that modifications were successful.

The operations *GetListOfImages* and *CopyImage* rely on file system operations, thus their testing will be done manually using a real life like setup on the development machine and in the laboratory environment.

The unit testing framework can be found in the `test\Unit` folder of the repository.

3.5.4. Integration Testing Framework

For integration testing I created a console version of the client that uses the same command format as the unit testing framework, except it has no *DumpMachines* command, since it doesn't maintain an internal list of machine states.

I have also created a version of the unit testing framework that references the real *System.ServiceModel* namespace and starts hosting the WCF service, however it still uses stubs instead of all the other dependencies.

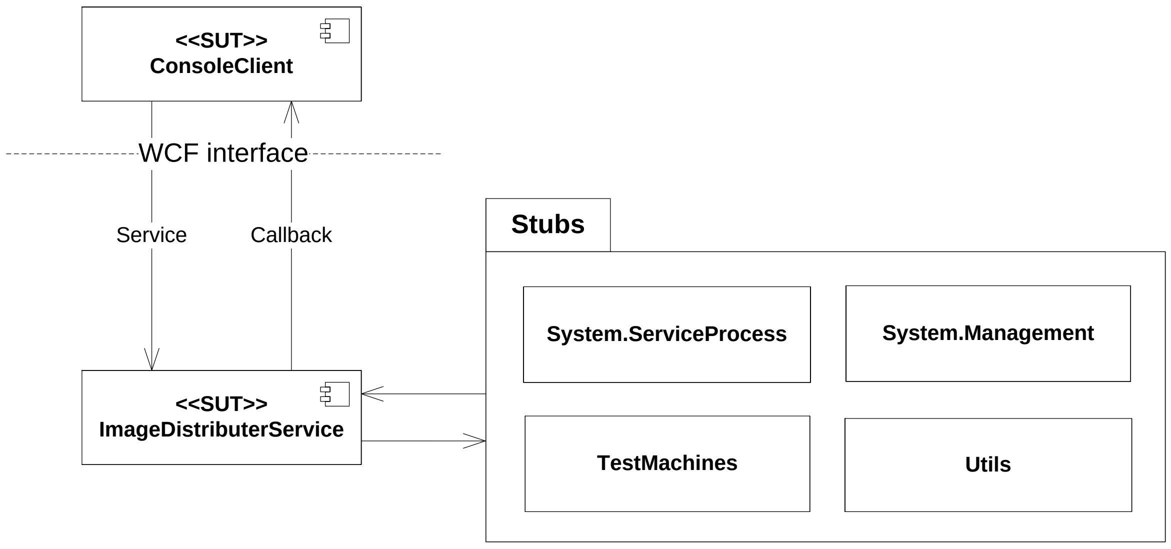


Figure 17: The architecture of the integration testing framework.

This setup enabled me to design a common set of test scripts for both unit and integration tests, with only minor differences, e.g. configuration verification at load time doesn't apply in integration testing.

The integration testing framework can be found in the `test\Integration` folder of the repository.

The batch file that runs all test scripts in the folder first starts the WCF host then starts running the test scripts. After all the scripts have finished running, the WCF host is shut down.

3.5.5. Test Scripts

The test scripts are placed in separate subfolders, the folder names being formatted as `ut_<testname>` for unit tests and `it_<testname>` for integration tests. The test name includes the postfixes _neg or _pos indicating whether the test scripts should have a negative or a positive outcome.

Each test script consists of the following elements:

- `config.xml` file for the service implementation (only in unit tests),
- `machine.xml` to provide test machine information (only in unit tests),
- `testinput.txt` containing commands for the test script,
- `expected.txt` containing the expected output of the test script.

The following batch files are used to evaluate test scripts and output results to the console or a file:

- `run_all_tests.cmd` runs all the tests found in the current directory (based on the `ut_*` or `it_*` filters).
- `log_all_tests.cmd` runs the previous batch file and redirects the output to the `unit_tests.log` file.
- `run_test.cmd` runs a single test, which is passed to it via the first command line argument.

The output of a test scripts is similar to the following:

```
------------------------------------------------------------
Running test:     TestName
Test result:      SUCCESS/FAIL
------------------------------------------------------------
```

I have defined a set of machines that is suitable for every test script that involves querying or amending their properties:

```
<TestMachineStates>
  <machines>
    <machine name="labpc1" turnedOn="true" freeSpace="6.5"
             uftpdServiceState="Running" ipAddress="10.40.100.10"
             macAddress="001111EF9273" />
    <machine name="labpc2" turnedOn="false" freeSpace="4"
             uftpdServiceState="Stopped" ipAddress="10.40.100.20"
             macAddress="001111890C88" />
    <machine name="labpc3" turnedOn="true" freeSpace="5"
             uftpdServiceState="Stopped" ipAddress="10.40.100.30"
             macAddress="001111B546FB" />
  </machines>
</TestMachineStates>
```

These machines have identical configuration as the machines in the laboratory with the same name.

I have defined the following test scripts for the service component (in alphabetical order):

Name	*DemandGroupMembership_neg*
Description	Tests whether access is granted to a client with no authorization. The *GroupName* setting in the configuration file is set to a group that the developer is not member of. Since *DemandGroupMembership* is a private method, it is tested by calling *GetMachineStates*.
Applies to	Unit test
Expected results	An error is returned with the message “Client has no authorization” and an exception is thrown because of the unsuccessful authentication (the exception message can be localized along with the framework).

Name	*GetMachineStates_neg*
Description	Tests the returned machine states for the *GetMachineStates* method with disallowed machine names.
Applies to	Unit and integration test
Expected results	The states are turned back empty with indication that they were disallowed.

Name	*GetMachineStates_pos*
Description	Tests the returned machine states for the *GetMachineStates* method with allowed machine names and compares them to the internal list.
Applies to	Unit and integration test
Expected results	The states are turned back with the same information as the internal machines dump.

Name	*InvalidConfig1_neg*
Description	Attempts to load the service implementation with a malformed *MachineNetworkInfo* entry in the config file. This test focuses on the verification features in the constructor of the service class. The following invalid line is inserted (contains invalid IP addresses): `invalid,257.1.1.1,255.0.0.0.0,ABCDEFABCDEF`
Applies to	Unit test only
Expected results	The service returns a “MachineNetworkInfo line in wrong format” error.

Name	*InvalidConfig2_neg*
Description	Same as previous test case. The following invalid line is inserted (contains invalid MAC address): `invalid,255.1.1.1,255.0.0.0,001111B546FBBB`
Applies to	Unit test only
Expected results	The service returns a "MachineNetworkInfo line in wrong format" error.

Name	*InvalidConfig3_neg*
Description	Same as previous test case. The following invalid line is inserted (contains a single string without any commas): `invalidline`
Applies to	Unit test only
Expected results	The service returns a "MachineNetworkInfo line in wrong format" error.

Name	*StartUftpService_neg*
Description	Tests the outcome of the *StartUftpService* method when called with machines that are turned off or the service is already started on them.
Applies to	Unit and integration test
Expected results	Errors returned for all machines.

Name	*StartUftpService_pos*
Description	Tests the outcome of the *StartUftpService* method when called with machines that are turned on and the service is not running on them.
Applies to	Unit and integration test
Expected results	The method throws no exceptions and the services are started on the appropriate machines.

Name	*StopUftpService_neg*
Description	Tests the outcome of the *StopUftpService* method when called with machines that are turned off or the service is not running on them.
Applies to	Unit and integration test
Expected results	Errors returned for all machines.

Name	*StopUftpService_pos*
Description	Tests the outcome of the *StartUftpService* method when called with machines that are turned on and the service is not running on them.
Applies to	Unit and integration test
Expected results	The method throws no exceptions and the services are stopped on the appropriate machines.

Name	*TurnOffMachines_neg*
Description	Tests the outcome of the *TurnOffMachines* method when called with machines that are already turned off.
Applies to	Unit and integration test
Expected results	Errors returned for all machines.

Name	*TurnOffMachines_pos*
Description	Tests the outcome of the *TurnOffMachines* method when called with machines that are turned on.
Applies to	Unit and integration test
Expected results	The method throws no exceptions and the appropriate machines are turned off.

Name	*TurnOnMachines_neg*
Description	Tests the outcome of the *TurnOnMachines* method when called with machines that are already turned on.
Applies to	Unit and integration test
Expected results	Errors returned for all machines.

Name	*TurnOnMachines_pos*
Description	Tests the outcome of the *TurnOnMachines* method when called with machines that are turned off.
Applies to	Unit and integration test
Expected results	The method throws no exceptions and the appropriate machines are turned on.

Logging is turned on in the configuration files by default, thus in case of an unexpected error, the log file can be examined for details.

These tests are configured to run on the development machine because configuration files must contain references to the executing environment, such as the machine name and user group names. Ideally the development environment should have the same parameters as the department server where final testing will be administered before deployment, otherwise a separate set of configuration files have to be created for the final testing environment.

3.5.6. System Testing

System testing will be performed manually according to a predefined script. The system passes the test if all the elements in the script have been achieved successfully.

Although no restriction was implemented, the service is intended for serving one client at a time. The general idea of multicast copying is to relieve the network of unnecessary load, performing two or more distribution processes simultaneously would negatively affect network performance. Therefore system testing can be performed by a single testing client at a time.

The configuration in the development environment should resemble that in the laboratory. If a build passes all tests in the development environment, it should be copied to the `dist` folder of the repository, into a subfolder indicating the build version. In the laboratory the system tests should be repeated with the latest build in the `dist` folder.

The tester should perform the following operations during a system test:

1. Rebuild the solution file (in the development environment or using the appropriate batch files in each project folder),
2. Verify parameters in the configuration files for both the service and the client,
3. Start the Windows service of the service component (with the batch files or the Services MMC module),
4. Start client user interface (`ImageDistributerClient.exe`),
5. Query images on server (press *Get Images* button),
6. Query machine states (select machines and press *Get States* button),
7. If possible try turning machines on and off (select machines and press *Turn On*/*Off* button, then *Get States* to verify changes),
8. Try starting and stopping UFTP daemon on machines (select machines and press *Start*/*Stop Service* button, the *Get States* to verify changes),
9. Copy image files to client machines (select image folder and target machines, press *Start Copy* button).

Notes are to be taken about unexpected events and the system’s logs should be archived for further analysis.

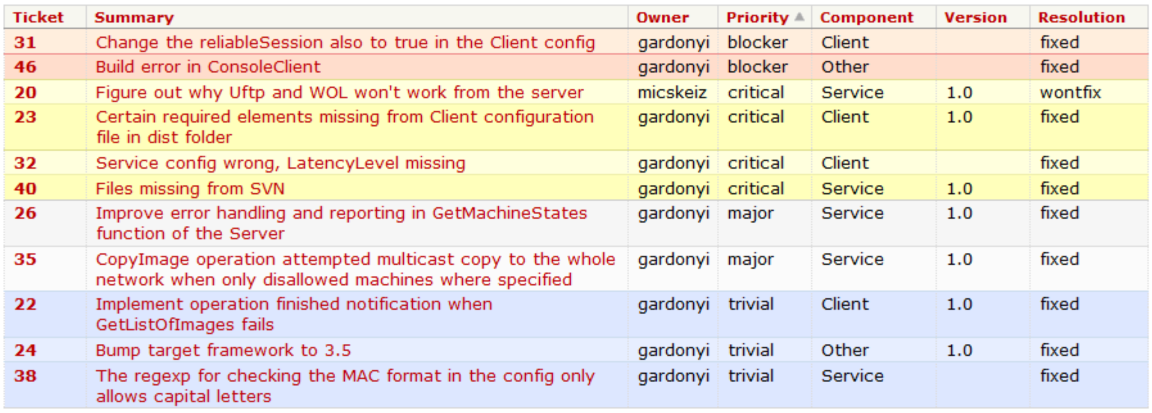
4. Testing Results

The latest build of the application fulfills all requirements and successfully passed all test scripts and manual testing sequences specified in the previous chapter. The last testing session in the target environment was considered as a successful acceptance testing; therefore we have labeled it as version 1.0.0.0 and made it ready for deployment.

The following chapter describes the issues we met while testing the system.

4.1. Fixed Issues

The following query results from the issue tracker show the list of tickets that were resolved during the current phase of the development:



Ticket	Summary	Owner	Priority ▲	Component	Version	Resolution
31	Change the reliableSession also to true in the Client config	gardonyi	blocker	Client		fixed
46	Build error in ConsoleClient	gardonyi	blocker	Other		fixed
20	Figure out why Uftp and WOL won't work from the server	micskeiz	critical	Service	1.0	wontfix
23	Certain required elements missing from Client configuration file in dist folder	gardonyi	critical	Client	1.0	fixed
32	Service config wrong, LatencyLevel missing	gardonyi	critical	Client		fixed
40	Files missing from SVN	gardonyi	critical	Service	1.0	fixed
26	Improve error handling and reporting in GetMachineStates function of the Server	gardonyi	major	Service	1.0	fixed
35	CopyImage operation attempted multicast copy to the whole network when only disallowed machines where specified	gardonyi	major	Service	1.0	fixed
22	Implement operation finished notification when GetListOfImages fails	gardonyi	trivial	Client	1.0	fixed
24	Bump target framework to 3.5	gardonyi	trivial	Other	1.0	fixed
38	The regexp for checking the MAC format in the config only allows capital letters	gardonyi	trivial	Service		fixed

Figure 18: Ticket results for resolved defects.

The categories of the issues in detail:

- **Configuration related:** tickets #23, #31 and #32 referred to inconsistencies in the numerous configuration files of the components. Different files are used for unit, integration and systematic testing that are also different from the ones used in the target environment. The issue was partly solved by reformatting all configurations to differ only in meaningful parameters, thus verification is manually possible with simple text comparison. A more robust configuration file manager solution is still to be designed.
- **Repository related:** ticket #40 refers to files missing from the repository. This usually happened after adding new files to the project or restructuring the existing hierarchy. The issue was resolved by the final reviewing of the repository structure and by introducing the practice of testing a new checkout after major changes.
- **Coding related:** unit testing and system testing revealed functionality flaws such as the ones described in tickets #22, #24, #26, #35, #38, #46 that needed amendments in the components' code.
- **Network related:** ticket #20 refers to network related issues when the UFTP tool and the WOL features did not work as expected. These issues had to be resolved independently of the applications testing.

4.2. Code Analysis Results

Visual Studio's built-in code analysis generated warnings in the following categories:

Type	*Microsoft.Naming*
Description	Indicated spelling errors in class and namespace names. The analyzer considered abbreviations like "uftp" and names containing the word "distributer" incorrect. Also suggested spelling for compound words like "filename" to be changed to "fileName".
Resolution	The abbreviations were obviously correct in spelling in the context of the project. The word "distributer" is a correct form of spelling in the English language, but the analyzer probably prefers the "distributor" spelling which is considered identical, although the latter is used far more often on the Internet (34). I have decided not to change the word to the alternative spelling, because that would also require correcting all the documentation and configuration files created up to date.

Type	*Microsoft.Design*
Description	Suggested missing elements of Microsoft .NET design patterns, e.g. strong naming of assemblies and the catching specific exception classes rather than the base *Exception* class.
Resolution	Strong naming was considered not yet necessary for this application will only be used internally. The catch blocks in question contain reporting and logging calls, without concern for what specific exceptions they report to the user or log to the log file, thus no changes are required.

Type	*Microsoft.Performance*
Description	Suggested that certain non-static methods of the implementation didn't use the *this* parameter, thus they would provide the same functionality when declared static.
Resolution	Added static modifier to the *DemandGroupMembership* method and the *Callback* property.

Type	*Microsoft.Usage*
Description	Suggested that specific exceptions should be created rather than the base *Exception* class. Also suggested that exceptions should be re-thrown instead of passed as parameters to another method to preserve stack location where the exception was initially caught.
Resolution	Although a rather formal suggestion, I modified the two created *Exception* classes to *FormatException* in the *ImageDistributerService* constructor, as they are thrown when the *machineNetworkInfo* setting is in wrong format. The latter exceptions were passed to logging functions where there was no need to keep track of the location where it was caught, the logging framework takes care of that with the Activity Tracing feature.

Type	*Microsoft.Security*
Description	Suggested possible security vulnerability because values from the configuration file are directly passed to *PrincipalPermission* class' constructor.
Resolution	This being an internally used project, the configuration file is guaranteed to be stored in secure place along with the service executable.

Type	*Microsoft.Globalization*
Description	Suggested that *string.Format* and *type.Parse* methods should be supplied with a specific culture info to make sure persisting of formatted values is not compromised by different environment settings.
Resolution	In an application that is likely to be used in environments with different culture settings, this can be considered an issue, but this project is created for a specific target environment.

4.3. Testing in the Target Environment

We regularly ran tests in the target environment to test newly implemented features and performed overall system tests. These testing sessions highlighted the following issues:

- The components must provide sufficient debug information even outside the development environment, because advanced debugging tools are typically not available in the target environment. To achieve this we have included configurable logging in each component, with optional verbose logging to help debugging.
- Multicast copying was tested in the development environment by distributing smaller images, than the ones in the laboratory, hence when testing with a real image the client timed out and disconnected from the service during the operation. This was resolved by setting a higher timeout period in the WCF configuration.
- Also while testing the Query images on server function in the development environment, the folder structure was moderate compared to the one that already existed on the laboratory server, which resulted in the feature not working in the laboratory. By tracing the WCF packets we discovered that the size of string parameters that could be passed to the remote methods was maximized in the configuration files and had to be set higher for the function to work properly.
- The Windows service did not fail immediately if the logging assemblies could not be loaded, and only sent unclear error messages to the Event Log when we tried to connect with a client. Since the logging framework was to help in debugging the service, our only option was to use an individual debugger to resolve the issue. An additional resolution was to add logging functions to the initialization code of the Windows service, hence if there's any problem with assembly loading the service will fail to start, thus narrowing the list of potential errors.

5. Installation Instructions

The following chapter discusses the installation and maintenance instructions of the application, divided by each component.

The `dist` folder contains the files required to install the system components. The component folders contain the executables, the configuration files and a *Lib* folder that contains the referenced assemblies and DLLs. The files are arranged under subfolders indicating release versions, e.g. `0.9.*` or `1.0.0.0`. The whole contents of the components' folder have to be deployed in order for them to work.

5.1. Service Component

Following are the installation instructions for the service component.

5.1.1. Requirements

.NET Framework 3.5 (or later) must be installed on the target machine.

5.1.2. Installation

The service has to be installed with the `installutil.exe` tool (comes with the .NET Framework). There is a batch file named `install_service.cmd` to execute the installation. There are two more batch files to start and to stop the service, without having to open the Services MMC module.

The service installer will be prompt for a username and password for under whose account the service will run. This user must have the rights to access the administrative shared folders (`c$`, `d$`) of the client machines, to execute remote WMI calls and to register an HTTP address on the machine (by default only Domain Administrator accounts have these rights). The username must be given in the `DOMAIN\UserName` format, otherwise the installation will fail.

5.1.3. Settings

The settings of the component are stored in the `ImageDistributerService.config` file beside the executable. The following table contains the significant configuration elements and their meanings. The elements indented are subelements of the previous unindented ones.

logConfiguration		Contains elements controlling the methods of logging. In the default configuration only the errors are logged to a log file and severe errors are also logged to the Event Log (under the source named "ImageDistributerService"). The Enterprise Library (32) comes equipped with a graphical editor tool, but some of the settings can be easily modified by editing the following elements in a text editor.
	listeners	These objects tell the logger where to log the events. By default there are two listeners installed, a *FlatFile TraceListener*, that logs to a text file and a *Formatted EventLog Listener* that logs to the Windows Event Log. The *fileName* attribute of the former specifies the filename in which the log file is created or appended and for the latter the *source* attribute indicates the Event Log source name.

	logFilters		There are two filters installed by default, the *LogEnabled* filter and the *Category* filter. Setting the *enabled* attribute of the *LogEnabled* filter to *false* disables logging entirely. By default the *Category* filter is set to let through the log entries only from the “Error” category. Putting XML comment tags (`<!-- -->`) around the “add” element of the *Category* filter enables verbose logging by letting through log entries from every category.
system.diagnostics			This element enables logging of WCF messages, so that the communication can be analyzed with the Service Trace Viewer utility. In the distributed `.config` file it is turned off by default.
system.serviceModel			WCF related settings.
	behaviours		This element contains *serviceBehavior* elements, which customize details of the service’s debug behavior.
		serviceMetadata	Enables WSDL metadata generation, which can be acquired via HTTP protocol.
		serviceDebug	Enables inner service exception details to be sent back to the client with *FaultExceptions*.
	endpoint		Under the *services/service* elements, this element provides the endpoint configuration, namely the exact protocols and URI on which the service should be available. More than one can be specified. The default setting in the distributed configuration file specifies an endpoint with *netTcpBinding* on port 9000. (There is also an endpoint named *mex* (short for *Metadata EXchange*) that serves metadata on port 9090.)
	bindings		The default setting enables encryption and digital signatures and message ordering. The default inactivity timeout is set to 1 hour, because the copy processes can take a long time to finish and it is possible that the client would time out in a shorter period.
applicationSettings			This element contains parameters of the business logic.
	GroupName		Only the group that is specified here has access to call the business logic. Users that do not belong to the group are denied access. **Format:** `DOMAIN\NameOfGroup` **Note:** After adding a user to a group, the changes are only applied after the user logs out and then logs back in.
	GroupToAddWriteAccess		The group specified here will get write access to the distributed files and folders. **Note:** In case this is a built-in group (e.g. Users, Administrators) the copy operation will fail with an *IdentityNotMapped* exception. This is probably a bug in the .NET Framework.
	MachineNetworkInfo		An XML serialized as an `ArrayOfStrings`, contains a the list of machines and their network information in the following format: `machineName,IPAddress,NetMask,MACAddress` The format of this information is verified when the service initializes and in case of syntax error it will not start and logs an exception to the Event Log.
	ImageFolderRoot		Path of the folder on the server that contains the images to be distributed.

	RemoteTempFolderPart	The path of the temporary folder to be used on the client machines. This is the location where the multicast copied files are first saved. Must contain an administrative shared folder (c$, d$, etc.) as root. (It is required to retrieve the free space available on the partition.) **Default value:** `c$\users\temp`.
	RemoteImageFolderPart	The path of the folder containing the images on the client machines. Must contain an administrative shared folder (c$, d$, etc.) as root. This is where the unicast copied files are saved and the multicast copied files are moved here after the transfer finishes.
	FileSizeLimitForMulticast	Files smaller than this value in bytes will be unicast copied to the target machines. This is required because multicast copying is not efficient for small files. **Default value:** `10000000` (~10 MB).
	UftpServiceName	The name of the Windows service that controls the UFTP daemon on the clients. **Default value:** `UftpService`.

	UftpServerConfiguration	An XML structure describing the parameters of the UftpServer. The parameters are described in detail on the UFTP homepage (1). The XML structure has corresponding elements to each parameter.
		An example configuration:
		
		`<UftpServer>`
		  `<Unicast>false</Unicast>`
		  `<Verbose>3</Verbose>`
		  `<Rate>0</Rate>`
		  `<Weight>0</Weight>`
		  `<MinTime>0</MinTime>`
		  `<TimeToLive>0</TimeToLive>`
		  `<LatencyLevel>0</LatencyLevel>`
		  `<Interface>192.168.1.2</Interface>`
		  `<Port>0</Port>`
		  `<Hosts>`
		    `<host>host1</host>`
		    `<host>host2</host>`
		  `</Hosts>`
		  `<LogFile>filename.ext</LogFile>`
		`<PublicMulticastAddress>230.4.4.1</PublicMulticastAddress>`
		`<PrivateMulticastAddress>230.5.5.x</PrivateMulticastAddress>`
		`<UftpServer>`
		
		The above configuration has every parameter set, but all elements are *optional* except for the root.
		Note: The XML structure must be surrounded by the `<![CDATA[ ]]>` tags in order to be processed correctly.

5.1.4. Operation

The component logs the following severe errors to the Event Log and also to the log file:

MachineNetworkInfo line in wrong format: {0}
A certain line in the *MachineNetworkInfo* configuration setting has a syntax error. {0} is replaced with the problematic line.

UftpServerConfiguration in wrong format
The *UftpServerConfiguration* setting has a syntax error. The message will also contain the problematic XML structure and the validation exception details.

System.TimeoutException or System.CommunicationException
The Windows service could not initialize the WCF service host. The WCF service is probably misconfigured.

The component logs the following non-severe errors to the log file. These can occur during an established client-server communication session thus the client is also notified of these events, but the exception details are only logged to the log file.

Client has no authorization.
The user that connected to the service with the client doesn't belong to the group specified in the *GroupName* setting.

Could not acquire UftpService status from {0}
An error occurred while acquiring UftpService status from a certain client ({0}). This could mean that the service cannot execute a WMI query because it has insufficient rights or the client is not responding, perhaps because the firewall blocked the WMI query.

Could not acquire free space from {0} (drive {1}:)
An error occurred while acquiring free disk space information from a certain client ({0}). This could mean that the service cannot execute a WMI query because it has insufficient rights or the client is not responding, perhaps because the firewall blocked the WMI query. It could also mean that the drive letter specified in the *RemoteImageFolderPart* setting doesn't exist on the client.

Could not retrieve states from {0}
An unknown error occurred while retrieving status information from a certain client ({0}). The detailed description of the exception is logged to the log file.

Could not send wake-up packet to {0}
An error occurred while sending a wake-up packet to a certain client ({0}). Probably a network error.

Could not shutdown {0}. WMI returned null or empty result.
WMI returned null or empty result when querying the *Win32Shutdown* object from a certain client ({0}). This could mean that the service cannot execute a WMI query because it has insufficient rights or the client is not responding, perhaps because the firewall blocked the WMI query.

Could not shutdown {0}.
An error occurred while calling the shutdown method of a certain client ({0}). The detailed description of the exception is logged.

Could not start remote service on {0}.
An error occurred while trying to start the UftpService on a certain client ({0}).

Could not stop remote service on {0}.
An error occurred while trying to stop the UftpService on a certain client ({0}).

Could not enumerate subfolders in images folder.
An error occurred while enumerating subfolders in the images folder on the server.

Cannot create access rule.
An error occurred while creating file system access rules. Check configuration for errors and verify that the service has sufficient permissions.

Image directory does not exist on server.
Check configuration for mistyped path in the *ImageFolderRoot* setting.

Cannot delete contents of remote temp folder.
Verify that the temporary folder exists on the selected clients and whether the service has write access to them.

Unknown error occurred while copying.
An unknown error occurred during the copy process. The detailed description of the related exception is also logged to the log file.

Unable to invoke UftpServer.
Check that the uftp.dll file is in the Lib folder relative to the path of the service executable.

Could not move files to destination folder.
An error occurred while moving the copied files to the destination folder on the clients. The detailed description of the related exception is logged to the log file.

Could not retrieve directory info.
An error occurred while retrieving directory information in the image folder being distributed. The detailed description of the related exception is logged to the log file.

Error occurred while copying: path\filename.ext.
The UFTP server returned an error while copying a certain file. The detail of the error is logged to the log file.

Cannot copy to disallowed machines.
The user tried copying images to a machine that is not configured in the configuration file of the service.

5.2. Client Component

Following are the installation instructions for the client component.

5.2.1. Requirements

.NET Framework 3.5 (or later) must be installed on the target machine.

5.2.2. Installation

The client does not require installation. It only needs the `ImageDistributerClient.config` file and the `Lib` folder containing the Enterprise Library Logging Application Block assemblies to be in the same folder as the executable.

The client can be run on any machine in the domain, the only restriction is that the executing user must be in the group specified in the *GroupName* setting in both the client's and the service's configuration file. Obviously the configuration file of the client can be tampered with; nevertheless the service will not give access to unauthorized users. The setting in the client's configuration only serves warning purposes.

5.2.3. Settings

The settings of the component are stored in the `.config` file beside the executable. The following elements and their effects are considered significant:

logConfiguration		Contains elements controlling the methods of logging. In the default configuration only the errors are logged to a log file. The Enterprise Library (32) comes equipped with a graphical editor tool, but some of the settings can be easily modified by editing the following elements in a text editor.
	listeners	These objects tell the logger where to log the events. By default there is only one listener installed, namely a FlatFile TraceListener that logs to a text file, the filename is specified by the *fileName* attribute.
	logFilters	There are two filters installed by default, the *LogEnabled* filter and the *Category* filter. Setting the *enabled* attribute of the *LogEnabled* filter to false disables logging completely. By default the *Category* filter is set to let through the log entries only from the "Error" category. Putting XML comment tags (`<!-- -->`) around the "add" element of the *Category* filter enables verbose logging by letting through log entries from every category.
system.serviceModel		WCF related settings. These settings must match the settings in the service in order for the connection to be established. The main difference is that instead of being defined under the *services* element the endpoints are defined under the *client* element.

applicationSettings		This element contains parameters of the client component logic.
	GroupName	Only the group that is specified here has access to call the business logic. Users that do not belong to the group are denied access. The group name specified in the service's configuration overrules this setting. **Format:** `DOMAIN\NameOfGroup` **Note:** After adding a user to a group, the changes are only applied after the user logs out and then logs back in.
	machineNames	Comma separated list of machine names to be displayed on the UI. By specifying the list of machines in the client configuration, it is possible to create individual configuration files for the discrete sets of machines in the laboratory. The machines specified here must also be specified in the configuration of the service component, because access will only be granted for the latter.
	RemoteImageFolderRoot	Path of a shared folder name on all clients that the user has access to. This is the folder that opens in an Explorer window when the user clicks on the *View Images* button for the selected client. The installation script for the UFTP daemon component will be responsible for creating the shared folders and as long as the users in the specified group have access to the folders in the file system, they will be able to reach it through the shared folder.

5.2.4. Operation

The component logs the following errors to the event log on the user interface and also the client log file:

Could not call the remote service.
An error occurred while calling the WCF service. Check whether the service is running and the configuration contains the correct endpoint specifications.

Could not add the returned image names to the treeview.
The service returned an invalid XML folder hierarchy. Enable verbose logging to output the XML string to the log file.

Could not close connection on exit.
An error occurred while closing the proxy instance on exiting.

There are other errors that can occur and show up in the event list, the list of those exceptions can be found in the previous section.

5.3. UFTP Daemon

Following are the installation instructions for the UFTP daemon.

5.3.1. Requirements

.NET Framework 3.0 (or later) must be installed on the target machine.

5.3.2. Installation

The `dist\UftpService` directory contains a batch file named `setup.cmd` which performs the operations needed to install the daemon on the client. These commands are the following:

- Creates the `C:\Program Files\Uftp` directory and copies the executable files into it,
- Creates the temporary directory (`C:\Users\Temp` by default) in which the files are saved during distribution,
- Creates a firewall rule to enable WMI and ping services and also enables the `Uftpd.exe` executable to open a port on the firewall,
- Creates a shared folder pointing to the folder containing the images on the machine, so that they can be accessed by the client component,
- Installs the Windows service that runs the UFTP daemon. It also inserts the contents of the `uftpserviceinstall.reg` file to the registry, which contains the command line parameters for the service.

The directory and file paths in the batch file reflect the scenario in which the daemon was used with the PowerShell script, in case of configuration changes, the appropriate parameters will need to be amended.

5.4. Group Policy Settings

Group policy settings can be used to centrally control the configuration and access rights of client machines, which is much safer than allowing local modifications. Since the laboratory operates in an Active Directory domain, it was obvious to use group policies to distribute settings to the clients. Therefore the firewall related settings in the UFTP daemon’s installation script has been commented out and deployed centrally by group policy settings.

These settings are available in the *Computer Management* console, via the following path:

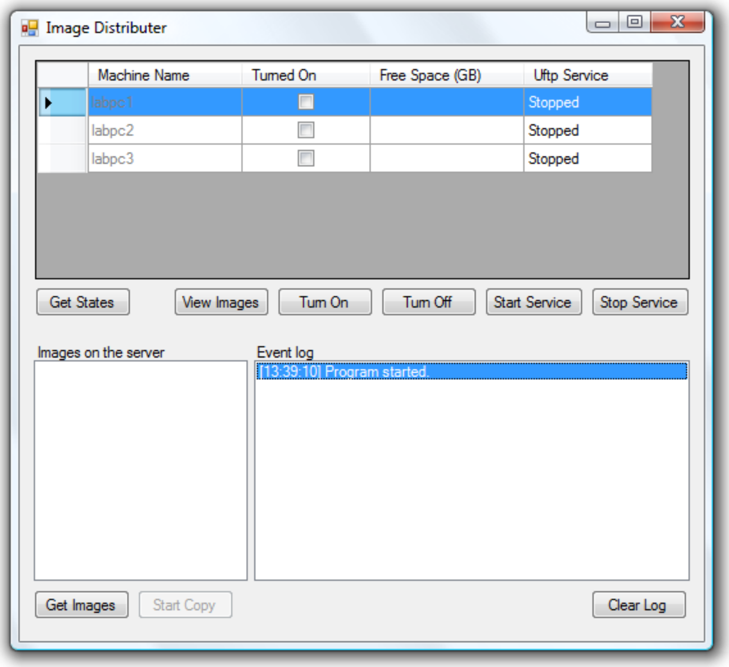
```
Computer Configuration\Administrative Templates\Network\Network
Connections\Windows Firewall\Domain Profile
```

The following policies have to be enabled:

- Windows Firewall: Allow remote administration exception
- Windows Firewall: Allow ICMP exceptions
- Windows Firewall: Define program exceptions
    - Value: `C:\Program Files\Uftpd\uftpd.exe:localsubnet:enabled:UFTPD`

6. User Manual

The end users of the system will be using the client component, in other words the user interface:



Machine Name	Turned On	Free Space (GB)	Uftp Service
labpc1	☐		Stopped
labpc2	☐		Stopped
labpc3	☐		Stopped

```
[13:39:10] Program started
```

Figure 19: The user interface after startup.

The list control on the bottom right side is the **Event log**. All events and errors are reported in this list. The events in the list can be cleared by pressing the **Clear Log** button.

The grid on the top contains the list of machines configured in the configuration file of the client. Initially it contains no data other than the machine names. Press the **Get States** button to retrieve information for the selected machines. After pressing the button, the name of the machines stay gray, but as the information is received their background turns green. The background of the selected machines that are not configured on the server will turn to red instead of green and an error message is reported in the **Event log**.

Other features related to the selected machines:

View Images	Opens a file browser window pointing to the shared folder containing the distributed image files on the selected machine(s).
Turn On	Sends a turn on signal to the selected machine(s). Changes aren’t reflected immediately, only after retrieving the information for the selected machines again.
Turn Off	Sends a turn off signal to the selected machines. Changes aren’t reflected immediately, only after retrieving the information for the selected machines again.
Start Service	Starts the UFTP daemon on the selected machines. Changes aren’t reflected immediately, only after retrieving the information for the selected machines again.
Stop Service	Stops the UFTP daemon on the selected machines. Changes aren’t reflected immediately, only after retrieving the information for the selected machines again.

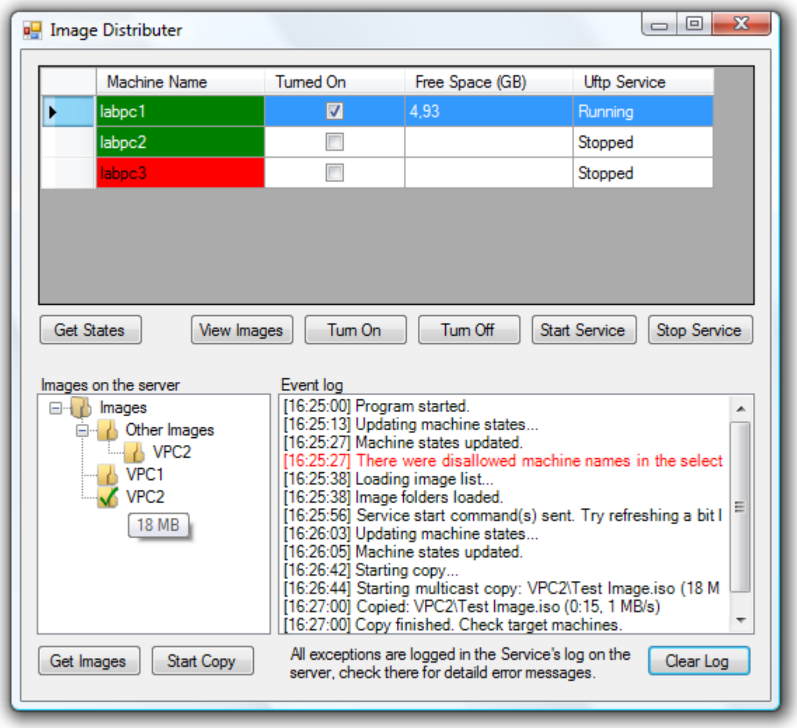
All of the above features first verify that the operation in question is necessary, e.g. the UFTP daemon cannot be started on machines that are turned off, and machines that are already turned on cannot be turned on again. A notification is logged of these situations into the **Event log**.

The treeview control on the bottom left side displays the image folders that exist on the server, which can be retrieved by press the **Get Images** button. The folders are displayed hierarchically. The size of a certain folder will be displayed in a tooltip by hovering with the mouse cursor over a certain tree node.

To start a distribution process the target machines and the image folder to be copied must be selected first. This doesn't necessarily have to be a leaf node, since the subfolders are copied recursively. Distribution is started by pressing the **Start Copy** button. Status events and errors during will be displayed in the **Event log**.

Please keep in mind that the connection between the client and the service times out after 1 hour. Should the client be left inactive for a long time, it might need to be restarted to successfully administer a distribution process. If the distribution takes longer than an hour, the administrator must increase the timeout period in the client and the service configuration likewise.

The following screenshot shows the user interface after querying the information of the test machines, including one deliberately disallowed, loading the list of images, and finally performing the distribution of an image folder:



Machine Name	Turned On	Free Space (GB)	Uftp Service
labpc1	☑	4.93	Running
labpc2	☐		Stopped
labpc3	☐		Stopped

```
[16:25:00] Program started.
[16:25:13] Updating machine states...
[16:25:27] Machine states updated.
[16:25:27] There were disallowed machine names in the select
[16:25:38] Loading image list...
[16:25:38] Image folders loaded.
[16:25:56] Service start command(s) sent. Try refreshing a bit l
[16:26:03] Updating machine states...
[16:26:05] Machine states updated.
[16:26:42] Starting copy...
[16:26:44] Starting multicast copy: VPC2\Test Image.iso (18 M
[16:27:00] Copied: VPC2\Test Image.iso (0:15, 1 MB/s)
[16:27:00] Copy finished. Check target machines.
```

All exceptions are logged in the Service's log on the server, check there for detaild error messages.

Figure 20: The interface after a test copy.

7. Conclusions

The following chapter serves as the epilogue of the thesis. It summarizes the achieved goals and lessons derived from the development process. It also reveals the future of the project.

7.1. Evaluation of the Solution

The development of the application is considered successful as it fulfills requirements specified on all levels and passed acceptance testing. The following is the detailed analysis of the implemented requirements.

Implementation of high-level requirements (section 1.4):

- The application uses the very same UFTP tool, thus the efficiency of multicast copying did not change,
- The authorization is solved by integration with the existing Active Directory environment, with access granted only to selected users,
- The user interface was designed according to end user preferences and successfully passed acceptance testing.

Use cases (section 2.1.1) are implemented as operations of the service component as described in section 3.1.

Implementation of non-functional requirements (section 2.1.2):

- **Fast Copying:** The UFTP tool was ported to a Dynamic Link Library (DLL) which allowed more seamless integration with the service, but the actual workings of the tool was not changed, nor used in a more complex way than with the original PowerShell scripted solution.
- **Central Manageability:** The application's client can be used to manage the whole distribution process from turning on the machines to turning them off after finishing, from a single client machine.
- **Asynchronous Operation:** Interaction between the client and the service component are performed as one-way WCF methods, thus the client is not being blocked during the execution of the actual operations.
- **Security:** The components communicate via a secured WCF channel and authorization checking is enforced by the service component, while the client will display warnings if access will not be granted or a disallowed machine was found in the configuration.

As additional requirement of the thesis, systematic testing was introduced to ensure the quality of the current and future versions of the application.

7.2. Lessons Learned

This thesis went over the development cycles of a software application that is only intended for inside use in the department's laboratory. Nevertheless the process can be drawn parallel with the development of larger scale software systems, which I am hoping to be part of some day.

However small it may be, this project served as a valuable lesson of teamwork, e.g. files were sometimes missing from the repository, because they weren't automatically added upon creation. Thus adding new files and restructuring the hierarchy requires extensive care and the occasional smoke testing of the repository itself.

Even with a highly profound test plan, unexpected situations will arise, for instance something that passes all tests in the development environment, doesn't work at all in the target environment. It is probably a configuration error, a buffer size set too low, which doesn't cause any symptoms during the initial testing if it was done with a smaller amount of test data, but identifying the cause is only possible with the sufficient amount of debugging information available. This led to the recognition of the need for debugging without an advanced Integrated Development Environment. The laboratory machines do not have Visual Studio installed and the convenient debugging tools were not available while testing there. As a solution to this we implemented efficient verbose logging of internal operations. The .NET Framework was also very helpful in this respect, because it contains the complete tool chain to build projects without the need for a separate SDK.

Such experiences could not have been earned otherwise than contributing in a similar project and without the devoted help of my advisor.

7.3. Future Plans

The project doesn't end here, although we have achieved the initial goal to present a usable system, still there are unresolved tickets in the issue tracking system. Some of them minor defects, some of them enhancements, or new ideas for features, e.g. the ability to abort the distribution process or reporting detailed progress of the multicast copying to the client. On the development side the automatic versioning and the management of configuration files need further improvements.

It is also considered to create a web interface, i.e. thin client instead of or alongside to the current thick client component.

Meanwhile the department staff can begin using the application for its designated purpose starting next semester.

References

(1) Bush D. UFTP homepage. [Online]. Available from: http://www.tcnj.edu/~bush/uftp.html.

(2) IBM. IBM - Provisioning Software - Tivoli Provisioning Manager - Software. [Online]. Available from: http://www.ibm.com/software/tivoli/products/prov-mgr/.

(3) Microsoft Corporation. System Center Configuration Manager: Home Page. [Online]. Available from: http://www.microsoft.com/systemcenter/configurationmanager/en/us/default.aspx.

(4) Buttyán L, Györfi L, Győr S, István V. Kódolástechnika jegyzet. [Online]. 2006. Available from: http://www.crysys.hu/courses/kodolastechnika/bscinfkod.pdf.

(5) Cohen B. The BitTorrent Specification Protocol Specification. [Online]. Available from: http://www.bittorrent.org/beps/bep_0003.html.

(6) Wikipedia. Comparison of BitTorrent clients. [Online]. Available from: http://en.wikipedia.org/wiki/Comparison_of_BitTorrent_clients.

(7) Rasterbar Software. libtorrent Homepage. [Online]. Available from: http://www.rasterbar.com/products/libtorrent/.

(8) Microsoft Corporation. Windows Communication Foundation. [Online]. Available from: http://msdn.microsoft.com/en-us/netframework/aa663324.aspx.

(9) Microsoft Corporation..NET Framework Development Center. [Online]. Available from: http://msdn.microsoft.com/en-us/netframework/default.aspx.

(10) Douglas K. B. Service-oriented architecture (SOA) definition. [Online]. Available from: http://www.service-architecture.com/web-services/articles/service-oriented_architecture_soa_definition.html.

(11) Lieberman P. White Paper: Wake On LAN Technology. [Online]. 2002. Available from: http://www.liebsoft.com/index.cfm/whitepapers/Wake_On_LAN.

(12) Microsoft Corporation. Shutdown Method of the Win32_OperatingSystem Class. [Online]. Available from: http://msdn.microsoft.com/en-us/library/aa393627.aspx.

(13) Zoltán B. Windows Communication Foundation - a kommunikációs réteg. [Online]. 2007. Available from: http://www.devportal.hu/Portal/Detailed.aspx?NewsId=6c10d424-df70-4b66-b64e-e2dd2404b8cb.

(14) Microsoft Corporation. Windows Communication Foundation Architecture Overview. [Online]. 2006. Available from: http://msdn2.microsoft.com/en-us/library/aa480210.aspx.

(15) Microsoft Corporation. Windows Communication Foundation Documentation. [Online]. 2007. Available from: http://msdn2.microsoft.com/en-us/library/ms735119.aspx.

(16) Peiris C, Mulder D. Hosting and Consuming WCF Services. [Online]. 2007. Available from: http://msdn2.microsoft.com/en-us/library/bb332338.aspx.

(17) Microsoft Corporation. Microsoft Developer Network Academic Alliance Homepage at BME-VIK. [Online]. Available from: http://msdnaa.bme.hu/.

(18) Open Source Community. Subversion. [Online]. Available from: http://subversion.tigris.org/.

(19) Edgewall Software. Trac. [Online]. Available from: http://trac.edgewall.org/.

(20) Microsoft Corporation. Windows SDK. [Online]. Available from: http://msdn.microsoft.com/en-us/windowsserver/bb980924.aspx.

(21) Microsoft Corporation. Mdbg Download. [Online]. 2006. Available from: http://www.microsoft.com/downloads/details.aspx?familyid=38449a42-6b7a-4e28-80ce-c55645ab1310&displaylang=en.

(22) Stall M. Mike Stall's.NET Debugging Blog. [Online]. 2005. Available from: http://blogs.msdn.com/jmstall/archive/2005/11/08/mdbg_linkfest.aspx.

(23) Microsoft Corporation. SysInternals. [Online]. Available from: http://www.microsoft.com/technet/sysinternals/default.mspx.

(24) Open Source Community. TortoiseSVN. [Online]. Available from: http://tortoisesvn.tigris.org/.

(25) Green P. Chromium Blog: Putting It to Test. [Online]. 2008. Available from: http://blog.chromium.org/2008/11/putting-it-to-test.html.

(26) Fowler M. Mocks Aren't Stubs. [Online]. 2007. Available from: http://martinfowler.com/articles/mocksArentStubs.html.

(27) Wikipedia. List of unit testing frameworks. [Online]. Available from: http://en.wikipedia.org/wiki/List_of_unit_testing_frameworks.

(28) Wikipedia. List of tools for static code analysis. [Online]. Available from: http://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis.

(29) Microsoft Corporation. FxCop. [Online]. Available from: http://msdn.microsoft.com/en-us/library/bb429476.aspx.

(30) J. Foster J. Calculate Code Metrics using Visual Studio 2008. [Online]. 2008. Available from: http://www.jamesjfoster.com/blog/2008/10/11/CalculateCodeMetricsUsingVisualStudio2008.aspx.

(31) Microsoft Corporation. Windows Server 2003 Resource Kit Tools. [Online]. 2003. Available from: http://www.microsoft.com/downloads/details.aspx?FamilyID=9D467A69-57FF-4AE7-96EE-B18C4790CFFD.

(32) Microsoft Corporation. Enterprise Library. [Online]. Available from: http://msdn.microsoft.com/en-us/library/cc467894.aspx.

(33) Microsoft Corporation. The Logging Application Block. [Online]. 2008. Available from: http://msdn.microsoft.com/en-us/library/dd139916.aspx.

(34) Google Fight. Distributer vs. Distributor. [Online]. Available from: http://www.googlefight.com/index.php?lang=en_GB&word1=distributer&word2=distributor.

(35) Microsoft Corporation..NET Framework Development Center. [Online]. Available from: http://msdn.microsoft.com/en-us/netframework/default.aspx.

(36) Foster JJ. Calculate Code Metrics using Visual Studio 2008. [Online]. 2008.