Department of Computer Science

Anders Brännström and Rickard Nilsson

Investigating and Implementing a DNS Administration System

Degree Project (20p)

Master of Science in Computer Engineering

Date:	07-01-18
Supervisor:	Hans Hedbom
Examiner:	Donald Ross
Serial Number:	D2007:03

Karlstads universitet 651 88 Karlstad

Tfn 054-700 10 00 Fax 054-700 14 60

Information@kau.se www.kau.se

Investigating and Implementing a DNS Administration System

Anders Brännström and Rickard Nilsson

This report is submitted in partial fulfillment of the requirements for the Master's degree in Computer Science. All material in this report which is not my own work has been identified and no material is included for which a degree has previously been conferred.

Anders Brännström and Rickard Nilsson

Approved, 070118

Advisor: Hans Hedbom

Examiner: Donald Ross

Abstract

NinetechGruppen AB is an IT service providing company with about 30 employees, primarily based in Karlstad, Sweden. The company began to have problems with their DNS administration because the number of administrated domains had grown too large. A single employee was responsible for all the administration, and text editors were used for modifying the DNS configuration files directly on the name servers. This was an error prone process which also easily led to inconsistencies between the documentation and the real world.

NinetechGruppen AB decided to solve the administrative problems by incorporating a DNS administration system, either by using an existing product or by developing a new system internally. This thesis describes the process of simplifying the DNS administration procedures of NinetechGruppen AB.

Initially, an investigation was conducted where existing DNS administration tools were sought for, and evaluated against the requirements the company had on the new system.

The system was going to have a web administration interface, which was to be developed in ASP.NET 2.0 with C# as programming language. The administration interface had to run on Windows, use SQL Server 2005 as backend database server, and base access control on Active Directory. Further, the system had to be able of integrating customer handling with the domain administration, and any changes to the system information had to follow the Information Technology Infrastructure Library change management process.

The name servers were running the popular name server software BIND and ran on two different Linux distributions – Red Hat Linux 9 and SUSE Linux 10.0.

The investigation concluded that no existing system satisfied the requirements; hence a new system was to be developed, streamlined for the use at NinetechGruppen AB.
A requirement specification and a functional description was created and used as the basis for the development. The finalized system satisfies all necessary requirements to some extent, and most of them are fully satisfied.

Contents

List of Figures

List of tables

1 Introduction

1.1 Executive summary

NinetechGruppen AB, a Swedish company with about 30 employees, offers services such as development, operation, and administration of IT systems. The company was having troubles with their DNS administration since the number of administrated domains had become several hundred, and all administration was made by editing text files on the name servers. This error prone approach easily led to inconsistencies between the documentation and the actual information on the name servers. Hence, the company decided to launch a project aimed for finding a solution to the problem.

A system was to be used that automated the administrative process, and the most important requirements on the system were:

- The system had to consist of a web administration interface running on Windows, developed with ASP.NET 2.0 using C# as programming language and SQL Server as database.
- The domain handling were to be integrated with customer handling – and all information were to be gathered in a database.
- Customer handling and domain handling were to be integrated by gathering all data in a database.
- The Information Technology Infrastructure Library (ITIL) change management process, see section 2.4, was to be used in the system.

First, an investigation was conducted to find existing DNS administration products and see if those met the requirements of the company. Three of the systems seemed most interesting – The Men & Mice Suite, Nixu Namesurfer, and NetDirector. Those were scrutinized closely and compared to a set of criteria, but in the end – none of the system could satisfy the company's needs. Therefore, the project moved into the next phase, a construction of an administration system streamlined for NinetechGruppen's use.

The first step was to create a requirement specification, upon which the implementation was based. The requirements were separated into two parts, shall- and should-requirements.

The shall-requirements were necessary while the should-requirements were more of a wish-list.

Several possible solutions to various problems were discussed and tested, for instance how to handle communication between the web server and the name servers. In the end, a system was constructed that met all the shall-requirements to some extent, and fully satisfied most of them. Unfortunately, there was not time available to implement many should-requirements. However, the system became a stable product with the necessary functionality in place.

The system can administer customers, domains, and resource records – and the updates are automatically replicated to the name servers. Transactions are used to protect the system data from inconsistencies and Active Directory group membership is the basis for access control.

As future work, the system might be extended with additional functionality such as group operations, searching features, and an interface for adding and removing managed name servers.

1.2 Reading guidelines

This thesis is primarily written for readers with at least basic knowledge in computer science. The background chapters provide a description for relevant areas, but since this thesis alone cannot cover every single related topic, additional knowledge on a basic level is required to fully understand the thesis contents.

To get the most out of the thesis, the reader ought to read the chapters sequentially. However, section 2.3 to 2.8 might be skipped at will if the reader feels comfortable with the subjects.

Thesis structure:

- **Chapter 2 – Background**
  This chapter provides background information along with an explanation of the related areas – DNS, ITIL, Active Directory, SQL Server, ASP.NET, C#, Web application architecture, and the Enterprise Library.
- **Chapter 3 – Investigation**
  This chapter describes the investigation, where several DNS administration tools are tested and evaluated.
- **Chapter 4 – Implementation**
  This chapter describes some different implementation choices, reasons to why the chosen ideas were used, and a high level system description.

- **Chapter 5 – Results and evaluation**

  This chapter evaluates the system against the company requirements and the criteria used in chapter 3. Further, an explanation of the most difficult problems encountered during the project is included.
- **Chapter 6 – Conclusions**

  This chapter provides some general conclusions and a discussion of future work.
- **Appendix A – Requirement Specification**

  This appendix is a translated version of the shall- and should-requirements found in the original requirements document [51].
- **Appendix B – System Description**

  This appendix describes the interfaces for the business object model.

2 Background

2.1 Introduction

NinetechGruppen AB, referred to as Ninetech in the rest of this thesis, is a company with about 30 employees with offices in Karlstad and Västerås in Sweden and Oslo in Norway. Ninetech offers e.g. development, operation, administration, and support of IT systems and environments - ranging from web hotels to complete systems. Ninetech also acts as an Internet service provider and handles Internet access and domain registrations.

Ninetech is responsible for the DNS administration of several hundred domains for different companies. The administration of the domains was earlier performed by a single employee using a text editor and a spread sheet. The process included editing cryptic text files and at the same time keeping the documentation in the spread sheet up to date. This was a very error prone process and could therefore only be performed by an employee who more or less knew the system by heart. The time and money spent on administrating the domains could be better invested elsewhere in the organization.

The company is in the process of introducing ITIL into their organization. ITIL (see section 2.4 for more information) is a set of best practices for organizations delivering IT services, e.g. describing change management processes (see section 2.4 for more information). Ninetech decided to start a project aimed for finding a solution to the current administrative problems, and the solution had to incorporate the ITIL change management process. By doing this, Ninetech hoped that other employees would be able to help out with the DNS administration in a simple, safe, and efficient manner. Thus in the long run benefit both the customers and the company.

The first step of the project was to formulate a requirement specification in collaboration with Ninetech. The solution should be an administration system according to the requirements further described in the next section.

Secondly, an investigation of existing solutions was performed and the following step would depend on the outcome of the investigation. Either an existing solution would be deployed, and possibly extended, or a completely new system would be developed.

The investigation concluded that no suitable existing solution existed that adequately satisfied Ninetech's needs. Hence, the project's final step was to design and implement a DNS administration system completely according to the formulated requirements.

2.2 Project requirements

The system had to conform to the ITIL standards and adhere to the current IT-infrastructure of the company. Further, a web user interface was required – making the administration possible with a standard web browser. For a summary description of the requirements in English, see appendix A or the full requirements document in Swedish [51].

Ninetech basically had three options. First, they could acquire a system off the shelf, commercial or open source. Before purchasing a commercial product, the cost of the product needed to be examined. Ninetech did not wish that the cost of the system should exceed the expected salary of the developers if a new system was to be implemented. Second, Ninetech could develop a whole new system internally. Finally, Ninetech could modify and/or build upon an existing system to fit their needs. Extendibility was important in this scenario if the system would need to be customized to meet the requirements.

If Ninetech decided to implement a whole or a part of a system themselves, they did not wish to make the source code publicly available in case they wanted to distribute or sell the system further on. Further, if modification was considered necessary, the system's license had to be scrutinized to avoid any potential license issues. If they chose to develop a new system internally Ninetech would have all rights to the source code themselves, thus avoiding license issues.

During project startup, Ninetech was using two name servers with different operating systems, Red Hat Linux 9 and SUSE Linux 10, and the name server software in use was ISC BIND version 9. Further, most web servers in Ninetech's server farm ran Windows and Internet Information Server, thus the web user interface was to be developed using ASP.NET 2.0, see 2.7, with C#, see 2.8, as programming language. The system needed to be able of administrating the name servers remotely through the web interface, all according to the requirements.

The system was going to be accessible from the internal company network only, and the access control mechanisms had to be handled through Active Directory. Information about customers and domains was to be stored in a SQL Server database, see 2.6. The system also needed to be able to list, add, modify and delete customers and domains, as well as the associated information.

Figure 2-1 depicts an overview of the target system.

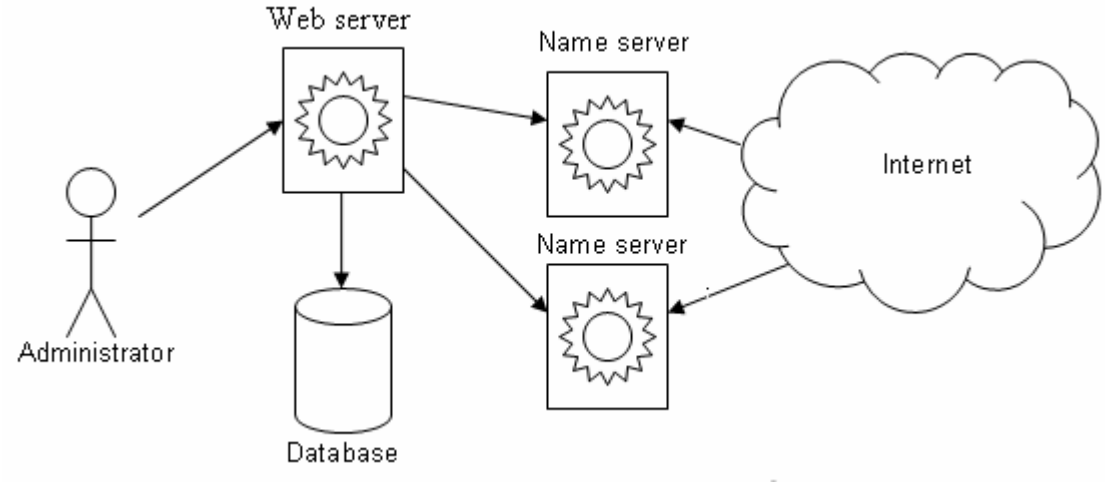


Figure 2-1. System overview

As stated earlier, Ninetech were in the process of integrating the ITIL standards into the organization. Thus, the system needed to handle administrative domain changes in a way that conformed to the ITIL standards for change management, i.e. users should only be able to make requests for changes. A Change Manager has authorization to decide whether the users' requests should be approved or not. To conform to the change management process, the system ought to present a summary of requested changes and provide functionality for request approval/rejection to the Change Manager.

Upon approval of a change request for a customer or domain, all changes were to be recorded into the database, and propagated to the name servers afterward. Finally, the name server processes had to be automatically restarted to make the changes take effect.

The rest of chapter two provides a description of some areas closely related to the project, starting with the domain name system itself.

2.3 Domain Name System

Hosts in TCP/IP networks can be identified in different ways. Two common identifiers in use today are hostnames, e.g. www.google.com and IP-addresses, e.g. 216.239.51.100. While it is more convenient for humans to use the mnemonic hostnames [41], routers require the fixed-length, structured IP-addresses. One main task of the Domain Name System (DNS) is to translate between hostnames and IP-addresses to overcome the different preferences of humans and machines [38].

The DNS is both a distributed database, which is implemented in a hierarchy of name servers, and an application-layer protocol which describes how name servers and hosts should communicate. Other application-layer protocols, for instance HTTP and FTP, frequently use DNS to look up IP-addresses of hosts they are communicating with before setting up the actual data connection.

2.3.1 The Domain Namespace structure

The naming convention DNS uses is based on a hierarchical tree structure, partially illustrated in Figure 2-2 called the Domain Namespace. The single node on top of the tree is called the root node. Underneath the root node lies the top domains, e.g. **.**com, **.**org and **.**se. Organizational domains, which can contain either hosts or additional sub domains, are in turn positioned beneath the top domains.

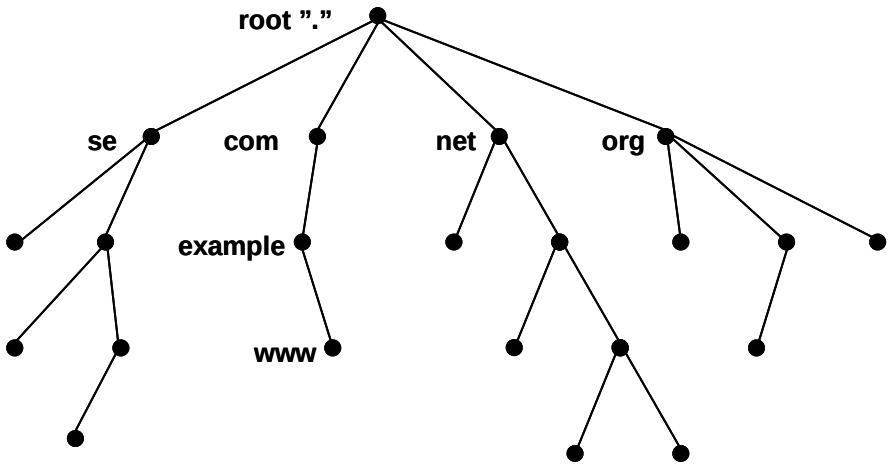


Figure 2-2. The Domain Namespace structure

2.3.2 Fully Qualified Domain Names

A Fully Qualified Domain Name (FQDN) consists from left to right of a host, optional sub domains, an organizational domain and a top-level domain. A trailing dot is added which indicates the root node [1], for instance: `ftp.subdomain.organization.com.`

The trailing dot is normally left out in everyday use, but without the trailing dot the name could in theory be interpreted relative to some other domain name other than the root.

A FQDN is used when referring to a specific host in the namespace. To ensure the uniqueness of names in the network, two child nodes with identical names cannot co-exist on the same node in the namespace.

2.3.3 Name servers

The software that handles information about a specific part of the domain namespace is called a name server. BIND, see 2.3.11, is an example of popular name server software. The portion of the namespace that the name server has complete information about is called a zone, for which the name server is said to be authoritative [63].

2.3.4 DNS and the Internet

Due to the scale of the Internet, a single centralized DNS server would be insufficient. The huge traffic volume, the long distances, and the fact that the entire Internet would be essentially useless if the server would crash render a centralized solution unfeasible. Instead, the DNS mappings are distributed to, roughly speaking, three different kinds of name servers around the globe [38]:

- **Local name servers** – When a host issues a DNS query message, for instance to find the IP address of the host www.google.com, the message will initially be sent from the issuing host to the local name server. In a company network, the local name server may be located on the same local area network as the host. For a home user, the local name server is usually maintained by the residential Internet Service Provider. If the message requests a translation for a host located in the same local area network, the name server can send a reply with the requested information instantaneously. If the requested host is located elsewhere and no cached information about the host is available, the local name server will have to forward the request to other name servers with additional information about the requested hostname.
- **Root name servers** – There are about a dozen of root name servers, which are queried if a local name server does not have a mapping which has been requested by a client. The root name servers know the addresses of authoritative name servers in the different top level domains [1]. Thus, the root name servers can always forward the request to a more knowledgeable name server.
- **Authoritative name servers** – A DNS server is by definition authoritative for a host if it always has the information required to translate the host's hostname into the current IP-address. Every single host has an authoritative name server, and when a translation request is received for which the name server is authoritative, the name server responds with the requested information.

2.3.5 A DNS query example

In this example, the host `a.src.com` requests the IP-address of the host `b.dst.net`, see Figure 2-3.

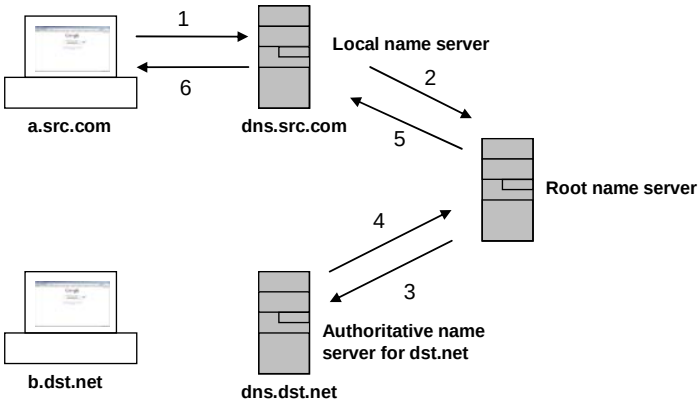


Figure 2-3. Recursive DNS query

Initially, a DNS query is sent from `a.src.com` to the local name server, `dns.src.com` (the IP address of the local name server is used to send this query), which cannot immediately translate the hostname into an IP-address. The local name server now acts like a client and forwards the query to a root name server, which knows the address to the authoritative name server for `dst.net`. Subsequently, the root name server forwards the query to the `dst.net`-authoritative name server. Since `dns.dst.net` contains the mapping between hostname and IP-address of `b.dst.net`, a reply containing the correct IP-address is sent back to the querying client. The reply travels back through the request chain until it finally arrives at the source host `a.src.com`. This kind of querying is called recursive resolution. One major issue with recursive resolution is that the root servers receive a lot of traffic. Instead, a technique called iterative resolution can be used, see Figure 2-4.

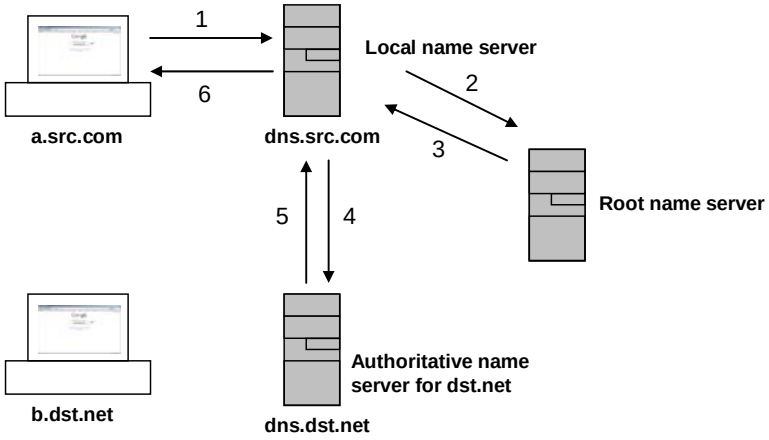


Figure 2-4. Iterative DNS query

In this case, the name server does not forward the query but simply returns the address to the next server in the chain. For instance, the root name server returns the IP-address of the authoritative name server for `dst.net` to the local name server, which subsequently sends the query to the received IP-address. Iterative and recursive resolution can be used together. Typically, all resolutions are recursive except the ones involving the root name servers to lessen the burden on those, as in Figure 2-4 [1][38].

The communication between the clients and the name servers uses the User Datagram Protocol (UDP) on port 53 [63].

2.3.6 Caching

Since the name resolution process consists of a number of steps for each query, name servers often store the retrieved DNS information for a certain, arbitrary, time. Due to this, if a request is received for information already in the cache, the name server does not have to query other servers yet again, but uses the information in the cache to speed up the process. For instance, if another host which is using the same local name server as `a.src.com` in Figure 2-3 requests the IP-address of `b.dst.net` a short while after the request from `a.src.com`, the required information might already exist in the local name server cache. If this is the case, the local name server simply replies with the IP-address immediately.

Another caching feature is negative caching. When a host requests a translation for a host that turns up to be unreachable, the name servers have the possibility to retain information about this as well as positive results [3].

2.3.7 Master servers and Slave servers

The DNS specifications define two different types of name servers, primary master name servers and slave name servers. The slave servers act as backups to the primary master servers in case they should experience problems.

The primary master is where the source zone information, stored in zone files, is located. The name server software loads the zone information from the zone files and uses this information when replying to different requests. The slave servers behave differently; they retrieve the zone information from the primary master name servers automatically. The transfer of information between master and slave is called a zone transfer.

Thus, by using a master and one or several slaves, you can achieve redundancy, load balancing, and decrease administration time because changes to the primary master server configuration are automatically reflected to the slave servers [15].

During zone transfers, the Transport Control Protocol (TCP) is used instead of UDP when the amount of data exceeds 512 bytes [63].

2.3.8 Resource Records

Inside a zone file, see 2.3.9, information about the zone is mostly contained in entries called resource records. The more common records are [1]:

- SOA (Start of Authority) – This is a required record and there can be only one SOA record in each zone file. The SOA record contains information about the primary name server for the zone, the e-mail address to a person responsible for the zone, along with information used by other name servers:
    - Refresh – Specifies the interval of time indicating how often slaves check if the information is up to date.
    - Retry – The amount of time a slave server waits before trying again if the master server is unreachable.
    - Expire – If the slave server loses contact with the master server, the slave server stops serving information about the specific zone after the amount of time specified by the expire-property.
    - Negative caching – Specifies for how long negative responses may be cached by querying name servers.
- NS (Name Server) – Indicates a name server for the zone.
- A (Address) – Maps a hostname to an IP-address.
- CNAME (Canonical Name) – Maps an alias to its canonical name. If the name server looks up a name and finds a CNAME record, a new lookup is performed using the canonical name.
- MX (Mail eXchanger) – Specifies mail exchangers for the zone. Each MX record also has a preference value used to prioritize between different MX records.
- PTR (Pointer) – Maps an IP-address to a name.

Except for the above mentioned resource records, there is a TTL-statement (Time To Live) which applies to all the other records in the zone file. The TTL is sent along with the DNS response and specifies the interval of time that other name servers may store the DNS response results in their local caches [1]. Other records exist as well but are outside the scope of this thesis.

2.3.9 Zone file example

```
$TTL 3h ex.com. IN SOA primarynameserver.ex.com. root.ex.com. (
              2006100103    ; Serial number
              10800         ; Refresh after 10800 seconds (3 hrs)
              1h            ; Retry after 1 hour
              1w            ; Expire after 1 week
              1h )          ; Negative caching TTL of 1 hour

localhost.ex.com.    IN A          127.0.0.1
www.ex.com.          IN A          132.125.14.35
mailserver.ex.com.   IN A          132.126.14.37

test.ex.com.         IN CNAME      www.ex.com.

ex.com.              IN MX         10 mailserver.ex.com.
```

Example of a PTR-record, stored in a separate file:

```
35.14.125.135.in-addr.arpa.  IN PTR www.ex.com.
```

2.3.10 Internationalized Domain Names

An Internationalized Domain Name (IDN) is an Internet domain name containing non- or extended-ASCII characters, e.g. å, ä and ö. One of the flaws with the original DNS design was precisely that domain names could only contain characters found in the ASCII character set. Thus, countries whose alphabets contained non- or extended-ASCII characters could not use these characters in their domain names. Eventually, RFC3490 [22] introduced a way for encoding non- or extended-ASCII characters into fully functional domain names.

Rather than extending DNS itself, domain names containing foreign characters are encoded into a compatible format by the applications, for instance web browsers, upon querying. This format is called ACE, ASCII Compatible Encoding [1]. There were some different versions of ACE in development but an agreement was made to standardize a specific version called Punycode [16][50]. Punycode-encoded domain names include a particular prefix, *xn--*, and as a result, this prefix is disallowed in normal domain names to avoid naming conflicts.

The incompatible characters are represented by a string of valid characters, which potentially make the ACE-encoded domain names virtually unreadable for humans. For instance, when a web browser tries to access the site `www.ääö.com`, the browser first translates the IDN into the ACE-encoded version, `www.xn--4cab6c.se`, and uses this domain name when querying the name servers. Thus, the name servers are not concerned with the special characters but store only the mappings for the ACE-encoded domain names [1].

2.3.11 BIND

BIND (Berkeley Internet Name Domain) is a popular implementation of the DNS protocols maintained by the Internet Systems Consortium [34]. BIND is compatible with most UNIX versions and has also been ported to Microsoft Windows NT, 2000 and 2003.

BIND reads information from a configuration file usually called *named.conf*. This file specifies, among else, the location and names of the zone files, whether BIND acts as master or slave for the zones, and access rights for slave servers. An example of a zone declaration follows:

```
zone "example.com" {
type master;
file "named.example.com";
};
```

This record indicates that the name server is a primary name server for the zone *example.com*, and specific information about this zone is located in the file *named.example.com*. The location of the zone file itself is specified by a directory-clause elsewhere in the configuration file or in the file-statement.

If the server acts as a slave server for the zone, the type is changed to slave and another clause is included that specifies the master servers for the zone. BIND connects to specified master servers to download zone information. Example:

```
masters { 123.124.125.126; 234.235.236.237};
```

The name server must also be kept updated with a list of root name servers, which is stored in a file along with the zone files and should be updated on a regular basis.

2.4 Information Technology Infrastructure Library (ITIL)

One of the goals of the project was to construct an administration system in a way that conformed to the ITIL-processes. Therefore, the following sections provide an overview of ITIL, focusing on the parts most relevant for the project.

2.4.1 What is ITIL

The Information Technology Infrastructure Library is what might be considered a collection of best practices for IT service management; a framework containing guidelines which has proved to work well according to numerous years of combined experience from companies, experts and other IT practitioners. Since the first ITIL elements were released in 1989, the

ITIL publications have grown tremendously and nowadays there are e.g. courses, books, certifications and tools available for ITIL adopters [44][58].

The ITIL guidelines attempt to cover all aspects of IT service management, from the people involved, the processes and products used or constructed as well as the partners, for example vendors and suppliers. The guidelines' ambitions are to align IT with business goals and customers, and improve the quality of service and IT effectiveness within the limits of current budget restrictions [44][58].

ITIL is process driven and designed to be flexible enough to be of use for organizations of virtually any size. However, adopting organizations has to adapt the ITIL-guidelines to fit their own distinctive environment; for instance by selecting a set of processes that matches the needs of their organization [44][58].

2.4.2 CMDB

A commonly used expression in ITIL contexts is CMDB, an acronym for Configuration Management Database. This database should provide information about all configuration items in the system and their relationships. A configuration item (CI) is a component that provides IT services, for instance a database, a computer, or a system manual. The level of descriptive detail varies but it is important to define in what way the configuration items are interconnected.

For instance, if one system handles customer accounts it should be possible to link this information to other systems in the organization which deal with the same customers. All information should be available from a central location, the CMDB. The CMDB itself does not have to be one physical database, but can be comprised of several, integrated databases [44].

2.4.3 The Framework

Figure 2-5 illustrates the core modules of ITIL and how they connect to the business and the technology. The core modules will be described shortly in forthcoming sections with focus on the middle section, service management, with the two core modules Service Support and Service Delivery. Service Management is not a core module in itself.

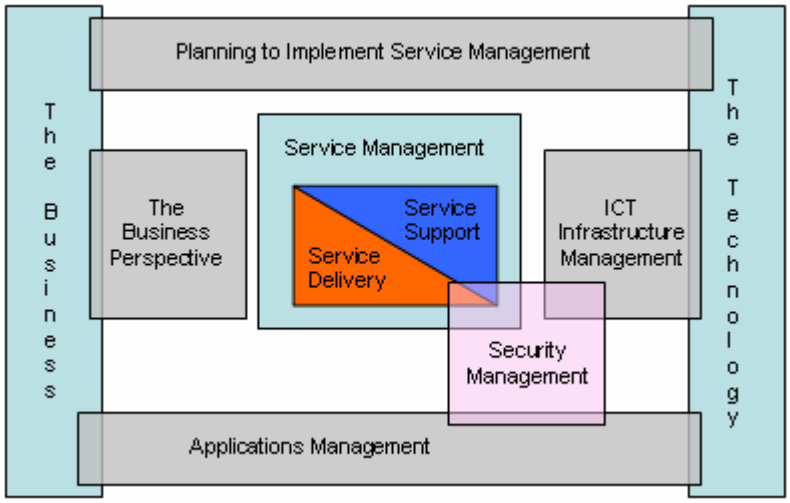


Figure 2-5. The ITIL Framework

2.4.4 Service Support

Service support is one of the core areas of ITIL and mostly concerns the day-to-day maintenance activities involved when providing IT services. Service support has six different areas [60]:

- **Service Desk** – The service desk is not a process, but some kind of physical unit in the organization. The service desk should provide an interface to the other service support processes and act as a central point of contact for IT users, becoming a filter between the users and the IT Service management. The service desk ought to receive for instance incident reports and act upon them. Simple request might be handled directly by the service desk and more complicated issues will be forwarded to other instances.
- **Incident Management** – An incident is anything that is not part of the standard workflow and which might cause a decrease in the quality of the affected service. An example of an incident could be a server crash. Incident management should among else classify and prioritize problems and register them in the CMDB. The objective is to restore normal service as soon as possible with the least amount of interference with other services.
- **Problem Management** – Tries to proactively prevent problems by for instance analyzing trends, along with resolving existing problems. A problem is defined as the cause of an incident and problem management tries to find the source of the problem and registers all relevant information in the CMDB.

- **Change Management** – All proposed requests for changes have to be assessed by a Change Manager, who controls the process and makes sure standard and effective routines are followed. A request for change (RFC) should contain information about which configuration items it affects, the identity of the request originator and information about the RFC status, e.g. if its accepted, pending or rejected. The Change Manager is supposed to estimate the impact of the proposed changes, approve or reject the requests for change, and also test the changes afterwards.
- **Release Management** – Should be used when large and/or critical software or hardware roll-outs are due. Testing and backup-plans are essential parts along with documenting both software and hardware in the CMDB. A release could be considered to be a set of approved Changes from Change Management.
- **Configuration Management** – The configuration management should provide a logical model of the IT infrastructure by identifying, labeling and documenting all assets in the CMDB. Configuration management should subsequently control all modifications and additions of configuration items to the CMDB along with verifying that information in the CMDB is correct.

Figure 2-6 illustrates in a simplified way the relations and messages sent between the different service support areas. For instance, changes and releases might go straight from the users to their respective management process. Incidents always pass through the incident management process if the service desk is unable to handle the problem itself.

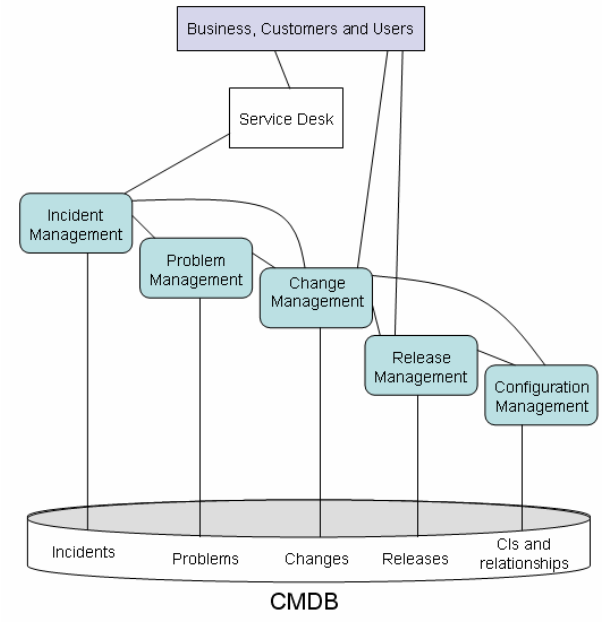


Figure 2-6. Service support areas and relationships

2.4.5 Service Delivery

The service delivery processes are mainly concerned with improving the quality of the IT services provided, making plans etc. There are five different processes involved [60]:

- **Service Level Management (SLM) –** SLM should negotiate and establish Service Level Agreements between customers and the provider of some resource. Further, SLM should monitor the services afterwards to make sure the service levels meet the established agreements. An agreement example might be a minimum amount of downtime of an Internet connection. The purpose is to keep up and progressively improve overall IT service quality.
- **Financial Management for IT Services –** Tracks the cost of IT services and is responsible for ensuring that services are running in a cost effective way [58].
- **Capacity Management –** Should make sure that there always are adequate resources available in a cost effective manner by inspecting current and future capacity requirements and creating a capacity plan.
- **IT Service Continuity Management –** Responsible for creating recovery plans that will be used if a serious incident occurs. The plan should make sure that IT resources will be back online within established time limits without causing unnecessary disruption to other services. Another responsibility is to perform risk analysis and risk management.
- **Availability Management –** Responsible for making sure resources are available when a customer or user needs them. Some key considerations for availability management are availability, reliability and maintainability of resources. Metrics ought to be taken and revised, and an availability plan with focus on improving availability in the long run created.

2.4.6 Information Communication Technology Infrastructure Management (ICT IM)

This module has focus on the tools and technology within the IT services, covering aspects such as design, building, testing and deployment of various resources. ICT IM is further responsible for creating a suitable environment for IT services with, for instance, technical support and educational possibilities [44].

2.4.7 Planning to Implement Service Management

This module deals with improving or implementing ITIL itself in an organization. The organization should try to find out what they want to achieve – the vision – along with finding out where the organization currently stands in terms of e.g. people, processes and technology. A plan for starting and keeping the process running ought to be established along with measurable milestones [44].

2.4.8 Application Management

Application development and service management has often been treated as separate processes which is an undesirable situation. An important task for application management is to connect these concepts and integrate service management thinking into the development process. This improves the chances that IT services will properly support the business objectives [44].

2.4.9 The Business Perspective

The focus of the business perspective is to align IT services with the business, e.g. making IT personnel understand how they can contribute to and assist the business objectives. The business perspective is also concerned with establishing and improving the relations between providers of IT services, customers and suppliers [44].

2.4.10 Security Management

Security management is concerned with the process of defining, planning and managing a level of security for IT services. Risk and vulnerability assessment is another important aspect of this module [44].

2.4.11 ITIL adopters

To conclude the explanation of ITIL, it might be interesting to know of some organizations that have decided to adopt ITIL. There are several well known companies worth mentioning, for instance Microsoft, Hewlett Packard, Guinness and Procter & Gamble. As of today, ITIL is the most widespread framework in use for IT Service Management [36].

2.5 Active Directory

One of the project requirements was that the DNS administration system had to base access rights on Active Directory group membership; hence an understanding of these concepts is important. Therefore, this section provides the reader with an overview of Active Directory.

2.5.1 Directories

Initially, it makes sense to introduce the concept of a general directory, which is, according to Oppermann [53], basically a container of data. An example could be a telephone catalogue, regardless whether it is online or offline. Information in a directory is usually stored in a logical, hierarchical way to provide an easy way to locate the correct information [17].

A network directory is an online directory that contains information about resources in a network, e.g. user accounts and printer services. Active Directory can be considered an extended version of a network directory since it has been designed to store additional data as well. Microsoft has for instance implemented their dynamic DNS service in Active Directory [53].

The directory acts as a centralized point of management; it authenticates and authorizes different resources, e.g. users and computers, and governs the relationships between different entities [36].

Other kinds of directories exist as well, but are outside the scope of this thesis.

2.5.2 Before Active Directory

Early versions of network operating systems handled account management on a per-server basis, see Figure 2-7.

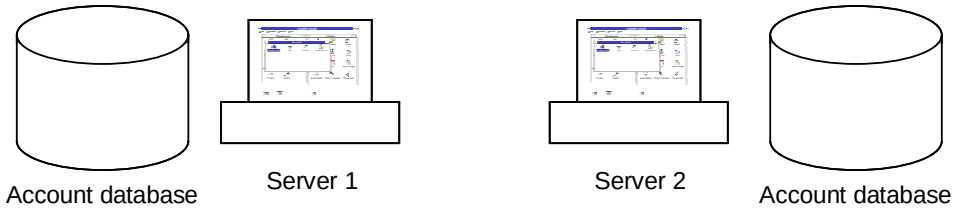


Figure 2-7. Early network operating systems' account management

Eventually, the lack of a single point of management and the ever increasing size of the databases became problematic for the administrators. In Windows NT4 the concept was improved by having a single point of management and allowing several servers to share a central user account database, as in Figure 2-8. This is an example of a domain-based network, where users can access the network from any server using the same credentials.

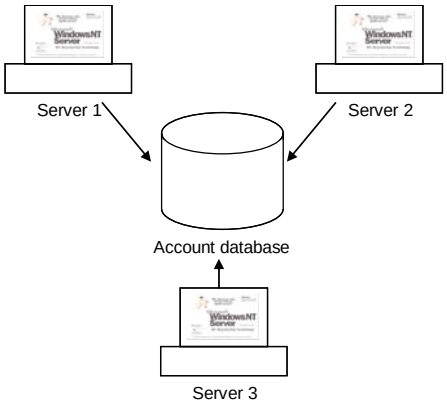


Figure 2-8. Domain account management – e.g. Windows NT

Network directories are the next step, where all information about all resources across the entire network is stored in the directory. For instance, the directory could store user specific information, start menu-settings, personalized icons and such. These settings will subsequently be downloaded to any computer from which the user accesses the network [36]. As mentioned earlier, Active Directory belongs to this directory category.

2.5.3 Naming schemes

Active Directory supports several common naming schemes; this provides compatibility with different systems and allows objects to be accessed in different ways. Some common supported naming schemes are DNS (see 2.3), URL (Uniform Resource Locator), UNC (Universal Naming Convention) and the LDAP URL (see 2.5.5). Normally, Active Directory assigns domain names based on DNS; a domain could for instance be called *example.com*.

2.5.4 The Active Directory structure

The three basic building blocks of Active Directory are domains, trees and forests. A domain is simply a logical collection of resources, e.g. users, computer and printers, and has a common directory database while also sharing security policies. Since a domain represents a logical group, the physical entities that the domain objects symbolize might be separated geographically [17].

When the first domain in the network is created it is called the root domain and becomes the foundation of the Active Directory namespace. If the first domain is called *ex.com*, all subsequently added domains are placed beneath the root domain and forms a hierarchical tree structure branching downwards. When more domains are added to the namespace, they are named *something.ex.com* and it might look like in Figure 2-9 [36].

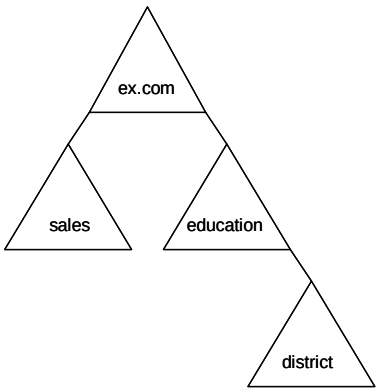


Figure 2-9. A domain tree

Thus, by combining several domains, a tree is formed. Additionally, several domain trees might also be grouped together, logically enough becoming a forest. For instance, a forest might be created out of the domain trees branching from *ex.com*, *abc.com*, and *def.com*, see Figure 2-10 [2].

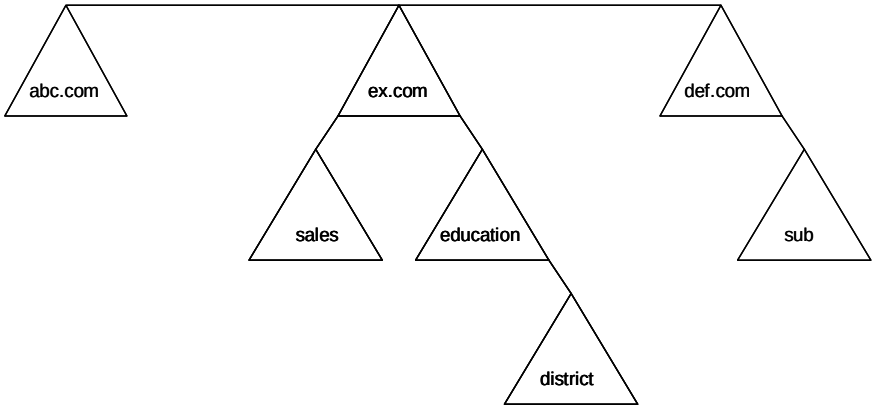


Figure 2-10. An Active Directory forest

Servers that host domains are called Domain Controllers. A domain controller is authoritative for a single domain and provides directory and authentication/authorization services to the clients.

Another important concept is the trust, which is an agreement between two domains to allow the trusted domain to access resources in the trusting domain. However, the trust itself does not imply that users from one domain can access resources in the other domain; the administrators still have to give users from the other domain proper permissions. The trust merely gives the administrators the capability of granting permissions to the users of the other domain [17][36].

Trusts can be initiated on different levels and on the top level is the forest trust, where all domains in both forests trust each other. Domains in the same tree trust each other by default [2].

2.5.5 Accessing the directory – LDAP and X.500

X.500 is a set of ISO standards defining distributed directory services. LDAP (Lightweight Directory Access Protocol) is an Internet standard, based on X.500, which defines how to access directory services. Active Directory is based on these concepts, thus LDAP is the protocol used for exchanging information with the directory [69]. This also provides interoperability between Active Directory and other systems that incorporate LDAP and X.500.

The Active Directory structure has the appearance of a tree, with sub trees extending from the root. Entries in the tree can either be containers or leafs. A Distinguished Name provides every object in the tree with a unique name. Distinguished names are based on an LDAP URL, which looks like: LDAP://domainname/CN=Common name/OU=Organizational Unit/DC=Domain Component.

An example of a distinguished name could be:

CN = Jose Gonzales, OU = Staff, DC=example, DC=com

CN is the common name of the object, for instance user account and printer names. The OU tag defines the organizational unit, see 2.5.7. These are containers, thus containing other objects. The organizational unit Staff in the example above cannot contain two objects with the same common name.

DC is an acronym for Domain Component, which specifies the DNS domain name path of the object. In the example above, a user account Jose Gonzales exist in the organizational unit Staff in the *example.com*-domain [17], [65].

2.5.6 Schema

The Active Directory schema formally defines all objects that might be stored in the directory. Objects are defined by a class along with a collection of attributes, for instance the User class with attributes such as name, home directory and address.

The schema also describes the relationship between different objects and other details about the database structure. The default schema is usually sufficient but if the need for additional object classes arises, the schema might be extended.

Schema information is shared by all Domain Controllers in an Active Directory forest and changes to the schema are automatically replicated to all Domain Controllers in the forest [36].

2.5.7 Organizational Units

Organizational units (OU) are used to organize resources inside a domain. For instance, there might be a Sales OU, a Management OU and an Accounting OU inside a domain of a company. Administrators can be assigned to the separate organizational units to lessen the burden on the network administrators.

The organizational units are populated with resources, e.g. users, groups, printers and other organizational units. Group policies may be set on an OU which applies to all related resources inside the OU. For instance, the administrator might take control over the users' environments and disallow them to change the desktop background image [17].

2.5.8 Users and Groups

For a user to be able to access network resources, the administrator must have created an account which the user is able to log on with. There are three different kinds of user accounts available in Active Directory.

- Local user accounts – These accounts only apply to the local computer where the account is located and may not be used to access network resources.
- Built-in user accounts – Specialized accounts mainly used for administrative purposes, e.g. the administrator account.
- Domain user accounts – Accounts which give the user access to the network. However, permissions still have to be configured properly.

Administrators grant access to resources through the user accounts, for instance giving certain users write permissions to a spreadsheet while others merely have the possibility of reading the spreadsheet information.

Since setting permissions on user-basis is very time consuming this kind of administration is mostly based on the concept of groups, which basically is a collection of users and other groups. For instance, by adding all users that should have access to an application to a group and then allowing the group access to the application, the users in the group receive the permission to access the application as well. Akin to user accounts, there are different kinds of groups. Several built-in groups exist but the administrators can also create new groups to fit the needs of their organization.

Computers also have accounts in the Active Directory and administrators may use these accounts as well as user accounts and groups when performing administrative tasks, such as allowing access to a printer [69].

2.6 SQL Server

Regarding the storage of customer and domain information, Ninetech stated that SQL Server had to be used for this purpose. SQL Server is a Relational Database Management System (RDBMS) developed by Microsoft. This section describes SQL Server and some related concepts in enough detail to provide the reader with an understanding of the subject.

2.6.1 Databases

Management of information has always been a challenge in software development, and there are many variations of databases to meet different demands. Nowadays, there are efficient and secure database management systems available, both commercial and non-commercial, and the use of such a system might be a step toward secure, efficient applications as well [59]. The use of existing, trustworthy database management systems offer several advantages, for instance the speed of data retrieval and insertion, automatic reporting and built-in security [57].

Apart from the physical database itself (where the stored information is located), a database management system consists of other components, e.g. [59]:

- Schema – specifies the structure of the database contents.
- Database Manipulation Language (DML) and Database Definition Language (DDL) – Computer languages that provide access to database contents and allows schema modifications.
- Database engine – the software that handles access to the database.

2.6.2 Relational database management systems

SQL Server is a relational database management system, which is a database management system that incorporates the relational model. A relation might be thought of as a two dimensional table, consisting of a set of rows. There can be no duplicated rows, and each row indicates a database object. The columns define the attributes of the object and each column has a predefined data type to which all rows must adhere [5][59].

2.6.3 SQL

The Structured Query Language (SQL) is the standard database language for relational database management systems. SQL covers, among else, the database definition language and the data modification language. An example of a DDL-language statement is `create`, which is used to create e.g. relations. A create-statement that creates the structure of Table 2-1 (not the data) is specified as:

```
CREATE TABLE Customer (
        customerNo int primary key,
        customerName varchar(100),
        customerEmail varchar(56)
)
```

Table 2-1. Customer table example

customerNo	customerName	customerEmail
435	Joe	joe@example.com
77	Bob	bob@example.com
243	Eve	eve@example.com

The customer number is specified as the table's primary key, which must be unique for each entry (row) and guarantees that all rows in the relation are unique. Keys are often used to link data in one table to data in another. Two additional examples of DDL-statements are `drop` and `alter`, which are used to remove tables and modify schemas.

The `select` query is an example of a DML component and is used to retrieve data from one or more relations. Select queries can also combine several tables when retrieving data to create more advanced queries. A simple example of a select query which retrieves all the e-mails in the Customer table where the customerNo is less then 100 looks like:

```
SELECT customerEmail FROM Customer WHERE customerNo < 100
```

Other kinds of DML-statements are `update` and `insert` statements [59].

As the examples above indicate, SQL is a declarative language; it describes what data to retrieve, insert, or modify rather than describing how to perform the retrieval, insertion or modification [64].

2.6.4 Transact-SQL

Transact SQL (T-SQL) is an SQL extension used in SQL Server and some key features of T-SQL are [64]:

- The possibility to use variables in SQL statements.
- Flow-commands, e.g. IF/ELSE-constructs.
- Integrated user authentication with Microsoft Windows.

2.6.5 Stored Procedures

As the name implies, a stored procedure is a database object *stored* on the server and of *procedural* nature. The stored procedure can contain constructs like `IF-ELSE` and `BEGIN-END`, accept parameters, return values and call other stored procedures [30].

The procedure itself is a collection of T-SQL commands, which is compiled together and can only be executed as a whole. A benefit with this is that if several commands are going to be executed after each other, data does not need to leave the server in intermediate steps, thus reducing network traffic [19].

Stored procedures are practical to use when a certain set of SQL-commands will be executed together frequently. It is also beneficial that intermediate steps cannot be executed separately since they often do not make sense if the stored procedure is not executed as a whole.

Stored procedures also provide encapsulation, if the interface stays the same, the user can call the stored procedure regardless of the implementation [6].

The `CREATE PROCEDURE` statement is used to actually create the stored procedure. An example featuring a parameter follows:

```
CREATE PROCEDURE procedure_booktitles
   @publisher
AS
SELECT book_title
FROM book
WHERE publisher = @publisher
ORDER BY price
```

This procedure uses the given parameter (prefixed with @) to select books from the specified publisher and returns the titles ordered by the price [12].

Another feature available in stored procedures is the possibility of using input validation by comparing the in-parameters against suitable values using IF-statements [6].

SQL Server can also grant different user access rights to a stored procedure, e.g. certain users might be prohibited to execute a certain query while others have access to it.

2.6.6 SQL Server

Apart from the SQL Server engine itself, which acts on commands and reads/modifies data, several other services are included in the SQL Server package. For instance:

- Analysis tools – Used to analyze data in various ways.
- Reporting Services – Used to create reports based upon data sources. The reports can later be accessed by using, for example, a web browser.
- Notification services – Allows, among else, messages to be sent when certain events are trigged.
- Integration services – Provides exporting and importing functionality to and from various other applications and formats.
- Workstation components – Installs e.g. the tool used after installation for building database solutions, the SQL Server Management Studio (SSMS), which provides a GUI for most database tasks. SSMS provides for instance a Query Editor where stored procedures and other queries can be written, wizards for generating queries by selecting tables and columns etc, and tools for creating user accounts as well as mapping the accounts to the correct databases [19].

Microsoft has also released SQL Server Express, which is a free but more limited version of the SQL Server product.

2.6.7 SQL Server security

Access to database contents can be based either on Windows accounts or user accounts created in SQL Server. If the SQL Server administrator wants to give a user access to a database while Windows authentication mode is used, the administrator has to allow the proper Windows user account to access the database. This can be done either by allowing access to the specific user account or to a Windows group the user belongs to. If the database is operating in an Active Directory environment, network users that need access to specific data can be placed in a group that has been set up with appropriate permissions.

When using Windows authentication mode, the SQL Server administrator trusts that Windows has already authenticated the user.

The alternative is to create specific accounts in SQL Server with a username and password, which the connecting users have to provide in order to access the database contents [19].

Additionally, it is also possible to explicitly allow user access to for instance a certain table or stored procedure, based upon user needs. If a user only should be able to see the contents of a table, the user should not be able to execute procedures that modify table data.

2.6.8 Transactions and rollbacks

Transactions are used to protect data against, among else, inconsistencies and have the following four properties [59]:

- Atomicity – the transaction must be completed as a whole or not at all.
- Consistency – the database must be in a consistent state both before and after the transaction, inconsistencies are only allowed during the execution.
- Isolation – if two transactions are performed concurrently, the result should not differ from the scenario where the transactions are executed sequentially.
- Durability – if the transaction finishes successfully the result should be permanently stored.

SQL Server allows the programmer to define customized transactional units by surrounding the commands of choice with `BEGIN TRAN` and `COMMIT TRAN`-statements. If something goes wrong inside the `BEGIN TRAN-COMMIT TRAN`-block, all changes made prior to the error becomes nullified, and the database contents gets restored to the consistent state before the transaction took place, see Figure 2-11. This restoration of data is called a rollback and is possible because SQL Server logs all information needed to undo non-committed changes in a log called the transaction log.

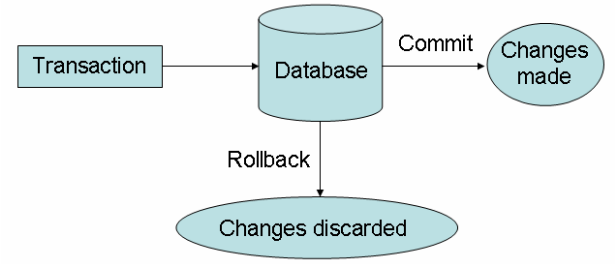


Figure 2-11. Transaction

Transaction isolation is performed by using locks, which block access to certain resources and subsequently hinder other transactions to access these resources before the lock is removed. This can potentially lead to a deadlock, a situation where two (or more) transactions lock resources other transactions need, thus blocking the involved transactions from continuing. SQL Server automatically discovers deadlocks and terminates the transaction which has performed the least amount of work [6].

2.7 ASP.NET 2.0

Another project requirement was that ASP.NET 2.0 had to be used if a new DNS administration system was to be developed. This section will provide an overview of ASP.NET 2.0 along with more detailed descriptions of parts that were most important throughout the project.

2.7.1 Background

Earlier frameworks for developing server-side web applications usually belong to one of the following categories:

- Scripts that are interpreted by some resource on the server.
- Compiled, small, separate applications executed by server-side calls.

Both categories have their respective problems. Scripted solutions have a tendency of executing slow, and the second approach tend to require a lot of server memory since the server creates a new instance of the application for each client [43]. A well known example of the latter approach is CGI, while classic ASP – the predecessor of ASP.NET – is an example of a script based solution. ASP rapidly became a popular web development technique but had a few problems. Apart from the performance issues, there was e.g. no application structure. ASP and HTML code were intermixed, thus creating code that often grew long and hard to debug. Further, there was state management limitations (see 2.7.8 for more information about state management) [42] and difficulties in creating an integrated development environment (IDE) [43].

2.7.2 ASP.NET

To resolve the issues of earlier web development solutions, ASP.NET was introduced. With ASP.NET, Microsoft decided to introduce a fundamentally new way of developing web content and blur the line between application and web development. The new model is based on the .NET-framework, of which ASP.NET is a part. The .NET-framework is a group of different technologies, e.g. the Common Language Runtime, the .NET programming languages, and a vast class library. These and others will be explained in following sections.

In ASP.NET, object orientation is a key concept. Buttons and other controls on web pages are objects which can be programmed quite similar as in application development [43][44]. A related, alluring part is that you can make use of a broad set of predefined classes known as the *.NET Framework class library*. These classes provide a great deal of help for the programmer with components such as textboxes, dropdown-lists and radio buttons ready for use.

The available classes are not limited to web development functionality but also contain general purpose classes for various circumstances, for instance database access, string handling, graphics, and mathematical operations [42].

The most commonly used integrated development environment for ASP.NET development is Visual Studio, another part of the .NET-framework. As this is written, the latest version is Visual Studio 2005, which contains for instance compilers for the different .NET-languages, WYSIWYG (What You See Is What You Get) capabilities, an integrated test web server, and advanced debugging tools.

Some drawbacks of ASP.NET are that it is platform (Windows) and web server (Internet Information Server) dependent [43].

2.7.3 Compilation and execution

Another benefit of ASP.NET over classical, interpreted ASP is that ASP.NET is compiled, which generally improves performance. An interesting related feature is that this allows interconnected web pages to be coded in different programming languages. Usually, C#, see 2.8, or Visual Basic.NET (VB.NET) is used but other programming languages are supported as well.

The following figure illustrates the process of compiling an ASP.NET-resource.

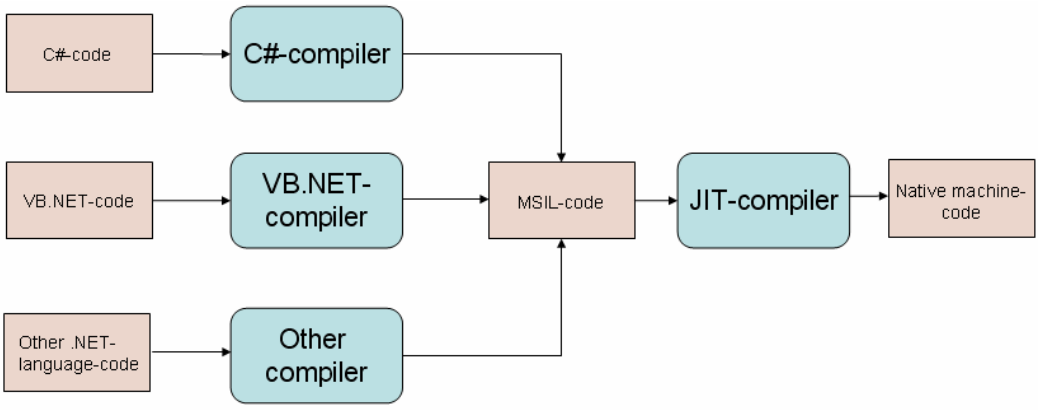


Figure 2-12. The compiling process

All web pages, regardless of programming language used, will initially be compiled into an intermediate code called MSIL (Microsoft Intermediate Language). This is the reason why different programming languages can be utilized. Next, the MSIL code will be compiled into machine code by a Just-In-Time (JIT) compiler upon a web page request [43].

The compilation process will only occur the first time a page is requested and not in subsequent request. A recompilation is only necessary if the source code has been modified or if some configuration was changed [42].

The MSIL code will be executed by the Common Language Runtime (CLR), the program compiling/debugging/executing engine in .NET. All code, including web application code, is executed within the CLR since ASP.NET is running as a service inside the CLR. [42][43]. For additional information about CLR see 2.8.2.

2.7.4 ASP.NET 2.0

Version 2.0 of ASP.NET is mostly a refinement of previous versions (1.0 and 1.1) and does not introduce any groundbreaking new concepts. Instead, the 2.0-release adds new functionality, additional classes, simplifications and bug fixes to the existing architecture [8]. The thesis will not elaborate on differences between different versions of ASP.NET.

2.7.5 Code behind

ASP.NET provides a technology to separate logic from presentation. This was a problem in classic ASP and is commonly seen as a good practice. This separation makes it easier for the developer to focus on design and behaviour separately, and the code will also become cleaner and easier to debug.

This technology is called code behind, and splits the web page into a mark-up file and a code-behind file. The mark-up file defines the appearance of the page while the code-behind file provides the code which is executed on certain actions, for instance when the page is loaded or when a button is clicked [14][43].

When a web page is requested by a client, ASP.NET will first compile the files separately into partial classes, which subsequently are combined and executed as a whole class [14].

2.7.6 Configuration

One standing challenge of software development is how to deal with configuration issues. For instance, certain properties should often be modifiable without having to alter the source code and recompile the program. ASP.NET simplifies configuration by providing easily understood, XML formatted configuration files. The most commonly used one for configuring web applications is web.config. This file is stored on the server along with the web pages and can be modified anytime with some kind of text editor [43]. Web.config contains different configuration sections, for instance an AppSettings-section which is reachable from the code by using the ConfigurationManager-class from the class library. Inside AppSettings, the developer can store custom name-value-pairs used for various causes. Common uses are to store paths, usernames and passwords. For instance, the IP address of an FTP-server could be stored like this:

```
<add key="FTPSite" value="127.0.0.1"/>
```

Now, the IP address is retrievable from the source code by using the key FTPSite. The developer can consequently change the address to the FTP server in web.config at any time without ever touching the source code.

Another common configuration is to define connectionStrings. A connectionString is used by the application upon database connection instantiation. The programmer can also define customized configuration sections if the need arises [4].

ASP.NET 2.0 is also bundled with the Web Site Administration Tool which provides a graphical user interface for manipulating the configuration file. The tool is mostly designed to manage security related settings, e.g. authentication-, and authorization methods for users.

Using the web site administration tool can simplify the process and helps avoiding errors, but manual editing of the configuration files is required if the tool cannot manage the specific type of configuration needed [33].

ASP.NET 2.0 also provides functionality for encrypting information in the configuration files to further improve security, since the configuration files often store sensitive information [43].

2.7.7 Postbacks

A postback occurs when a displayed web page is being sent back to the server for processing, for instance when a user clicks the search-button in a web search engine. The page, including the search keyword, is sent back (posted back) to the server which is able to perform a search based on the given keyword.

ASP.NET can determine whether a page has been posted back or not, allowing appropriate actions to be taken if needed. Afterwards, the updated page is sent back to the client's browser and displayed to reflect any changes [14].

2.7.8 State

Since HTTP is a stateless protocol, ASP.NET must utilize some special technique if information should be stored between different web requests. Several techniques exist, each with their respective pros and cons. Which one that ought to be used depends on the situation.

View State – Should only be used when a single web page is concerned, since all information in view state vanishes if the user navigates to another page. Any kind of serializable object can be stored in view state, and the information that is to be stored is written to a hidden field in the web page.

A benefit is that view state does not affect server performance. On the other hand, the information is available to the client which is a potential security risk. The data must also be sent back and forth between server and client, which is bandwidth consuming. And, as stated above – it only works within the bounds of a single page.

View State is used by many native controls in ASP.NET to allow their properties to remain after postbacks, see 2.7.7.

Query String – Often used when transmitting small amounts of information from one web page to another. A string is appended to the URL which indicates certain properties. For instance, this query string might indicate that the user wants to display only LCD monitors in a web shop:

```
http://example.com/filter?q=monitors+LCD
```

The query string approach is quite limited, only strings are supported and the query string is clearly visible, which is a security risk. However, it is useful for minor tasks.

Session State – Can store all serializable objects, and in contrast to view state, session state is not limited to a single page. Objects can be transmitted from one page to another which makes session state a good choice when storing for example e-commerce shopping carts. This is possible as the information is stored in server memory. From a security point-of-view, this is positive since the client has no way of tampering with the information, but it can also lead to server performance issues. The data will also disappear after a certain period of time, which is configurable.

Other techniques exist as well, e.g. application state, which is quite similar to session state but available for all users at all time and can be used to store global information. Cookies are also frequently used for storing information for later use but suffer from the same problems as query strings [42][43].

2.7.9 Web forms

Web forms are the actual content (the finalized web pages) a client web browser receives. The main difference between a web form and a web page is that web forms might be developed with the same control-based interface as a normal Windows application [43].

2.7.10 Server controls

Server controls are basically .NET-classes which represent visual elements on a web form. There are a number of server control categories, such as HTML Server controls, web controls, rich controls and validation controls.

HTML Server controls are essentially ordinary HTML elements that are running on the server. They have built-in state management and fire events. ASP.NET is event driven, which means that after the controls are added, the programmer can decide upon which events that should be handled and in what way. For instance, a certain method gets called each time a user clicks a button [14][43].

Web controls are a more advanced version of controls and provide more functionality. Additional events are available, more modifications are possible and they closely resemble ordinary controls in a Windows application.

Rich Controls are advanced controls with many features. One example of a rich control is the GridView, a very handy control for displaying data. The GridView provides functionality such as editing, selection, and filtering, and has a wide range of formatting options.

Validation controls are used to validate user input in e.g. textboxes. They provide an easy to use interface and can simplify error checking considerably by for instance matching input data against regular expressions [43].

2.7.11 Global events

A useful part of the ASP.NET event model is the use of global event handlers; which are to be specified in a file named global.asax. Some examples of global events are Application_Start, Application_End, and Application_Error. The last one provides a handy location for logging errors because it is invoked whenever an unhandled exception is thrown in the application [42][43].

2.7.12 ADO.NET

To access and retrieve information from data sources, ASP.NET generally makes use of ADO.NET, the data access technology integrated with ASP.NET. ADO.NET basically consists of a set of classes with data retrieving functionality.

ADO.NET utilizes *data providers* to access data from a data source. A data provider is a set of classes optimized for a particular data source. Different data providers are included with ADO.NET to provide access to e.g. SQL Server, MySQL and Oracle databases. The only difference in the calls lies in which set of classes, i.e., which data provider that is used.

The steps included in the process are always the same – a connection is created, the database is queried and the results are used. Regardless of what data source used, the data provider generally provides a Connection object used to set up the connection, a Command object to query the database, and a DataReader or a DataAdapter to represent the retrieved data.

The DataReader is a lightweight, forward read-only, single-use component, and a DataAdapter is used to create a DataSet – a more advanced but more resource demanding component [24].

2.7.13 Security

When security should be considered, there are essentially two different strategies. Either you allow anonymous web site access and restrict access, if needed, by for instance force users to enter a username and a password in a login dialog. This strategy is useful for Internet-based, publicly available web sites.

The second approach is to restrict anonymous users from entering the site and rely on Windows authentication instead, where the users are authenticated by a Windows user account. To simplify administration, access can be based on Active Directory group membership. This approach would obviously not work very well for public web sites but is useful on intranets for allowing access to certain users [42].

ASP.NET supports both approaches and provides built-in functionality for assisting the developer. However, if Windows authentication is used, ASP.NET relies on Internet Information Server (IIS) to perform the authentication by using existing user accounts.

Different techniques exist for Windows authentication as well, and one interesting technique is the integrated Windows authentication, where the login procedure is transparent to the user. In this case, an identity token is passed from the windows user account to IIS, which uses the token to authenticate the user.

It is easy to control security programmatically this way, for instance by using the User.IsInRole()-method to restrict or allow access to certain resources based on role membership [43].

Accordingly, ASP.NET provides ample functionality to implement security in a simple and secure way.

2.8 C#

One of the requirements Ninetech had on a new administration system was that C# (pronounced C sharp) was to be used. C# is a relatively new programming language developed by Microsoft. This section will provide a short introduction to C#.

2.8.1 Overview

C# was created by Microsoft in the late 1990s, and version 1.0 was released in the year 2000 along with .NET. As this is written, the current version of C# is 2.0, often referenced as C# 2005 [39].

C# is primarily based upon the existing, highly popular languages C++ and its predecessor C. Several ideas were also borrowed from Java, for example the runtime environment error checking, see 2.8.2. Thus, by balancing it carefully, C# tries to combine the security of Java with the power of C/C++ [62].

Since the basic foundations of these languages are very much alike, the process of learning C# is rather smooth for developers with experience in, especially, C++ or Java. In fact, learning the syntax of C# is probably a lesser task than learning what functionality that is available in the vast class library included in .NET [32], see 2.7.2. Even for an inexperienced developer, C# is thought of as a somewhat simple language to learn, despite being a feature-rich, high performance language with a component-based, structured object-orientation approach. The relatively few keywords (if, else, for, while, etc.) used in C# are considered to be a large reason to the rather smooth learning process [39].

C# is considered to be the primary language for .NET development, especially regarding development closer to the system level where C# provides more control than the sister language Visual Basic.NET, which in essence is intended for higher level tasks [31].

2.8.2 Managed code

C# and other .NET-languages runs under the Common Language Runtime, which means that in contrast to the earlier primary languages in the C-family, C and C++, C# code runs in a managed environment. This brings both benefits and drawbacks. On the positive side, the code will be more secure since the environment performs several security checks [32]. For instance, writing to memory dedicated to the operating system is prohibited [7].

The Common Language Runtime also provides one of the greatest benefits of C# and the other .NET-languages, the support for mixed-language programming [62]; consult section 2.7.3 for more information.

The runtime environment also contains an automatic garbage collector, which removes unused code and data and frees the programmer from these responsibilities. However, this also reduces the fine-tuning memory handling capabilities compared to e.g. C++.

Another negative aspect of the Common Language Runtime is that environmental checks impose a slight overhead on the application, which makes .NET-languages more suitable for

applications without critical performance requirements. For e.g. high performance demanding games, a language like C++ might be a better choice since well written code in C++ will yield a higher performance grade [7].

Worth mentioning in the context is that it is in fact possible to write unmanaged code that operates outside the CLR. Unsafe coding should not be taken lightly since the application will require higher permissions to run, and it also compromises the interoperability with other programming languages [31].

2.8.3 Application types

There are several types of applications that can be written using .NET and C# [39], for instance:

- Console applications – runs in a console window with text-mode output (if any).
- Windows applications – standard Windows applications.
- Pocket PC applications – for handheld computers.
- Web applications – are developed together with ASP.NET, see 2.7. Web applications are accessible using a web browser.
- Web services – reusable application software components that provide services to other applications. The services are accessible via standard Internet protocols and might for instance provide current exchange rates or perform different types of conversions [18]. Web services are also developed together with ASP.NET.

2.9 Web application architecture

There are countless descriptions and definitions of what an application architecture is, and it seems to be very little agreement among people in the field. For instance Perry and Alexander [55] define architecture like this:

> "Architecture is concerned with the selection of architectural elements, their interactions, and the constraints on those elements and their interactions necessary to provide a framework in which to satisfy the requirements and serve as a basis for the design."

Mary Kirtland [37] has a somewhat less formal description. She states that an application architecture is a conceptual view of the structure of an application. All enterprise applications

contain presentation code, business logic code and data access code, and thus Kirtland means that application architectures differ in how this code is packaged.

Fowler [25] on the other hand states that application architecture has mainly two parts. Firstly, the highest level breakdown of a system into its parts, and secondly, decisions that are hard to change. With the second part Fowler means that all the decisions that you wish you could get right early on is architectural. He argues that if a decision shows to be easier to change than you once thought, then it is no longer architectural.

However application architecture is defined it is agreed to deal with decisions which influence maintainability or scalability [45]. The architecture should also provide guidance to developers regarding when and how objects are defined during the development process [9].

The choice of an application architecture is always a serious challenge. Web applications are not an exception. The remainder of this sub chapter describes different, existing architectures which some are more suitable for web architecture than others.

2.9.1 Client-Server

In the 90's the client-server architecture gained popularity, often using a graphical user interface as client and a relational database as server. The client-server systems are often seen as a 2-layer architecture where the rich client (sometimes also referred to as a fat client), which contains presentation code and other data processing code, constitutes one layer and the database server constitutes the data layer [25].

Though client-server systems work well in environments where the user count is relatively low the architecture fails to provide scalability if applied when the user count is unknown or very large, e.g. in the case of a web application. In addition, opportunities for reuse are limited because presentation code is often mixed with business logic code [37].

2.9.2 MVC

MVC (Model-View-Controller) has its roots in Smalltalk-80 and the Xerox PARC in the 70's [10] and it has become a fundamental idea behind graphical user interfaces in most object-oriented programming languages since then [45].

Originally, MVC was called a paradigm since the pattern terminology had not yet been applied in the software community. It was not until 1995 with the publishing of [28] by *Gang of Four* that the software design pattern idea was born. However, MVC was never called a single design pattern by Gang of Four, rather a collection of design patterns working together.

Since MVC is such a well known concept in the field it is almost a mandatory part in any text about modern software architecture and design, e.g. both [25] and [11] describe MVC as an architectural pattern.

MVC separates a system into three parts with the purpose of solving some of the problems with architectures like e.g. client-server. First, presentation is separated from content (View and Model respectively) such that the model manages behavior and data of the application domain whereas the view manages the presentation of information. The model-view separation enables one model to have many views thus opens for reuse, but more importantly, the separation enables the model to be built and tested independently from any visual presentation [11]. This is one of the key benefits of MVC [54].

Secondly, presentation is separated from control (View and Controller respectively). The controller should react to user actions (e.g. mouse and keyboard inputs) and inform the model and / or the view to change accordingly [11]. This second separation has shown itself to be less important. In fact, many Smalltalk versions did not make this separation and neither does most modern object-oriented frameworks.

2.9.3 Layered architecture

Application layering is a way of thinking about software construction where the principal subsystems are conceptually organized in the form of a layer cake [25]. Typically, each layer in the cake is stacked on top of another layer, interacting only with the adjacent layer directly underneath [9]. The interaction is restricted by a well defined contract which enables you to think of a layer as an isolated entity which can be developed without knowing much about the other layers. A layer can also be replaced by another layer as long as it fulfills its contract (the idea of software contracts was first described by Bertrand Meyer in 1988 [48]). Further, each layer hides the underlying layers such that layer two only uses the services of layer three, but is unaware of layer four and layer five (in case of a five layered architecture) [25].

Layering has proven to be a useful abstraction technique for software construction. However, there are many different layering schemes describing how to partition your application elements in different numbers of layers.

As mentioned, the traditional client-server architecture can be partitioned into two layers, presentation layer and data access layer. Object technology encourages the abstraction and reuse of business processes and data, thus decoupling application logic results in a three-layer system [9] as in Figure 2-13.

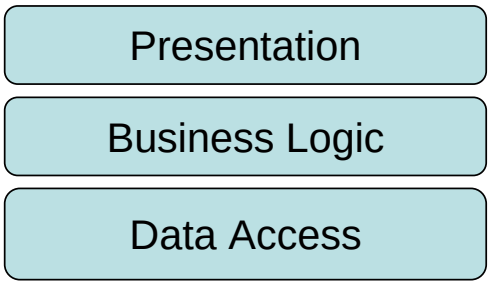


Figure 2-13. Basic 3-layered abstraction

The three-layer architecture is such a well known approach that Microsoft [54] describes it as the 3-layered application pattern which is classified as an architectural deployment pattern. The pattern is stated to be the solution to organizing the application such that business logic can be reused and provide deployment flexibility.

Brown, et al. introduce a five layer architecture with the additional Controller/Mediator layer and a Data Mapping Layer, thus decoupling the domain (or business model) ever further.

Layered architecture has become one of the most common techniques that software designers use to break apart a complicated software system [25]. Layering provides reusability, scalability, and maintainability, thus being essential in supporting the iterative development process. To quote Brown, et al.:

> "With the explosion of the Internet and related technologies, enterprise application requirements have made the existence of layered application architecture imperative." [9]

Layered architecture has proven itself over and over as we see it in operating system architecture and networking architecture [25].

2.9.4 Using ASP.NET

When using ASP.NET much of the architecture is already implemented and ready to use. First of all the code behind mechanism described in section 2.7.5 enables, in fact, the separation of presentation code (view) from the model-controller code in the MVC pattern. Further, the .NET framework and especially ASP.NET is designed to fit into the Windows DNA (Distributed interNet Applications) architecture which was originally designed by Kirtland [37].

Windows DNA is the Microsoft approach to writing distributed applications using Microsoft technologies and products.

Windows DNA is a three-layered architecture partitioned into a presentation layer, a business layer and a data access layer just like the ones described previously. The main difference between Windows DNA and the 3-layered application pattern lies in the way data is passed up from the data access layer. In Windows DNA all the layers operate on record sets that result directly from SQL queries issued by the data access layer. This introduces a tighter coupling when both the business layer and the presentation layer are aware of the database [25].

Another advantage of using ASP.NET is that Microsoft provides extended class libraries with tools and support which can save a lot of time implementing common functionality over and over again or investing in your own common library. An example of such a library is the Enterprise Library described next.

2.9.5 Enterprise Library

Enterprise Library, a free .NET 'add-on' developed by Microsoft, consists of a collection of application blocks that provide additional functionality often needed during application development; e.g. cryptographic, caching, and data access functionality. The application blocks are provided as source code and can be extended and/or modified by developers if necessary [23].

Version 2.0 of Enterprise Library, released in January 2006, includes the following application blocks; see Figure 2-14 [49]:

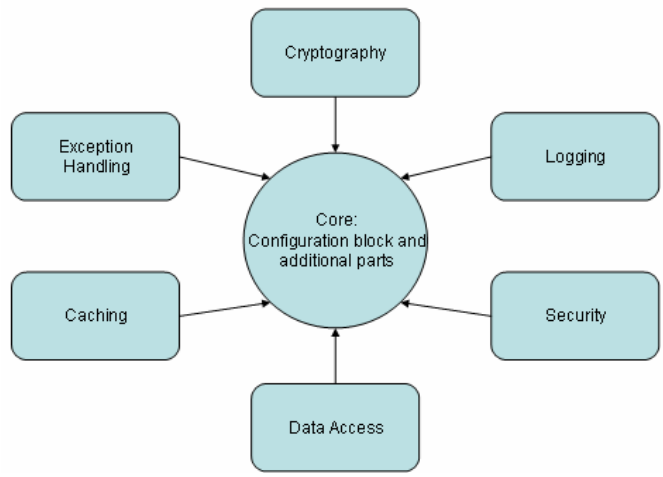


Figure 2-14. Enterprise Library

- Caching – Used for incorporating local caches into applications.

- Configuration – This block is arguably the most important one, since all the other blocks depend on it. The configuration block acts as the glue that connects the separate application blocks into a library [23].
- Cryptography – Contains symmetric encryption and hashing functionality.
- Data Access – Contains functionality for incorporating standard database functionality into the application.
- Exception Handling – Provides additional exception functionality, e.g. exception policy creation.
- Logging – Contains functionality to include standard logging features in the applications.
- Security – Used for including security and authorization functionality into applications.

The blocks generally provide a set of classes and interfaces that encapsulate best practices and new functionality recommended by Microsoft. For instance, the data access block supplements the ADO.NET-code – the same code will now work regardless of relational database used [67].

A pre-release of version 3.0 of Enterprise Library has been released as this is written. However, only functionality from version 2.0 has been utilized in the project, and no further discussion of 3.0 will be provided.

3 Investigation

3.1 Introduction

This chapter describes how an investigation was performed to find out whether any DNS administration software that satisfied Ninetech's requirements existed. The investigation began with a wide search for any type of DNS administration system that could administer the BIND name server software. The method used involved including all systems available for download through a web page, either free or an evaluation version.

Nine different systems were found in the search and from these three systems were chosen for closer inspection with focus on how well they met a set of criteria that were chosen based on the system requirements.

3.2 System descriptions

This section describes the systems found in the wide, first search.

3.2.1 DNS Control

DNS Control [46] is a web based DNS management system licensed under the GNU General Public License [26]. The system is developed by Justin Mazzi and designed for administration of BIND 9. DNS Control is written in PHP and stores information in a MySQL database. DNS Control has to be installed on the same computer as the name server software and is accessible using a web browser.

3.2.2 GBIND Admin

GBIND Admin [36] is a Linux desktop application for administration of BIND. GBIND Admin is licensed under the GNU General Public License [26] and is developed by Magnus Loef. The application should be installed and used on the same machine as the DNS server software.

3.2.3 Men & Mice Suite

The Men & Mice Suite [47] provides a management layer on top of standard DNS and DHCP server software and is a commercial product developed by the Icelandic software develop-

ment company Men & Mice. They are, according to their web site, market leading in DNS administration worldwide.

The suite consists of a central server, a web server, DNS server controllers, and a management console. The DNS server controllers are installed on each managed server and controlled via the central server using the management console, a JSP-based web user interface, or a command line interface.

3.2.4 mysqlBind

MysqlBind [68] is running on the Apache web server and is used to administrate BIND servers using a CGI-enhanced web interface. MysqlBind, which is developed by Gary Wallis, is licensed under the GNU General Public License [26] and uses a MySQL database server to store information. MysqlBind must be installed on the same computer as BIND.

3.2.5 NetDirector

NetDirector [20] is an open source administration system for managing a number of different services, e.g. HTTP, DNS and DHCP in a distributed network. NetDirector is developed by Emu software Inc. which also offers paid support. The software is licensed under the NetDirector Public License [21].

The NetDirector system consists of several separated parts which include a Java Servlet-based centralized server, called the Server Manager, running on the Tomcat web server. Agents, which are written in Python, handle the administrative part and can be attached to the Server Manager. The agents are specialized for specific services, e.g. DNS, HTTP, etc.

3.2.6 NIXU Namesurfer Suite

The NIXU Namesurfer Suite [52] is a commercial product developed by Nixu Ltd. in Finland that contains several different administrative tools for DNS, DHCP and IP Address management. The suite ships with an internal BIND server, a configuration daemon, a web server, and an integrated SQL server. Administration can be performed remotely through a command line interface or a Perl-based web user interface running on the web server.

3.2.7 ProBIND

ProBIND [35], licensed under the GNU General Public License [26], is a web based administration tool for one or more BIND servers. ProBIND is written in PHP and uses a MySQL database for storing zone information. The information is subsequently written to the DNS servers upon request from the user. The web interface runs on an Apache HTTP Server.

The last update of ProBIND was released in 2001 and it seems that no further development is being done on the project. ProBIND must be installed on the same computer as BIND.

3.2.8 WeBBind

WeBBind [56] is a graphical web interface written in PHP for managing BIND servers. It should be installed on a server running BIND as well as the Apache HTTP Server which makes it accessible using a standard web browser. It is an open source project without a specified license developed by Danilo Paliani and is distributed "as is".

The development of WeBBind seems to have ceased in April 2002.

3.2.9 Webmin

Webmin [13] is a web based user interface for administration of Unix which includes the possibility of BIND administration. It is written mainly by Jamie Cameron, runs on an integrated web server and uses CGI-scripts. Webmin must be installed on the same computer as BIND.

3.3 Closer inspection

The investigation so far clearly indicated that the administration systems could be divided into three separate categories. The ones in the first category are those where an application is executing on the computer running BIND and the administration is performed via the application's graphical user interface. The second category consists of script language based systems that must be running on a web server on the same computer as the name server software. The third category consists of well modularized systems, such that the BIND administrative part is separated from the web user interface, thus allowing them to be located on different computers.

The categorization in short:

1. Local administration only-systems
2. Script-based web applications installed on the computer running the name server
3. Systems where the web- and administrative parts are separated

The only system belonging to category one is GBIND Admin. Category two consists of DNS Control, mysqlBind, ProBIND, WeBBind and Webmin. Most of the examined systems belong to this category. In the third category we find our two commercial products, Nixu Namesurfer Suite and the Men & Mice Suite, as well as the open source system NetDirector.

First of all, category one was excluded due to the obvious reason that these systems yield no possibility of remote administration of the name server. Local administration only is not an

option. Next, all of the script-based solutions of category two depend on the fact that the web server they are running on is installed locally on the name server computer. According to the requirements, see Figure 2-1, the web server and name servers should be deployed on separate computers. Hence, the second category could be excluded. The third category could however not be excluded immediately since these systems can be deployed in a similar manner as in Figure 2-1. Thus, the solutions in the third category were chosen for further evaluation.

3.3.1 Men & Mice Suite

The Men & Mice Suite is a multi platform network administration system which supports administration of DNS, IP address management, and DHCP. The suite has a core component called the Men & Mice Central. It is a server that provides authentication and keep user- as well as audit trail information stored in a database. The central communicates with DNS Server Controllers which are installed on each managed name server.

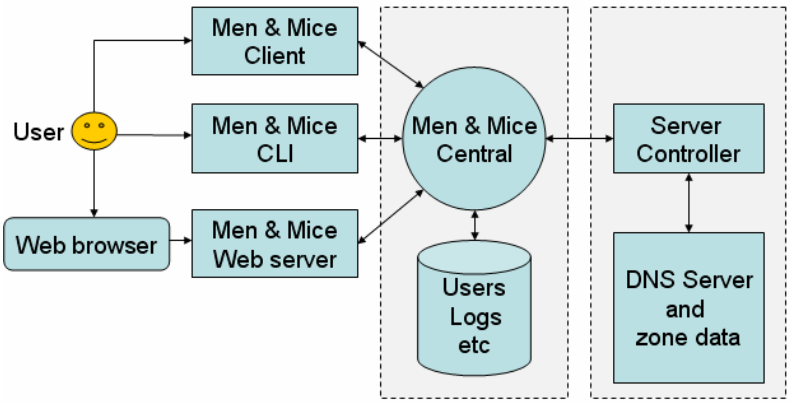


Figure 3-1. Men & Mice Suite – system overview

The users have some different choices for the administrative process. The Men & Mice Client (a Windows application) is shipped with the suite and provides a graphical user interface for administrating the managed hosts. The users can also administer the servers through a web based user interface, which is accessible by connecting to the suite's integrated web server. The web user interface is built on top of the Men & Mice CLI, which is a command line interface that can be used for customization purposes.

3.3.2 NetDirector

NetDirector is a Linux and Solaris system administration tool with an embedded CMDB which provides management for HTTP, DNS, DHCP, LDAP, Kerberos, File and Print services (FTP, Samba, NFS), E-mail, Users, and Groups. It is also possible to extend the functionality by developing other modules using Java.

NetDirector consists of three primary parts, see Figure 3-2. Server Agents are installed on each managed server and handle configuration changes on the servers, e.g. replacing files and loading new configurations. The Server Agents are controlled by the Server Manager, which is responsible for relaying configuration requests from the clients to the managed servers. Several Server Agents are typically attached to one Server Manager for centralized administration. The final primary part is the web based graphical user interface, which is accessible through a standard web browser.

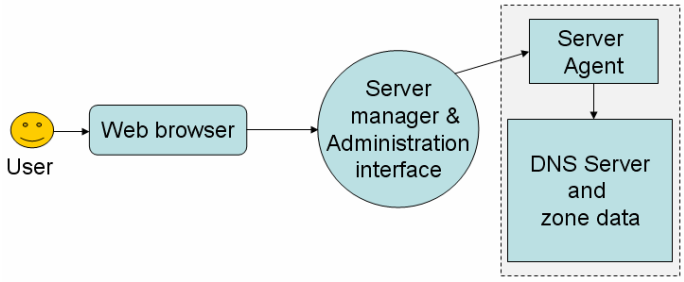


Figure 3-2. NetDirector – system overview

3.3.3 Nixu Namesurfer Suite

Nixu Namesurfer Suite supports DNS, DHCP, and IP address management. The suite consists partly of the Namesurfer primary component which includes an embedded name server and SQL database. The SQL database is Solid's Embedded Engine. There is also a specialized, integrated SSH/SSL-enabled Apache proxy included on which a web based graphical user interface runs. There is also a command line interface available through which scripts can be created to update DNS information automatically.

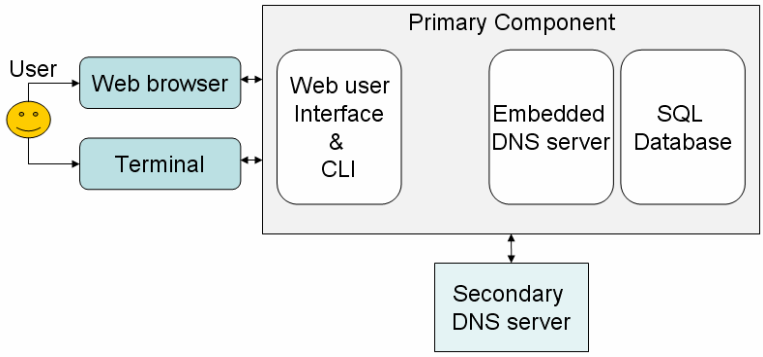


Figure 3-3. Nixu Namesurfer Suite – system overview

3.4 Criteria of evaluation

The following criteria were chosen for system evaluation.

3.4.1 Cost

Cost was defined as the price of the system itself and additional costs such as support and system update prices. According to the requirement section cost was a factor when choosing an administration system and should not exceed the developers' expected salary if a new system was to be implemented.

3.4.2 Documentation

In the documentation criteria, several forms of documentation were relevant, installation guides, user manual, design documents, etc. This kind of information makes it easier to start using the system and simplifies the troubleshooting of problems that might arise.

3.4.3 Extendibility

Extendibility is in what way, if any, it is possible to extend the system later on; for instance if additional functionality is needed.

3.4.4 Functionality

According to the requirement specification the system had to be able to handle and administrate both customers and domains and connect these two together. It should for example be possible to list domains that belong to a specific customer.

3.4.5 License

What kind of license the system is licensed under was an important matter due to the consequences this might have when the system is used or altered, e.g. the GPL license forces people that modifies the software to openly share those changes.

3.4.6 Platform

According to the specifications the web server should be running Windows and .NET. The system should also use SQL Server as database server.

The system should store information about unexpected system events in the Windows Event Log.

3.4.7 Process

Change requests should follow ITIL's change management process standards, see 2.4.

3.4.8 Security

The security requirements from the company consisted of two parts. On one hand, the security of the communication, solvable by for instance encryption and on the other hand the access control capabilities. Different users needed different access levels and this had to be based on users and groups in Active Directory, since the system had to use Active Directory to handle access rights.

3.4.9 Traceability

Traceability was defined as the system's capability to record information about changes that users make to the database.

3.5 Evaluation

3.5.1 Test setup

All the applications were tested and evaluated according to the criteria in 3.4. The test setup consisted of:

- One computer running Red Hat Linux 9, acting as a primary name server.
- One computer running SUSE Linux 10, acting as a slave name server.
- One computer running Windows XP, from where the administration took place.

Both name servers were running the latest versions of BIND, see 2.3.11; otherwise clean installations were used. The systems were subsequently installed according to any available documentation.

3.5.2 Men & Mice Suite

3.5.2.1 Cost

The cost of the system is approximately 2150 USD. This amount was considered acceptable when compared to a quick estimate of the expected developer salary for developing a new administration system.

3.5.2.2 Documentation

Both installation- and administrative procedures are well documented and accessible from the company website. The Men & Mice suite lacks database documentation.

3.5.2.3 Extendibility

The command line interface (CLI) can be used for building customized user interfaces.

3.5.2.4 Functionality

The Men & Mice Suite has the required functionality concerning domain handling but requires extensive customization to enable customer handling. A new graphical user interface must be built on top of the command line interface and a database with customer data must be created and integrated with the domain handling.

3.5.2.5 License

The Men & Mice Suite is a commercial product and the source code is not publicly available.

3.5.2.6 Platform

According to the requirements all zone information should be stored in a database along with customer information. The Men & Mice Suite does not use a database to store the zone files but manipulates the files directly on the name servers.

There is no .NET-based web user interface with administrative capabilities available, to adhere to the requirements, one must exist.

Windows Event Log is used which satisfies the requirements.

3.5.2.7 Process

Users cannot make change requests for later approval, all changes take effect immediately. This does not conform to the ITIL standards.

3.5.2.8 Security

All communication is secured by encryption, and access control can be implemented in different ways. Either you can create own users and groups and assign different rights, or you can base authentication on a native user authentication method, e.g. Active Directory, in line with the requirements.

3.5.2.9 Traceability

The system stores zone change history in a database in the Men & Mice Central. The log saves information about who made the change, when it was made, what was replaced and what it was replaced with. This satisfies the requirements.

3.5.3 NetDirector

3.5.3.1 Cost

The core application is free of charge, and support and customization can be purchased for an annual price of 199 USD per managed node. For Ninetech, the price would be 398 USD á year for the two DNS servers, which is an acceptable amount.

3.5.3.2 Documentation

Installation- and administrative procedures are well documented and can be downloaded from the company website. There is no database documentation available.

3.5.3.3 Extendibility

NetDirector is open and extendible such that anyone with experience in Java can develop custom modules. Additionally, as mentioned in 3.5.3.1, Emu software Inc. can develop custom modules as well for a price.

3.5.3.4 Functionality

NetDirector has all the required functionality concerning the domain handling. However, the system lacks the ability to handle customers which is a requirement. Thus, customization is needed to address this issue.

3.5.3.5 License

Since NetDirector is licensed under the NetDirector Public License it requires that any further development of the system must also be distributed under the same license and made publicly available. The requirements state that this should not be the case.

3.5.3.6 Platform

The NetDirector Server Manager is initially only available for Linux but since it is Java-based and uses the Tomcat web server it is theoretically possible to port the program to the Windows platform. However, this is still incompatible with the requirements that states that IIS should be used as web server, not Tomcat. Further, SQL Server is not used as database server and Windows Event Log is not used – which is incompatible with the requirements.

The server agents are installed locally on each managed server. The agents are written in Python which meet the requirements.

3.5.3.7 Process

Users cannot make change requests for later approval, all changes take effect immediately. This does not conform to the ITIL standards.

3.5.3.8 Security

Communication between the Server Manager and the managed hosts is secured using SSL. However, the communication between the web user interface and the Server Manager is not encrypted which is a potential security risk.

Access control is implemented by an integrated role based system which is administered through the web interface. However, it is not possible to base the access control on Active Directory entities as required.

3.5.3.9 Traceability

NetDirector has the required traceability possibilities. The system tracks what was changed, who made the change and when the change occurred. Administrators also have the possibility to rollback changes.

3.5.4 Nixu Namesurfer

3.5.4.1 Cost

The company behind Nixu Namesurfer did not offer a final prize for their product.

3.5.4.2 Documentation

An installation guide is included along with a user manual of low quality. Database documentation is available.

3.5.4.3 Extendibility

A command line interface is available which could be used as a back end for customized user interfaces.

3.5.4.4 Functionality

Nixu Namesurfer has the required domain handling functionality but lacks the ability to handle customers, which is a requirement. Customization is required to address this issue.

3.5.4.5 License

Nixu Namesurfer is a commercial product and the source code is not publicly available.

3.5.4.6 Platform

The Nixu Namesurfer primary component is available exclusively for Linux, and is very Linux distribution dependent. It did not work on either of the tested distributions (Red Hat 9 and SUSE 10) so it was tested on Fedora 4 and Red Hat Linux Enterprise Server 4 as well. The last distribution was the only distribution where the system behaved correctly.

Nixu Namesurfer uses an integrated Apache web server to allow remote administration. Further, Solid's SQL Server is used instead of SQL Server. Neither Apache nor Solid's SQL Server is acceptable according to the requirements.

Windows Event Log is not used since Nixu Namesurfer is Linux based. This does not satisfy the requirements.

3.5.4.7 Process

Users cannot make change requests for later approval, all changes take effect immediately. This does not conform to the ITIL standards.

3.5.4.8 Security

Remote connections from users are secured using SSL. SSH is used to secure remote BIND configuration updates from the primary component.

For access control the administrator can assign users to groups and assign them different rights. However, this cannot be done using Active Directory as the requirements state.

3.5.4.9 Traceability

An undo/redo log is available that records information about who made the change, when the change was made and what was changed, thus satisfying the requirements. Administrators can also rollback committed changes.

3.6 Conclusion

The Men & Mice Suite is a competent system which did not yield any installation problems and could be configured easily with the help of high-quality documentation. It has most of the functionality required, the domain handling is handled very well and no obvious features are missing regarding this. Group operations are possible, but only when using the Men & Mice Client. Traceability and security are well implemented too, all traceability requirements are met, all communication can be encrypted, and it is possible to base access control on Active Directory. However, there is one major drawback specific to The Men & Mice Suite – it does

not use a database for storing domain data, thus it obviously lacks database documentation as well. Instead it operates directly on the DNS configuration files on the name servers.

NetDirector on the other hand is freely available which makes it a strong candidate. The documentation is well written and covers the required topics, with the database as the only exception. Another interesting feature is the ability to rollback committed changes. However, while NetDirector is free, the necessary support is costly and thus makes the system less desirable. Further, there is no way of using Active Directory for user authorization which is an important requirement, and the error handling could be improved regarding the lack of informative messages that are presented to the user.

Nixu Namesurfer has the required domain handling functionality. Further, all communication is secured using different kinds of encryption techniques and it is also the only system which provides database documentation. There is some group operation functionality included, and it is possible for administrators to rollback committed changes. However, the system has many flaws. First, it is difficult to install and deploy. According to the documentation, which is of low quality, it should support a number of Linux distributions but it has problems with at least three of the most common ones. The only distribution where the system behaved correctly was Red Hat Linux Enterprise Server 4, which Ninetech is not using. Further, just like NetDirector, Nixu Namesurfer lacks the ability to use Active Directory for authorization.

There are some problems common to all the evaluated systems. Neither of them has the required web user interface developed in ASP.NET nor use a SQL Server database. Further, none of the systems has integrated customer handling which is a critical requirement.

Additionally, none of the systems adheres to the ITIL processes of change management by supporting requests for changes from users. Hence, whether any of the systems are chosen as a whole or extended, extensive customization is necessary to obtain all the required functionality.

To implement customer handling, an external database with customer information must be created and connected to the respective system's domain information. Since the requirements state that there should be one database that contains both customer and domain information, this alone makes none of the systems a suitable choice. The Men & Mice Suite does not even use a database for storing zone information, thus it might be very difficult to connect the external customer information to the zone information. This will potentially be a problem in NetDirector as well, since no database documentation is available.

Another problem is that if the administration of the systems should be performed using the web user interfaces, these interfaces has to be used as back ends to the required .NET-based web user interface. In this case, two web servers are needed, which create potential security risks and consume unnecessary resources. In NetDirector, this is the only option since no command line interface is available. Nixu Namesurfer is too Linux distribution dependent and does not work on the distributions Ninetech is using. On top of it all, neither NetDirector nor Nixu Namesurfer can use Active Directory for user authorization.

Table 3-1. Requirement compatibilities

Requirement	Men & Mice Suite	Nixu Namesurfer	NetDirector
.NET	No	No	No
SQL Server	No	No	No
Access control – AD	Yes	No	No
Windows Event Log	Yes	No	No
Documentation	Yes*	No	Yes*
ITIL-processes	No	No	No
Customer handling	No	No	No

* Database documentation is missing

Combined, the above mentioned issues make using any of the systems an unsuitable choice. They all lack support for many crucial requirements (see Table 3-1); hence there are more advantages in developing a new system streamlined for the organization.

3.7 Summary

In the first part of the investigation, a wide search was made for DNS administration systems. Nine different systems were chosen: DNS Control, GBIND Admin, Men & Mice Suite, mysqlBind, NetDirector, Nixu Namesurfer Suite, ProBIND, WeBBind, and Webmin. The systems were categorized as local administration only, script based web applications, and systems where the web- and administrative parts were separated. Based on the requirements, only the systems in category three were chosen for closer inspection. The systems in category three were:

- The Men & Mice Suite – a commercial multi platform network administration system which supports administration of DNS, DHCP, and IP address management.

The suite is separated into a central server, a server controller on each managed host, and a command line interface on which a desktop client and web user interface are built.

- NetDirector – an open source administration system which supports management for various Linux services, e.g. HTTP, DNS, and DHCP. NetDirector is divided into a server manager and server agents on each host. The server manager is implemented as a web application and has a web user interface.
- Nixu Namesurfer Suite – a commercial system for administration of DNS, DHCP, and IP address management. The suite's primary component is accessible through an administration web user interface or a command line interface. Nixu Namesurfer uses an embedded name server with secondary BIND servers.

The systems were evaluated against the following criteria: Cost, Documentation, Extendibility, Functionality, License, Platform, Process, Security, and Traceability. All the systems had sufficient domain handling capabilities but they all lacked certain functionality which led to the conclusion that neither system satisfied Ninetech's requirements.

The missing functionality can be summarized as follows: None of the systems had integrated customer handling or support for ITIL change management processes. None of the systems used the required database- or web server, and no .NET-based web user interface was available. Thus, the investigation concluded that a new system was to be implemented to fully satisfy Ninetech's needs.

4 Implementation

4.1 Introduction

This chapter will initially provide a discussion of possible solutions to various problems encountered during the project, and the final choices made will be explained later on. An architectural description is provided along with an explanation of how the system is supposed to be used. Screenshots are used to illustrate the explanations.

The chapter ends with a more detailed discussion about a couple of the more interesting and difficult problems encountered, such as how transactions were handled.

Since code and other internal details are confidential, certain information is not included in the thesis.

4.2 Possible solutions

This section describes some different possible solutions available when implementing an administration system as described in the requirements (section 2.2 and appendix A) and which fulfills the criteria in the previous chapter.

Chapter 3 concluded that no system was found that adequately satisfied Ninetech's needs. However, a short analysis of that chapter concludes that the system can be divided into two physical parts; a web user interface part and a name server part which may contain several name servers. According to the requirements these parts should reside on different physical machines, or tiers (Figure 2-1). The terminology of tiers seems to differ in the field though Fowler, et al. [25] made an attempt to explain what a tier is and what it is not. This thesis will use Fowler's terminology. Based on the investigation in chapter 3 two possible solutions appears. The system can be developed using part of an existing solution as back end or the system can be developed from scratch.

In the first option there is only one suitable choice when it comes to an existing system, namely the Men & Mice Suite. It does not store the domain information into a database and the suite provides a command line interface which can handle the communication with the Linux name servers. Thus, the only part that could be used in the Men & Mice Suite is the command line interface.

The second option is to base the Linux communication on something else and there exist another two possible solutions. 1) A client or service can be constructed and installed on each name server machine. The service would receive commands from the web server tier and take the appropriate actions on the name servers to implement the commands, either by retrieving data from the database itself or receiving the data from the web server tier. 2) The web server tier connects remotely to each name server and transmits files and commands which are executed on the Linux machines.

Both of these options require means for communication and Table 4-1 lists the considered technologies.

Table 4-1. Communication technologies

Name	**Technology**	**Security**	**Note**
SSH	Secure shell	Encrypted	Requires configuration*
SCP	Secure file transfer	Encrypted	Requires configuration*
SSL	Secure socket	Encrypted	Requires installation*
SFTP	Secure file transfer	Encrypted	Requires configuration*
SAMBA	Windows file sharing	Encrypted	Requires installation*
TELNET	Remote shell	Unencrypted	Disabled due to security issues*
FTP	File transfer	Unencrypted	Disabled due to security issues*

* Can vary between Linux distributions and versions

Each of these technologies can be used in either case; however some are more suitable than others, e.g. Telnet and FTP are usually disabled by default in most Linux distributions since they are considered insecure due to the lack of encryption. SAMBA is a file sharing suite which e.g. provides file services to Windows systems. While SAMBA provides encryption it implements Windows SMB (server message block) which requires many open ports, thus introducing more insecurity than it counteracts [66]. The benefits of SAMBA is that no file transfers are needed since the file sharing suite makes it possible to mount the name server's file systems locally on the web server. Thus, no network communication is necessary when it comes to programmatically connecting to the servers since that is performed on the operating system level.

SSH (Secure Shell) [29] is a network protocol that allows a secure channel to be established between peers and is the only secure technology for executing shell commands like `ls`, `pwd`, `cd`, `rm`, etc., remotely. SSH has in itself no support for file transfers, but other protocols

can be used on top of SSH. SSH is not integrated with the operating system thus an implementation of the protocol is needed in the web server tier to programmatically establish a connection to the name servers. An implementation can either be constructed in the project or a solution developed externally can be acquired.

SCP (Secure Copy) and SFTP (SSH File Transfer Protocol) [29] are both secure file transfer protocols though SFTP has more functionality than SCP. Further, SFTP is a stand alone protocol that does not require an underlying protocol such as SSH, though it is usually used with SSH-2. SCP on the other hand fully relies on SSH to handle communication. SCP only provides basic file transfers to and from the server. Whether SCP or SFTP is chosen both technologies require an implementation in the web server tier which can be developed or acquired externally.

SSL (Secure Socket Layer) [29] provides endpoint authentication and secure communication using encryption but does not provide any services like file transfer or shell access. Thus, applying SSL to the project would require a service component to be constructed and deployed on each internal name server. The component would provide an interface to the web server tier which would communicate with the name servers only through that interface. If SSL is chosen an implementation is required in the name server tier.

According to the requirements the system should be implemented using ASP.NET as a web application. The rest of this section describes possible solutions for web application architecture and design.

4.2.1 Architecture

At Ninetech, a software architecture is used in most new projects regardless of what type of system that is to be constructed, e.g. a desktop application or a web ditto. The architecture implements the *3-layered application* pattern described in chapter 2.9 with small divergences. This project was strongly encouraged to apply the architecture used at Ninetech in the system. However, the architecture was neither formalized nor documented which means that it exists in each coworker's mind only, thus having as many interpretations as there are developers. Hence, this would imply that this project also formed its own interpretation of the architecture and the architecture used and documented in this thesis is the authors' interpretation, not necessary the architecture that is most often used at Ninetech. The architecture implements the *3-layered application* pattern with the following divergences:

The domain model, called Business Object in the architecture, is implemented using the *Separated Interface* design pattern [25], i.e. the interfaces of each business object are ex-

tracted into a separate module (see Figure 4-1). The purpose is to break the dependency between the business objects used in the Presentation layer and the implementation, thus enabling objects that are pure mappings of data entities in the database to exist only in the Data Access layer, not also as a gateway in the Business Logic layer.

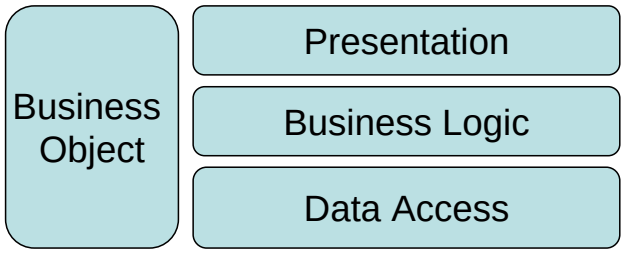


Figure 4-1. 3-layered application architecture + Business Object

The instantiation of business objects are, however, managed in the Business Logic layer using the *Plug-in* design pattern [25], i.e. the actual instance is decided at configuration time in distinction to compile time.

4.3 System overview

The web administration interface has been developed primarily for use with Firefox 2 and Internet Explorer 7, but should be compatible with any modern web browser. The system provides the users with five different views; see Figure 4-2. Users can navigate to any of the list views or the change request compilation (if they are Change Managers – see 4.5.4 for an explanation of roles) at any time by clicking on the appropriate link in the menu bar. The arrows in the figure illustrate the navigation possibilities available to the system users.

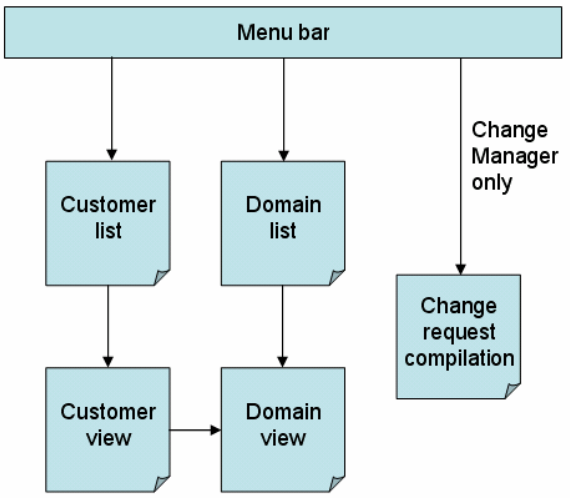


Figure 4-2. System views

The default page of the system is the customer list which provides a list of all the customers in the system, see Figure 4-3.

Figure 4-3. Customer list

From the customer list, all users can:

- Inspect a specific customer by clicking *Detaljer* (details).

Administrators also have the ability to:

- Request a new customer by clicking *Lägg till kund* (add customer). The user will be presented with an empty customer view, see Figure 4-4, where customer details are entered. Domains may also be added at the same time. By clicking *Begär förändring* (request change) – the new customer becomes waiting for approval.
- Request removal of one or more customers by selecting the checkboxes to the left of the customer name and subsequently click *Begär ta bort* (request removal).

The customer view, see Figure 4-4, provides details about a specific customer.

Figure 4-4. Customer view

From the customer view, all users can:

- Inspect customer name and number.
- See a list of the customer's domains.
- Inspect a specific domain by clicking *Detaljer* (details).

Administrators also have the ability to:

- Request changes of customer name and number by entering the new values and click *Begär förändring* (request change).
- Request new domains by clicking *Lägg till* (add). The administrator will be presented with a domain view, see Figure 4-6, where domain details should be entered. When finished, the administrator is returned to the customer view and submits the actual request by clicking *Begär förändring* (request change).
- Request removal of one or more domains by selecting the checkboxes to the left of the domain names, click *Ta bort* (remove) and finally click *Begär förändring* (request change).

The domain list, see Figure 4-5, provides a list of all domains the system.

From the domain list, all users can:

- Inspect a specific domain by clicking *Detaljer* (details).

Administrators also have the ability to:

- Request a new domain by clicking *Lägg till domän* (add domain). The user will be presented with an empty domain view, see Figure 4-6, where domain details should be entered.
- Request removal of one or more domains by selecting the checkboxes to the left of the domain name and subsequently click *Begär ta bort* (request removal).

Figure 4-5. Domain list

The domain view, see Figure 4-6, provides details about a specific domain.

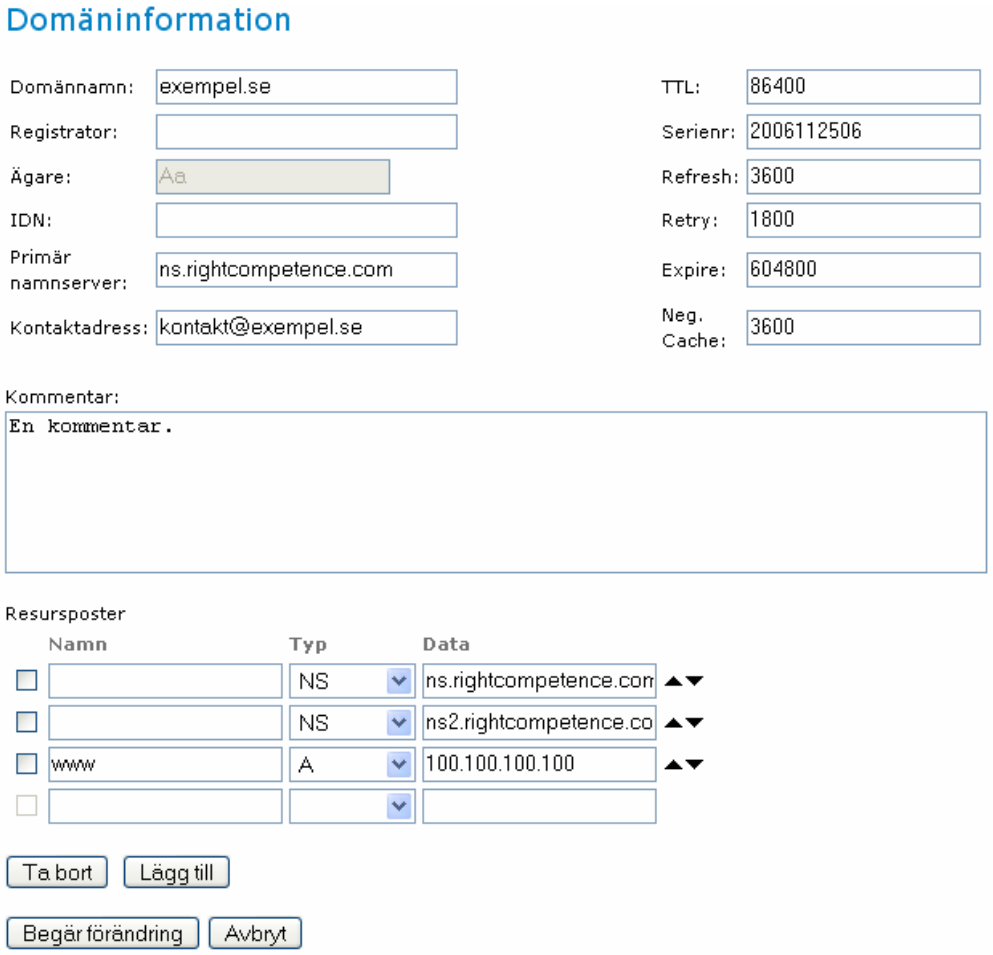


Figure 4-6. Domain view

From the domain view, all users can:

- Inspect domain and resource record details.

Administrators also have the ability to:

- Request changes of domain and resource record information by entering the new values and click *Begär förändring* (request change). It is the administrator's responsibility of entering correct values – the only existing data validation is a check for empty fields.
- Add new resource records by clicking *Lägg till* (add).
- Remove one or more resource records by selecting the proper checkboxes next to the resource records and subsequently click *Ta bort* (remove).
- Alter resource record ordering by clicking the arrows to the right of the resource records.

Change Managers can further click *Begärda förändringar* (requested changes) in the menu bar to access the change request compilation, see Figure 4-7.

Figure 4-7. Change request compilation

The change request compilation view provides the change manager with a view of all existing change requests in the system. If an administrator made several different changes before clicking the request change-button, the changes are grouped together and must be approved as a whole or not at all. If the change is supposed to be approved – the radio button beneath *Ja* (yes) ought to be marked, otherwise the other radio button. This means that the changes must be either approved or denied, it is not possible to set aside a change for later approval.

The change manager can see who made the requested changes and at what time the requests were made. The list includes a complete description of the requested changes, which the change manager must inspect before approving/denying the requests. When the button is clicked – all approved changes are activated. The administrators who requested the changes are automatically notified of the approval/denial by e-mail. The e-mail addresses are automatically retrieved from the respective user's Active Directory user account.

If an error occurs during the activation of the new changes – an informative message will be displayed, and no changes are activated. The exception to this rule is if a name server does not start up properly after an update. In this case, the changes are activated, but the name server process must be restarted manually on the computer with the inoperative name server.

4.3.1 Modifying internal name servers

Internal name servers cannot be modified from the web administration interface in the current version of the system. Adding and removing new name servers or modifying the IP addresses of existing ones has to be done directly in the database – using for instance SQL Server Management Studio.

Web.config is used to alter, add, and remove other name server details the system utilizes – such as username, password, and file paths.

4.4 System description

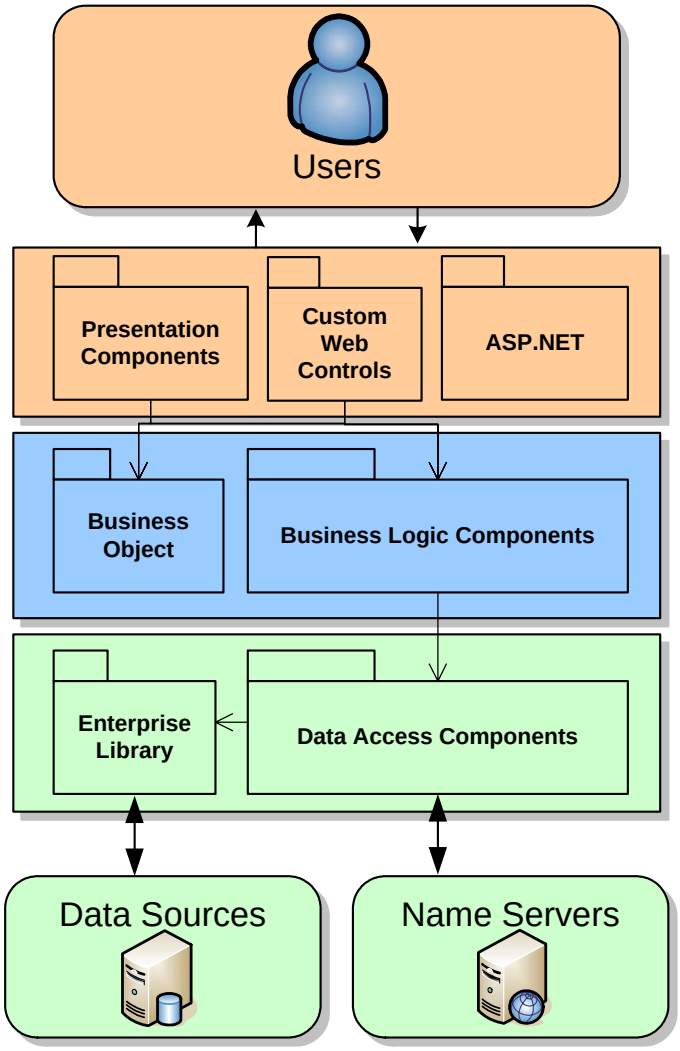


Figure 4-8. Architectural model

The architecture of the system is based on the authors' interpretation of the architecture mostly used at Ninetech as described in section 4.2.1. The system is partitioned into three layers of abstraction stacked on top of each other as visualized in Figure 4-8. Each layer provides a service to the adjacent layer above in the stack by defining a public interface. The Business Object package in the Business Logic layer is a little special since it does not contain any implementation classes, only interfaces for the business object model presented in detail in appendix B. The Business Object package is what one might call an interface repository (in

distinction to an implementation repository which may contain both implementation classes as well as interface classes).

The Presentation layer contains components that handle the visual view of the system and the navigation of the web application. A large part of the components are part of the ASP.NET framework as described in chapter 2.7. The application specific components of the Presentation layer are divided into two separate parts, one for the web page templates and the corresponding controller code, and one for the custom web controls. The latter contains custom controls which implement common behavior in the graphical user interface and enable web pages to reuse behavior when appropriate (Web controls are custom Server Controls which where described chapter 2.7).

The Business Logic layer contains parts from the Business Object interface repository, components that enforces the business rules of the system. And, in addition, components that play the roles as factories – which are used by the Presentation layer to instantiate business objects.

The Data Access layer contains components which manage communication with databases and external resources like the name servers in the system. Part of the Data Access layer is the Microsoft Enterprise Library described in chapter 2.9.5 which is used to fully abstract any database specific code from the application specific code. Thus, all communication with databases is managed through the Enterprise Library. Additionally, a third party component, which is described later, which implements the SSH protocol is also part of the Data Access layer. SSH is used for secure connections with the name servers, this is described in the next section.

4.5 Detailed descriptions

This section will provide descriptions for several problem areas encountered during the project.

4.5.1 State

All entities (customers, domains, and resource records) in the system have a state – Clean, Dirty, New, or Deleted, see Table 4-2. The states are used to avoid potential problems that might occur during request approval/denial; for instance – if one user requests a deletion of a domain – another user shall not be able to add additional requests on the same domain. If this would happen, and both requests are approved, the second request will operate on a domain which no longer exists in the system.

The state management also caused some other concerns, e.g. when a customer is requested for removal, all of the customer's domains and resource records are to be deleted as well, and must be marked accordingly. It was also decided that removals should only be possible on entities with no active change requests, which has the consequence that if an administrator for instance requests a change for a field in a domain, it will not be possible to remove the customer the domain belongs to either. Hence, the system needed logic for setting states properly in a number of situations.

At first, an iterative solution was used for setting the states. This solution caused several problems; see section 5.4.2 for further discussion. Instead, it was decided that the entity that was marked for removal must mark every other resource that becomes involved correctly; see Figure 4-9.

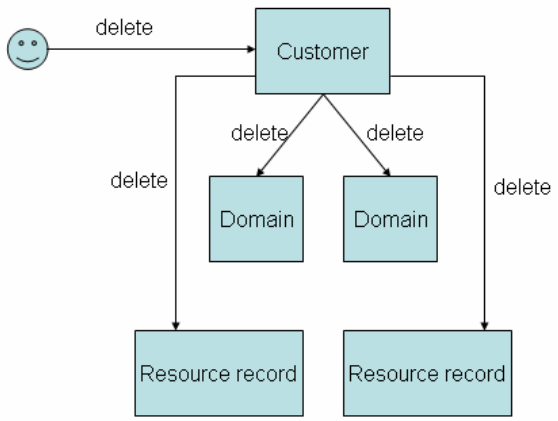


Figure 4-9. State setting

The system utilizes colors in the graphical user interface to notify the customer of the current state of the different resources. Resources with no active change requests are not coloured (clean-state) at all. Resources with requested changes are grey (dirty-state). When customers/domains/resource records have been requested for removal – they become red (deleted-state). New, not yet approved customers/domains/resource records are coloured green (new-state). Additionally, the relevant fields and buttons are disabled to make sure that users cannot make inappropriate change requests.

When specific fields have an active change request – they are grayed out, see Figure 4-11, and cannot be altered until the associated request is no longer active.

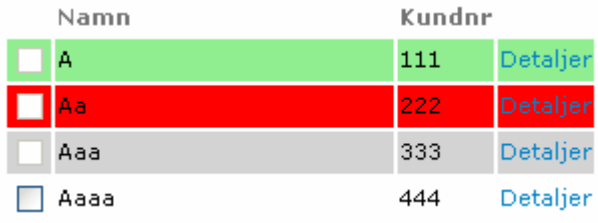


Figure 4-10. State coloring in the user interface

Figure 4-11. Field change request

The following table provides a description of the access rights and what functionality that is available for the different resources in all possible states.

Table 4-2. States

BO / State	Clean	Dirty	New / Deleted
Customer	Inspect fields Inspect domains Remove customer Change fields Add domain Remove domain* Change domain**	Inspect fields Inspect domains Modify unchanged fields Add domain Remove domain* Change domain**	Inspect fields Inspect domains
Domain	Inspect fields Inspect resource records Remove domain Change fields Add resource record Remove resource record*** Change resource record***	Inspect fields Inspect resource records Modify unchanged fields Add resource record Remove resource record*** Change resource record***	Inspect fields Inspect resource records
Resource record	Inspect fields Remove resource record Change resource record	Inspect fields	Inspect fields

* The domain status must be Clean

** The domain status must be either Clean or Dirty.

*** The resource record status must be Clean

A special case is the resource record reordering functionality. The order can be altered regardless of resource record state, but – it can only be done once. When a change request is made that modifies the order, the involved resource records become dirty and no further reordering is possible until the request is approved/denied.

4.5.2 Logging

4.5.2.1 Error logging

The system uses the ASP.NET-file global.asax for logging application errors. The global event handler Application_Error is used to capture all unhandled exceptions, and the event handler subsequently examines the exceptions and writes the exception information to the Windows Event Log. The usage of this event handler made error catching and logging a trivial matter.

In certain cases, the system catches exceptions and inserts a customized message before rethrowing the exceptions. This approach is used when the user are to be provided with information that the system cannot provide on its own. E.g. when a name server fails to start – the system inserts information about which name server that has failed, along with the original exception message. The catch-clauses are also often used in these cases to display a message to the user in the web interface.

4.5.2.2 Change Request Logging

All change requests are stored in the database, along with a textual description and a state. The change requests assume a pending-state when they are created and are updated to an approved- or denied state when a change manager takes the decision.

The change requests are never removed from the database – hence all changes are stored if they have to be revised at any time. Revising might be necessary if, e.g., a user proposed a change that was improper and the change manager missed the problem during the inspection before approving the request.

4.5.3 Initial database population

Since Ninetech's name servers already had several hundred active domains – information about those preexisting domains had to be inserted into the administration system. A manual insertion of this information was impractical due to the time it would take; hence a custom

tool was developed with the sole purpose of populating the database with the preexisting information.

To be able of joining the initial domains and customers together, the company's documentation was used to create text files that mapped domain names, customer names, and customer numbers to each other.

To populate the database, all relevant zone files should be placed in a folder. Upon execution of the tool, it reads all the zone files sequentially, derives domain names from the name of the zone files, and DNS configuration settings by parsing the contents of the zone files. The domain names are mapped to customers by using the created text files mentioned earlier, and all information is thereafter inserted into the database.

4.5.4 Security

The security of the administration system was carefully designed to try to avoid potential misusage or sabotage, even if the latter is improbable since the system is accessible from the company internal network only. This section is focused on the security implemented using ASP.NET. For a discussion of communication security with the name servers, consult section 4.5.5, and for database security information, see 4.5.6.

Access to the system itself is controlled using the Windows authentication mode, and the first security check is made when the user tries to access the system, regardless of what page that is requested. The configuration file web.config specifies that everyone is denied access except users that belong to specific Active Directory groups. If the user belongs to any of the specified groups, initial access is granted. The users also obtain different access rights depending on which Active Directory group they belong to.

The system roles and the related access rights are:

- **User** – A user can only view information, and only information that is currently active. That is, the user cannot see any proposed changes or new resources; which is why no coloring is available for members of this role.
- **Administrator** – An administrator is an extension to the user role and can make requests for changes to the information in the system.
- **Change Manager** – Has the same rights as an Administrator but can also view a compilation of the requested changes and approve/deny the requests.

Note that a Change Manager belongs to all three groups; an Administrator belongs to the Admin- and User group, while a User only belongs to the User-group.

4.5.4.1 Page security

All pages in the system derive from a class named AppPage, a customized class that derives from the standard Page-class – see Figure 4-12 – and acts as the base class for all pages in the system.

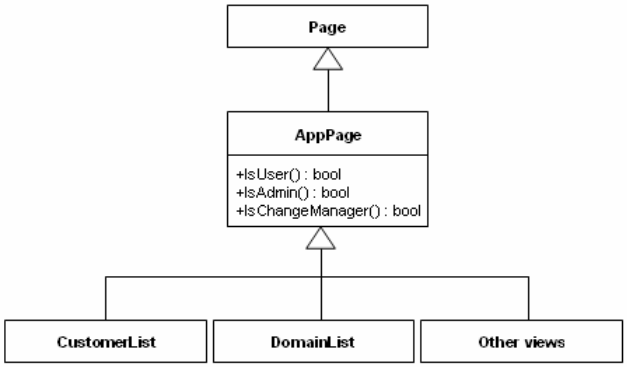


Figure 4-12. Page structure

The AppPage-class contains three methods used for security checks – `IsUser()`, `IsAdmin()`, and `IsChangeManager()`. These methods use the `IsInRole()`-method to validate the user by controlling if the user currently belongs to the associated role (group in Active Directory).

The different pages utilize the methods derived from the AppPage-class to enable, disable, or block access to specific parts of the system. E.g., all request functionality is disabled if the user does not belong to the Admin group. And, as mentioned above, the change request compilation is accessible only for Change Managers.

Access is always denied by default as stated in the security design principle by Saltzer [61].

The views and the provided functionality are further explained in section 4.3.

4.5.5 Name server communication

Several solutions existed for how to solve the communication with the name servers. This section will provide an explanation of the chosen solution and why it was selected.

Due to the developers' experience and knowledge in C#, it was decided to create all the required files using C# on the web server to reduce the implementation time needed. The generated files should then be sent to the name servers utilizing some file transfer supporting protocol.

The zone files are generated from the information in the change requests, and the configuration file is generated from the information in the database. Since the configuration file also contains static parts, a template file is used to provide these parts.

Using this approach, research indicated that two, quite small, separate tasks would be sufficient for handling everything related to the name servers:

1. Transmit commands to the name servers – for restarting the name servers, removing, and renaming files.
2. Transmit files to the name servers – zone files and a configuration file, named.conf.

Since both of these tasks are manageable with standard Linux commands and a file transfer service, the easiest and selected approach was to let the web server tier connect remotely to the name servers and simply transmit the necessary files and commands.

Hence, some kind of communication technology had to be used – for a list, see Table 4-1. Since the time frame did not allow the developers to create an own implementation of any of the considered communication technologies, it was decided to use an existing API for .NET with the required functionality.

Several commercial API's and one free (open source) API [27] were found and tested. The open source API was chosen, because it was free of charge and had implementations of the SSH and SCP protocols, which satisfied all the requirements. Both SCP and SSH are active by default in most major Linux distributions. The API was also easy to use and the license did not yield any problems.

Using the API, it is simple to setup a connection to the name servers by providing a username, a password, and an IP-address. The username and password are retrieved from the ASP.NET configuration file web.config, which makes it easy to change those if necessary. The IP-address has to be changed in the database.

Testing indicated that the only part of the communication process that took a measurable amount of time was the connection initiation. Due to this, it was decided to create only one connection to each relevant name server and keep the connection open until all commands have been processed. The commands are generated from the change requests and stored in a list, and when all change requests have been processed – a new configuration file is created (if needed) and scheduled for transmission along with a restart command. Finally, the list of commands is processed and the commands and files are transmitted to the name servers.

Figure 4-13 illustrates the name server updating process.

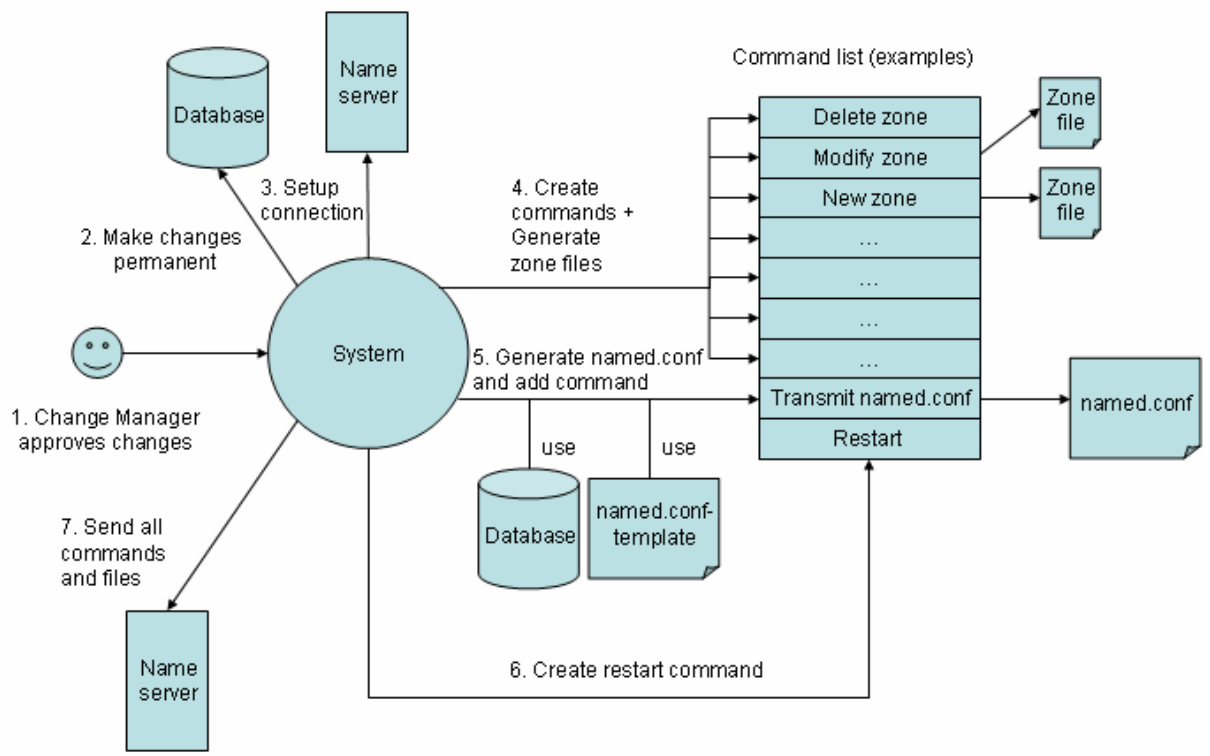


Figure 4-13. Name server updating

As mentioned earlier, SSH is used to transmit commands, and SCP is used to transmit the files from the web server to the name servers. A beneficial side effect of using SSH and SCP is that all communication becomes encrypted, which prevents attackers from trying to retrieve the password to the Linux servers by sniffing the network traffic.

4.5.6 Database communication

Database communication is performed exclusively using stored procedures. The procedures are called from the data access layer to insert, retrieve, or update information in the database. The procedures are named according to their functionality, e.g. Customer_Insert or Name-server_SelectByAlias, which makes it easy to start using the procedures. Further, as long as the interface stays the same, the implementation of the stored procedure can change at any time.

Stored procedures are generally thought of as being resource-economic, and to further ease up on resources, the choice fell on using simple DataReaders, instead of more advanced features available when reading data from the database.

The communication with the database is secured by setting permissions on the stored procedures. Access to the procedures is not granted by default, but has to be granted explicitly –

a well known security principle [61]. The different roles in the system can execute different sets of procedures; the change manager role is the only one with access to all procedures.

The database communication is handled using the functionality found in the Data Access-block of the Enterprise Library, see 2.9.5.

4.5.7 Transactions

When a Change Manager approves the change requests, there are two major steps that follow.

1. The database is updated with all new information.
2. All the updates are replicated to the name servers.

This introduced several problems, because if something goes wrong during either of these steps – all changes must be rolled back to the state before the approval.

SQL Server has built in functionality for rolling back failed transactions, but to use this functionality, the transactions could not be committed before the name servers had been updated correctly as well.

For instance, if there are three requested changes – a new domain, a change of a customer name, and a removal of another domain – all these transactions must be kept uncommitted until both step 1 and 2 are completed successfully. If something goes wrong, no changes at all should be made.

To solve this problem, a class named StorageContext was used, which has been internally developed at Ninetech. The class acts as a container to store, and keep open, the connections and transactions involved when connecting to the database and the name servers. Certain modifications to the class were needed since the original version of the class was only concerned with SQL Server; name server functionality had to be added.

When the requests are approved, the StorageContext keeps all connections open and alive until everything is complete. At this stage, the Commit-method in the StorageContext is called, which loops through all the internal connections to the database and the name servers and finalizes the transactions.

If an error occurs during, for instance, the removal of the domain on the name servers, a Rollback-method is called in the StorageContext, which performs a rollback on all SQL Server transactions and also performs a rollback on the name server information.

The name server rollback had to be manually implemented. This is solved by first creating temporary files when the updates are made, and renaming those to the correct filenames when the Commit-method is called.

4.6 Summary

This chapter begins with an examination of different solutions to the problem of communicating with the name servers. Several different communication techniques were discussed and reviewed, and the chapter later describes the chosen solution – the web server connects remotely to the name servers and uses SSH and SCP to transmit files and commands.

The chapter also describes the software architecture that Ninetech normally uses, and upon which the administration system was built – the 3-layered application pattern – with the three layers Presentation, Business Logic, and Data Access.

The chapter further contains a system overview. Screenshots and descriptions are used to provide the reader with an understanding of the available functionality. Essentially, the system consists of five different views.

- A customer list which presents all available customers
- A customer view where customer details can be inspected and modified
- A domain list which presents all available domains
- A domain view where domain details can be inspected and modified
- A change request compilation where a change manager can approve/deny the requests

The various administrated resources (customers, domains, and resource records) in the system all have a state – which is used to provide the administrators with information of the current state of the resources, e.g. if a resource is to be approved or if it is requested for removal.

The system has three different roles used to control access rights. Which role the user assumes depends on group membership in the company's Active Directory system.

The system further logs all events in Windows Event Log, keeps a record of all change requests, communicates securely with the name servers, and utilizes transactions to avoid data inconsistencies.

5 Results and evaluation

5.1 Introduction

This chapter will describe the finalized system by comparing the system functionality to the requirements in Appendix A. Further, the developed administration system will be evaluated against the same criteria as the chosen systems in chapter 3.5.

5.2 Results against the requirements

This section will compare the system against the requirements, see Appendix A, and explain whether the system complies with the requirements or not.

The section will focus primarily on the shall-requirements since almost no should-requirements have been implemented. The shall-requirements were, as the name implies, required, while the should-requirements were more of a wish-list. These were to be implemented if there was time available.

5.2.1 Shall-requirements

5.2.1.1 Process

The system fulfills requirement 1 by making sure that no changes can be made without being scrutinized by a Change Manager. The Change Manager retrieves a compilation of all active change requests and can subsequently approve/deny the requests.

5.2.1.2 Platform

The platform requirements (number 2-5) are all satisfied. ASP.NET has been used with C# as programming language to develop the web administration interface. SQL Server serves as backend database and users get access to the system depending on group membership in Active Directory. The system is also only accessible from the company's internal network.

5.2.1.3 System

Requirement 6 is satisfied as no changes are allowed on the name servers. The system does not have any functionality to discover if a user has manually edited the configuration files on the name servers, hence such changes will probably be overwritten by the system in the next update.

To satisfy requirement 7 – a database was created containing, among else, information about customers and domains. The following figure visualizes the database schema.

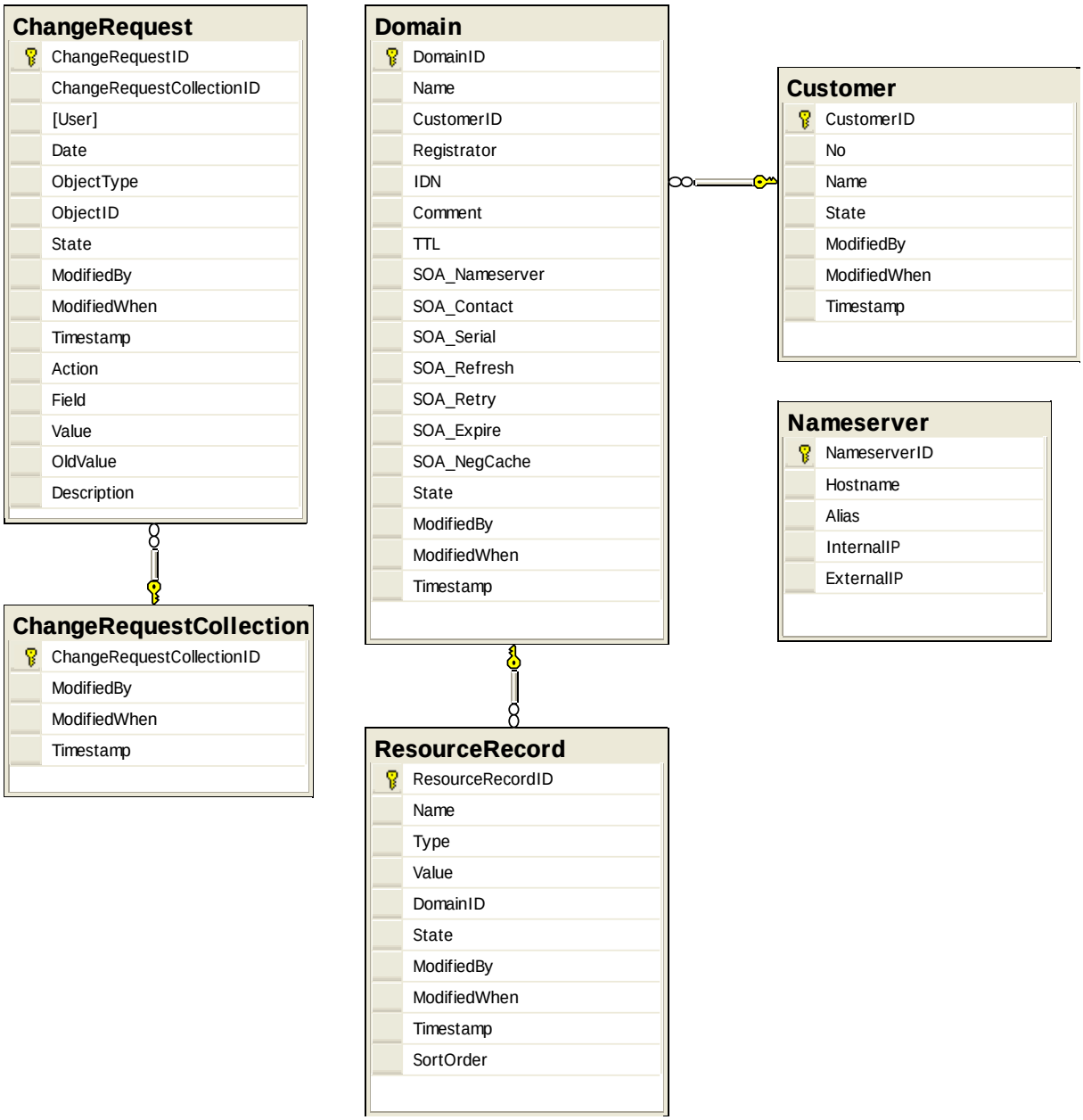


Figure 5-1. Database diagram

For a customer – the properties name and number are stored in the database. For a domain – the name, customer, registrator, IDN, comment, TTL, SOA_Nameserver (primary name-server), SOA_Contact (contact address), SOA_Serial, SOA_Refresh, SOA_Retry, SOA_Expire, and SOA_NegCache are stored in the database. For a resource record – the properties name, type, and value are stored in the database. For a name server – the properties hostname, alias, internal IP and external IP are stored in the database. Hence, all information needed to satisfy requirement 8, 9 and 10 is present.

For a terminology explanation, consult section 2.3 – DNS.

5.2.1.4 Functionality

Requirement 11-14 and 18 is satisfied by the functionality found in the customer list, the customer view, the domain list, and the domain view. The exception is requirement 14.e – change owner. This requirement collided with should-requirement number 38, which the development has been based upon. Therefore, changing owner is not possible in the current version of the administration system.

The customer list provides a list of all customers in the system, and provides functionality for adding and removing customers.

The customer view provides functionality for inspecting and changing the customer name, number and owned domains. It is also possible to add new domains or remove existing ones. The domain list provides a list of all domains in the system, and provides functionality for adding and removing domains.

The domain view provides functionality for inspecting and changing the information for a domain and the associated resource records. Further, it is possible to add new resource records, and delete or change the ordering of existing resource records. For screenshots and usage, see section 4.3.

Regarding internal name servers (requirement 15), only 15.c is fully satisfied. This information can be altered using the web.config-file in ASP.NET, see section 2.7.6. Changing IP and port (15.a and 15.b) is possible without source code modification but has to be done directly in the database.

Requirement 16 is satisfied; the administration system inserts information about all unexpected system events into an event log named .NET – see 4.5.2.1.

Requirement 17 is satisfied, primarily using colouring in the web interface. Resources with requested changes become grey. When customers/domains/resource records have been requested for removal – they become red. New, not yet approved customers/domains/resource records are coloured green, see Figure 4-10 for an example.

Further, status messages are displayed in certain cases, such as when an input validation check fails.

Requirement 18 is satisfied as stated above along with requirement 11-14.

Requirement 19 and 20 are fulfilled by letting the Change Manager have access to a change request compilation – where the Change Manager has to approve or deny the re-

quested changes and carries out the decisions by clicking the button. For screenshots and usage – see section 4.3.

Requirement 21 is satisfied, all relevant name servers are updated after the database updates.

Requirement 22 is satisfied, an e-mail is sent from the e-mail address of the Change Manager that approved/denied the request to the request issuer's e-mail address. The e-mail contains the request contents along with information about if the request was approved or denied. The e-mail addresses are retrieved from the users' Active Directory-accounts.

Requirement 23 is satisfied – a DNS-query is made with nslookup locally on the name servers and if the result is an error message, it is interpreted as a problem with the name server process. This is tested in several distributions of Linux. If the name server process has stopped running, information about the problem is inserted into the system event log; hence requirement 24 is satisfied as well. The Change Manager will also get an error message about the problem in the graphical user interface.

5.2.1.5 Security

Requirement 25 and 26 are both satisfied – there are three different roles in the system and which role the user assumes depends on group membership in Active Directory, see section 4.5.4 for more information.

Requirement 27 is a borderline case but since internal name server information is administered from the web.config, see 2.7.6, certain privileges are necessary to get access to this information. Thus, the requirement is considered partially fulfilled but with room for improvement.

5.2.1.6 Data integrity

Requirement 28 is satisfied by using transactions, see 4.5.7 for further information.

Requirement 29 is satisfied, information about errors is always written to the Windows Event Log.

5.2.1.7 Traceability

Requirement 30 is satisfied, all requests are stored in the database along with a textual description. When the Change Manager has decided upon approval/denial of a request, the request is marked appropriately in permanent storage. For additional information see section 4.5.2.2.

5.2.1.8 Documentation

Requirement 31 is satisfied – user manual, installation manual, and database documentation are available. However, these are internal Ninetech documents and not included in this thesis.

5.2.2 Should-requirements

The only should-requirements that have been satisfied are number 32 and 44. The system uses the standard SCP and SSH-services for communication with the name servers, which also provides encryption.

5.3 System Evaluation

This section will evaluate the developed system against the same criteria as the chosen systems in chapter 3.5 to provide a comparison to other, existing systems.

5.3.1.1 Cost

The system is free in the aspect that it is internally developed. The developers have however received a monthly salary, but no specific sums will be included in the thesis.

5.3.1.2 Documentation

All requested documentation is available: user manual, installation procedures, and database documentation.

5.3.1.3 Extendibility

Since the company owns the source code, the system is extendible at will.

5.3.1.4 Functionality

The system has all required domain and customer handling functionality.

5.3.1.5 License

The system is internally developed and the company owns the source code. The license of the open source .NET-API used for SSH communication does not cause any problems.

5.3.1.6 Platform

All the platform requirements are met since the system was built with those specifically in mind.

5.3.1.7 Process

The system makes use of the ITIL change management processes and conforms to the ITIL standards.

5.3.1.8 Security

Communication between the web server and the name servers is encrypted and access control is based on Active Directory group membership, thus satisfying the requirements.

5.3.1.9 Traceability

All proposed changes are logged in the database together with a textual description. There is information about who made the change, when it was made, what was replaced and what it was replaced with. This satisfies the requirements.

5.4 Problems

The project encountered several problem areas which are briefly described in this section.

5.4.1 Design complexity

A problem was encountered with the original change request design, where four tables were created in the database to contain the change request information.

- One table for information common to all types of change requests.
- One table for information specific to new/delete-change requests.
- One table for information specific to field-change requests.
- One table used to group change requests together.

Unfortunately, having four different tables had the effect that complex table join logic was necessary when creating queries for data insertion and removal. The creation of those queries took more time for the developers than what was available, so an alternative solution which allowed simpler join logic was desired.

The problem was solved by simplifying the design considerably, using two database tables instead of four. One table for storing all information about the change requests and one table for grouping change requests together. However, this requires a large table containing all information and some fields are unnecessary for some change requests. On the other hand, implementation became manageable, which is why this approach was chosen over the more advanced one.

5.4.2 States

One major problem during the system development was the state management of the customers, domains, and resource records.

The first solution to the state management was based on an iterative process where, for instance, the customer that gets marked for removal is responsible for marking the domains the customer owns for removal. The domains are in turn responsible for marking their resource records (and their owner in certain cases) properly. Unfortunately, this approach led to state-setting loops that were very hard to avoid, see Figure 5-2.

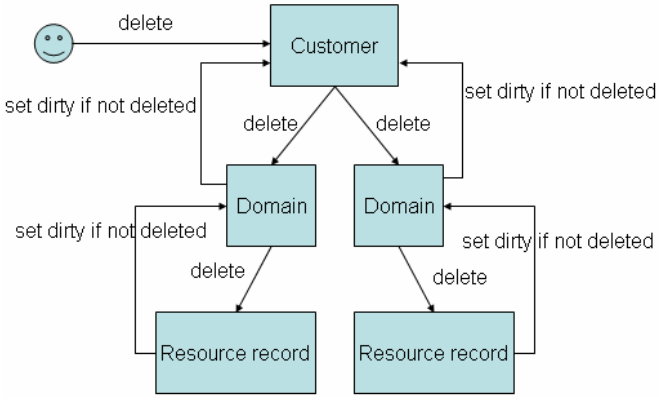


Figure 5-2. States – solution one

This figure describes one of the many troublesome scenarios. Note that this scenario is, in fact, a simplified version of the real case. The purpose is to demonstrate the difficulties involved and not for providing the complete picture.

When a customer was marked for removal, the customer also marked the domains for removal. In this case, each domain that belonged to the customer had to:

- Mark their customer – in case the domain itself had been requested for removal by an administrator. But, this must not happen if it was the customer that had marked the domain for removal in the first place.
- Mark their resource records for removal.

In the next step – each resource record had to mark their domains properly – in case the resource record itself had been requested for removal, but not if the domain was not already marked for deletion.

This was just one of the scenarios and not the most difficult one to solve either. The special cases kept growing and it quickly became too complicated. The state setting process was later simplified; see Section 4.5.1 for a description of the final solution where loops are avoided. This dramatically simplified the implementation.

5.4.3 Transactions

Transactions (described in section 4.5.7) became a problem mostly because the required functionality was incorporated into the system at a very late stage in the development. Without experience, it is sometimes tricky to determine how complicated or large certain tasks are, and the difficulties of including transactions were significantly misestimated.

The transaction feature forced a major refactoring – almost every method that was concerned with the database, including numerous others, had to be rewritten. Since the system was more or less complete, this was not a trivial matter.

Lesson learned – transaction handling ought to be in mind while developing the system as part of other tasks, and not handled as a completely separate task near the end of the development. At least, this was definitely the case for this project

5.5 Summary

The current version of the system is fully functional and has gone through extensive manual testing with successful results. Regarding the requirements, all the shall-requirements in Appendix A have been dealt with to some extent, and most of them are fully satisfied.

The should-requirements are a different matter, only two are fulfilled but there was not time available to implement any more of those. Creating a stable system with all required functionality included was the main goal and this goal has been reached.

When compared to the other evaluated systems in section 3.5, it is not especially perplexing that the system satisfies all criteria (see 5.3) to a further extent. After all, the system was developed with the chosen criteria in mind. The other systems may have more functionality in other aspects but lack several features that Ninetech needs; particularly the ITIL process compatibility and customer handling functionality. Apparently, it is difficult to find existing systems that satisfy requirements that is out of the ordinary.

Finally, the project encountered several problems during the implementation, problems which led to experiences that were not anticipated in advance. The problem areas were design complexity – too much design which led to code complexity, object state management – a state machine which was too complex to manage was developed and had to be redesigned to fulfil the requirements, and transactions which were introduced too late in the project and caused a great deal of refactoring.

6 Conclusions

6.1 General conclusions

There are quite a few applications available intended for DNS administration, but nevertheless – finding a system that had both integrated customer handling and follows the ITIL change management process thinking was not possible.

The solution was to implement a new system from scratch and streamline the system for the specific needs that Ninetech had. This way, Ninetech could choose precisely what kind of functionality the system should contain while also having access to the source code afterwards – which yields the possibility of extending the system if the need arises.

One important lesson learned during the development is that if a problem seems too hard – it is often better to stop banging your head against the wall and proceed with other tasks instead. Normally, there is not time available to spend several days with the same problem, instead – the focus needs to be on adding other functionality, and deal with the problem later if it becomes absolutely necessary. And even in this case, a drastic simplification is often the best solution. A theoretically optimized solution might become confusing and hard to grasp, and in the end, ending up unused anyway. This was the case in some situations during the project.

6.2 Future work

No future work is planned, but from the developers' point of view – a user interface for updating information about internal name servers (see requirement 15.a and 15.b) would be the most useful first addition. This functionality is possible to add without any greater difficulties since it will not affect the rest of the system, and it makes it possible to administer the name server information more easily. At the moment, it has to be done directly in the database.

Any future work depends naturally of the needs of the company and the above is merely the personal opinions of the developers.

Regarding the should-requirements, the easiest requirements to implement would probably be:

- Requirement 39, 40 and 45 – default values

- Requirement 41 – name server administration
- Requirement 42 – validation (which exist today in a limited fashion)
- Requirement 43 and 46 – name server restarts

The rest of the should-requirements probably require a significant amount of work to implement. Most of them require extensive modification of the existing system, which is significantly more problematic than simply adding more functionality, which is more or less sufficient for the requirements in the bullet list above.

References

[1] Albitz, Paul and Liu, Cricket. *DNS and BIND - 5th Edition*. O'Reilly, 2006.

[2] Allen, Robbie. *Active Directory Cookbook*. O'Reilly, 2003.

[3] Allen, Robbie et al. *DNS on Windows Server 2003*. O'Reilly, 2003.

[4] Armstrong, Damon. *Pro ASP.NET 2.0 Website Programming.* Apress, 2005.

[5] Bagui, Sikha Saha and Earp, Richard Walsh. *Learning SQL on SQL Server 2005*, O'Reilly, 2006.

[6] Ben-Gan, Itsik, Sarka, Dejan and Wolter, Roger. *Inside Microsoft SQL Server 2005: T-SQL Programming.* Microsoft Press, 2006.

[7] Braden, Rickard P. *Unlocking Microsoft C# v2.0 Programming Secrets.* Wordware Publishing, 2006.

[8] Brown, Fritz and Onion, Keith. *Essential ASP.NET 2.0.* Addison Wesley Professional, 2006.

[9] Brown, Kyle, et al. *Enterprise Java Programming with IBM WebSphere, 2nd edition*, IBM Press, November 6th 2003.

[10] Burbeck, Steve. *Applications Programming in Smalltalk80(TM): How to use Model-View-Controller (MVC)*, http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html, 1992

[11] Buschmann, Frank, et al. *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns*, John Wiley & Sons, August 8th 1996.

[12] Butler, James and Caudill, Tony. *ASP.NET Database Programming Weekend Crash Course*, Hungry Minds Inc. 2002.

[13] Cameron, Jamie, *Webmin*. http://www.webmin.com/, September 11th, 2006.

[14] Cogswell, Jeff et al. *ASP.NET 2.0 All-In-One Reference Guide for Dummies*. Wiley Publishing Inc, 2006.

[15] Cole, Dr. Erik et al. *Network Security Bible*. Wiley Publishing Inc, 2005.

[16] Costello, A. *RFC3492 - Punycode: A Bootstring encoding of Unicode for Internationalized Domain Names in Applications (IDNA).* http://www.ietf.org/rfc/rfc3492.txt, March, 2003.

[17] Cross, Michael et al. *Planning, Implementing, and Maintaining a Windows Server 2003 Active Directory Infrastructure Study Guide & DVD Training System*. Syngress Publishing Inc, 2003.

[18] Deitel, Harvey and Deitel, Paul. *Visual C# 2005: How to Program, Second Edition*, Prentice Hall, 2005.

[19] Dewson, Robin. *Beginning SQL Server 2005 for Developers – From Novice to Professional.* Apress, 2006.

[20] Emu software Inc, *NetDirector*. http://www.netdirector.org/, September 13th, 2006.

[21] Emu software Inc, *NetDirector Public License version 1.1.* http://www.emusoftware.com/content/view/161/220/, September 11th, 2006.

[22] Faltstrom, P et al. *RFC3490 - Internationalizing Domain Names in Applications (IDNA).* http://www.ietf.org/rfc/rfc3490.txt, March, 2003.

[23] Fenster, Len, *Effective Use of Microsoft Enterprise Library: Building Blocks for Creating Enterprise Applications and Services*. Addison Wesley Professional, 2006.

[24] Foggon, Damien. *Beginning ASP.NET 2.0 Databases – From Novice to Professional.* Apress, 2006.

[25] Fowler, Martin. *Patterns of Enterprise Application Architecture*. Addison-Wesley, 2003.

[26] Free Software Foundation, *GNU General Public License version 2.* http://www.gnu.org/licenses/gpl.html, June, 1991.

[27] Gal, Tamir. *SharpSSH - A Secure Shell (SSH) library for .NET,* http://www.tamirgal.com/home/dev.aspx?Item=SharpSsh, October 13th, 2006.

[28] Gamma, Erich, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley, Reading, Mass., 1995.

[29] Gollmann, Dieter. *Computer Security*, John Wiley & Sons, 1999.

[30] Gulutzan, Peter and Pelser, Trudy. *SQL Performance Tuning*, Addison Wesley, 2002.

[31] Hilyard, Jay and Teilhet, Stephen. *C# Cookbook, 2nd Edition,* O'Reilly, 2006.

[32] Hoffman, Kevin. *Microsoft Visual C# 2005 Unleashed*. Sams Publishing, 2006.

[33] Homer, Alex and Sussman, Dave. ASP.*NET 2.0 Illustrated.* Addison Wesley Professional, 2006.

[34] ISC, Inc. *Internet Systems Consortium*. http://www.isc.org/about/home/, October 24th, 2006.

[35] Johansen, Fleming. S, *The ProBIND project*. http://probind.sourceforge.net/, September 11th, 2006.

[36] King, Robert R. *Mastering Active Directory for Windows Server 2003*. Sybex, 2003.

[37] Kirtland, Mary. *Designing Component-Based Applications*. Microsoft Press, 1998.

[38] Kurose, James F. and Ross, Keith W. *Computer Networking – A Top-Down Approach Featuring the Internet*. Addison-Wesley, 2nd Edition, 2003.

[39] Liberty, Jesse, and MacDonald, Brian. *Learning C# 2005*. O'Reilly, 2006.

[40] Loef, Magnus, *GBIND Admin.* http://85.214.17.244/gadmintools/index.php?option=com_content&task=view&id=14&Itemid=33, September 11th, 2006.

[41] Lowe, Doug. *Networking All-In-One Desk Reference For Dummies*. Wiley Publishing Inc, 2004.

[42] MacDonald, Matthew. *Beginning ASP.NET 2.0 in C# 2005.* Apress, 2006.

[43] MacDonald, Matthew and Szpuszta, Mario. *Pro ASP.NET 2.0 in C# 2005*. Apress, 2005.

[44] Macfarlane, Ivor and Rudd, Colin. *IT service management*. itSMF Ltd, 2003

[45] Madeyski, Lech and M. Stochmiałek. Architecture of Modern Web Applications, *Software Engineering after the year 2004: New Challenges*, J. Górski and A. Wardzi´nski, Eds. Wydawnictwa Naukowo-Techniczne, pp. 373–388, 2004.

[46] Mazzi, Justin, *DNS Control,* http://r00tshell.com/dns-control/, September 12th, 2006.

[47] Men & Mice, *The Men & Mice Suite*. http://www.menandmice.com/solutions/suite, September 11th, 2006.

[48] Meyer, B., *Object-Oriented Software Construction, Second Edition,* Prentice Hall, Englewood Cliffs, New Jersey, 1997.

[49] Microsoft Corporation, *Enterprise Library for .NET framework 2.0*. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag2/html/EntLib2.asp, January, 2006.

[50] Momoi, Katsuhiko, *Internationalized Domain Names (IDN) Support in Netscape 7.1/Mozilla 1.4*. http://devedge-temp.mozilla.org/viewsource/2003/idn/index_en.xml, July, 3rd, 2003.

[51] NinetechGruppen. *Kravspecifikation för DNS-administrationsssytem*. NinetechGruppen internal document.

[52] Nixu Oy., *Nixu Namesurfer Suite*. http://www.nixu.com/products/namesurfer/index_en.html, September 11th, 2006.

[53] Oppermann, Charles. *Microsoft Windows 2000 Active Directory Programming*. Microsoft Press, 2001.

[54] Pattern & Practices, Microsoft Corporation. Enterprise Solution Patterns Using Microsoft .NET, version 2, *MSDN Library*, http://msdn2.microsoft.com/en-us/library/ms998469.aspx, July 14th, 2003.

[55] Perry, Dewayne E., and Alexander L. Wolf. Foundations for the study of software architecture. *ACM SIGSOFT Software Engineering Notes 17*, *4* (Oct. 1992), pp. 40–52.

[56] Paliani, Danilo, *WebBind*, http://www.afn.org/~afn23397/, September 11th, 2006.

[57] Parihar, Mridula et al. *ASP.NET Bible,* Hungry Minds Inc, 2002.

[58] Pultorak, David. *Introduction to ITIL for the IT Executive*. http://wp.bitpipe.com/resource/org_938652273_953/11680_34816_edp.pdf, October, 2006.

[59] Riccardi, Greg. *Principles of Database Systems with Internet and Java Applications*, Addison Wesley, 2001.

[60] Rudd, Colin. *An introductory overview of ITIL*. itSMF Ltd, 2004.

[61] Saltzer, Jerry, and Mike Schroeder. *The Protection of Information in Computer Systems*. Proceedings of the IEEE. Vol. 63, No. 9 (September 1975), pp. 1278-1308. Readable on-line at http://cap-lore.com/CapTheory/ProtInf/

[62] Schildt, Herbert. *C# 2.0: The Complete Reference, Second Edition*. McGraw-Hill Osborne Media, 2006.

[63] Seijsing, Johan. DNSSec, *Nätverk & Kommunikation*. Number 11, 2006.

[64] Stephen, Ryan K et al. *Teach yourself SQL in 21 Days, Second Edition*, Sams, 1997.

[65] Tchekmarev, Aleksey. *Windows .NET Server 2003 Domains & Active Directory*. A-List Publishing, 2003.

[66] Terpstra, John H. and Jelmer R. Vernooij. *Official SAMBA-3 HOWTO and Reference Guide, 2nd edition*, Prentice Hall PTR, August 8th 2005.

[67] Thangarathinam, Thiru. *Using the Enterprise Library Data Access Block for .NET 2.0.* March 16th, 2006.

[68] Wallis, Gary, *mysqlBind,* http://freshmeat.net/projects/mysqlbind/, September 12th, 2006.

[69] Watts, David and Willis, Will. *Windows® Server™ 2003 Active Directory® Infrastructure Exam Cram 2*. Que Publishing, 2004.

A Requirement Specification

A.1 Introduction

This is a translation of the project requirement specification, which originally was written in Swedish. The shall-requirements were, as the name implies, crucial, while the should-requirements were to be implemented if there was time available.

A.2 Shall-requirements

A.2.1 Process

1. Domain handling shall occur according to the ITIL processes.

A.2.2 Platform

2. The system shall consist of a web administration interface running on Windows Server, and SQL Server shall be used as database management system.
3. The system shall only be accessible from the company's internal network.
4. The web administration interface shall be developed using ASP.NET 2.0 with C# as programming language.
5. Access rights shall be based upon Active Directory.

A.2.3 System

6. All changes are going to be made via the system and not directly on the name servers.
7. Information about customers and domains are going to be gathered in a database.
8. Customer name and number for each customer shall be stored in the database.
9. The following shall be registered in the database for each domain:
    a. The customer who owns the domain, the registrator who handles domain registration, and a comment.
    b. Domain name and IDN-address, used for translation of domain names using symbols with no native support in DNS, e.g. å, ä, and ö.

c. Resource records (see the following table) stored in the DNS configuration files.

Table A-1. DNS information

Type	Description	Example
TTL	Caching time period	`$TTL 1d ; Time To Live (1 day)`
SOA (Start Of Authority)	Primary name server	`@ IN SOA dns.example.com.`
	E-mail to contact person	`administrator.example.com.`
	Serial number	`2006082901 ; Serial`
	Refresh interval	`10800 ; Refresh (3 hours)`
	Retry interval	`3600 ; Retry (1 hour)`
	Expiration time for zone data	`604800 ; Expire (1 week)`
	Caching time for negative results	`10800 ; Min (3 hours)`
NS	A name server for the zone	`example.com. IN NS dns.example.com.`
A	Maps a name to an IP-address	`t.example.com. IN A 192.16.0.1`
PTR	Maps an IP-address to a name	`1.0.16.192.in-addr.arpa. IN PTR t.example.com.`
CNAME	Alias	`best.example.com. IN CNAME test.example.com.`
MX	E-mail server	`example.com. IN MX 0 mail.example.com.`

10. Hostname, alias, internal IP-address and external IP-address shall be stored in the database for each internal name server.

A.2.4 Functionality

11. The system shall be able to administer customers (users of Ninetech's services). More specifically, the system shall be able to:

    a. Display customer information
    b. Display the customer's domains, see requirement 12
    c. Remove an existing customer
    d. Add a new customer
    e. Change information for an existing customer

12. The system shall be able to administer domains. More specifically, the system shall be able to:

    a. List domains
    b. Display domain information for a specific domain, see requirement 13
    c. Remove an existing domain
    d. Add a new domain

13. The system must be able to administer all information for each domain, see requirement 9. More specifically, the system shall be able to:

    a. Change domain name
    b. Change registrator
    c. Change comment
    d. Change IDN-address
    e. Change domain owner

14. The system shall be able to administer resource records, defined by requirement 9.c, in each domain. More specifically, the system shall be able to:

    a. Display resource records
    b. Remove an existing resource record
    c. Add a new resource record
    d. Change an existing resource record

15. The system shall have parameterization functionality available for acquiring properties of internal name servers. More specifically, the system shall be able to:

    a. Change IP-address
    b. Change port number if relevant
    c. Change username and password

16. The system shall catch unexpected system events and save information about the events in Windows Event Log.

17. The user shall receive feedback for all events.

Workflow

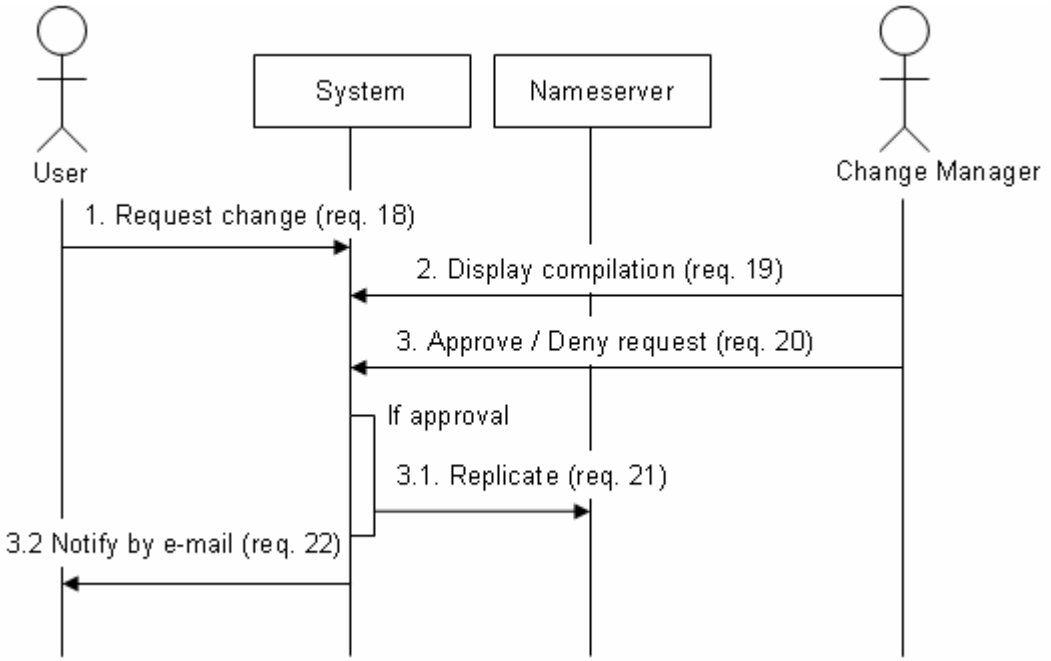
18. The user shall be able to request a change of the information in the database. A change is defined as:

    a. Remove or add a customer (see requirement 11)
    b. Remove or add a domain (see requirement 12)
    c. Change customer name and/or customer number for a customer (see requirement 11)
    d. Change one or more resource records along with information for a domain (see requirement 13 and 14)

19. Before changes are carried out, a compilation of all requested changes along with the name of the issuing users shall be displayed.

20. The user shall be able to choose which requests to approve and which requests to deny.
21. Upon changes of the information in the database – the changes shall be replicated to the name servers.
22. After a request for change has been approved or denied, an e-mail shall be sent to the user with information regarding the decision.

The workflow for requirement 18-22 can be illustrated as follows:



Operational security

23. The system shall check if a name server is running after an update by verifying that the name server process is running on the name server.
24. If a name server fails to start after an update, the system should log this in Windows Event Log.

A.2.5 Security

25. There shall be three roles with different levels of access rights to the system.
    a. Can only view information
    b. Can, in addition to view information, request changes (see requirement 18)
    c. Change Manager can, in addition to a. and b., also view a compilation of active change requests (see requirement 19) and approve/deny the requests (see requirement 20)

26. Which role the user has (see requirement 25) shall be based on group membership in the company's Active Directory-system.
27. Only a Change Manager (see requirement 25) shall be able to administer the information about internal name servers (requirement 15).

A.2.6 Data integrity

28. If an error occurs during an update of the database or a name server, the system shall be rolled back to the database state before the update.
29. In the case of a failed update (requirement 28), the system shall insert information about the failed update in Windows Event Log.

A.2.7 Traceability

30. The system shall log all database changes. Information about time, from- and to values, and the identity of the request issuer shall be stored in the database.

A.2.8 Documentation

31. The system shall be documented using the company's standard template. The following documents shall exist:
    a. User manual
    b. Installation manual
    c. Database documentation

A.3 Should-requirements

Of the following requirements, number 34-37 have the highest priority.

A.3.1 Platform

32. Any necessary software in the Linux environment should be Linux distribution independent and possible to install automatically with GNU GCC and make.

A.3.2 System

33. The system should have parameterization functionality for using language packages in addition to Swedish.

A.3.3 Functionality

34. The system should allow that one or several operations according to requirement 12.c and 13 can be performed on one or several domains simultaneously.
35. The system should allow the user to provide search conditions for deciding which domains to display. The search conditions ought to be based on domain name, customer, IDN-address, registrator, comments, resource record information (requirement 9), and/or name server (requirement 10).
36. The user should be able to mark domains from different searches.
37. The user should be able to receive a compilation of the marked domains for further administration.
38. The system should be able to move domains between customers.
39. The system should automatically suggest default values when new domains are created.
40. The system should allow configuration of default values, see requirement 39.
41. The system should allow that the number of internal name servers can be altered during operation. More specifically, the system should be able to:
    a. Add a new name server
    b. Remove an existing name server
    c. Change an existing name server
42. The system should validate all input to the database according to the following rules:
    a. IP-addresses can only consist of numbers and dots, i.e. *x.x.x.x* ; *x = 0..255*
    b. Domain names should only contain valid characters according to RFC 1035.

Operational security

43. The system should provide the user with a possibility of restarting a name server process after an error has been discovered as in requirement 24.

A.3.4 Security

44. The communication between web server and name servers should be encrypted.
45. Only the Change Manager (requirement 25) should be able to configure default values as in requirement 40.

46. Only the Change Manager (requirement 25) should be able to restart a name server process as in requirement 43.

B System Description

B.1 Design

The class diagram in Figure B.1-1 represents the business object model in the system. For details, consult the following sections.

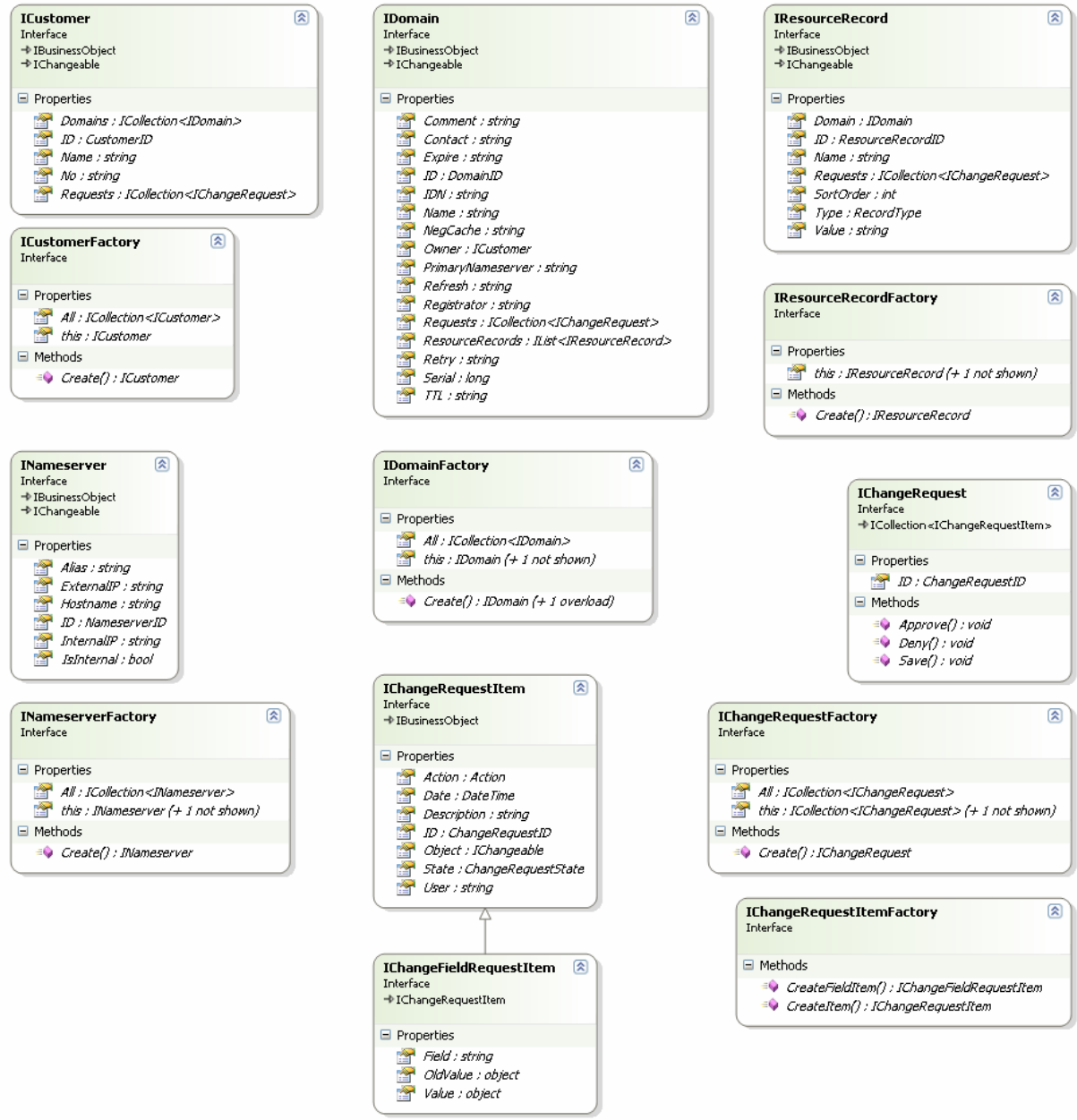


Figure B.1-1. Class diagram – business object model

B.2 Object identifiers

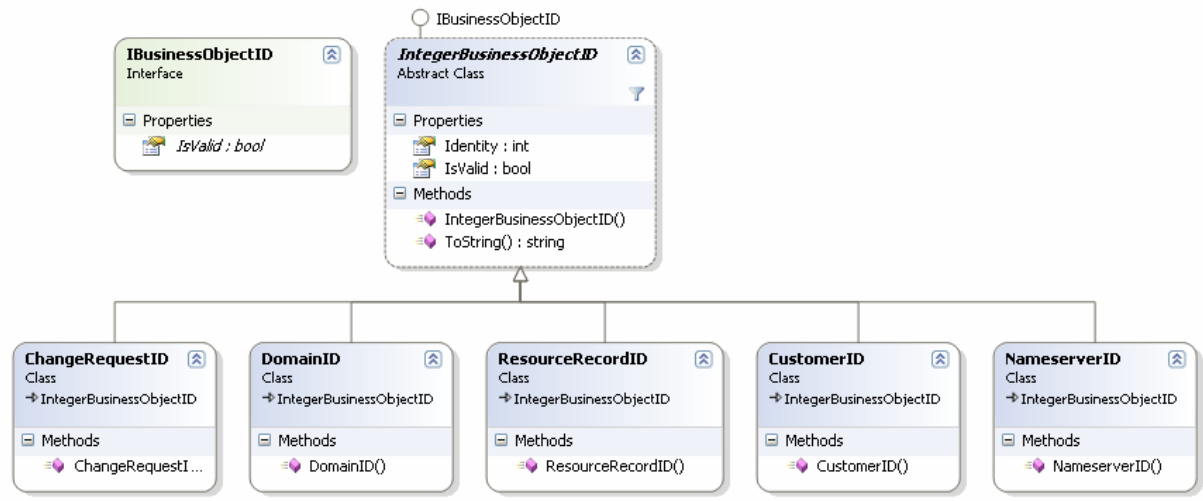


Figure B.2-2. Class diagram for object identifiers

B.2.1 Common.BusinessObject.IBusinessObjectID

Supertype for all objects that represent business object identifiers

Properties

```
bool IsValid { get; }
```

Returns true if the identity number is valid, otherwise false.

B.2.2 Common.Identity.IntegerBusinessObjectID

The basetype of all objects that represent business object identifiers.

Properties

```
int Identity { get; }
```

Returns the identity number.

B.2.3 973Common.Identity.CustomerID

Identifies uniquely a business object of the type Customer.

B.2.4 Common.Identity.DomainID

Identifies uniquely a business object of the type Domain.

B.2.5 Common.Identity.ResourceRecordID

Identifies uniquely a business object of the type ResourceRecord.

B.2.6 Common.Identity.NameserverID

Identifies uniquely a business object of the type Nameserver.

B.2.7 Common.Identity.ChangeRequestID

Identifies uniquely a business object of the type ChangeRequest.

B.3 Common types

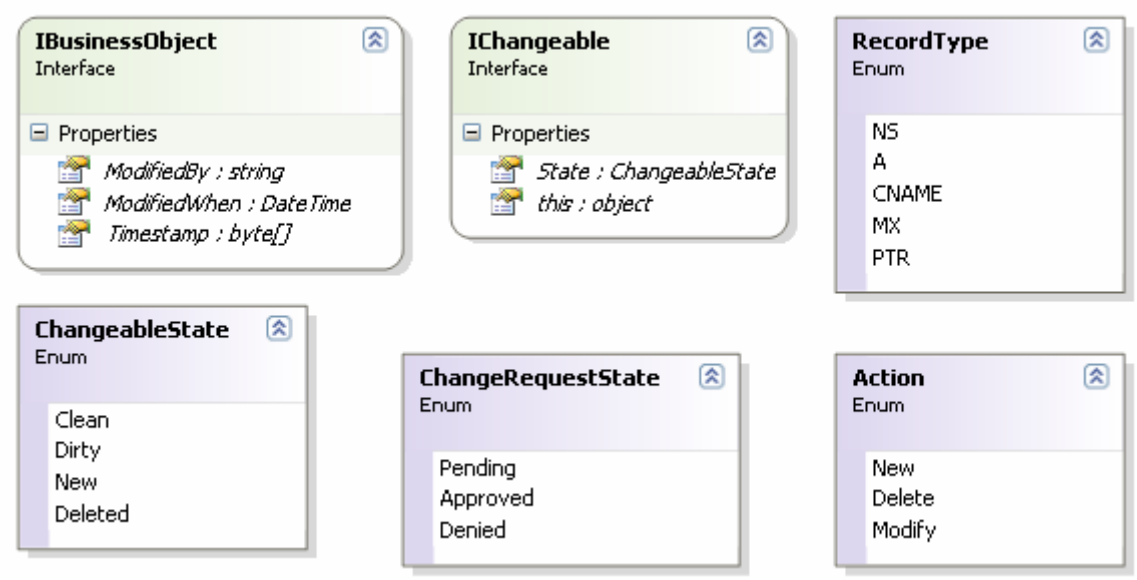


Figure B.3-3. Class diagram for common types

B.3.1 Common.BusinessObject.IBusinessObject

The basetype of all business objects.

Properties

```
string ModifiedBy { get; }
```

Returns the identity of the last user that modified the permanent representation of the object.

```
DateTime ModifiedWhen { get; }
```

Returns the date and time of the most recent change to the permanent representation of the object.

```
byte[] Timestamp { get; }
```

Returns a sequence of bytes indicating the most recent point of time the permanent representation of the object was altered.

B.3.2 Common.BusinessObject.IChangeable

Objects that implement the IChangeable-interface can be stored (permanently) and removed. Implementing objects also receives a status property which indicates read- and write-access rights – Clean, Dirty, New, and Deleted.

Properties

```
object this[string field] { set; get; }
```

Returns or stores the value of the given field in the object. Implementing objects document their own, respective fields.

```
ChangeableState State { get; }
```

Returns the current access right-state.

B.3.3 Common.BusinessObject.ChangeableState

Defines the different object (Customer/Domain/Resource Record) states – Clean, Dirty, New, and Deleted.

B.3.4 Common.BusinessObject.ChangeRequestState

Defines the different possible states for change requests:

- Pending – The change request is waiting for approval
- Approved – The change request has been approved
- Denied – The change request has been denied

B.3.5 Common.BusinessObject.RecordType

The resource record types – NS, A, CNAME, MX, and PTR.

B.3.6 Common.BusinessObject.Action

The change request action types – New, Delete, and Modify.

B.4 Customer

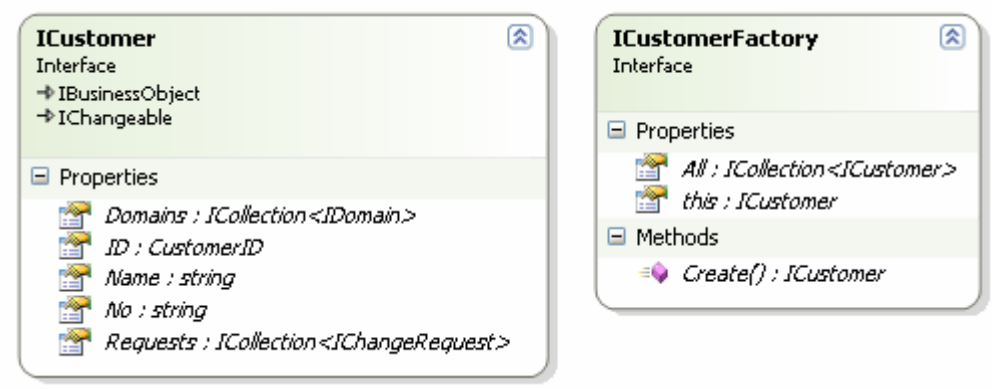


Figure B.4-4. Class diagram for customer interfaces

B.4.1 BusinessObject.ICustomer

The customer has a unique ID for reference to the customer object and the properties customer name and customer number. A customer can have zero or more domains, see B.5. A customer can also have one or more requests for changes regarding the name, number, or domain information, see B.8. However, there can be only one active change request for each property, e.g. only one name change request can be active at a time for a specific customer.

Properties

```
CustomerID ID { get; }
```

Returns the unique reference ID of the customer.

```
string No { set; get; }
```

Returns or stores the customer number.

```
string Name { set; get; }
```

Returns or stores the customer name.

```
ICollection<IDomain> Domains { get; }
```

Returns the collection of domain objects that represent all the domains that belong to the customer.

```
ICollection<IChangeRequest> Requests { get; }
```

Returns a collection of currently active change requests for the customer. This includes any active change request for the domains that belong to the customer.

B.4.2 BusinessObject.ICustomerFactory

A factory that creates customer objects.

Properties

```
ICollection<ICustomer> All { get; }
```

Returns a collection of customer object that represent all customers in the system.

```
ICustomer this[CustomerID id] { get; }
```

Returns a customer object that represents the customer with the specified identifier.

Methods

```
ICustomer Create(string no, string name)
```

Returns a new customer with the specified properties.

B.5 Domain

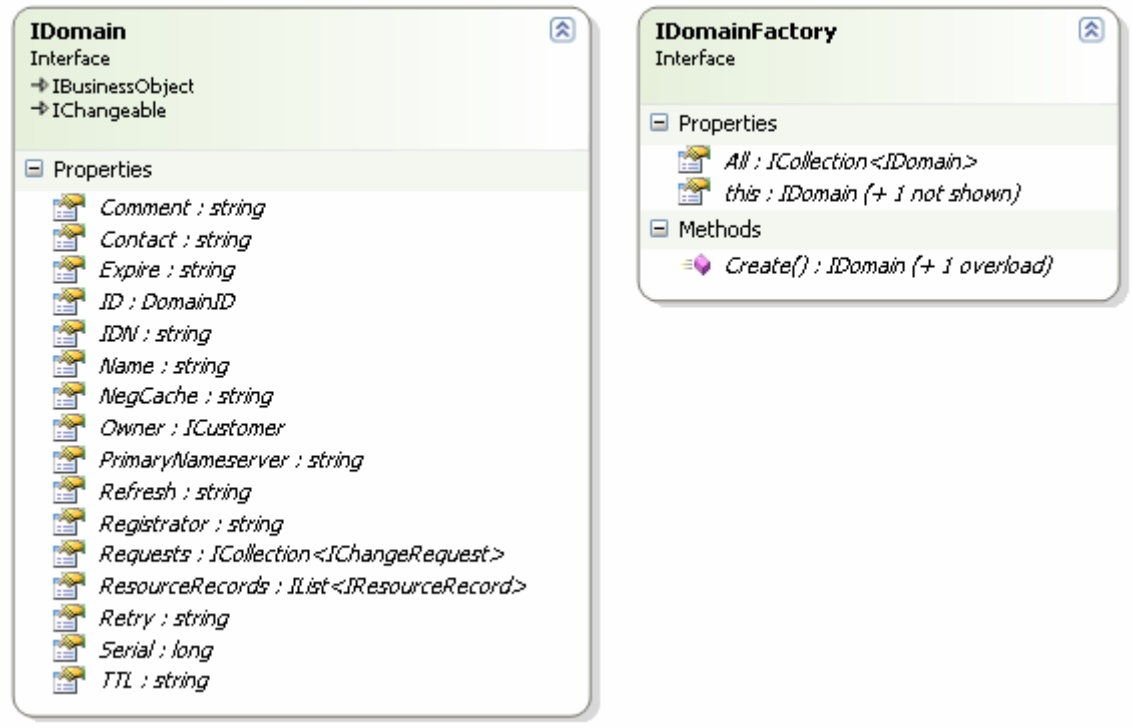


Figure B.5-5. Class diagram for domain interfaces

B.5.1 BusinessObject.IDomain

A domain has a unique ID and several properties that can be stored and retrieved. The properties represent the domain name, registrator, owner, and DNS-configuration settings, see 2.3 for an explanation of the DNS settings. A domain can have zero or more resource records, see B.6, of different types.

Properties

```
DomainID ID { get; }
```

Returns the domain's unique reference ID.

```
string Name { set; get; }
```

Returns or sets the domain name.

```
string Registrator { set; get; }
```

Returns or sets the domain registrator.

```
ICustomer Owner { set; get; }
```

Returns or sets the domain owner.

```
string Comment { set; get; }
```

Returns or sets the comment to the domain.

```
string IDN { set; get; }
```

Returns or sets the domain IDN.

```
string TTL { set; get; }
```

Returns or sets the TTL (Time To Live), the time period that querying name servers cache the domain DNS-information.

```
string PrimaryNameserver { set; get; }
```

Returns or sets the primary name server of the domain.

```
string Contact { set; get; }
```

Returns or sets the e-mail address to the contact person for the domain.

```
long Serial { set; get; }
```
Returns or sets the serial number.

```
string Refresh { set; get; }
```
Returns or sets the refresh time used by slave servers for updating domain information.

```
string Retry { set; get; }
```
Returns or sets the time interval used by slave servers when the primary name server is unreachable.

```
string Expire { set; get; }
```
Returns or sets the time interval the domain information is active after failed updates.

```
string NegCache { set; get; }
```
Returns or sets the time interval that querying nameservers cache a negative query-result.

```
ICollection<IChangeRequest> Requests { get; }
```
Returns the collection of currently active change requests for the domain.

```
IList<IResourceRecord> ResourceRecords { get; }
```
Returns a list that represents all the domain's resource records.

B.5.2 BusinessObject.IDomainFactory

A factory that creates domain objects.

Properties

```
ICollection<IDomain> All { get; }
```
Returns a collection of domain objects that represents all domains in the system.

```
IDomain this[DomainID id] { get; }
```
Returns a domain object representing the domain specified by the given ID.

```
ICollection<IDomain> this[CustomerID id] { get; }
```
Returns a collection of domain objects that represents all domains for the customer with the given ID.

Methods

```
IDomain Create()
```

Returns a new domain object without any information.

```
IDomain Create(string name, ICustomer owner, string TTL,
               string primaryNameserver, string contact,
               long serial, string refresh, string retry,
               string expire, string negCache)
```

Returns a domain with the specified properties.

B.6 Resource record

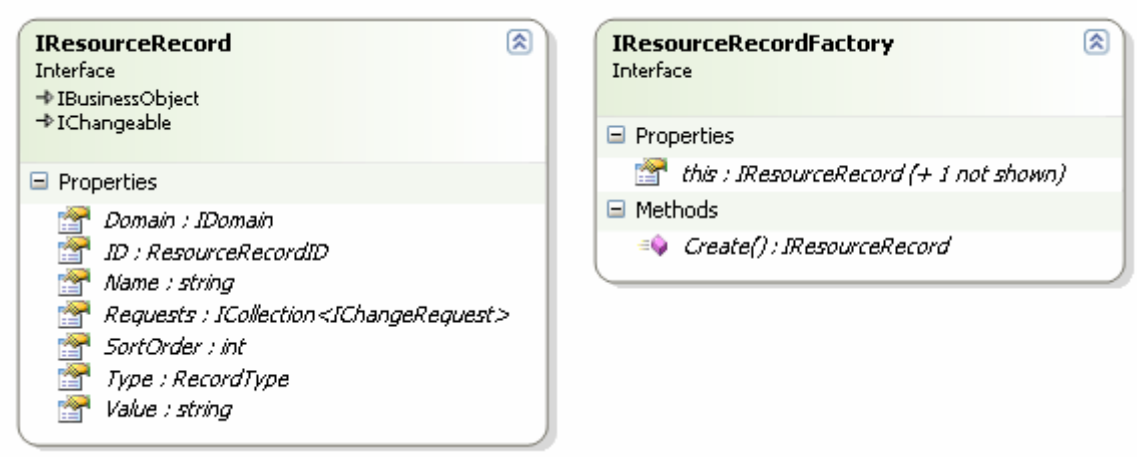


Figure B.6-6. Class diagram for resource record interfaces

B.6.1 BusinessObject.IResourceRecord

A resource record has a unique ID and the properties name, type, and value. Resource records cannot have the Append-access right, only Write or Read-only.

Properties

```
ResourceRecordID ID { get; }
```

Returns the unique reference ID of the resource record.

```
string Name { set; get; }
```

Returns or sets the resource record name-property.

```
RecordType Type { set; get; }
```
Returns or sets the resource record type, i.e. NS, A, MX, CNAME, or PTR.

```
string Value { set; get; }
```
Returns or sets the value of the resource record.

```
IDomain Domain { get; }
```
Returns a domain object representing the domain that the resource record is associated to.

```
int SortOrder { set; get;}
```
Returns or sets the sort order of the resource record relative to the other resource records for the `Domain`.

```
ICollection<IChangeRequest> Requests { get; }
```
Returns a collection that represents all currently active change requests for the resource record.

B.6.2 BusinessObject.IResourceRecordFactory

A factory that creates resource records.

Properties

```
IResourceRecord this[ResourceRecordID id] { get; }
```
Returns a resource record object that represents the resource record with the specified ID.

```
IList<IResourceRecord> this[DomainID id] { get; }
```
Returns a list of resource records that represents all the resource records for the domain with the specified ID.

Methods

```
IResourceRecord Create(IDomain domain, string name,
                       RecordType type, string value)
```
Returns a new resource record with the specified properties.

B.7 Nameserver

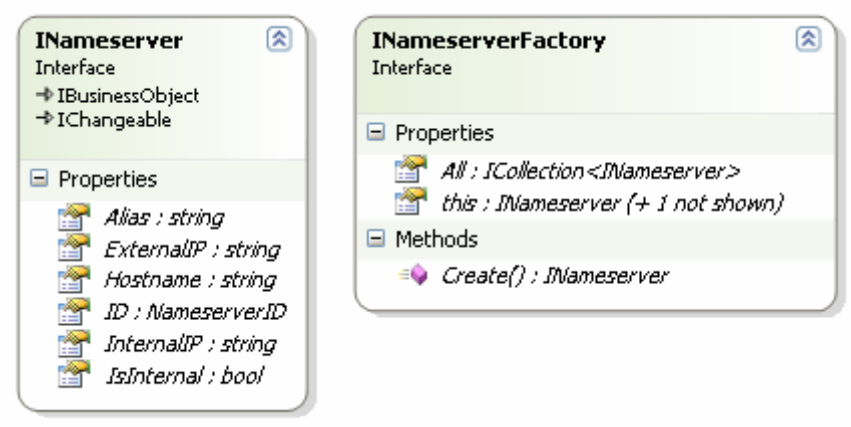


Figure B.7-7. Class diagram for nameserver interfaces

B.7.1 BusinessObject.INameserver

A nameserver has a unique ID and the properties hostname, alias, internal-, and external IP-address.

Properties

```
NameserverID ID { get; }
```

Returns the unique reference ID of the name server.

```
string Hostname { set; get; }
```

Returns or sets the name server's hostname.

```
string Alias { set; get; }
```

Returns or sets the name server's alias (an internal nickname).

```
string InternalIP { set; get; }
```

Returns or sets the name server's internal IP-address.

```
string ExternalIP { set; get; }
```

Returns or sets the name server's external IP-address.

```
bool IsInternal { get; }
```

Returns false if the name server is an internal name server, otherwise false.

B.7.2 BusinessObject.INameserverFactory

A factory that creates name server objects.

Properties

```
ICollection<INameserver> All { get; }
```

Returns a collection of name server objects that represents all name servers in the system.

```
INameserver this[NameserverID id] { get; }
```

Returns a name server object that represents the name server with the specified ID.

```
INameserver this[string alias] { get; }
```

Returns a name server object that represents the name server with the specified alias.

Methods

```
INameserver Create(string hostname, string alias,
                   string internalIP, string externalIP)
```

Returns a new name server with the specified properties.

B.8 Change Request

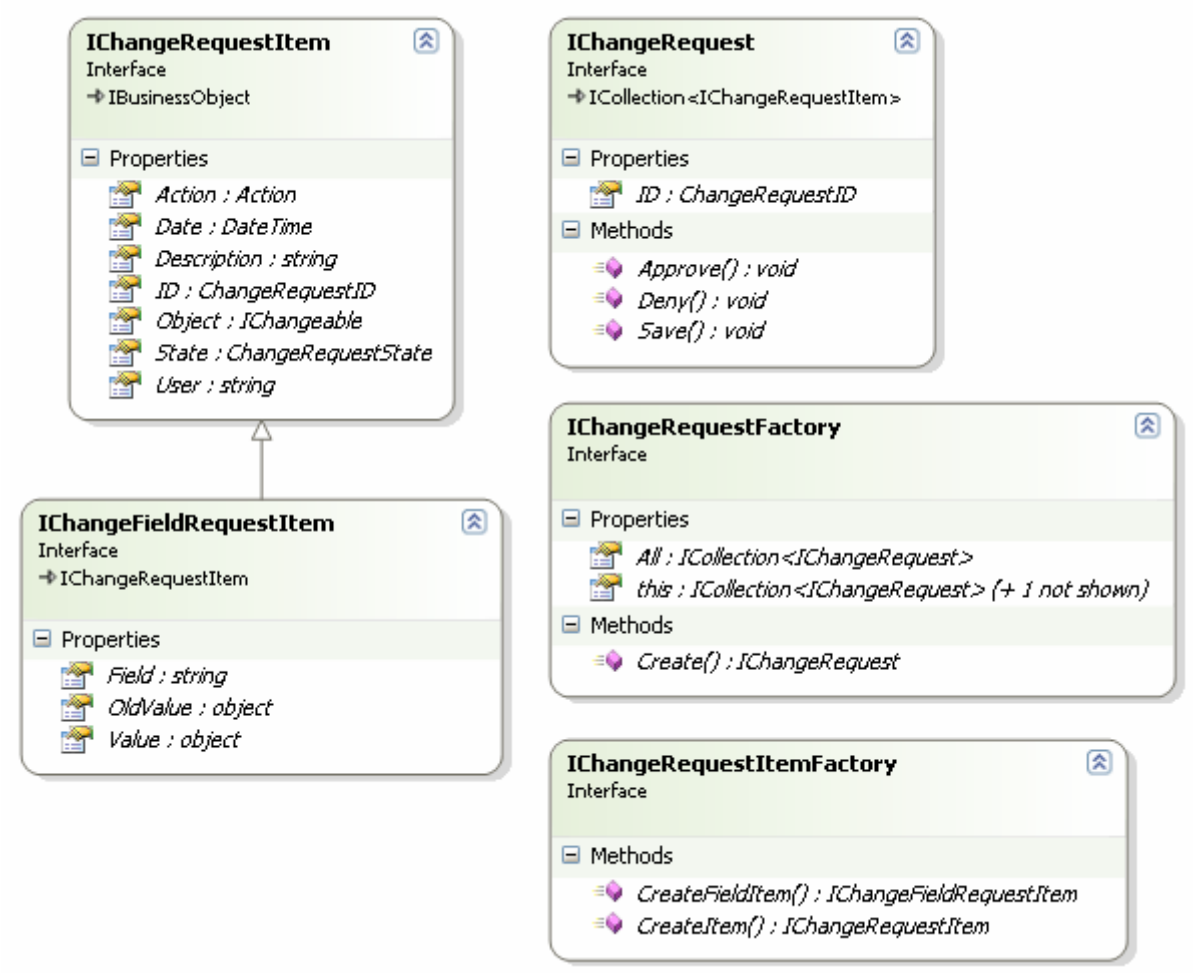


Figure B.8-8. Class diagram for change request interfaces

B.8.1 BusinessObject.IChangeRequest

Each change request is identified by a unique ID and represents a collection of change request items. A change request collection can only be approved or denied as a whole.

Properties

```
ChangeRequestID ID { get; }
```

Returns the unique reference ID of the change request.

Methods

```
void Approve()
```

Approve the change request – activate the changes.

`void Deny()`

Deny the change request.

`void Save()`

Stores the change request permanently.

`void override Add(IChangeRequestItem item)`

Appends a change request item to the collection.

B.8.2 BusinessObject.IChangeRequestItem

Each change request item is identified by a unique ID. The change request item also contains the name of the issuing user and creation date/time. A request can be a change of a single field in a customer, domain, or resource record. It can also be a removal or creation of a customer, domain, or resource record.

Properties

`string User { get; }`

Returns the username of the issuing user.

`DateTime Date { get; }`

Returns the date and time of when the change request was created.

`Action Action { get; }`

Returns the change request's type - new, delete, or modify.

`IChangeable Object { get; }`

Returns the object that should be changed.

`ChangeRequestState State { get; }`

Returns the status of the change request – pending, approved, or denied.

`string Description { get; }`

Returns a descriptive text about the change request item.

B.8.3 BusinessObject.IChangeRequestFactory

A factory that creates change requests.

Properties

```
ICollection<IChangeRequest> All { get; }
```

Returns a collection of change request objects representing all currently pending change requests.

```
ICollection<IChangeRequest> this[IChangeable obj] { get; }
```

Returns a collection of change request objects representing all currently pending change requests for the specified object.

Methods

```
IChangeRequest Create()
```

Returns a new change request.

B.8.4 BusinessObject.IChangeFieldRequestItem

A request for change for a single field in a customer, domain or resource record.

Properties

```
string Field { get; }
```

Returns the name of the field that the change affects.

```
string Value { get; }
```

Returns the proposed new value for the field.

```
string OldValue { get; }
```

Returns the earlier value of the field if the request has been approved, otherwise null is returned.

B.8.5 BusinessObject.IChangeRequestItemFactory

A factory that creates change request items.

Methods

```
IChangeRequestItem CreateItem(IChangeable obj, Action action,
                              string user, DateTime date);
```

Returns a new change request item with the specified properties.

```
IChangeFieldRequestItem CreateFieldItem(IChangeable obj,
                    string user, DateTime date,
                    string field, object value);
```

Returns a new change request for the specified field.