KOA Evaluation, Demonstration Installation and Implementation

Final Year Project Final Report

Alan E. Morkan

A thesis submitted in part fulfilment of the degree of BA (Hons.) in Computer Science with the supervision of Dr. Joseph Kiniry and moderated by Dr. Ahmed Patel.

Department of Computer Science

University College Dublin

10 March 2005

Abstract

This document serves as an account of the execution of the project entitled *KOA evaluation and demonstration installation and implementation*. As an introduction, the inception of the KOA system is described together with an overview of the domain of internet and electronic voting. Subsequently, an analysis of the functional overview and components of the KOA system are presented. An outline is then sketched of the process undertaken in carrying out a demonstration installation of the system on an open-source platform. Penultimately, there is a discussion of the implementation of the missing functionality and problems encountered. Finally, the conclusions drawn from the project are considered alongside an enumeration of the necessary future work to be performed in order to develop the KOA system into a secure and trustworthy means of remote internet voting.

Acknowledgements

Special thanks to those who have helped in various ways throughout the execution of the project. To Martijn Warnier for translating the design specification documents from Dutch into English; To Wim Kegel of LogicaCMG for information on how the KOA system was originally developed and released under the GPL licence; To Fergus Boydell for his infinite knowledge of the bash shell; To Edward Abrahamson for proof-reading and in depth knowledge of English; and finally and most especially to Dr. Joseph Kiniry for his patience and good-humouredness in dealing with every query, and as an inspiration to learn at all times.

Table of Contents

1 Introduction 5

2 Background Research 6

2.1 Java 2 Platform, Enterprise Edition 7

2.1.1 Tiers 7

2.1.2 Components 8

2.1.3 Deployment 9

2.1.4 Servlets 9

2.1.5 JavaServer Pages (JSPs) 9

2.1.6 Java Message Service (JMS) 10

2.1.7 Java Database Connectivity (JDBC) 10

2.1.8 Java Authentication and Authorization Service (JAAS) 10

2.1.9 Java Naming and Directory Interface (JNDI) 11

2.2 Application Servers 11

2.3 Database Servers 12

2.4 Apache Ant 12

2.5 Concurrent Versions System (CVS) 12

2.6 Business Object Notation (BON) 13

2.7 Java Modeling Language (JML) 14

2.8 ESC/Java2 14

2.9 The Party List System 15

2.10 Encryption 15

3 System Analysis 16

3.1 Design Overview 16

3.1.1 KR (Voter Register) 18

3.1.2 VSL (Virtual Polling Station) 19

3.1.3 ESB (Electronic Ballot Box) 20
3.1.4 TSM (Telephone Voting Machine) 21
3.1.5 WSM (Web Voting Machine) 22
3.2 Released System 23
3.2.1 Enterprise Java Beans 23
3.2.2 Utility Classes 26
4 Installation 26
4.1 Compilation 26
4.2 Packaging and Deployment 27
4.3 Database Setup and Connectivity 27
5 Implementation 27
5.1 User Interfaces 28
5.2 Properties 28
5.3 Classes 28
6 Problems Encountered 29
7 Appraisal 30
8 Conclusions 30
9 Future Work 30
10 References 31

1 Introduction

The *Kiezen op Afstand* (KOA)[1] system is the first open-source Internet voting system to be developed in the world[1]. The Dutch Government outsourced the project to the tender winner LogicaCMG, a Dutch consulting firm. This company designed, developed, tested and deployed the KOA system. The Dutch Government released the source code for the KOA system under the GNU General Public Licence (GPL) [2] in July 2004 after the system was used on a trial basis in the European Parliamentary Elections.

The trial was restricted to roughly 16,000 Dutch expatriates who were required to register beforehand. Expatriates could vote either via the Internet or by telephone. The telephone votes were fed into the KOA tally system. 5,351 used one or other system[3]. When registering, voters chose their own personal access code. Subsequently, a customised information packet was mailed to the voter. This package contained general information about the election itself (date, time, etc.), and also voter-customised details that were known to only that voter. These details included information for voter authentication, including an identification code and the previously chosen access code. Also included is a list of all candidates. Each voter receives 1 of the 1,000 codes that are allocated to each candidate[2].To vote, a registered voter visits the secure KOA website. The voter is instructed how to vote and enters the voter code and access code to verify their identity. The voter then enters the 9-digit candidate code of the candidate of their choice. The system shows the voter the political party, the place of residence and the actual name of the chosen candidate. The voter must then confirm the accuracy of this vote and is subsequently provided with a transaction code receipt. This code can later be used to check that the voter's choices were included properly in the final tally for the election.

The advantages of such a system include:

- User friendliness as voters would be more acquainted with PCs than with Electronic Voting Machines (EVMs).
- The voter can decide the time and place in which to vote. This is sometimes called autonomous voting.
- There exists no limitation to the number of candidates or parties per election. In contrast, EVMs are standalone units with a fixed memory. Thus they can only store a certain number of candidates and votes, which are uploaded after the close of polls.

[1] "Kiezen op Afstand" can be literally translated as "Remote Voting".

[2] For security reasons each candidate has 1,000 different codes assigned to him/her. The candidate lists that are sent to the voter contain one of these codes. This makes it possible to generate a large number of different candidate lists. This security measure is used for voting via telephone. If a connection is monitored it is not easy to match the candidate code to any specific candidate without a complete list of all candidate codes. The complete list can not be generated from any given candidate list[4].

Unfortunately, however, a number of pieces of functionality, constituting roughly 10% of the deployed KOA system, are proprietary and owned by LogicaCMG. In addition, the released KOA system contained no high-level design documentation and very little information on how to build the system. Thus, the code released under the GPL licence is incomplete. This means that it is only possible to inspect the (partial) source, not to compile and run it. Therefore, although the KOA system in its current form guarantees the desired transparency of a voting system, it is still impossible to test it to verify that the system is correct.

Therefore, the purpose of this project is to complete the KOA system. This consists of identifying, reverse engineering, and writing from scratch (including English documentation and formal specifications) the missing parts. The completed system will then be re-released as the world's first open-source Internet voting system.

Such an open-source system would be of particular relevance and importance in Ireland. An attempt at moving towards universal use of electronic voting in nationwide elections was abruptly shelved in June 2004. The EVMs purchased by the Irish Government are a combination of a generic hardware architecture manufactured by a Dutch company, Nedap[5], and software developed specifically for the Irish politico-legislative context by a subsidiary of Nedap, called Powervote[6]. The Commission on Electronic Voting[7], set up by the Irish Government in response to concerns over the proposals in relation to electronic voting, had very limited time to report on the suitability of the Nedap/Powervote system. Consequently, only minimal analysis and testing of the Powervote software was carried out[8]. Nevertheless, noticeable bugs were detected and thus the decision was taken not to use the EVMs due to fears over the security and integrity of such a proprietary system without an auditable paper trail. Currently, two reviews of the Nedap-Powervote EVMs are taking place. The first involves an examination of the possibility and difficulty of altering these machines to comply with new E.U. and Irish law. The other is envisaged as a more comprehensive review of the existing system by the Commission on Electronic Voting[9]. Considering that it is unlikely that sufficient experts will be given access to the Powervote source code and also the aggressive time constraints dictated by the review process, it is doubtful whether such a review will yield an acceptable technical certification. Consequently, the availability of a secure, open-source, auditable and testable system, such as a completed version of the KOA system, could direct the required pressure to achieve a policy change towards a transparent and trustworthy means of voting.

This report is divided into three main sections. The first part discusses the background research that was necessary in order to grasp a full understanding of the scope of the problem. This includes a description of the basic architecture of distributed applications, an explanation of the value of design by contract and formal specifications and also the tools and technologies required to arrive at a potential solution. The second part focuses on the actual work completed in analysing the system, installing a demonstration system and implementing the missing functionality. The final section discusses the problems encountered, an appraisal of the effectiveness of the approach employed, conclusions drawn and the necessary future work in developing the KOA system.

2 Background Research

This section outlines the major areas in which background research was necessary. It includes areas of general theory necessary in order to gain a thorough understanding of the project, together with the tools and technologies required to reach a potential solution.

2.1 Java 2 Platform, Enterprise Edition

The Java 2 Platform, Enterprise Edition (J2EE) is a framework for creating client-server applications[10]. These are applications whose source code is not necessarily centrally stored, which are accessed by users in different locations and who communicate across a network.

2.1.1 Tiers

The architecture of a J2EE application is organised according to a number of tiers. Each tier consists of a number of components. The tiers of a typical J2EE system are:

- The client-tier runs on the client's machine. The client-tier consists of components that manage web pages, applets or user applications.
- The web-tier runs on the J2EE server. The web-tier consists of components that deal with requests or responses to the client-tier.
- The business-tier components run on the J2EE server. The business-tier contains Enterprise Java Bean (EJB) components, which coordinates data flow between the client-tier and the Enterprise Information System Tier.
- The enterprise information system-tier (EIS) runs on the EIS server. Usually, the EIS-tier contains a database.

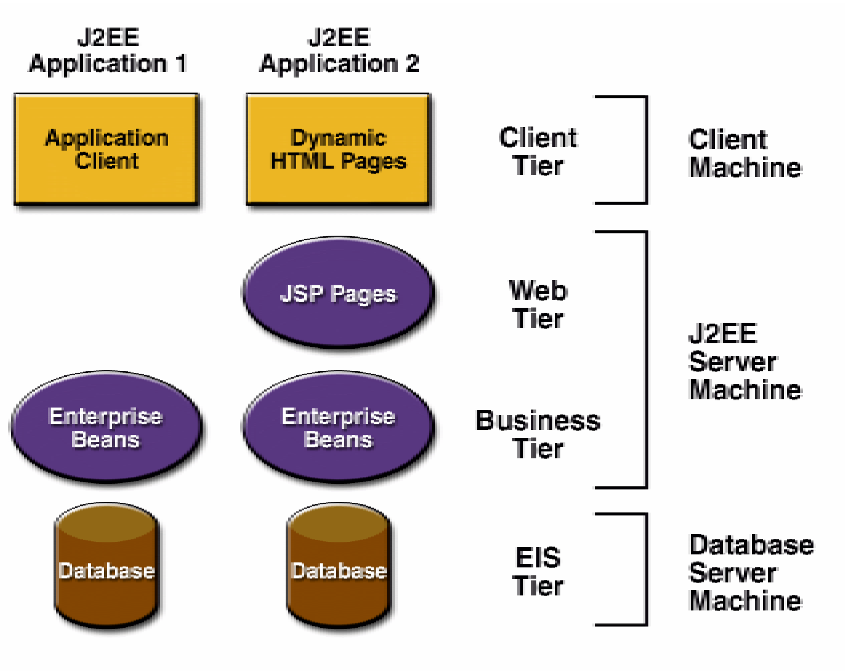

Figure 1. Diagram of the multi-tier architecture of a J2EE application

2.1.2 Components

A J2EE component is a self-contained functional software unit. It is written in Java and compiled in the same way as any other Java program. The difference between J2EE components and "standard" Java classes is that J2EE components are assembled into a J2EE application, verified to be well-formed and deployed (installed) to a server that runs and manages them.

Components of the client-tier are dynamic HTML pages, applets or applications.

Components of the web-tier can be either servlets or Java Server Pages (JSPs). (See section 2.1.4 for a description of servlets and section 2.1.5 for an explanation of JSPs)

Components of the business-tier are Enterprise Java Beans (EJBs). An enterprise bean is a body of code with fields and methods that implements modules of business logic. It is a building block that can be used alone or with other enterprise beans to execute business logic.

There are two main kinds of enterprise beans: session beans and entity beans. A session bean represents a transient conversation with a client. When the client finishes executing, the session bean and its data are gone. In contrast, an entity bean represents persistent data maintained in a database. An entity bean can manage its own persistence or can delegate this function to its container (called Container Managed Persistence (CMP)). If the client terminates, or if the server shuts down, the entity bean, its primary key, and any remote references survive the crash.

Containers are the interface between a component and the low-level platform-specific functionality that supports the component. Before a Web, enterprise bean, or application client component can be executed, it must be assembled (packaged) into a J2EE application and deployed to its container.

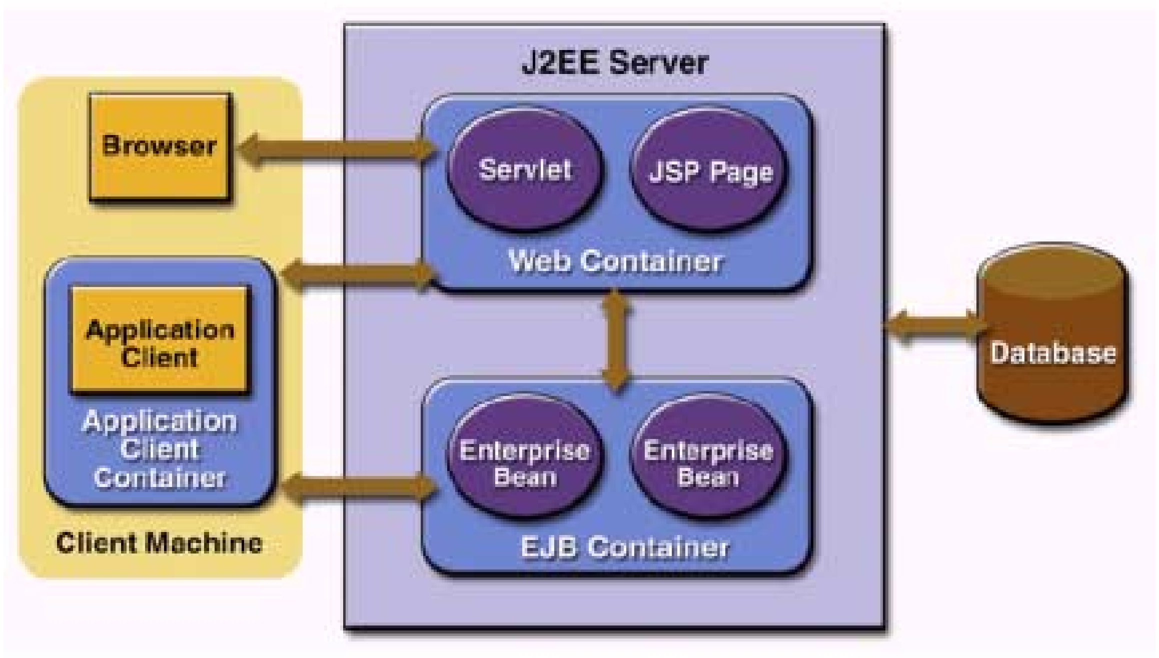

Figure 2. J2EE Server and Containers

2.1.3 Deployment

J2EE components are packaged separately and bundled into a J2EE application for deployment. Each component, its related files, such as GIF and HTML files or server-side utility classes, and a deployment descriptor are assembled into a module and added to the J2EE application. A J2EE application is composed of one or more enterprise beans, Web, or application client component modules.

A J2EE application and each of its modules has its own deployment descriptor. A deployment descriptor is an XML document that describes a component's deployment settings. Because deployment descriptor information is declarative, it can be changed without modifying the bean source code. At runtime, the J2EE server reads the deployment descriptor and acts upon the component accordingly.

A J2EE application with all of its modules is delivered in an Enterprise Archive (EAR) file. An EAR file is a standard Java Archive (JAR) file with an ".ear" extension on its filename. An EAR consists of one or more JAR and Web Archive (WAR) files.

2.1.4 Servlets

Servlets are Java classes that extend the capabilities of servers that host applications. Servlets were developed as a response to the shortcomings of the Common Gateway Interface (CGI)[53] such as platform dependence and lack of scalability. Servlets dynamically process requests and construct responses. Requests are processed using a service method. The most common service methods are *doGet* and *doPost* which respond to the ‘get’ and ‘post’ actions of the HTTP protocol[52]. The general pattern for a service method is to extract information from the request, access external resources, and then populate the response based on that information. The life cycle of a servlet is controlled by the container in which the servlet has been deployed. When a request is mapped to a servlet, the container performs the following steps:

1. If an instance of the servlet does not exist, the web container
    - loads the servlet class,
    - creates an instance of the servlet class, and
    - initialises the servlet instance by calling the *init* method.
2. The servlet then invokes the service method (usually *doGet* or *doPost*), passing the request and response objects.
3. If the container needs to remove the servlet, the container finalises the servlet by calling the *destroy* method.

2.1.5 JavaServer Pages (JSPs)

JavaServer Pages (JSPs) facilitate the insertion of snippets of servlet code directly into a text-based document (such as HTML, SVG, WML, and XML). JSPs provide all of the dynamic capabilities of servlets but many developers also feel that they provide a more natural approach to creating static content. A JSP page has two types of text: static data and JSP elements. The static data can be in any text format while the JSP elements determine how the page constructs dynamic content.

A JSP page processes requests as a servlet. Thus, the life cycle and many of the capabilities of JSP pages, in particular the dynamic aspects, are determined by Java Servlet technology (see section 2.1.4). When a request is mapped to a JSP page, the web container first checks whether the JSP page's servlet is older than the JSP page. If the servlet is older, the web container translates the JSP page into a servlet class and compiles the class.

2.1.6 Java Message Service (JMS)

Messaging is a method of communication between software components or applications. A messaging system is a point-to-point facility. Each client connects to a messaging agent that provides facilities for creating, sending, receiving, and reading messages. Messaging enables distributed communication that is *loosely coupled*. A component sends a message to a destination, and the recipient can retrieve the message from the destination. However, the sender and the receiver do not have to be available at the same time in order to communicate. In fact, the sender does not need to know anything about the receiver and vice-versa. The sender and the receiver need to know only which message format and which destination to use. In this respect, messaging differs from tightly coupled technologies, such as Remote Method Invocation (RMI), which require an application to know a remote application's methods.

The Java Message Service (JMS) API is a messaging standard that allows J2EE application components to create, send, receive, and read messages. It enables distributed communication that is loosely coupled, reliable, and asynchronous.

2.1.7 Java Database Connectivity (JDBC)

The JDBC API allows SQL commands to be invoked from Java programming language methods. The JDBC API is used in an enterprise bean when the default container-managed persistence is overridden or when a session bean accesses the database. With container-managed persistence, database access operations are handled by the container, and the enterprise bean implementation contains no JDBC code or SQL commands. The JDBC API can also be used from a servlet or a JSP page to access the database directly without going through an enterprise bean. The JDBC API has two parts: an application-level interface used by the application components to access a database, and a service provider interface to attach a JDBC driver to the J2EE platform.

2.1.8 Java Authentication and Authorization Service (JAAS)

The Java Authentication and Authorization Service (JAAS) provides a method for a J2EE application to authenticate and authorise a specific user or group of users to run it. The J2EE application programming model insulates developers from mechanism-specific implementation details of application security. The J2EE platform provides this insulation in a manner that enhances the portability of applications, allowing them to be deployed in diverse security environments.

Security for components is provided by their containers. A container provides two kinds of security: declarative and programmatic security. *Declarative security* expresses an application's security structure, including security roles, access control, and authentication requirements, in a form external to the application (in a deployment descriptor). *Programmatic security* is embedded in an application and is used to make security decisions. Programmatic security is useful when declarative security alone is not sufficient to express the security model of an application.

JAAS applies *users* and *roles* property files to regulate access to protected resources. A *user* is an individual (or application program) identity that has been defined in the Application Server. A *role* is an abstract name for the permission to access a particular set of resources in an application.

A *role* can be compared to a key that can open a lock. Many people might have a copy of the key. The lock doesn't care who the user is, only that it is the right key.

2.1.9 Java Naming and Directory Interface (JNDI)

The Java Naming and Directory Interface (JNDI) provides naming and directory functionality. It provides applications with methods for performing standard directory operations, such as associating attributes with objects and searching for objects using their attributes. Using JNDI, a J2EE application can store and retrieve any type of named Java object. J2EE naming services provide application clients, enterprise beans, and web components with access to a JNDI naming environment.

A *naming environment* allows a component to be customized without the need to access or change the component's source code. A container implements the component's environment and provides it to the component as a JNDI *naming context*. A J2EE component locates its environment naming context using JNDI interfaces. A component creates a *javax.naming.InitialContext* object and looks up the environment naming context in *InitialContext* under the name *java:comp/env*. A component's naming environment is stored directly in the environment naming context or in any of its direct or indirect subcontexts. A J2EE component can access named system-provided and user-defined objects.

The J2EE platform allows a component to name user-defined objects, such as enterprise beans, environment entries, JDBC datasource objects, and message connections. An object should be named within a subcontext of the naming environment according to the type of the object. For example, enterprise beans are named within the subcontext *java:comp/env/ejb*, and JDBC DataSource references in the subcontext *java:comp/env/jdbc*. Because JNDI is independent of any specific implementation, applications can use JNDI to access multiple naming and directory services, including existing naming and directory services such as LDAP, NDS, DNS, and NIS. This allows J2EE applications to coexist with legacy applications and systems.

2.2 Application Servers

The original KOA system was developed using WebSphere, which is an IBM product[11]. 30 of the 59 missing classes from the KOA system were automatically generated by the IBM WebSphere IDE. However, given the goal of making the KOA system entirely open-source, using a proprietary product, such as IBM WebSphere, is not an option.

The JBoss Application Server version 4.0 is the first open-source J2EE 1.4-certified server[12]. It is possibly also the most popular application server in the world considering it has recorded more than 200,000 downloads in a single month. It includes an integrated copy of Apache Tomcat version 5[13]. Tomcat is considered the best open-source servlet container in the world and has driven the servlet specification and is the servlet container reference implementation.

Fundamentally, the JBoss architecture consists of the JMX MBean server, the microkernel, and a set of pluggable component services, the MBeans. This makes it easy to assemble different configurations and creates the flexibility to tailor them to meet the requirements. It is not necessary to run a large, monolithic server all of the time. It is possible to remove the components that are not necessary and also to integrate additional services into JBoss by writing custom MBeans.

2.3 Database Servers

The JBoss Application Server is bundled with the Hypersonic SQL database. Hypersonic SQL is an open-source database written in Java[14]. The problem with Hypersonic SQL is that it is not multithreaded, so it is not appropriate for most server-side developments. It is also limited to a subset of SQL. Most notably, it does not support foreign keys. Perhaps most significantly, it lacks the backup and management tools required for critical applications such as a voting system.

Consequently, a different database had to be investigated and chosen. The two most popular open-source databases are MySQL[15] and PostgreSQL[16]. MySQL was chosen for a number of reasons one of which is that it is faster than PostgreSQL[17]. MySQL also has a much larger user base than PostgreSQL and therefore the code is more tested and has historically been more stable than PostgreSQL[18]. MySQL is used much more in production environments than PostgreSQL, mainly due to the fact that MySQL AB has provided commercial support for MySQL from the day it was released, whereas PostgreSQL had originally been unsupported.

2.4 Apache Ant

Apache Ant[40] is a Java-based build tool. It was designed to be an improvement on previous build tools such as *make*[41] and *jam*[42]. These improvements include:

- It is platform independent.
- Tab syntax was considered problematic by some users of *make*. This is not the case with Ant.
- Ant is written in Java and can be extended using Java classes.
- Instead of writing shell commands, the configuration files are XML-based, calling out a target tree where various tasks get executed.

Nonetheless, it is probably a less expressive build tool than ones such as *make*.

2.5 Concurrent Versions System (CVS)

CVS[43] is a version control system. CVS maintains a history of all changes to the source files. CVS allows the concurrent editing of files by insulating the different developers from each other. Every developer works in their own directory and CVS merges the work when each developer commits their work. Consequently, a group of developers can work on the same project without the worry of overwriting each other's efforts. It is ideal for medium-sized teams, potentially in a distributed context.

As an outline, CVS can be used by:

- Importing source code files to a centralised "repository".
- Checking some source code files to be edited to a "sandbox".
- Committing any changes made back to the repository.

The repository contains directories and files, in an arbitrary tree. The *modules* feature can be used to group together a set of directories or files into a single entity. A typical usage is to define one module per project.

Each version of a file has a unique *revision number*. Revision numbers look like `1.1', `1.2', `1.3.2.2' or even `1.3.2.2.4.5'. By default revision 1.1 is the first revision of a file. Each successive revision is given a new number by increasing the rightmost number by one.

Log information and metadata is stored for every file revision. This log information can be tracked. Consequently, multiple different versions can be both compared and merged.

CVS is not limited to linear development. The *revision tree* can be split into *branches*, where each branch is a self-maintained line of development. Each branch has a *branch number*. The branch number is created by appending an integer to the revision number where the corresponding branch forked off. Having branch numbers allows more than one branch to be forked off from a particular revision.

2.6 Business Object Notation (BON)

The Business Object Notation (BON) is a method and complementary textual and graphical specification language for the analysis and design of object-oriented systems[19]. BON emphasises seamlessness, reversibility and software contracting. Its aim is to create a natural flow between analysis, design, and implementation by solely using concepts present in object-oriented programming.

Seamlessness and reversibility relate to the ability to not only go from analysis and design to executable code, but also to reflect source code changes back into design and analysis. If a method is to serve throughout the life cycle of a system, seamlessness and reversibility are essential. In BON, unlike the Unified Modeling Language (UML) for example, there are no entity-relationship diagrams and state-charts[20]. The reason is that despite the potential expressiveness of such a modeling concept, it is incompatible with object-oriented languages and so effectively prevents reversibility. BON uses classes as the main structuring mechanism and uses only inheritance and client dependencies to describe how the classes are related to each other. Consequently, reversibility is inherent because the major object-oriented programming languages directly support these concepts.

Despite the fact that this notation is based only on class abstractions related through inheritance and client dependencies, it is still extremely expressive because BON also contains two crucial means of specification: strong typing and class contracts.

Software contracts are one of the most important constructs for writing correct, reliable software. The idea is to use *assertions* to define the semantics of each class. The prerequisites and resulting behaviour of each operation are specified through *preconditions* and *postconditions*, and the overall class consistency through the *class invariant*. These semantic specifications then form the basis for a contract between each class, the supplier, and all classes using its operations, the clients. A software system is viewed as a network of cooperating clients and suppliers whose exchange of requests and services are precisely defined through decentralised contracts.

Finally, these elements are combined with a clustering mechanism to group related abstractions. This creates the possibility to scale up from small examples to large, complex real-life systems. The BON notation uses nested clustering and element compression to be able to zoom between the various levels of a large structure. Classes may be grouped into clusters according to various criteria (part of a subsystem, heirs of a common ancestor, related functionality, etc.). Clusters and classes may in turn be grouped into new clusters on higher levels. Depending on the current focus of interest, many different views of the same underlying system can be presented. This is achieved by compression. Most elements in BON can be compressed, which means that one or several graphical and textual elements are represented by a simpler element, its compressed form. The user may freely choose the amount of detail shown for each system part.

2.7 Java Modeling Language (JML)

The Java Modeling Language (JML) is a formal specification language for Java[21]. It is a concrete example of the concept of software contracting outlined in BON (see section 2.6). It specifies the behaviour of Java classes and records design and implementation decisions taken. It does this by adding assertions to the Java source code. These are of three types: preconditions, postconditions and invariants.

Preconditions and postconditions define a contract between a class and its clients. The client must ensure a precondition and may assume a postcondition. Methods may assume a precondition and must ensure a postcondition. Invariants are properties that must be maintained by all methods.

One of the main goals of JML is ease of use for any Java programmer. It does this by adding JML assertions as comments in Java source files between /*@…@*/ or after //@. Properties are specified as Java Boolean expressions augmented by a few extra operators: \old, \forall, \result, etc.

JML annotated Java code can be parsed, type-checked and tested for violations of assertions during execution. It can also be proven that contracts are never violated at compile-time using the ESC/Java2 Extended Static Checker.

2.8 ESC/Java2

ESC/Java2 tries to prove that an implementation conforms to a JML specification at compile-time, in a fully automated fashion[22]. It is a process of program verification, not merely program testing. It consists of three phases: a parsing phase (syntax checks), a type-checking phase (type and usage checks), and a static checking phase (reasoning to find potential bugs). This last phase runs a behind-the-scenes mathematical prover called ‘Simplify’. Parsing and type checking produce cautions or errors. Static checking produces warnings.

Due to the fact that ESC/Java2 delivers particular warnings, the developer is forced to specify certain properties. If the developer understands the code, then these properties are obvious. Consequently, if ESC/Java2 has forced these properties to be documented, then understanding the code becomes easier. One of its advantages is that ESC/Java2 is that it is independent of any test suite. Results of runtime testing are only as good as the test suite. Therefore, ESC/Java2 provides a higher degree of confidence for the developer.

Finally, it should be noted that ESC/Java2 is not entirely sound or complete. There are errors that it may miss and there are errors that will be reported that are impossible. However, it does find a lot of potential errors quickly.

2.9 The Party List System

The Netherlands uses a form of Proportional Representation (PR) called the Party List System[49]. A Party List system presents multi-member constituency electorates with political parties putting forward slates or lists of candidates. It represents the principal PR system in operation, although there are many variations of it, based on constituency size, thresholds and quorums, rules about whether the electorate can have preferences for individuals on party lists, whether the Greatest Remainder or Highest Average systems are used and which formulas are used within them.

In Dutch national elections there are 150 seats to be filled. The mandate is for 4 years. The electoral system is PR with closed lists[3]. Seats are distributed on a national level with parties receiving a seat for every 0.67% of the popular vote. The remaining seats are filled using the d'Hondt[4] method. There is a single national constituency.

In European Parliament Elections in the Netherlands there are 31 seats to be filled. The mandate is for 5 years. The electoral system is PR with Preferential Voting[5]. Seats are distributed on a national level with parties receiving a seat for every 0.67% of the popular vote. The remaining seats are filled using the d'Hondt method. There is a single national constituency.

2.10 Encryption

The bases of encryption are ciphers. A cipher substitutes characters with different symbols. Random substitution ciphers provide excellent security. The simplest form of random substitution is to break a message into blocks of 4 bytes and add a pseudo-random number to each block. To recover the plaintext (the input message), it is simply a matter of subtracting the same series of pseudo-random numbers from the ciphertext (the encrypted message). The key is simply the seed value for the pseudo-random number generator. This form of encryption is widely used (the file encryption options offered by word processors often use this method) as it is simple and highly reliable.

Block ciphers break the message into fixed-length blocks. Each block of plaintext is then converted into a block of ciphertext using a sequence of arithmetic operations and/or substitutions. The best known of these is DES (the Data Encryption Standard). DES uses 64-bit blocks with a 64-bit key (although only 56 bits are significant; the other 8 are parity bits). To create the ciphertext, the bits within a block are shuffled and XOR'ed with the key in a sequence of 16 substitutions called "rounds". Applying the same process (with the same key) to the ciphertext restores the original plaintext, so the process is *symmetric*. Advances in computer

[3] **Closed List*:*** A list of candidates (in rank order of priority to be given seats) drawn-up for elections, which may not be adjusted by the voter. This is the most common variation.

[4] **D'Hondt method**. Named after Victor d'Hondt, this method takes the votes obtained by each party list and divides them by 1, 2, 3, 4, etc until all the seats are filled. The quotas obtained are ranked from the largest to the smallest, and seats are allocated to the lists with the highest averages.

[5] **Preferential Voting*:*** Party List PR rule variant which enables voters, once they have voted for one party list, to choose their preferred ranking of candidates.

hardware mean the relatively short key used by DES is now vulnerable to a brute-force attack. A decent supercomputer could check all possible DES keys in just a few days. In the future, even a desktop PC will be able to crack any DES-encrypted message. The use of longer keys for symmetric ciphers still has weaknesses. One such weakness is that both sender and receiver must share the key while keeping it secret from everybody else. This poses a particular problem for Internet transactions since a secret key would no longer be secret if it were sent over the Internet. Even if it was embedded in a browser it could be discovered by reverse-engineering the program.

The solution lies in a group of ciphers known as *asymmetric* or public key/private key systems. In asymmetric systems the key used to encrypt a message is not the same as that used to decrypt it. If a message has been encrypted using one key of a pair it cannot be decrypted even by someone else who has that key. Only the matching key of the pair can be used for decryption. There are a number of asymmetric key systems but the best known and most widely used is RSA. The letters stand for the names of the three co-inventors: Rivest, Shamir, Adleman. Originally patented, the patent expired in September 2000 and the algorithm is now in the public domain. The Secure Sockets Layer (SSL) used for secure communications on the Internet uses RSA. The basic security comes from the fact that, while it is relatively easy to multiply two huge prime numbers (‘huge’ meaning numbers with several hundred decimal digits) together to obtain a product, it is computationally difficult to go in the reverse direction. It is this one-way nature of RSA that allows an encryption key to be generated and disclosed to the world, and yet not allow a message to be decrypted.

However, asymmetric encryption is really only practical for short messages as it is computationally expensive. A common workaround when encrypting long messages is to use RSA to encrypt a short preamble containing a DES key selected at random. The main body of the message is then sent encrypted with that key. A recipient with the corresponding private key can decrypt the preamble and use the key it contains to decipher the rest of the message. Modern web browsers use exactly this method to conduct secure communications.

3 System Analysis

This section describes the components that constitute the KOA system. First, it describes a high-level view of the design of the system. Second, it outlines how this is implemented using the J2EE framework.

3.1 Design Overview

The following information is a combination of the analysis of the source code of the KOA system and a reading of the first two chapters of *Kiezen op Afstand Specificatie Functionele Eisen*[4].

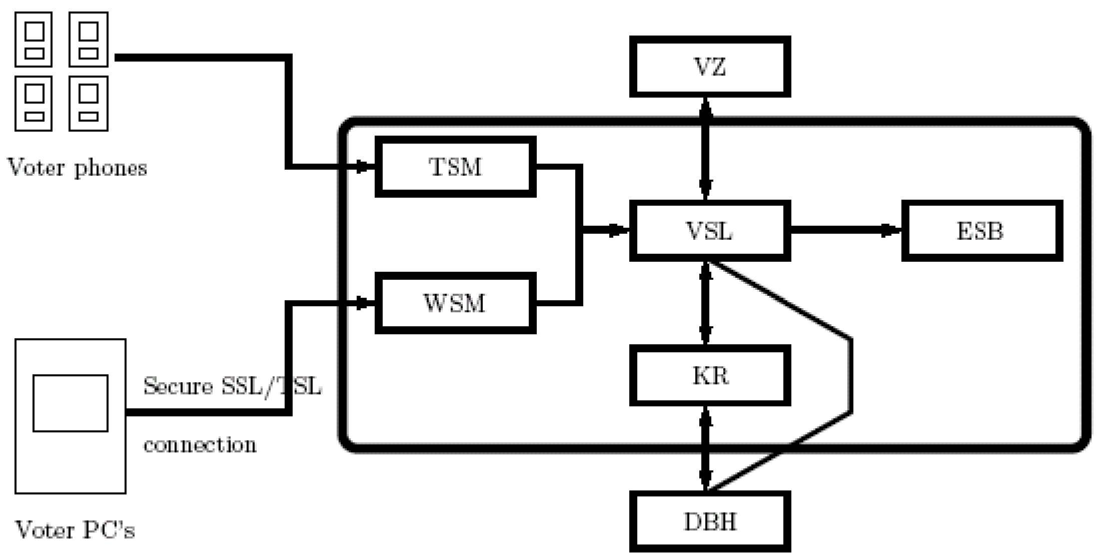

Figure 3. Functional Components of the KOA system

The components of the KOA system are:

- The Virtual Polling Station (VSL)[6]. This is the central controlling function of the KOA system. The KOA system contains only one VSL which can serve multiple electoral areas[7]. Multiple voting machines can be connected to a VSL. They are all connected in the same manner, whether the connection is via the internet or via telephone.

- The Electronic Ballot Box (ESB). Each electoral area has one ESB. All votes cast are encrypted and stored in the ESB.

- The Voter Register (KR). Before the start of the elections all of the (anonymous) information of authorised voters is stored in the KR. In addition, the KR registers if a voter has already cast his/her vote. There is only one KR.

- The Web Voting Machine (WSM). This component is the link between the voter's PC and the virtual polling station. There is only one WSM.

- The Telephone Voting Machine (TSM). This component is the link between the voter's telephone and the virtual polling station. There is only one TSM.

[6] All abbreviations come from Dutch.

[7] The words "electoral area" will be used in this document as a translation of the Dutch "district". The word "ward" would also be possible as an equivalent concept in the Irish electoral context.

The returning officer[8] of the polling station (VZ) connects to the VSL via a secure connection. S/he controls the VSL and receives reports from the VSL such as the final vote tally. Each VSL can have multiple returning officers, but only one returning officer is active at any given moment. See section 3.1.2 for more information on the tasks of the returning officer.

The Data Management System (DBH) has the following functions:

- Reading the candidate lists into the system and delivering the generated candidate codes to the customer. This function takes place before the actual poll has opened.

- Loading the KR with the anonymous records of authorised voters.

- Maintaining the voter register after the start of the election (adding voters and stopping the registration of voters).

3.1.1 KR (Voter Register)

The Voter Register contains all data related to authorised voters. The Voter Register consists of two main parts: data to authenticate the Voter and data concerning a voter's ballot.

In order to verify if a voter is authorised to vote the following information is recorded:

- The voter code used to identify authorised voters. This voter code is unique and is generated by the KOA system when the data is loaded. The voter code is a 9-digit number.

- The access code which has been chosen by the voter. The access code is a 5-digit number[9]. This code is delivered and stored in encrypted form using a one-way encryption algorithm. For security reasons the system does not differentiate between the situations "access code not present" and "access code incorrect".

- An indication whether the voter has been "removed". By default, this is not set to true. Only after the election has been started can the voter be "removed".

- Electoral Area and Constituency[10]. The electoral area is used to validate the candidate codes. When a vote is cast both the electoral area and constituency are stored with the vote. This ensures that a result per electoral area can be calculated. Results per constituency are also possible, although this function was not used in the European Parliament Elections experiment.

The data stored concerning a voter's ballot is:

[8] "Returning Officer" is the translation being used for the Dutch term "Voorzitter" which is literally translated as "president" or "chairperson".

[9] The 5 digit code is a compromise. A longer code provides more security, but increases the probability of an input error. Banks use a 4 digit PIN, but this is in combination with a physical token (the bank card). A parallel example is the Dutch Revenue Commissioners who also use a 5 digit PIN for online transactions.

[10] "Constituency" is being used as a translation of the Dutch term "kieskring".

- An indication if s/he has already voted or not.
- The date and time at which the vote was cast.
- The transaction code receipt.
- A record of any failed attempts by a voter to login, consisting of:
  - The counter of the number of attempts made.
  - The total number of attempts made.
  - The time of the last attempt.

After a certain number of attempts to login all further attempts are blocked for a certain period. The parameters for this function can be changed. The default is a maximum of three attempts and a blocking period of one hour. The total counter per voter will never be returned to zero. It is used to signal a systematic attempt over a longer period of time to tamper with the data of a certain voter.

3.1.2 VSL (Virtual Polling Station)

The VSL is responsible for operating the election process. The VSL distinguishes functions carried out by the returning officer and functions which are part of the voting process.

The functions of the returning officer are:

- Initialisation. This prepares the system for the start of the elections. The VSL module controls the initialisation of all the other components of the KOA system and receives a confirmation message from them. If all components are ready the election can start.
- Opening the election. This can only happen when the system is properly initialised. At the start of the elections the VSL gets the public key of an RSA (see section 2.10) key pair from the returning officer. This key is used to encrypt all incoming votes.
- Interrupting the elections. Once the elections are interrupted, the VSL refuses any incoming commands from the voting machines. All components receive a command to halt their activities. Both the ESB and the KR transmit a confirmation by sending an electronic fingerprint[11] to the VSL. The VSL returns them to the returning officer.
- Re-initialising. This can only happen if the elections are interrupted. A new electronic fingerprint is generated which is compared to the one that was generated when the elections where interrupted. If the fingerprints do not match, the returning officer must decide if the elections can continue.

[11] A ‘Fingerprint’ is a snapshot of particular system data at a given point in time. It is a concept that will be used throughout this report.

- Continuing the elections. This is only possible if the system has been interrupted and re-initialised using one of methods explained above. The public key of the returning officer is retransmitted and compared to the original public key. This prevents the votes in the ESB from being encrypted with multiple keys.

- Closing the elections. Hereafter, the VSL refuses to accept any new votes cast. All components of the system receive a command to stop their activities. The ESB and KR transmit a confirmation by sending an electronic fingerprint to the VSL. The VSL delivers these to the returning officer. Voters receive the message that the elections are closed.

- Opening the ‘ballot boxes’. This can only happen if the elections are closed and can only be performed once during any given election. This private key of the returning officer is sent to the ESB which uses it to decrypt the votes in a random order (to making tracing voters by the order in which they voted impossible). They are then transformed into a specially formatted file. The counting process begins and the result (in the form of a report) is delivered to the returning officer.

- Querying incidents/events. Events/incidents that are detected in the system which are a potential threat to the election process are logged and can be queried by the returning officer.

The second function of the VSL module is concerned with the actual process of casting a vote. This is carried out by communicating with the voting machines (TSM, WSM). Please refer to section 3.1.4 and section 3.1.5 for further details.

3.1.3 ESB (Electronic Ballot Box)

All votes are encrypted and stored in the Electronic Ballot Box (ESB). There is one ESB per electoral area. It contains:

- The contents of the candidate code of the candidate that received the vote (or a fictive candidate with semantics “blank vote”[12]).

- The name and initials of the candidate.

- The position on the candidate list.

- The list number.

- The political party of the candidate.

- The constituency is also stored in order to give election results per constituency.

[12] Under Dutch law it is possible to cast a vote without voting for a candidate. This is called a ‘blank vote’. Under Irish law, it is currently legal to cast a vote with no candidate preference, but illegal to indicate on the ballot that there is no preference.

Blank votes are represented in the system as a persistent party list with only one candidate. For security reasons blank votes are treated as much as possible as ordinary candidates.

The votes are encrypted when they are stored using the public key of the returning officer of the polling station. This guarantees two things: that the votes can only be opened by the returning officer of the polling station and that the corresponding VSL is responsible for the votes cast. Random data is added to the votes when they are encrypted. This ensures that votes within the same constituency and for the same candidate are encrypted differently for each vote, making it impossible to make inferences from encrypted votes.

3.1.4 TSM (Telephone Voting Machine)

The TSM is a so-called Voice Response system. It asks the voter questions in a predetermined order. The voter responds by pressing keys on his/her telephone.

The voting process using this facility is as follows:

- The voter hears a short welcome message as confirmation that s/he has dialled the correct telephone number.

- The voter is then asked to first give his/her voter code and then the access code that has been chosen by the voter. The TSM checks the incoming data with the VSL.

- The voter is then asked to give the number of the candidate of their choice[13]. Due to security concerns an acknowledgment is given by reading the candidate code back to the user. This confirms that the user has voted for the correct candidate. The name of the candidate could be spoken, but this would ignore security restrictions (i.e., eaves-dropping). If the voter does not confirm his/her vote then s/he can continue with a new vote (phase 3) or cancel the process.

- The TSM sends the voter data to the VSL and receives a transaction code. This is a random number that is generated in the same manner and adheres to the same criteria as the voter code. This code is read out to the voter. Then the connection is closed. Once the VSL receives the vote it is definitively cast (unless a technical error occurs when the vote is stored). This is still the case even if the voter disconnects before the transaction code is received.

Before the system is initialised each caller receives a short standard message stating that the elections have not yet started. The same holds when the system is interrupted (“Elections interrupted”) and when the TSM has no live connection with the VSL (“The voting service is not accessible at this moment”). After the elections have ended the standard message is substituted for a message which states that the elections have closed. The TSM checks at three points if the VSL is accessible: when the Voter is verified, when the candidate is checked and when the vote is cast.

[13] The Dutch electoral system uses a List System (see section 2.9). Ireland uses the Proportional Representation-Single Transferable Vote (PR-STV) system. This would necessarily make the interface and voting process for the TSM and WSM more complicated.

3.1.5 WSM (Web Voting Machine)

The Web Voting machine (WSM) is a standard web-server.

The following guidelines were used in the implementation of the WSM:

- The interface has been kept as simple as possible. This ensures that a minimum of data has to be transferred, which ensures that voters with a slow connection can still use this facility.

- The requirements for the browser are kept to a minimum. The most important requirement is that the browser should support encrypted connections. Almost all browsers conform to this. The browser must also allow session cookies. This is almost always the case, even for PC's behind a firewall. Users that try to connect to the standard HTTP port are immediately redirected to the secure pages.

The menu structure of the web-interface is as follows:

1. The voter first sees a couple of pages which briefly explain the voting process.

2. A verification screen is then displayed which requires the voter to enter the voter code and access code. The WSM verifies these with the VSL. If the data is incorrect the voters can retry a predetermined number of times. If verification continues to fail an error message is displayed. Further attempts are not possible. The voter must login again. It could also be the case that the polling station is not yet open or that the voter is blocked. In these cases an appropriate error message is displayed and the voter is unable to try to login again. If it is the case, during the verification process that the voter has already voted then the appropriate error message is displayed together with the transaction code they have already received.

3. The voter then sees a screen that allows him/her to enter the 9 digit candidate code.

4. The vote is transferred to the VSL in order to verify if the candidate code is correct. (i.e., the candidate code is contained within the candidate list that corresponds to the electoral area for this particular voter) If this is correct the system displays a screen with the name, place of residence, and the political party of the candidate. The voter can either confirm this vote or return to the previous page.

5. The WSM then transfers the vote to the VSL which sends back a transaction code by way of confirmation. This transaction code is then displayed on the final screen.

The system checks if the polling station is open in the same as with the TSM. The WSM becomes active when the elections have been started. Until then a standard message is displayed that states that the elections have not started yet. When the elections are interrupted another message is displayed ("elections interrupted"). If the WSM can (temporarily) not connect to the VSL the message "the voting service is temporarily not available" is displayed. When the elections are closed the standard message is replaced by a message which states that the polling station is closed.

The WSM is only used for voting. A separate web-server has to be used for information purposes and other such activities.

3.2 Released System

The components of the KOA system described above are implemented by 489 Java classes, 59 of which are missing from the system released under the GPL licence. The remaining 430 classes are distributed among 13 Enterprise Java Beans and a number of utility classes. Due to the size of the system and the limited length of this report, class diagrams for each of the packages and Javadoc documentation can be downloaded in archive format (.zip or .tar.gz) from http://sort.ucd.ie/docman/?group_id=13.

3.2.1 Enterprise Java Beans

The functionality of the WSM is provided by the following two EJBs:

The **KOASAREntityEJB** manages the persistent storage of the 'stemcode'[14]. It may retrieve and modify an access code and provides a facility to search the database for the corresponding voter code.

The **KOAStemprocesSessionBean** controls the voting process. It manages the flow between identification, candidate choice, confirmation and the receipt of the transaction code. Given the necessary data of the voter, candidate chosen, and voter pincodes, the bean sends the encrypted vote to the ESB database.

First, it validates the vote. It verifies the vote to ensure the input is correct, the voter exists and the voter is eligible to vote (i.e., the voter code and access code are correct, the voter has not been blocked, the voter has not voted already and that the voter has not been 'removed').

Second, it encrypts the vote using the public key provided by the returning officer.

Finally, the vote is saved and the voter is given a new unique transaction number. This uses a random number generator and then checks to see if the returned transaction number has already been used. If so, it continues to generate random numbers until a unique number is produced. It then sets the property that this number is now used. The fact that the voter has now already voted is saved. The encrypted vote is passed to the ESB EJB to save the vote in the database.

The functionality of the VZ is provided by the **Controller EJB**. The entity bean manages the persistent storage of the current state of the KOA system. It retrieves, modifies and queries the current system state. When initialised, the current state is automatically set to 'Prepare'.

The session bean provides methods to initiate the main system state changes: Prepare, Initialise, Open, Suspend, Reinitialise, Block, and Close. When the state is changed all clients that are subscribed to the controller are notified of this state change and the change is then persistently stored using the Controller Entity bean. It also collects all of the 'counters' from the other components for consistency checks.

The functionality of the DBH is provided by the **KOADatabeheerSessionEJB**. It manages the upload and removal of 'kieslijsten' or 'kiesregisters' from the database. They are uploaded from XML files. For each candidate these files contain:

[14] The Dutch word 'stemcode' means 'vote code'. Both terms will be used in this report.

- the name and number of the candidate's constituency,
- the name and number of the candidate's electoral area,
- the name and number of the candidate's political party
- the name and list position number of the candidate
- 10 candidate codes

Any previous 'kieslijst' database entries are deleted, the XML files are parsed and committed to the database and an XML report of this action is stored. When the 'kieslijst' entries are removed from the database the following tables are removed: 'kieslijsten'[15], 'lijstposities'[16] and 'kandidaatcodes'[17].

The 'kiesregister' upload process proceeds in a similar fashion to that for the 'kieslijsten'. However, a fingerprint is also stored in the database over the static part of the KR system. When the 'kiesregister' entries are removed from the database all of the entries from the 'kiezers'[18] table are also removed.

Finally, it facilitates the insertion of a list of 'kiezers' into the database. This method ensures that the voter is not already in the database, that its electoral-area and constituency exist and are linked in the database and that it possesses a 'stemcode'.

The functionality of the KR component is provided by the **KR EJB**. The entity bean manages the updating and persistent storage of information relating to 'kiezers'. The main information stored is the name, vote code, voter code, hashed access code, electoral area number, constituency number, transaction number, boolean values indicating whether the voter has been 'removed', 'already voted', and 'account blocked', the number of login attempts left and the timestamp of the last successful login.

The session bean controls the interactions between the 'kiezers' and the system. It defines how the system should react when a state is about to change and manages voter login. The static part of the KR system is the name, vote code, etc. and the dynamic part are the boolean values (already voted, etc.) and the number of login attempts left. When the KR system is initialised, the bean checks if the database is filled with data, checks to see if all voters have not already voted and checks that all timestamps are empty. Provided this is the case a new static fingerprint is created, compared with the fingerprint created when the voters were imported into the system and finally this fingerprint is stored.

[15] The Dutch word 'kieslijst' means 'electoral list'. Both terms will be used in this report.

[16] The Dutch work 'lijstposities' means 'list positions'. It is an important concept in the List System for elections (see section 2.9). Both terms will be used in this report.

[17] The Dutch word 'kandidaatcodes' means 'candidate codes'. Both terms will be used in this report.

[18] The Dutch word 'kiezer' means 'voter'. Both terms will be used in this report.

When the state of the KOA system is about to be changed the KR system opts to do one of a number of things. If the KOA system is about to be initialised the KR system initialises also. If the KOA system is about to enter the ‘Votes Counted’, ‘Open’ or ‘Prepare’ state, the KR system does nothing. If the KOA system is about to enter the ‘Block’ state the KR system simply creates a new static fingerprint. If the KOA system is entering one of the other states the KR system creates new static and dynamic fingerprints and compares them to the initial and most recent fingerprints respectively and then stores them.

The system manages voter login by first checking if the system is open for voting and whether the voter code and access code are not empty and are of the correct length and format. If the voter code exists and the access code is identified as matching the one in the database the number of successful logins is incremented and provided that the voter has not already voted then a new timestamp is created for the most recent successful login. If the password is incorrect, the number of invalid logins is incremented and if it equals the maximum number of invalid logins the voter account is locked and a timestamp is set for the next available opportunity for the voter to try and login.

The functionality of the VSL is carried out by the following two EJBs:

The **KOA Entity EJB** manages the persistent storage of the electoral areas, candidate codes, constituencies, electoral lists and list positions. It also manages the associations and multiplicities between these classes.

The **KOAKieslijstBean** provides functionality to retrieve the political parties and candidates for a given ‘kieslijst’ in a given ‘kieskring’. It allows the existence of a candidate to be verified and of the proper ‘type’ (i.e., it contains the correct fields with a certain type of input) for the constituency of a given voter provided that the state of the system ‘Open.’ It is able to save a fingerprint or compare it to a previous fingerprint.

The functionality of the ESB is carried out by the **ESB EJB**. The entity bean manages the persistent storage of the encrypted votes, the decrypted votes and the fingerprints of the ESB system. A vote consists of a vote number, a candidate number, a constituency number, an electoral area number, an electoral list number, a list position number, a list name and a candidate name. A fingerprint consists of an ID, a byte array that stores the fingerprint data, a timestamp and an indication of the system state.

The session bean controls the decryption and counting of votes in the system. Votes are encrypted during the voting process and are stored in the encrypted ESB table in the database. These votes are decrypted using the KOA encryption utility. The decrypted data is tokenised to fill the abovementioned vote fields. These decrypted votes are then saved to the decrypted ESB table in the database.

When the votes are counted the result is place in XML format in the ‘reports’ table. This file contains:

- the constituency name and number,
- the time and date of polling,
- the number of people who voted,

- the number of people who did not vote,
- the number of votes for each political party/list,
- the name, list position and number of votes for each candidate, and
- the number of blank votes

It checks for blank ballots by checking whether the list name is identical to the 'Blank Vote Indicator.' Votes are counted both by political party and by candidate for a particular constituency.

The remaining two EJBs provide utility functionality for the KOA system:

The **KOASchedulerEJB** bean manages jobs and their execution. The entity bean manages the persistent storage of the jobs and their properties (name, timestamp, context, priority, successor, type, retry count, status, etc.). The session beans manage the execution of the jobs and their relative position in the priority queue. Timestamps and commit/rollback calls are used to ensure the jobs are consistently executed and that any sort of deadlock/starvation is avoided. Jobs are allocated a priority and together with their age determine whether they are set as the successor job.

The **KOA Session EJB** logs audit records to the database using either an existing transaction or a new transaction. If the logging fails the system state is changed to 'Blocked.' This is important as the system would no longer be verifiably correct if the audit trail was interrupted.

3.2.2 Utility Classes

There are a large number of utility classes that complete the functionality of the KOA system. Adapter classes link the TSM system into the web-based VSL. Constants relating to errors, JNDI naming and functional and technical properties are accessed using the constants classes. Classes for database querying, event handling, exceptions, I/O, logging, reports, security and servlets are also provided.

4 Installation

This section describes the steps necessary to install the incomplete KOA system on the JBoss 4.1 application server using a MySQL 4.17 database server.

4.1 Compilation

The first step that was necessary was to get the source code to compile. This was a relatively straightforward although time consuming task. The error messages were analysed which indicated which steps needed to be taken in order to get the source code to compile. The greater part of the compilation errors were caused by the missing classes. These classes were added with the appropriate empty methods and instance variables. In addition, the IBM WebSphere IDE compiles certain import statements inherently without needing them to be stated explicitly in the source code class. It was therefore necessary to insert these import statements. One such missing import statement included an entire package to deal with uploading and processing files. This was researched and the file upload utility package provided by Apache[46] was chosen. Strangely, there was also a number of *catch* blocks that needed to be appended. Finally, the KOA system used a proprietary encryption utility developed by IAIK[44]. As has already been mentioned, the use of proprietary software is not an option in developing the KOA system. This encryption utility was replaced by an open-source alternative called BouncyCastle[45].

4.2 Packaging and Deployment

The compiled system was packaged and deployed by writing an Ant build file. This compiled the source code, packaged each of the EJBs into their respective JAR files, packaged the web-based components (JSPs, images, utility classes) into their respective WAR files, combined these packages into an EAR file and copied it to the 'deploy' directory of the JBoss server. In packaging the various components, it was important that each JAR file was named correctly according to JNDI naming scheme, that it contained its original deployment descriptor, its JBoss specific deployment descriptor (which maps EJBs to the database), the necessary external libraries, and all necessary bean and utility classes

4.3 Database Setup and Connectivity

Although the KOA system was deployed, the EJBs failed to initialise as they needed to connect to the database and create the necessary tables. JBoss needs a special Java library to manage JDBC connectivity and functionality with a MySQL database. The de facto standard library is Connector/J[47]. This is added to the *lib* directory of the JBoss server. Subsequently, a number of special XML files must be present in the JBoss server. These files contain information such as the name of the database, the host and port number on which the database server can be accessed, the username and password needed to access the database, and the JNDI name being used[25]. Finally, the database itself was created with the correct access privileges and permissions. It is called 'KOA01'[19]. When the EJBs are deployed and initialised during server startup, a number of tables are created (Decryptedesb, Districten, ESBSequencesEJB, Encryptedesb, Esbfingerprints, KRFingerprints, KRSequenceEJB, Kandidaatcodes, Kieskringen, Kieslijsten, Kiezers, Koa_state, Lijstposities, Sar, Scheduledjob, Transactioncode) together with JBoss specific tables (HILOSEQUENCES, JMS_MESSAGES, JMS_ROLES, JMS_SUBSCRIPTIONS, JMS_TRANSACTIONS, JMS_USERS, TIMERS).

5 Implementation

This section describes the implementation of the missing functionality of the KOA system. The development process was iterative with emphasis shifting between the different areas as diverse issues arose. See section 6 for an elucidation of the problems encountered. Nevertheless, the necessary changes will be divided into the areas of user interfaces, properties and classes with each being considered in turn.

[19] This name is hard-coded into the KOA system source files.

5.1 User Interfaces

There were two problems with the JSP web interfaces. First, it was necessary to translate the JSPs from Dutch into English, in order to make them usable and testable. Second, a bug was found in the JSPs whereby a button that was supposed to guide the user back to the interface homepage in fact had the same action as that of the 'submit' button on the same page. This was due to the fact that the 'return home' button had been incorrectly placed within the 'form' tag block of the HTML. This meant that the action of the 'form' block overrode any action the 'return home' button might itself have. Such a simple mistake in the design of the user interface is rather worrying. The fact that such a mistake could be made, not noticed in the testing and evaluation phase for the software, and actually used in the elections to the European Parliament, would suggest that there is in all likelihood further bugs in this software. This would imply that the KOA system should not have been used in such an important poll.

5.2 Properties

Some of the most important property files were not released with the published system. These were the JNDI property file and the security authentication property files.

Although the *JNDI..properties* file was included with the released system, all of the information that it had contained was deleted for publication purposes. Consequently, none of the system components were able to locate each other when they needed to communicate. Therefore, it was necessary to perform a further analysis of the KOA system source code and the deployment descriptors in order to discover what information should have been contained in this file. Although time consuming, this was eventually completed.

Both the KOAVoorzitterWeb and KOADatabeheerWeb are protected with a JAAS security mechanism (this is in addition to the other security elements of the system). This works by prompting for a username and password in order to access these interfaces for the first time. These usernames and passwords are stored in a special file called *users.properties*. The roles for which these username/password pairs are valid are contained in a special file called *roles.properties*. However, these files were not contained in the released system. When the system was initially deployed, it was not possible to access these interfaces, and no error message was generated. These files were created, given meaningful values and packaged with the rest of the deployed system.

5.3 Classes

The missing source code is divided into two parts. First, the base functionality for the web interfaces, called the 'ePlatform', consisted of the servlet, ticket, errors, commands, logging, services, event handling and utilities code on which the rest of the system depended. Second, there were the classes automatically generated by the IBM Websphere IDE that consists of EJB persistence and container classes, classes that link data through multiplicities and associations, converters for the TSM component, and security classes.

Although it was initially envisaged that the missing classes would be designed by contract by writing the JML specifications first and then writing the actual code, the complexity of system resulted in the conclusion that this was not feasible. The first aim was to be able to browse around the web interface. This depended mainly on the servlet functionality of the ePlatform. These classes were coded in a straightforward manner. They simply initialise certain services and command factories, pass the HTTP servlet request and response on to the appropriate KOA servlet and then redirect the user to the appropriate page once this request has been processed.

Once the system could be browsed, it was desirable that an actual vote could be cast. Unfortunately, it was soon discovered that it was not possible to use any of the functionality of the system until the system had been initialised by the returning officer. Disappointingly, after obtaining an appropriate public key and solving a number of problems, it was discovered that information in relation to the encryption of the public key (the SALT[20] byte array, the encryption key length, etc.) had been altered for publication purposes. Although, appropriate values were available from the University of Nijmeegen Security of Systems team [1](they needed to use the same encryption information to be able to decrypt the votes for the separate tally system[1]), problems persist with the padding[21] of the public key which have meant that it has been impossible to make further progress.

6 Problems Encountered

A number of difficulties arose that impeded the completion of the project. These include:

1. The scale of the KOA system. It contains 489 class files, together with numerous properties files and deployment descriptors. It is considered that most software developers can comprehend about 100 classes on average at any one time. Taking into account the level of method propagation it made the KOA system extremely difficult to understand.

2. The language barrier. All of the design specifications, the JSPs, some of the variables, some of the log and error messages and about half of the comments are in Dutch. This slowed down the comprehension of the system.

3. Although not mentioned in the project description, one of the major tasks was to port the system from the IBM WebSphere application server to an open-source alternative. Not having had experience or access to use IBM WebSphere made it difficult to appreciate its unique properties.

4. Commenting of the source code of the KOA system is extremely sparse.

5. Although JBoss 4.0 integrated without problems with MySQL 4.17, problems were encountered when trying to set it up to run on a remote server (http://correct.ucd.ie/KOAWeb) using MySQL 3.23. MySQL 3.23 does not support subqueries which caused problems for JBoss.

[20] A string of random (or pseudo-random) bits concatenated with a key or password to foil precomputation attacks.

[21] Extra bits concatenated with a key, password, or plaintext.

7 Appraisal

This section describes an assessment of the quality of the work carried out. Despite the abovementioned problems encountered, mistakes were made in the execution of the project.

- Initially, in the early months, a disproportionate amount of time was spent in researching and understanding certain topics such as BON, JML, ESC/Java2. Although these are extremely interesting areas, a solid understanding of J2EE technologies was garnered later than it should have been.

- The author was slow to adapt and form a methodical and analytical approach to deal with the scale of the system and the nature of EJBs.

- The tardiness experienced in getting the KOA system deployed on a remote server made it difficult to develop and test the system on campus.

8 Conclusions

This section describes the conclusions that can be drawn from the execution of this project.

- Personally, it was an excellent learning experience. The KOA system incorporates a large number of technologies as can be witnessed from the breadth of background research. A strong predilection towards open-source software has been acquired.

- Considering some of the flaws that have been noticed in the KOA system, it is worrying that the Dutch Government would use such a system, even in a limited fashion, in as important an election as that to the European Parliament. In spite of this, the Dutch Government should be applauded for releasing the system under the GPL. It is thanks to this transparency that these flaws can be discovered. No such option is given in any of the electronic voting systems currently in use.

- It will be extremely important that the KOA system be available entirely in English. As an international language it will facilitate a large number of people to develop and test it. Such numbers will be necessary in order to complete a secure, transparent and trustworthy remote internet voting system.

- It is sincerely disappointing that further progress towards completing the KOA system was not made in the time available. It was unfortunate that the opportunity was not taken to gain additional practical experience in the areas of formal methods and security. Nonetheless, the background research alone has fuelled an interest in postgraduate study and research in these areas.

9 Future Work

This section outlines the future work that is necessary towards completing a secure, transparent, open-source and trustworthy remote internet voting system. The developers who have subscribed to the project[48] will continue with the work that has already been undertaken.

- A paper will be written and submitted outlining the results of this project.
- The remaining functionality of the KOA system will be reverse engineered.
- The KOA system will be fully commented.
- High-level BON specifications will be written and generated for the full KOA system.
- An analysis of the system's test code coverage, CPU and memory usage, load capabilities, etc. will be carried out.
- Formal specifications in JML for the full KOA system will be written. Such specifications will be used to test the system for correctness and security flaws using runtime assertion checking, unit test generation, and extended static checking.
- A security audit of the KOA system will be carried out. This analysis will primarily consist of developing a threat analysis model.
- A full translation into English will be carried out. Although some translation has already taken place, much of the internal code and commenting is in Dutch or based on Dutch. As an international language, English will make it more portable to other countries and will allow more developers to work on the system.
- The JSPs will be rewritten so that they are Cascading Style Sheets, Level 2 compliant[50]. The number of tables present in the HTML for the web-based interfaces makes them extremely difficult to read and comprehend. The separation of style and content that is inherent in CSS2 would remedy this and also make it easier to adapt for different elections.
- Considering the importance of security for an internet voting system and given that the OpenBSD 3.6[51] is the most secure operating system available, it would probably be worthwhile to investigate the possibility of running the KOA system on the OpenBSD platform.

Internationally renowned security technologist, Bruce Schneier[54], has written: “A secure Internet voting system is theoretically possible, but it would be the first secure networked application ever created in the history of computers.” He is correct. A secure internet voting system *is* possible. However, it will take an enormous amount of work and collaboration.

10 References

1. Nijmeegs Instituut voor Informatica en Informatiekunde Security of Systems, **Kiezen op Afstand**, http://www.cs.ru.nl/sos/research/koa/

2. GNU Project (2004), **GNU General Public Licence**, http://www.gnu.org/copyleft/gpl.html

3. **Experimenten bij verkiezing leden van het Europees Parlement**, http://www.minbzk.nl/grondwet_en/verkiezingen/inspringthema_s/kiezen_op_afstand/per

sberichten/experimenten_bij (Dutch ministry press release indicating the results of the trial vote.)

4. LogicaCMG, **Kiezen op Afstand Specificatie Functionele Eisen** (translated into English by Martijn Warnier) available from http://sort.ucd.ie/docman/?group_id=13
5. **Nedap**, http://www.nedap.com/
6. **Powervote – nedap UK**, http://www.election.nl/bizx_html/IVS-IE/
7. **The Commission on Electronic Voting**, http://www.cev.ie/
8. **Part 4 Summary and Conclusion: 4.2 Testing, Accuracy and Secrecy**, http://www.cev.ie/htm/report/part4_2.htm
9. **Commission on Electronic Voting invites tenders for software assurance and testing services for e-voting system**, http://www.cev.ie/htm/press/press111104.htm
10. Armstrong, E., et al. (2004), **The J2EE 1.4 Tutorial**, http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html
11. **IBM WebSphere Software**, http://www-306.ibm.com/software/websphere/
12. **JBoss.com :: The Professional Open Source Company**, http://www.jboss.org/
13. **The Jakarta Site – Apache Jakarta Tomcat**, http://jakarta.apache.org/tomcat/
14. **Hypersonic SQL**, http://hsqldb.sourceforge.net/
15. **MySQL: The World's Most Popular Open Source Database**, http://www.mysql.com/
16. **PostgreSQL**, http://www.postgresql.org/
17. **MySQL|Information|Benchmarks**, http://ftp.csie.chu.edu.tw/Mirrors/www.mysql.com/information/benchmarks.html
18. **PostgreSQL vs. MySQL: Which is better?**, http://www.databasejournal.com/features/mysql/article.php/3288951
19. Waldén, K. and Nerson, J.-M. (1994), **Seamless Object-Oriented Software Architecture: Analysis and Design of Reliable Systems**, Prentice-Hall, Inc., also available from http://www.bon-method.com/
20. Pooley, R. and Stevens, P. (1999), **Using UML: Software Engineering with Objects and Components**, Addison-Wesley Longman.
21. **The Java Modeling Language (JML) Home Page**, http://www.jmlspecs.org/
22. **The ESC/Java2 website**, http://www.cs.kun.nl/sos/research/escjava/
23. Kiniry, J. (2004), **Electronic and Internet Voting in The Netherlands**, http://kind.cs.kun.nl/~kiniry/papers/NL_Voting.html

24. Kiniry, J. (2004), **Project 75: KOA evaluation and demonstration installation and implementation**, http://www.cs.ucd.ie/courses/undergrad/bsc/FourthYear/projects/Project_75/default.htm

25. **Getting Started with JBoss 4.0, Release 3**, http://www.jboss.org/

26. Waldén, K. and Data, E. (1998), **Business Object Notation (BON)**, http://www.bon-method.com/

27. Burdy, L., et al (2002), **An overview of JML tools and applications**, *International Journal on Software Tools for Technology Transfer (STTT)*, Springer-Verlag Heidelberg.

28. Leavens, G., et al (2000), **JML: notations and tools supporting detailed design in Java**, *OOPSLA '00 Companion*, Minneapolis, Minnesota, pp. 105-106.

29. Kiniry, J. (2004), **JML and ESC/Java2 Homework Exercises**, http://kind.cs.kun.nl/~kiniry/ecoop_tutorial.html

30. Cok, D., Poll, E., Kiniry, J. (2004), **ESC/Java2: Uses and Features**, http://kind.cs.kun.nl/~kiniry/ecoop_tutorial.html

31. Cok, D., Poll, E., Kiniry, J. (2004), **ESC/Java Extended Static Checking for Java**, http://kind.cs.kun.nl/~kiniry/ecoop_tutorial.html

32. Cok, D., Poll, E., Kiniry, J. (2004), **Introduction to JML**, http://kind.cs.kun.nl/~kiniry/ecoop_tutorial.html

33. Mills, A.J.S. (2002), **Ant Tutorial**, http://supportweb.cs.bham.ac.uk/documentation/tutorials/docsystem/build/tutorials/ant/ant.pdf

34. **Dutch-English Translation**, http://babelfish.altavista.com/

35. **ESS-Model**, Retrieved October 15th, 2004 from http://essmodel.sourceforge.net/

36. **Ososs - Programma Open Standaarden en Open Source Software voor de overheid**, http://www.ososs.nl/article.jsp?article=9698 (Report announcing the release of the KOA system by the Dutch government. Transparency is given as the reason for the decision. The source code can be downloaded from this site. Advice is given to potential users.)

37. Pieters, W. (2004), **SoS - Society - Electronic voting in the Netherlands**, http://www.cs.ru.nl/sos/research/society/voting/index.html

38. Libbenga, J. (2004), **Dutch e-voting software goes open source**, http://www.theregister.co.uk/2004/06/23/open_source_voting_software/

39. Hunter, Jason & Crawford, William (2001), **Java Servlet Programming (2nd Ed.)**, O'Reilly.

40. **Apache Ant**, http://ant.apache.org/

41. **GNU Make,** http://www.gnu.org/software/make/

42. **Jam – Perforce Software,** http://www.perforce.com/jam/

43. **CVS – GNU Project – Free Software Foundation,** http://www.gnu.org/software/cvs/

44. **IAIKs Java Crypto Products**, http://jce.iaik.tugraz.at/products/index.php

45. **bouncycastle.org**, http://www.bouncycastle.org/

46. **FileUpload**, http://jakarta.apache.org/commons/fileupload/

47. **Connector/J 3.1**, http://dev.mysql.com/downloads/connector/j/3.1.html

48. **MyGForge: KOA: Project Info**, http://sort.ucd.ie/projects/ucdkoa/

49. McGee, Simon & Isaacs, Adam, **Electoral Systems in Europe: An Overview,** http://www.ecprd.org/Doc/publica/OTH/elect_system.html

50. **Cascading Style Sheets, Level 2,** http://www.w3.org/TR/REC-CSS2/

51. **OpenBSD,** http://www.openbsd.org/

52. **HTTP – Hypertext Transfer Protocol Overview,** http://www.w3.org/Protocols/

53. **CGI – Common Gateway Interface,** http://www.w3.org/CGI/

54. **Schneier.com,** http://www.schneier.com/index.html