Natali Ivanova

First Degree Programme in Mathematics and Computer Science

Master’s thesis;

Internet Based Interface for Control of a Mobile Robot

Internetbaserat gränssnitt för fjärrkontroll av en mobil robot

Master’s thesis subject: Computer Science

Supervisor: Henrik I. Christensen

Examiner: Henrik I. Christensen

Taskgiver:
Department of Numerical Analysis and Computer Science
Centre for Autonomous Systems
Abstract

The growth of the World Wide Web provides a unique opportunity to connect robots through the Internet. This implies that people all over the world can control them and monitor their operation. This project describes an intuitive user web interface, designed for inexperienced people to remotely operate mobile robots such as a robotics vacuum cleaner. The project also describes the first step toward building such an Internet-based interface with focus on technologies that most people have on their standard PCs (Personal computers). The system uses a standard network protocol and has an interactive human-machine interface. Using a web browser, a user can steer the vacuum cleaner at home from other places such as for instance at work. This report also discusses the different solutions to implement network connection.

Referat

Internetbaserat gränssnitt för fjärrkontroll av en mobil robot

Contents

1 Introduction ............................................................................................................. 1
  1.1 Problems ............................................................................................................. 1
  1.2 Trilobite – World’s First Self-propelling Vacuum Cleaner .................................... 3
2 Related Work ........................................................................................................... 7
3 Design ...................................................................................................................... 11
4 Components ........................................................................................................... 15
  4.1 User Interface ...................................................................................................... 15
    4.1.1 Tree Basic Types of Web Document .............................................................. 15
    4.1.2 To Make the Framework .............................................................................. 16
    4.1.3 Heavy and Light Components ..................................................................... 18
  4.2 Network Communication .................................................................................... 19
    4.2.1 Applet as a Special Kind of Java Program .................................................... 19
    4.2.2 To Run Java Applets Imply some Risks ....................................................... 20
    4.2.3 The Original Applet Sandbox ....................................................................... 20
    4.2.4 Trust, Applets and Application ................................................................. 22
    4.2.5 Can a Client Connect from an Applet back to an Unprivileged Port on the Server? ................................................................. 23
    4.2.6 Two-Way Talk with java.net – the Solution to Work Around Security restriction ................................................................. 25
    4.2.7 Signing Code ............................................................................................... 27
    4.2.8 What is the Main Goal of Code Signing? .................................................... 27
    4.2.9 The Java Cryptography Architecture (JCA) .............................................. 29
    4.2.10 Certificates ............................................................................................... 31
    4.2.11 Secure Communication ............................................................................... 32
    4.2.12 Signed Code ............................................................................................. 32
    4.2.13 Digital Signatures ...................................................................................... 34
    4.2.14 Key Distribution Key Distribution ............................................................. 36
5 Implementation ......................................................................................................... 38
6 Conclusions ............................................................................................................. 44

References

Appendix A Some Risks that Applets Introduce
Appendix B What Untrusted Java Code Can and Cannot do
Appendix C Differences between different Approach of Code Signing
1 Introduction

Some of our most influential technologies, the telescope, telephone, and television, were developed to provide knowledge at a distance. Remote-controlled robots were first developed in the 1940’s to handle radioactive materials. Specialists use telerobots to explore deep in sea and space, to defuse bombs and to clean up hazardous spills in nuclear power plant accidents such as in Chernobyl. Most of these robots are accessible only to trained and trusted experts. Allowing an operator at one location to control a robot at some other location is called teleoperation. Teleoperation permits the performance of physical work at a remote site under operator control [1].

Later the direct teleoperation interface to the remote robot was replaced with an interface via a computer workstation connected via a local network. All these lead to a slightly different system from the conventional telerobots system’s concept.

“When robots are connected to the Internet, each pair of an operator site and a robot still considered being a telerobots system. But the system now shares communication lines with other resources and every system has the potential to communicate with the other system on the network. These new systems connected to computer networks can be called networked robot systems” [2].

A new class of remote-controlled robots is now available to the general public that allows anyone with an Internet access anywhere to participate. At the same time Hypertext Transfer Protocol (HTTP) become available. HTTP can be a standard communication protocol of a telerobotic system since robots connected to the Internet can be accessed from any Internet site via the protocol. It also becomes possible to use various robotic hardware/software distributed on the Internet together.

1.1 Problems

Although the Internet provides available communication channels for teleoperation there are several difficulties in forming such web-based tele-control.

Those difficulties are:

1. The delay and the throughput of the Internet are highly unpredictable.

How does time delays complicate the control?
Consider the case where the operator notices that the remote robot is moving and is about to smash into a rock, for example, she reacts by putting on the remote breaks, but the signal does not reach the remote robot in time to prevent a collision.

Communication delays on the Internet can be lengthy and vary significantly. Problems with time delays are particularly challenging on the Internet, where packets are routed dynamically and are not guaranteed to arrive in order. That is why autonomy plays an important role for web-controlled robots. It is desirable that the interfaces should enable web users to sensibly control a robot even if the communication is highly unreliable.

2. Online robots are constrained by the physics of operating a mechanical device through the need to communicate via a low-bandwidth connection. Typical bandwidths for the feedback device over the Internet are 100—500 kHz, when it used locally. The Internet experiments shows that time delays can reach 400 msec, which would cause standard feedback control to go unstable [1].

3. Online robots are constrained by the need to interact with a human via a simple interface. Web users typically lack the technical education and skill for this. Graphical displays and easy-to-use command interfaces pose challenges on the design of web interfaces and information gathered by the robot (such as graphical maps of an environment). Thus they may not only be useful for the robot but must also be easy to understand by people on the Web [3].

4. Internet robots need an innovative mechanism for brokering control among multiple web users. This is particularly important for the web, where the number of potential “users” of a robotic system is enormous.

5. Since the web users are a central part of the control loop in Internet robots, their behaviour become an important consideration in the system design.

The aim of this project is to build a web based interface for controlling a domestic robot such as the Electrolux vacuum cleaner “Trilobite” at home while the users being geographically distributed at other places such, for example, at work. The users should easily understand the system design. The initial prototype should utilize 90% of the shelves
software and would focus on technologies that most people have on their standard PCs (Personal computers).

1.2 Trilobite—World’s First Self-propelling Vacuum Cleaner

The world’s first automatic vacuum cleaner was designed over several years at the household appliances giant’s facilities in Stockholm and Västervik. A prehistoric animal provided inspiration for the shape of this copper red round machine, see figure 1.1.

![Figure 1.1 Trilobite.](image)

“The shape and the name have been borrowed from the trilobite, a prehistoric animal that cleaned the bottoms of the oceans 250–560 million years ago.” Using a sonar system to navigate, four motors and sophisticated electronics, the Electrolux Trilobite will take care of the vacuuming all by itself. “The machine ‘sees’ the same way as a bat does,” says Lars Dahl, Technical Project Manager at the Electrolux Group. “The acoustic sonar vibrates at a rate of 60,000 Hz and is coated with a thin gold plate to give the best performance. The sonar’s semicircular shape allows it to see up to 180 degrees.”

How does the Trilobite work?

The maximum speed is 0.4 meters per second, and the Trilobite has no problem avoiding collision with anything you place on the floor – for example the dog’s water bowl, see figure 1.2.
Figure 1.2 *The Trilobite’s path.*

It will simply calculate a new path and continue to clean. Special magnetic strips are placed in doorways, near stairs and other openings. These act as a wall, keeping the Trilobite in the room.

Individual motors power the two wheels with independent suspension. This technical solution means that the vacuum cleaner can easily navigate over cables and the edges of rugs. Its ability to maneuver is further enhanced by its round shape and the fact that it is only 13 centimeters high and has a diameter of 35 centimeters. It is able to clean under really low furniture and beds.

The Trilobite scans the room before beginning to clean. When the automatic vacuum cleaner is started it first goes to the nearest wall, then follows along every wall in the room. While vacuuming this path, it scans the walls to create an inner picture of the room. The wall phase then forms the basis for calculation of the time required for cleaning the open spaces that follows. The sonar detects any obstacles on the floor and a new path is quickly calculated.

When the batteries run low the Trilobite automatically returns to the charging station to recharge. If the machine needs charging before it has completed the cleaning, it will automatically resume cleaning once it is fully charged.

The Trilobite has an LSD display where the user can choose between three different cleaning programs: normal, quick and spot vacuuming. A display shows the information with text and symbols, see figure 1.3 and figure 1.4.

*Normal vacuum cleaning* is the program which the user utilizes the most. The normal vacuum cleaning begins with the Trilobite searching the wall and then following the walls of the room from the left. The Trilobite goes one round about the room which takes 15 minutes (with the loader it takes 25 minutes). Thereafter it finishes following the walls and the cleaning is started with automatic time counting. See figure 1.3 a.
1.3.a. Normal cleaning program.

*Quick vacuum cleaning* is the normal vacuum cleaning program, without following the walls. The cleaning goes fast and time can be chosen from the timer menu minutes. See figure 1.3 b.

1.3.b. Quick cleaning program.

*Spot vacuum cleaning* should be used when the user only wants to clean a limited area. This area is about 1 m², with start point in the center. The area will be cleaned up two times. See figure 1.3 c.

1.3.c. Spot cleaning program.
The following symbols are:

- Normal cleaning
- Quick cleaning
- Spot cleaning
- Timer
- Battery
- Loader is found
- Dust magazine
- Clock symbol

*Figure 1.4* Symbols on the Trilobite.

*Timer:* The user can choose the time the normal and quick vacuum cleaning can take.

*Battery:* The indicator on the Trilobite shows actual battery state. See figure 1.5.

*Dust magazine:* The indicator shows when the dust magazine is full.

*Figure 1.5* The Trilobite’s battery status.
2 Related Work

Today, robots can be controlled by anyone on the Internet and that is why it dramatically extends our scope and reach. The robots include cameras that are being set up to continuously transmit live scenes from different and unpredictable places. Those cameras allow us not only to look but also enable us to control the telerobots movements and actions.

What is a Telerobot?

![NASA's Sojourner telerobot on Mars.](image)

According to Ken Goldberg [4] “A robot can be broadly defined as a mechanism controlled by a computer. A telerobot is a robot that accepts instructions from a distance, generally from a trained human operator. The human operator thus performs live actions in a distant environment and through sensors can gauge the consequences. Telerobotic systems date back to the need for handling radioactive materials in the 1940s, and are now being applied to exploration, bomb disposal, and surgery. In the summer of 1997 the film Titanic included scenes with undersea telerobots and NASA’s Mars Sojourner telerobot successfully completed a mission on Mars.” See figure 2.1.

The “Cambridge coffeepot” was the first networked device, which appeared on the Internet as early as 1980. It had a text-based Internet interface to soda machines and the first Internet camera which was set up by researchers at Cambridge University to monitor the status of a coffeepot. See figure 2.2. The appearance of the “Cambridge coffeepot” and the rapid growth of the WWW over the past several years have resulted in a growing number of telerobotics sites and web
accessible devices on the Internet.

Figure 2.2 The Cambridge Coffeepot.

The Mercury project, released in September 1994, was the first system to permit Web users to remotely view and alter the real world. The Mercury project consisted of an industrial robot arm with a camera and a pneumatic system allowing users to pick up and manipulate various objects in a sandbox filled with buried artifacts in the robot workspace [5]. See Figure 2.3.

Figure 2.3 The Mercury project: User Interface.

The Mercury project belongs to the 1st generation of Internet robotics systems. The telerobotic garden extended this idea to tending a garden with the arm, which allowed users to dig and water the plants [6]. See figure 2.4.

Other devices that have become available over time were based on robotic arms or simple robots that are directly controlled by human operators. They are the Bradford robotic telescope [7], a Web-based
tele-manipulator [8], the “Forty Two” telerobot at Manchester [9], the
VISIT telerobot system via computer network [10], an Internet-based
remote teleoperation [11] and the “MAX” wireless teleoperation [12]

The 2nd generation of Internet robots has begun to appear with
autonomous mobile robots. Reid Simmons and his team at Carnegie
Mellon University describe the first mobile and autonomous Internet
robot, Xavier. Xavier has been online since 1995. It accepts commands
to visit classrooms and offices, broadcasting images and jokes as it
goes [13]. See figure 2.5. The main research focuses on interactions
between remote users and autonomous robots.

After being online for three years Xavier’s operation have resulted in
collecting useful experience and statistics. We have learned from Xavier
that the mobility of the robot has both positive and negative impacts on web
interaction. The positive effect was that the user felt that controlling a mobile
robot remotely was a unique experience. Most of the effects of mobility on
connecting with the web are however negative. The need for wireless
communication limits the bandwidth of the robot, which lessens the interacti-

Figure 2.5 Xavier—the first Mobile and autonomous Internet robot.
the most severe effect of mobility on online interaction is a sociological one: users can see what Xavier sees, but they can not see Xavier itself.

The autonomy of Xavier has mostly positive effects on online interactions. Autonomy mitigates the effects of low bandwidth and unreliable communication. Since the robot is being asked to travel to discrete locations, high bandwidth interaction is not strictly necessary. Even if communication is lost completely, Xavier can still achieve its current task. None of the navigation components are affected by loss of communication, so the robots safety is not affected. The only real negative impact of autonomy on online interaction according to Xavier’s statistics, is that commanding at a high level (travelling to discrete locations) is not as interactive as teleoperation. Some users have expressed an interest in being able to choose an arbitrary location on the map for Xavier to go to. The implementation of future online robots can be guided based on this experience. Also according to the statistics from Xavier the most important result is the need for high-quality feedback.

“Since the original design, Xavier’s computational power has tripled and standardized low bandwidth mechanisms and protocols such as Java and Real Video have been developed and become ubiquitous. It is now possible, with a low computational overhead, to generate a continuous low-bandwidth, real-time video feed. Similarly, it is possible to construct dedicated Java applets so that map and position information can be displayed rapidly and efficiently” [13].

An important part of the feedback mechanism is a guestbook where the user can leave comments.

The other interesting projects from the 2nd generation are:

1. The Khep-on-the-Web, a small mobile robot that moves within a maze, and allows users to view either from the robot’s perspective or from an external camera to choose driving directions [14].

2. The Museum tour-guide robot [3].

The Museum tour-guide robot project has been gathering some interesting experience. The project describes three web interfaces to control tour-guide robots and their interaction with people. The specific robots studied here were deployed in two museums, the Deutsches Museum in Bonn and the Smithsonian’s National Museum of American History in Washington, DC. Both robots, Rhino [15] and Minerva [16], performed the function of a robotic tour-guide, leading people through the museums exhibitions. The control interfaces
provide various forms of instant feedback to the users such as information about the users interacting with the robot, the current goals and the current actions. The interfaces are based on Java applets and that is why they provide frequent updates of the robot’s state even over low-bandwidth connections. The interfaces deliver images taken with the robot’s cameras and they are able to serve several people at the same time by different types of brokering techniques. The important lessons that we have learned from the project’s experience are the following [3].

1. Since delay and throughput on the Internet varies drastically, the communication must be extremely low bandwidth. High initial start-up cost, inferred by the Java-based simulator that web users has to download, are preferable over high-bandwidth communication during operation (like periodically downloading GIF images to display the latest robot position).

2. If one wants more than a single person to use the interface, interfaces that support exclusive control may overly restrict the access to the robot. An interface that gives web users momentary control are more effective than voting-based interfaces, where the robot’s actions are chosen by the majority. The voting interface does not give individual users sufficient control to be truly excited; only few web users used this interface for robot control.

3. Interleaving web-based control, with control by the people that directly interact with the robot, is a tricky issue. The technique developed for Minerva turned out to be effective in proving the same service to visitors in the museum and to virtual visitors.

4. Placing all-important on-line information on a single web page is essential.

Additionally, there is a long list of mobile robots, which can be controlled over the Web.

In order to let individual users perform sequences of actions, more complex interfaces are required. This is because CGI (Common Gateway Interface) scripts limits web-based user interaction to a relatively small set of fields, menus and forms, and a single “submit” action using Java applets allows a client to provide both a more sophisticated interface and use its own protocol to communicate with the server. A typical example is the Puma Paint project, which allows people to draw a complete painting by controlling a PUMA 760 robot arm located at Wilkes University, Wilkes-Barre, Pennsylvania [17].
3 Design

Normally, the users interact directly with the Trilobite through its interface. So in order to control it at a distance, we need to build an interface to communicate with the user. See figure 3.1.

User

[Diagram of user connecting to internet and Trilobite]

Figure 3.1 Web-based interaction.

How can we replace the direct interaction users have with the Trilobite with a web based interaction?

We need to set up an interactive interface that represents the remote site, which is easily understood by the web users.

The idea is to easily mount a small camera on top of the Trilobite so that it can deliver moving images. Now we can follow the Trilobite working alive through the video stream. See figure 3.2.

[Diagram of interface with control panel, video picture, and Trilobite’s status]

Figure 3.2 Interactive interface.

To perform actions we need a control panel containing direction buttons or text fields for the steering of the Trilobite. We also need a panel to get feedback on the actions currently being carried out and the status of the Trilobite. See figure 3.2.

It is also possible to control the Trilobite through a graphical map of the home area (such as a floor plan). The map shows the Trilobite’s
current position and the user can easily choose the direction for the Trilobite robot to move to by pointing on the map. Users should easily understand the graphical map, but the implementation of the map may be complicated because different users need different maps.

It is clear that several users cannot control one robot at the same time. Therefore we need some exclusive control of the people that can manipulate the robot in order to avoid concurrency. It means that only one user can control the particular robot, while the other users might only be able to watch. To enter the robot’s web site a user first has to register her name (or whatever name she likes to assume). To accomplish this we need a login facility. Furthermore we want to have a feedback mechanism like a guestbook where users can leave their comments.

The final set up of the system consists of a server site and a client site as shown in figure 3.3.

![Diagram showing the final set up of the hole controlling system](image)

**Figure 3.3** The final set up of the hole controlling system.

The web browser in the client site represents the user who owns the Trilobite. A user anywhere on the Internet makes a connection through the Internet to the workstation that host the server. The server sends the command to the Trilobite through a wireless LAN. The response is sent back to the user and the Java applet running on the client site update the user’s site. A camera is placed on the Trilobite to give the user a global
view. It delivers video images continuously via a wireless link to a web server on the workstation. See figure 3.3.

The remote site is a server site and is not necessarily in a remote place or near the robot. All the clients and servers are connected to a central web server, and only need to know the location of the web server. With this architecture, we can either put all image service, robot control service and web service in one computer or put them in several computers and connect them with TCP (Transmission Control Protocol/Internet Protocol) sockets.
4 Components
This chapter covers which tools can be used in the project and how we can visually create the user interface. It also describes how we can communicate with the robot through Internet.

4.1 User Interface
This chapter describes which types of web documents and which tools that can be used in the project.

4.1.1 Tree basic types of web document
To build the user interface mentioned in chapter 3 we need to know what techniques there are to accomplish this.

In general, all web documents can be grouped into three broad categories [18],

- *Static*. A static web document resides in a file that is associated with a Web server.

- *Dynamic*. A dynamic web document does not exist in a predefined form. Instead, a web server creates a dynamic document whenever a browser requests the document. Because a fresh document is created for each request, the contents of a dynamic document can vary from one request to another.

- *Active*. An active document is not fully specified by the server. Instead, an active document consists of a computer program that understands how to compute and display values. Thus, the contents of an active document are never fixed – they can continue to change as long as the user allows the program to run.

Since the static method to build web document does not suit the project, there are only two ways to build the interface to control our robot.

A widely used technology for building dynamic web documents is known as the Common Gateway Interface (CGI). The most of the early World Wide Web (WWW) controlled robots used CGI to interface between the web browser and the physical device being controlled. A CGI program can be written in any language that allows it to be executed on the system such as:
A web browser in response to a certain HTTP (Hypertext Transfer Protocol) request launches CGI processes, and the CGI result is sent as the HTTP response. HTTP is a stateless request response protocol; it is simple to implement but it can have shortcomings. As it is stateless, the browser and the CGI process must manage the state. This is usually achieved by passing extra identification information with each request, known as cookies. The request response paradigm means that after a client request has been processed by a server, there is no way for the server to contact the client. The client must always initiate the contact. This can be a problem when a client is interested in the state of a no stationary remote process; to receive constant updates the client must poll the server at regular intervals. Polling is inefficient as requests must be made even when there are no changes and the server must handle these requests using resources to process them. Furthermore new information is only received with each poll, not as it becomes available.

This is why the chief advantage of an active document over a dynamic one lies in its ability to update information continuously. For example, only an active document can change the display quickly enough to show an animated image. More important, an active document can access sources of information directly and update the display continuously [18].

Based on what is mentioned above, an active document will been chosen to represent the interface that is used to control the robot.

4.1.2 Making the framework

What should we use to build user interface?

To create a static page the Hyper Text Makeup Language (HTML) serves to fill a request from the user. This can be passed to perform some predetermined actions in the server. A dynamically generated HTML page will return the results to the client. It can be done with CGI, but CGI has a number of limitations such as its slow response speed (the need to update the hole page for little change). Moreover, a complete HTML page must be generated with each request while the resulting page is still static.
In contrast, Java provides the capability to implement network connections and thus avoid the limitation of CGI. The introduction of Java is probably the most significant one compared to earlier existing technologies, because the Java technology is designed to make Java independent of computer hardware. Keeping an applet independent of underlying computer hardware is essential for three reasons.

- First, in the Internet, users have many different types of computers.
- Second, machine independence guarantees that the document will produce correct output on all browsers.
- Third, machine independence dramatically reduces document creation costs because a programmer can build and test one version of an active document instead of building one version for each type of computer.

The Java run-time environment includes facilities that allow an applet to manipulate a user’s display, and the Java library contains software that provide a high-level graphics interface. Together, the run-time graphics support and graphics library are known as a graphics toolkit; the specific graphics toolkit in Java is known as the Abstract Window Toolkit (AWT), which is always included. The AWT does not specify a particular style of graphics or level of interaction. Instead, the toolkit includes classes for both low-level and high-level manipulation [17].

Rather than being static a Java applet also enables an interface to dynamically change its content due to the fact that the Java applet is executable within a web page. However, there are also security restrictions associated with Java, such as Java applets can only connect to the host they are served from. The solution to this approach is coming later in the project.

In conclusion, the applets are focused on using the client platform which is a good choice to deliver dynamic user experiences. At the same time, developers have also investigated using the server platform for this purpose. Java Servlet technology was created as a portable way to provide dynamic, user-oriented content. With each client running a Java applet and communicating with a server, the system can be called distributed, but there are many issues associated with the design and implementation of a distributed system.

If the Java applet has been chosen to create the graphical interface to control the robot there are two ways to build the graphical design using either light components or heavy components.
4.1.3 Heavy and light components

As mentioned in section 4.1.2 Java code can be run in any virtual machine – no matter what platform it is, but if we build UI components native, using the AWT make the Java code dependent on the platform specific resources.

Java AWT components look different on different platforms, because UI components look like those of the proper platform[19]. One of the issues with the 1.0 AWT is that creating new components requires creating subclasses of java.awt.Canvas or java.awt.Panel. Those components are called heavyweight components and each new component owns its own opaque native window (commonly known as a peer). “A lightweight component is one that ‘borrows’ the screen resource of an ancestor (which means it has no native resource of its own – so it is ‘lighter’)” [20].

The heavyweight components are the original UI components. Each maps directly onto a native UI component, and the underlying system is responsible for drawing and controlling the native component. “Advantages in speed and look and feel result, but heavyweight components have disadvantages as well. The applications must limit themselves to components that are available on all platforms, and cannot control many aspects of the components behavior so the AWT has to struggle to maintain a consistent view across these varied platforms” [20].

Hooks have been implemented that enable the creation of ‘lightweight’ UI components; these hooks are called the Lightweight UI Framework. To create a lightweight component is simple and requires some steps of lightweight development.

- Each Java class, which is going to represent some component should directly extend the java.awt.Component and java.awt.Container classes. This allows components to not have native opaque windows associated with them.

- To represent the component on the screen all the graphical behaviour is needed to be done (all the action commands need to be written, like for example the mouse action).

- The components event handling should be provided.

Simply changing their super class appropriately can easily change existing subclasses of Canvas and Panel to lightweight versions.
Lightweight components can be freely mixed with existing heavyweight components. This means that lightweight components can be made direct children of heavyweight containers, heavyweight components can be made direct children of lightweight containers, and heavyweights and lightweights can be mixed within containers [21].

4.2 Network communication

This section describes possible solutions on how applets on communicate through a network. Since we need some panel to perform actions and describe the status of the Trilobite, we have to use the networking functions from the Java class libraries. In order to use them, it is important to understand the security concept Java uses, since networking functions are dependent on the security model.

4.2.1 Applets as a special kind of Java Program

The Java programming language is a high-level language which was developed by Sun Microsystems Incorporated. Java is designed for developing programs that run on many different kinds of networked computers. As long as a computer has a Java VM, the same code should run on all of the most popular platforms such as Macintosh, Windows, Linux and Solaris.

Applets can be attached to a Web page, which means that they are embedded into a Web page’s hypertext markup language (HTML) definition and executed by Java enabled browsers. See figure 4.1.

First a program is run which translate it through a compiler into an intermediate language; Java byte codes. After that the Java byte codes need to be interpreted. Every Java Interpreter is an implementation of the Java VM (Virtual Machine). Such Java enabled browsers automatically download and begin running any Java applet they find embedded in a Web page. Java byte codes help make “write once, run anywhere” possible.

The ability to send data that can be automatically executed anywhere on the net opens up many new possibilities on the World Wide Web. However, Java’s potential is mitigated by some concerns. Security is always an issue when computers are networked.
The ability to send data that can be automatically executed anywhere on the net opens up many new possibilities on the World Wide Web. However, Java’s potential is mitigated by some concerns. Security is always an issue when computers are networked.

4.2.2 To run Java applets imply some risks

Applets as mentioned before can be dynamically loaded over the network and run locally. Applets, which are started by pointing the browser at a Web page containing an applet is, not required advertising their presence. Java is designed so that the user will not need to worry about security restrictions.

Java provides applets loading over the network with a very restrictive “sandbox”. The Java sandbox model has been widely distributed to millions of users with their Web browsers and is today widely used. In most cases, Java itself can detect rogue applets and programs before they do any harm. But most of the security risks are new to users and system administrators. Appendix A shows the risks that applets introduce.

4.2.3 The original Applet Sandbox

The HTML in a Web page containing a Java applet specifies which code is to be fetched from the Web server. Then JVM loads the applet
and creates a sandbox for it. Figure 4.2 shows how to provide executable content for the Web with the help of the Java environment.

Figure 4.2 *Java provides applets loading over the network with the original sandbox [22]*.

1. The code is fetched from the Web. The applet then invokes the CL (Class Loader), loading all the necessary Java classes before passing the byte code to the BCV (Byte-Code Verifier). The byte code, which is created by the Java compiler, passes into the BCV.

2. Then it is time for verifying. Applet byte code must pass through the CL in order to make sure that it is valid Java code and meets the language specifications. The role of BCV (All Java code passes thought the BCV):

   • Ensure that all byte code adheres to Java language specifications.

   • BCV presents attack by scanning the byte code and locating potential security risk and poor language syntax. (An attacker can create byte code by hand, because applets are precompiled byte code.

   • Checks for name space violations and object access that would violate the Java security model [23].

3. The third step is instantiation of the code as a class or a set of classes in a namespace. Then the applets is being executed.
4. The applet invokes a dangerous method. When an applet performs it first consults the SM (Security Manager) for approval.

5. The Security Manager is consulted before the method runs.

The SM decides if the action is permissible based on the origin of the application or applet. Whenever a possibly dangerous function is called from within the applet or application, the SM grants or denies access to specific resources based on the origin of the application or applet.

The Security Manager performs runtime checks based on the calling class' origin and may veto some activities.

Some Web browsers, such as Netscape and Internet Explorer, can automatically download and execute Java applets when a user accesses an HTML Web page including the <APPLET> tag.

4.2.4 Trust, Applets and Application

A Java program has some level of trust. Programs that are more trusted can be allowed to carry out potentially dangerous acts. Programs that are less trusted would have their powers and permissions curtailed.

Applets that are embedded in Web pages are the most restricted and clearly treated as untrusted because the source of an applet is often unknown or unfamiliar. Java applications are trusted almost entirely without restrictions. This means that applications could use the complete power of Java, including potentially dangerous functionality.

With the introduction of Java 2, it has become possible to create and manage security policies that treat programs according to their trust level. This means that users may wish to grant more access to certain applets; by default all applets are untrusted and cannot access local information, such as network configuration and user information. The applets become partially trusted and can be allowed to, for example, read and write a particular file or make a network connection to a particular server.

Locally installed applications are trusted because they reside on the local hard disk drive.

Figure 4.3 illustrates the way in which the old applet/application distinction can be recast in terms of black-and-white trust.

It also shows the impact that Java 2 has on the black-and-white trust model, transforming it into a shades-of-grey trust model.
The distinction between applets and applications found to be useful during the JDK 1.0.2 days no longer applies to mobile Java code based on the Java 2 model. A black-and-white distinction between trusted code and untrusted code underlies both JDK 1.0.2 and JDK 1.1. Under Java 2, code can be constrained or unconstrained regardless of whether it is applet or application code. See more about what untrusted Java code can and can not do in Appendix B.

4.2.5 Can a client connect from an applet back to an unprivileged port on the server?

Since there is a need to write an applet that will run inside a web browser anywhere on the network, it is necessary to deal with those restrictions for untrusted applets. This imposes some problems, since we need to establish network connections to the robot’s server. Assuming that the robot’s server is not running on the host we loaded the applet from, we have a major design problem. Therefore we need to find a solution that work around those security restrictions according to Appendix B.

The networking restrictions do not only depend on the source the applet loaded from. There are also differences depending on the type of proxy server that might be located between the client browser and the web server as well as the actual implementation of the environment the applets are loaded into. Additionally, the restrictions do also depend on the kind of networking operation that is attempted. Connections to arbitrary network sockets are not routed through HTML proxies, therefore they are to be treated differently than so called “URL-
Connections”, which are used to fetch information from web or ftp servers and may transparently access HTTP proxies.

According to [26] table 4.1 summarises where untrusted applets are allowed to connect using java.net.Socket and table 4.2 summarises where untrusted applets are allowed to connect to when using java.net.URLConnection.

For the building of the panel’s applications, URL connections cannot be used. Therefore we need direct access to the robot server's port.

A test that has been made by University Western Australia (UWA) summarised in table 4.3 shows the results for 1.339 unique hosts that downloaded the test applet. Of these 87 percentage were able to make connection. The remaining 13 percentage showed a range of errors, most of which probably can be attributed to firewalls. According to them the cause of the “UnknownHostException” is not fully understood, as the client was able to resolve the host name to download the applet in the first place. It has been suggested that this is due to proxy configurations where the proxy handles all DNS requests; straight DNS lookups within the Java VM therefore fail.

The above test was carried out using TCP (Transmission Control Protocol). Some other tests were also tried, but there is a bug in the Internet Explorer virtual machine that makes receiving UDP (User Datagram Protocol) datagrams impossible in untrusted applets [27].

Table 4.1 Where applets are allowed to connect to when using Socket.

<table>
<thead>
<tr>
<th></th>
<th>Appletviewer</th>
<th>Netscape</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>No Proxy</strong></td>
<td>Depending on the setting of the appletviewer.security.mode property, you can connect nowhere, only to the originating host, or anywhere.</td>
<td>Can only connect to the originating host.</td>
</tr>
<tr>
<td><strong>SOCKS Proxy</strong></td>
<td>Same as no proxy, assuming you set the socksProxyHost property.</td>
<td>No connections allowed (except under OS/2, where it is the same as with no proxy).</td>
</tr>
<tr>
<td><strong>HTTP Proxy</strong></td>
<td>If the appletviewer.security.mode property is set to “none” then all connections are allowed; else no connections are allowed.</td>
<td>No connections allowed.</td>
</tr>
</tbody>
</table>
Table 4.2 Where applets are allowed to connect to when using URLConnections

<table>
<thead>
<tr>
<th></th>
<th>Appletviewer</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Proxy</td>
<td>Depending on the setting of the appletviewer.security.mode property, you can connect nowhere, only to the originating host, or anywhere.</td>
</tr>
<tr>
<td>SOCKS Proxy</td>
<td>Same as no proxy, assuming you set the socksProxyHost property.</td>
</tr>
<tr>
<td>HTTP Proxy</td>
<td>Same as no proxy, assuming you set the appropriate properties.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Netscape</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Can only connect to the originating host (=the web server).</td>
</tr>
<tr>
<td></td>
<td>Same as no proxy, assuming Netscape has been properly configured to use the proxy.</td>
</tr>
<tr>
<td></td>
<td>Same as SOCKS Proxy.</td>
</tr>
</tbody>
</table>

Table 4.3 Connections made from remote hosts.

<table>
<thead>
<tr>
<th></th>
<th>1162</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td></td>
</tr>
<tr>
<td>Failures</td>
<td></td>
</tr>
<tr>
<td>Java.net.UnknownException:</td>
<td>64</td>
</tr>
<tr>
<td>Java.net.NoRouteToHostExeption:</td>
<td>56</td>
</tr>
<tr>
<td>Operation time out</td>
<td>56</td>
</tr>
<tr>
<td>Java.net.ConnectException:</td>
<td>39</td>
</tr>
<tr>
<td>Connection refused</td>
<td>39</td>
</tr>
<tr>
<td>Java.net.NoRouteToHostExeption: Host unreachable</td>
<td>12</td>
</tr>
<tr>
<td>Java.net.SocketException</td>
<td>6</td>
</tr>
<tr>
<td>All</td>
<td>1339</td>
</tr>
</tbody>
</table>

There are some solutions to get around the security restrictions.

4.2.6 Two-way talk with java.net – the solution to work around security restriction

If there is a need for client-side Java programs to interact in a complex manner with the server-side Java programs there is Two-Way Talk with java.net technique that allow the applet and servlet to communicate. This solution is used here.

In this situation the applet does not speak directly with the servlet.

The servlet can create an applet’s server class during initialisation, but in this case we started the applet’s server first. This applet’s server class creates a Server Socket and listens for incoming connections from
the applet. When incoming connection is received, the applet’s server class hands off the data connection to a Robot server. This is possible since the Java application do not has to deal with security problems. The whole use of the applet’s server is completely transparent to the programmer and the user. It suffices to start the server application on the web server once, the applet can then automatically detect the TCP port that is used on the web server and all further communication will be relayed by it.

It is important that the applet’s server is on the same machine as the CODEBASE of the Panel applet.

Figure 4.4 shows the communication between the Panel Applet and the Robot Server.

![Figure 4.4](image)

*Figure 4.4 The network communication between the Panel Applet and the Robot Server.*

The Panel applet server can be used for arbitrary TCP ports. In this particular case the default port has been used.

According to Larry O’Brien in the article “Applet to Servlet Communication” [28] if you are using UDP rather than TCP, you will not have a chance of getting through the client-side firewall. Also, while it may be rare that a client-side firewall is configured to refuse outgoing data, it is not impossible.

The java.net technique is quite clean and fairly easy to implement. Since it relies on nothing more than Java’s standard network interfaces,
it is easy to modify for a particular need. On the other hand, you must parse the various requests for information coming in at the server side and then, on the client side, interpret the results coming back. If you have many server-side methods that you wish to use, you should consider using RMI (remote method invocation).

RMI is appropriate for situations where you have either dynamic or large data provided by Java objects on the server. Although RMI does allow for OO (object oriented) distributed programming, it is only supported when both client and server are written in Java. RMI is not sufficient when the application demands mixed language development. Therefore the CORBA (Common Object Request Broker Architecture) must be used.

### 4.2.7 Code Signing

Another way to solve the problem with security restriction is signing an applet and asking for the higher networks rights. The signing of an applet can be a future solution to run untrusted applets in a trusted environment. So far there exist no standards has for signing applets besides two separate inconsistent systems, which have been involved for each of the major browser in (Java 2 and JDK). Using the signing method can be complicated for the end user, because she will be confronted with complicated boxes, which can create some confusion. This can lead to users not choosing vacuum cleaning via the Internet. Java has outgrown the original restrictive sandbox. The anticipated future of mobile code security, a complex mix of sandboxing and code signing, is now upon us with Java 2.

Java’s approach to trust is also based on digital signatures. However, instead of allowing only black-and-white trust decisions, Java 2 allows fine-grained access control decisions to be made. With the introduction of code signing in JDK 1.1, Java’s sandbox model underwent a state transition from a required model applied equally to all Java applets to a malleable system that could be expanded and personalized on an applet-by-applet basis. Java 2 further complicates the picture with the addition of access control. The addition of code signing to Java complicates things. As it now stands, the Java sandbox has been reduced to a default.

There some differences between Netscape’s and Sun’s approach to code signing as you can see in Appendix C.
4.2.8 What is the Main Goal of Code Signing?

The main goal of code signing is to gain better control over the security of mobile code. By adding code signing and expanding beyond a black-and-white trust model, we can achieve three things:

1. The ability to grant privileges when they are needed.
2. The ability to have code operate with the minimum necessary privileges.
3. The ability to closely manage the system’s security configuration.

We can judge the JDK 1.1 and Java 2 security models by how well they meet these objectives.

The first objective is about to give trusted code the privileges it needs to get its job done.

The second objective is to have code that operates with the minimum necessary privileges at all times. This means that

- We want to grant each applet or application the minimum privileges it needs.
- Rather than assigning a given applet’s entire collection of privileges to all of its classes, we want each class to get just what it needs.
- We want a class’ privileges to be “turned off” except for brief periods of time.
- We even want to reduce the privileges of some of the built-in system classes.

The third objective is manageability. The users in the reality “are overwhelmed and irritated when they are confronted with too many big complicated dialog boxes” according to [22]. That is why the choices must be boiled down so that the users get just the control they need without being asked any unnecessary questions. This is an important issue in the project and it is still difficult to achieve in the reality with the signing code.
Some Security Enhancements introduced in JDK 1.1

JDK 1.1 introduces a totally new model based on trusted applets, which allows an applet to do anything. JDK 1.1 appeared in the early spring of 1997 and included a number of improvements and changes to the base Java security model of JDK 1.0.2. From a security perspective, the most important changes were the addition of authentication and simple access-control mechanisms that rely on the use of cryptography. The crypto API, also introduced with JDK 1.1, provides a basic toolkit of cryptography algorithms that developers can use in their programs.

4.2.9 The Java Cryptography Architecture (JCA)

Java technology includes a cryptography toolkit that in its includes both an API and several packages that provide classes useful for writing secure applications. The Java Cryptography Architecture (JCA) has three packages in JDK 1.1:

- Java.security
- Java.security.acl
- Java.security.interfaces

Classes in Java has two purposes

1. To provide the cryptographic methods that Java’s designers used to implement the JDK 1.1 and Java 2 security models.
2. To provide cryptography functionality to Java developers charged with creating secure applications. Parts of a crypto API were released with JDK 1.1. The parts included both one-way hash functions and digital signature capability.

JDK 1.2 modifies the existing security packages, adds the java.security.spec and java.security.cert packages and offers the Java Cryptography Extension (JCE) (javax.crypto.* which currently is available for the US and Canada only).

The java.security package consists mostly of abstract classes and interfaces that encapsulate security concepts such as certificates, keys, message digest and signatures [29].

The security package has both a user side and a provider side. Programmers use the user side to create applications and are using crypto algorithms from the provider side. Companies provide algorithmic implementation on the provider side.
One of the capabilities supported by the JCA are digital signatures. Digital signatures make it possible to authenticate who has vouched for a piece of code. If you decide to trust a particular person, things can be set up so that the programs that person signs will be automatically trusted with digital signatures. The sender of a message is authenticated so the recipient knows whom the message came from and that the message itself has not been altered.

Beyond digital signatures, the crypto API released with JDK 1.1 includes support for digital fingerprints or message digest or calculations used on the message content. Message Digest is one-way hash functions that provide a way to fingerprint a program or data so that you can verify that it has not been changed since it was created. Fingerprinting hash functions, such as MD5 and SHA, make distribution over the Net easier to swallow. If you are certain that a program you are downloading from the Net is the original program, you will probably be more likely to use it. Many archives on the Web today make use of MD5. MD5 and SHA are useful when it comes to signing code because the act of signing is actually a complicated function of a secret crypto key and the data to be signed.

Figure 4.5 How code is digitally signed (A) and how digital signatures are verified (B) [22].
Figure 4.5 shows the important role that one-way hash functions play in code signing.

A. Signing code takes several distinct operations:

1. A one-way hash calculation is run on a piece of binary code, resulting in a small “thumbprint” of the code.

2. The hash is signed using the signer’s private key.

3. The signed hash and the original binary code are placed together (potentially along with other signed and unsigned code) in an archive JAR. Now the JAR can be shipped around as mobile code.

B. Validating signed code also takes several steps:

1. A piece of binary code and its associated signed hash are removed from the JAR.

2. A new hash is calculated using the same one-way hash algorithm that the signer used to create the signed hash.

3. The signature carried by the signed hash is cryptographically validated with the signer’s public key (possibly with reference to certificate authorities and trust chains).

4. If the signature checks out, the now decrypted original hash is available for comparison with the new hash. Though all three Java code signing schemes (Sun, Microsoft, and Netscape) share these two processes, there are enough differences that the systems do not inter-operate.

4.2.10 Certificates

Another feature that appeared in JDK 1.1 is certificate technology based on the X.509v3 open standard. “Certificates provide an authentication mechanism by which one site can securely recognize another. Sites that recognize each other have an opportunity to trust each other as well” [22]. A secure socket layer (SSL) initializes connection between two machines, after they have exchanged certificates.

A certificate is a piece of identification (credential) which consist of a subject’s public key, detailed information about the certificate’s owner (such as name, e-mail address and so on) the subject’s name, the
certificate’s issuer, the issuer’s digital signature, an expiration date, and a serial number.

To ensure the integrity of the certificate, it is signed by so-called certification authority (CA), a trusted entity whose public key is widely known and distributed. There are a handful of companies that have set themselves up as CAs in the world. These include Netscape, GTE, Verisign, and a few others. Most CAs offers different classes of certificates. The level of verification used determines the trustworthiness and price.

### 4.2.11 Secure Communication

Java 2 now includes a package for secure socket layer (SSL) communication. Similar to Netscape’s SSL, the Java SSL provides a secure communications channel by using encryption. SSL works by providing a mechanism for encrypting packets on the sending end, sending them over an untrusted channel, and decrypting them at the receiving end. SSL is useful for many business applications, including the transmission of proprietary information and electronic currency. Most Web servers and browsers now support SSL, allowing a browser to communicate with a Web server without anyone else overhearing the conversation. An outsider might overhear a conversation, but she certainly will not understand it. Since SSL is commonly used over the Web, it can actually be used to protect virtually any sort of network transaction.

Most browsers support SSL by providing a Secure “HTTP Connection” service that looks just like a normal Web connection to the user, but uses SSL underneath. This allows you to reap the benefits of SSL without having to learn anything except the way the browser tells you whether a connection is secure. The encryption technology underlying SSL is generally believed to be secure, but there are two potential problems. First, the U.S. government restricts the export of strong cryptography software. If your browser version includes dumbed-down exportable cryptography software, your communications might not be as secure as you think. Second, SSL is good at providing secure communications, but it is not as good at establishing whom you are communicating with. This leads into all the problems of authentication and key distribution.

### 4.2.12 Signed Code

The capability to digitally sign Java byte code (at least byte code files placed in a Java archive, called a JAR file) was introduced with JDK 1.1 and greatly expanded with Java 2. Digital signing capability is an important part of the new Java security regimen. This is exciting
because digital signing radically alters the amount of trust you can place in a piece of code.

Before looking at creating and using certificates, it is however necessary to look at why they are necessary.

One particular kind of cryptography tool allows a chunk of digital information to be signed by a person or organization, see Figure 3.1. Since a digital signature has special mathematical properties, it is difficult to forge. Your browser can verify a signature, allowing you to be fairly certain that a particular person or organization vouches for the code. This means either you can instruct your browser that to always accept applets signed by some party that you trust, or to reject applets signed by some party that you do not trust. The same thing goes for a nonbrowser-based VM, which can be instructed (through policy) how to treat application code signed by particular entities. “Browser-level security for Java applets is based on a subclass of SecurityManager. The specific subclass depends on the browser” [29]:

- AppletSecurityManager: from package sun.applet with JDK 1.1 and HotJava.
- AppletSecurity: from package sun.applet with JDK 1.2.
- AppletSecurity: from package netscape.security with Netscape Communicator 4.0.
- StandardSecurityManager: from package com.ms.security from Microsoft’s Internet Explorer.

The SecurityManager subclass provides the browser-specific granting or denying privileges to the applet. Browsers have tended to restrict everything that can not be proven safe.

In general, you do not know from whom you are downloading an applet. This means that applets are prevented from opening sockets to destinations other than the IP-address from which the applet originated and applets are also prevented from creating a ServerSocket for listening on a port. Why? An applet would be able to spoof a service on the client end.

According to [22] a nasty attack called IP spoofing allows a bad guy to send you network traffic that claims to come from someplace else. For
instance, you might think the traffic is coming from “whitehouse.gov”, when it is really coming from “cracker.org”. IP spoofing used to be considered just a theoretical possibility, but it has actually happened in recent years. The best-known example is an attack by the infamous cracker Kevin Mitnick on a machine managed by computer security worker Tsutomu Shimomura. Mitnick’s attack led to his eventual capture and conviction [Shimomura and Markoff, 1996].

An attack known as Web spoofing shows that even in the absence of IP spoofing, it is not always clear that you are visiting the site you may think you are visiting [Felten, et al., 1997]. An attacker can lure you into a “false Web” that looks just like the real one, except that the attacker can see everything you do, including anything you type into a form, and the attacker can modify the traffic between you and any Web server. All of this is possible even if your browser tells you that you have a “secure” connection. See figure 4.6.

![Figure 4.6](image-url)  

**Figure 4.6** A Web Spoofing attack can be carried out with extensive use of a browser’s mobile code capability [22].

The Princeton Team has implemented a demonstration of Web Spoofing that makes extensive use of JavaScript. Once an attacker has lured the victim to the attack server (shown as www.attacker.org), the attacker can control the victim’s view of the Web by acting as a rewriting proxy. Clever use of JavaScript makes all changes invisible to the victim and can even appear to offer encrypted traffic.

Even if you ignore the possibility of spoofing, using the return address of an applet (that is, knowing the Web site where you got the applet code) still is not good enough to base a trust decision on. A digital signature holds much more information. For example, such a signature
could tell you that although if a site is redistributing the applet you do not trust, someone you do trust originally signed it. Or it can tell you that although the applet was written and distributed by someone you do not know, your friend has signed the applet, attesting that it is safe. Or perhaps it can simply tell you which of the thousands of users at aol.com who signed the applet.

4.2.13 Digital Signatures

So how do you sign a piece of code? The key to certification and authentication is the use of digital signatures. The idea is simple: to provide a way for people to sign electronic documents so that the signatures can be used in the same way we use signatures on paper documents. In order to be useful, a digital signature according to [22] should satisfy five properties:

1. Verifiable: Anyone should be able to validate a signature.

2. Unforgivable: It should be impossible for anyone but you to attach your signature to a document.

3. No reusable: It should be impossible to “lift” a signature off one document and attach it to another.

4. Unalterable: It should be impossible for anyone to change the document after it has been signed, without making the signature invalid.

5. No deniable: It should be impossible for the signer to disavow the signature once it is created.

The digital signatures used for Java code are based on public-key cryptography. This means that data gets encrypted at one end, sent through the network, and are decrypted at the other end.

In public encryption, two complex mathematical keys are generated together by a key generation algorithm, the private and public key. They are related only in the following way: anything encrypted with one key can only be decrypted by the other key.

*Private key*. Private key is used for signing electronic documents. The owner should keep it to itself and never circulate it.

*Public key*. Public key is used to verify those signatures. This key should be widely circulated.

Whenever someone wants to send data to us, they encrypt the data using a public key and send it to us. The data travels encrypted,
therefore nobody other than us can read it, and only we have the private key, so we are the only ones who can read the data that is actually sent.

How does one sign a piece of data (e.g. a jar file)? These are the general steps from the sender’s end:

- Using a well-known, one-way hash algorithm, hash the data into a short sequence of bits, called the digest (or message digest).
- Use our own private key to encrypt this digest (sometimes with some other short attached data) to create a signature.
- Attach this signature to the unencrypted data and send it.

At the receiver’s end.

- Use the sender’s public key to decrypt the signature and recover the digest. This authenticates the sender because only its public key can be used for decryption.
- Using the one-way hash algorithm, hash the data into a short sequence of bits to create a digest, and then compare this digest with the decrypted one.

4.2.14 Key Distribution

A common problem occurs when using digital signature technology: how does one reliably and securely obtain the public key of all the parties with whom we would like to communicate?

The certificate can be obtained from the CA (certification authority), which is mentioned above. The digital certificate is actually the public key of the individual and some other attached information signed with the CA’s private key. The certificate can be freely circulated by the entity using any means. Any third party receiving the certificate and wanting to communicate with the entity can do so by using the public key contained within the certificate. Because the third part already has the well-known CA’s public key, she can ascertain the authenticity and integrity of the entity’s public key before usage.

Figure 4.7 shows the process by which a signature on a piece of signed code can be validated.
In this example, the private key of thing1’s signs a piece of code. The corresponding public key, available on thing1’s certificate can be used to validate the signature carried by the code. For added security and to make key management more reasonable, browsers typically validate the CA signature carried on the certificate.
5 Implementation

The project client who gets downloaded over the Internet is written in the Java programming language as a Java applet. The server part of the system has two parts: the server part which responds to requests by the applet clients and the robot’s server, which is performing the actions on the Trilobite.

As shown in figure 5.1, users (web clients) access the system through the Internet. The user is coming first to a html page, which is served by a http server. The html page has static contents and therefore we can use forms for the user to choose some parameter to be sent to the applet and at the same time generate another page, which is going to communicate with the robot. Since the Java Servlet is widely used to generate a dynamic content, it has been used to pick up the parameter and, depending on this parameter, generate the new page with the appropriate applet view.

Figure 5.1 The system architecture.
The applet therefore initiates the network communication with the robot server to send some command to the Trilobite. There are however security restrictions associated with Java such as Java applets which can only connect to the host they were served from. To work around those restrictions we use the applet server to communicate with the robot server using TCP (Transmission Control Protocol) sockets. Each panel in the Applet can be redrawn separately from the others. The clock panel simulates the clock and therefore do not need connection to the robot server.

The following resources have been used to implement the whole structure.

- Sun’s 5.8 OS
- Apache HTTP Server 4.0.1
- Tomcat
- Java

The design of the interface was based on the original design of the Trilobite. Since of the end-user is a person who owns the Trilobite it was natural to assume that the end-user is already familiar with the objects and actions in the vacuum cleaner’s environment. The design of the graphical interface based on the original design of the Trilobite, simply replicates and portrays the objects on a different medium, the screen. The screen allows the end-user to work in a familiar environment and in a familiar way.

In the final program there are four programs to choose from; the original programs “normal”, “fast”, and “spot” cleaning, and a new one; “manual” cleaning. In the new program the user can steer the robot’s direction with the steer button in Figure 5.4. As can been seen in Figure 5.2 the user can choose a program and the timer from an html page.

Since the normal cleaning is used the most it becomes the default state. When the user loads the choose program page, the normal cleaning and auto timer are already chosen.

The user only needs to press the button “Start”.

The graphical attributes of icons such as shape and colour are very useful for quickly classifying Objects, elements, or text by some common property (Gittens, 1986).
This principle was used to design the 2-D control panel, which is placed at the left on the screen. The control panel consists of different subpanels. These are battery panel, time panel and dust magazine panel for normal and fast cleaning. The battery panel, dust magazine, steer panel are used for spot and manual cleaning See figure 5.3 and 5.4.

Battery panel. The colour indicates how much of the loading is left.
Time panel. Indicates how many minutes are left.
Dust magazine panel. Indicates whether the dust magazine is full or not.
Steer panel. It consists of a button to control the robot. You can drive left, right, back and forward. There is one button, which is both the stop button to stop an action and the start button to start an action.
Display. The display shows errors and commands.

Cursor action and motion occurs in physically obvious and intuitively natural ways.
The buttons on the control panel are green if they have not been pressed and red when they are pressed. The start button is green and the stop button is red.

According to [29] here follow some of the general principles that have been used in the design of the interface.

The General Principles:
Aesthetically Pleasing
• Provided meaningful contrast between screen elements.
• Aligned screen elements and groups.
• Colour used effectively and simply.

Clarity
• Visual elements should be understandable, relating to the user’s real world concept and functions.
• Interface words and text should be simple.

Compatibility
• The organization of a system should match the tasks a person must do to perform the job.
• The structure and flow of function should permit easy transition between tasks.

![Figure 5.2 The html page to choose a program from.](image)

**Comprehensibility**

• The system should be intuitive and understandable, flowing in a comprehensible and meaningful order. The steps to complete a task should be obvious.

**Consistency**

• The system look, act and operate the same throughout.
Figure 5.3 The normal cleaning program.

Control

- Control is feeling in charge, feeling that the system is responding to your actions.

Familiarity

- Employ familiar concepts and use a language that is familiar to the user.
Figure 5.4 The manual cleaning program.

Predictability

- Tasks, displays and movement through a system have been anticipatable through the user’s previous knowledge or experience. Screen elements are distinct and recognizable.

Simplicity

- Provided defaults. Providing defaults for all system-configurable items. The user will not be burdened with the decisions.
6 Conclusions

We have found that the World Wide Web can act as an excellent interface to a mobile robot such as the vacuum cleaner “Trilobite”. Using applets is a good choice because it makes the interface more interactive and is platform independent. The interface allows any Internet user to remotely steer their vacuum cleaners at home.

The original design have been built to allow a user directly interact remotely with her vacuum cleaner at home from other places such as at work. The interface has been built using applets, which are connected to the “Trilobite” control system via a TCP socket connection. Someone else has written the Trilobite’s control system earlier. The interface has four different programs to choose from. Only two of them are working so far because of a limited number of commands on the already written Robot’s server.

The interface provides to be easy to understand graphical representation of information acquired by the “Trilobite”. The interface has been showed to some other persons and they have easily understood it.

In this project we considered only a point-to-point connection, which means that it is only possible to connect to one robot server, and not the case where are many users with different robot servers (different IP-addresses). Future work might concentrate on the problem of making system that handles many users. At the same time a connection to one particular robot’s server may be solved by using a database with user IP-addresses where only one user with a particular password can connect to one specific robot’s server.

The proposed system’s central server architecture can look like in Figure 6.1.

![Diagram](image)

Figure 6.1 The system structure with more than one user.
All the clients and servers are connected to a central web server, and only need to know the location of the web server. All IP-addresses can be stored in a central database. Each user with a particular password can make a connection only to one particular robot server. With this architecture, we can put robot servers and web service in several computers and connect them via TCP connection.

In a future work it could be good to have some guestbook to leave the comments and some simple chat for user to communicate with each other.

As mentioned above, choosing Applet was a good choice, but there are still some problems to overcome. Choosing Applet as our tool makes us capable to deal with the security restriction. There are some technologies that allow us to make secure network connection and go around Applet’s security restriction through signing code, but there is no standard so far and it is complicated to agree on one. This implies that the end user must have technical skills, which should not be a requirement in the case of our project. The development of such technologies can bring new possibilities to provide an easier way to accomplish network communication in the future.
References

All Internet references last visited 1 December 2002.


[3] Dirk Schulz, Wolfram Burgaard, Dieter Fax, Sebastian Thrun, Armin B, Web Interfaces for Mobile Robots in Public Places, Cremers of Department of Computer Science 3, University of Bonn, Germany, Department of Computer Science, University of Freiburg, Germany, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA.


[9] U. Nehmazow, A. Buhlmeier, H. Durer and M.Nolte, Remote control of mobile robot via Internet, Dept. of Computer Science,


[18] Douglas E. Corner, Computer Networks and Internets with Internet Applications.


www.securingjava.com

http://www.networkcomputing.com/904/904ws22.html


http://java.sun.com/docs/books/tutorial/applet/overview/security.html

http://www.innovation.ch/java/HTTPClient/security.html

[27] Ken Goldberg and Roland Siegwart, A distributed framework for online robots by Barney Dalton in *Beyond web cams*.

[28] *Applet to Servlet Communication* by Larry O'Brien  


Appendix A

Some Risks that Applets Introduce

Malicious and badly written applets can be grouped into four categories, see table 1

- Attacks that modify the system.
- Attacks that invade a user’s privacy.
- Attacks that deny legitimate use of the machine by hogging resources.
- Attacks that antagonize a user.

The source [22] says that importance “of the four attack classes varies depending on the situation. For a lone user browsing on a personal computer, system modification attacks and privacy attacks are serious, but the other two classes are only a mild inconvenience. For big enterprise servers, however, attacks that antagonize employees and customers or deny them service altogether are serious issues.”
Table 1 The four classes in order of severity. Hostile applets often fall into more than one category according to [22].

<table>
<thead>
<tr>
<th>ATTACK CLASS</th>
<th>EXPLANATION AND CONSEQUENCES</th>
<th>JAVA DEFENSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Modification</td>
<td>Most programming languages give programs the ability to read and modify data on the system where they are running. Java includes predefined classes with methods that can delete and otherwise modify files, modify memory, and even kill processes and threads. System modification attacks comprise the most critical risks. In the most serious cases, system modification involves intrusion into the system itself. The most severe class of attacks. Applets that implement such attacks are attack applets. Consequences of these attacks: severe.</td>
<td>Strong</td>
</tr>
<tr>
<td>Invasion of Privacy</td>
<td>If you value your privacy, this attack class may be particularly odious. They are implemented by malicious applets. Include mail forging. Consequences of these attacks: moderate.</td>
<td>Strong</td>
</tr>
<tr>
<td>Denial of Service</td>
<td>Also serious but not severely so, these attacks can bring a machine to a standstill. Also implemented by malicious applets. May require reboot. Consequences of these attacks: moderate.</td>
<td>Weak</td>
</tr>
<tr>
<td>Antagonism</td>
<td>Merely annoying, this attack class is the most commonly encountered. Implemented by malicious applets. May require restart of browser. Consequences of these attacks: light to moderate.</td>
<td>Weak</td>
</tr>
</tbody>
</table>
Appendix B

What Untrusted Java Code Can and Can not do

Applets can be related to the traditional client/server model in a straightforward manner: The Web server is the applet's server. It sends the applet to the client machine. The client is the machine on which the applet eventually runs. That means when you are surfing the Web and come across an applet, your machine is the client.

When an applet is loaded from the Web the value of the CODE attribute in the APPLET tag is checking if it exists in the classpath. If no value was explicitly set, then the default classpath is used of the internal machine. If the value exists in the classpath that the applet found loaded and then they are considered trusted. If no applet by the requested name is found in the classpath then applets loaded this way are considered untrusted. Untrusted applets that have not been granted special privileges are relegated to the default sandbox. The sandbox implements the default constraints. If an untrusted applet has been loaded across the network and the default security policies are intact, the applet has the following security restriction [24]:

*Security restrictions*

- Read files on the client file system.
- Write files to the client file system.
- Delete files on the client file system, either by using the File.delete() method, or by calling system-level rm or del commands.
- Rename files on the client file system, either by using the File.renameTo() method, or by calling system-level mv or rename commands.
- Create a directory on the client file system, either by using the File.mkdirs() methods or by calling the system-level mkdir command.
- List the contents of a directory.
- Check to see whether a file exists.
- Obtain information about a file, including size, type, and modification timestamp.
- Create a network connection to any computer other than the host from which it originated.
- Listen for or accept network connections on any port on the client system.
• Create a top-level window without an untrusted window banner.
• Obtain the user’s username or home directory name through any means, including trying to read the system properties: user.name, user.home, user.dir, java.home, and java.class.path.
• Define any system properties.
• Run any program on the client system using the Runtime.exec() methods.
• Make the Java interpreter exit, using either System.exit() or Runtime.exit().
• Load dynamic libraries on the client system using the load() or loadLibrary() methods of the Runtime or System classes.
• Create or manipulate any thread that is not part of the same ThreadGroup as the applet.
• Create a ClassLoader.
• Create a SecurityManager.
• Specify any network control functions, including Content-HandlerFactory, SocketImplFactory, or URLStreamHandleFactory.
• Define classes that are part of packages on the client system.

Applet Capabilities

Here are some other things that applets allowed to do according [25]:

• Applets can usually make network connections to the host they came from.
• Applets running within a Web browser can easily cause HTML documents to be displayed.
• Applets can invoke public methods of other applets on the same page.
• Applets that are loaded from the local file system (from a directory in the user’s CLASSPATH) have none of the restrictions that applets loaded over the network do.
• Although most applets stop running once you leave their page, they don't have to.

Today Java 2 allows the user to give a digitally signed applet access to more local resources. In order for a signed applet to run on a client machine, the administrator can restrict access to an applet based on its digital signatures or author information. Signing is particularly effective in identifying valid applets because signatures are quickly recognized and may be stored for future access. Setting up and administering these policies is not a trivial exercise.
**Netscape Navigator and Microsoft Internet Explorer**

All Netscape Navigator versions subsequent to 2.0 are Java enabled. All Microsoft Internet Explorer versions subsequent to 3.0 also include Java. There is a toggle switch for the entire Java environment. In both Netscape Navigator and Internet Explorer, Java is enabled by default.

The two browsers’ security policies are, at the present time, very similar. Both are somewhat strict. The following rules apply to all untrusted applets running under Netscape Navigator and Internet Explorer:

- Applets cannot read or write files locally.
- Applets cannot open a client-side network connection to any machine other than the applet's origin host. Applets after JDK 1.1 can open a server socket as long as the port numbers is greater than the privileged port number on the machine (usually 1024).
- Applets can read only nine system properties. This allows an applet to access information such as the vendor who created the Java VM, the VM version number, the file separation character (either \ or /), the character used to separate lines, and so on. Applets are not permitted to read any other system properties.
- If an applet is loaded using the file: URL, and it does not reside in a directory in CLASSPATH, it is loaded by an Applet Class Loader.

There is no reason that all future browser vendors will choose to implement similar security policies. That two major vendors now do so is probably an artefact of Java's short history. Once browsers begin to implement different policies, security issues will become more complex.

There is a split among vendors happening already with the 4.x versions of Netscape Navigator and Internet Explorer. Both products offer complex security models based on digital signatures and partial trust. Although the models are quite similar to each other and to the model defined by Java 2, there are many detailed differences that annoy developers and users.

One side effect of the common ground between different vendors’ Java security implementations was that early on in Java’s history, security holes often cut across all Java implementations. That is, the same problem would be exploitable in all Java VMs, regardless of the
browser. These days, it is more common for errors to be relegated to one browser or another [22].
Appendix C

Differences between different Approach of Code Signing

There are a number of differences according to source [22] between Netscape and Sun’s approach to code signing and between Sun’s approach to code signing in JDK 1.1 and Java 2:

**Differences Between Netscape Object Signing and JDK 1.1.x javakey**

1. Netscape Object Signing only works within Communicator. JDK 1.1 signed applets can work in any browser, although Netscape Navigator and Microsoft Internet Explorer both require the installation of the Java Plug-In for the applet to leave the sandbox.
2. Netscape Object Signing requires getting a certificate from a certificate authority such as VeriSign. JDK 1.1 users can generate their own certificates.
3. Netscape Object Signing requires no modifications to HTML tags. If the Plug-In is needed for JDK 1.1 (in case you want to use IE or Netscape), the `<APPLET>` tag must be changed by HTMLConverter.
4. Netscape Object Signing uses Netscape’s own classes to step outside of the sandbox. A Netscape-specific exception is thrown when permission to leave the sandbox is denied. JDK 1.1 javakey-signed applets do not need to include calls to any other non-java.* classes to leave the sandbox, and java.lang.SecurityException is thrown when permission is denied.
5. Netscape Object Signing prompts the user when an applet attempts to leave the sandbox, asking the user for permission to carry out the dangerous act. Actions are grouped, so the user can allow some actions (file reads) but not others (file writes). JDK 1.1 javakey-signed applets that are trusted get complete access to the host.

**Differences between JDK 1.1 Code Signing and Java 2 Code Signing**

1. JDK 1.1 trusts code completely or does not trust it at all; Java 2 allows policy to define what code can and cannot do. This reflects the change from black-and-white trust to shades-of-gray.
2. JDK 1.1 has one tool, javakey, for all code-signing related functions; Java 2 has keytool for certificate management and jarsigner for signing and verifying JARs.
3. JDK 1.1 does not support certificates from Certificate Authorities; Java 2 does allow Certificate Authorities to sign generated certificates, however it is unclear if any CAs currently offer this service.

Both Netscape and Microsoft have provided browser-specific methods for leaving their sandboxes. Both rely on external Certificate Authorities to manage identities, but the same certificate used for Netscape cannot be used for Microsoft. Netscape requires applets to use special classes to take advantage of code signing. Microsoft also provides a vendor-specific API for certain capabilities. Both take a similar approach when it comes to prompting the browser’s user when certain applets attempt to leave the sandbox.

Sun has moved from a black-and-white security policy that allowed trusted code to do anything it wants to a shades-of-gray security policy by which only certain code from certain people can do certain things, depending upon the configuration. However, in Java 2, unsigned code can be granted free reign of the system as well if the policy is configured as such. Having unsigned code play outside the sandbox is something that none of the other schemes allow.

Each of the four Java code-signing techniques discussed in this thesis vary in their complexity level, have their own special tools for signing and key management, have different levels of support from VM to VM, and take different approaches to the user’s interface to security controls. Considering that Java is meant to be a portable, mobile code system, the large number of compatibility issues surrounding code signing is worrisome. Developers want their applets to do more than the original JDK 1.0.2 sandbox model allowed, but with each vendor providing different ways for code to leave the sandbox, the goal of “sign once, leave the sandbox anywhere” seems highly unlikely.