CHAPTER 2

LITERATURE REVIEW

This chapter discusses the concepts and technologies involved in secure network-centric application access. Some relevant past work is also described. Since, the aim of this thesis is primarily to create a secure environment for network-centric access of engineering software, most of the work that has been reviewed is associated with engineering and especially CAD/CAM.

Section 2.1 gives an overview of the problem associated with communication over an open network like the Internet. Section 2.2 describes the security services required for secure communication over the Internet. Section 2.3 gives an introduction to some of the basic concepts of cryptography that was used in the development of the prototype for this thesis. Section 2.4 provides an overview of some older technologies for network-based application access. Section 2.5 deals with the current technologies for Web-based application access. Section 2.6 describes some technologies for securing the network-based application access. Finally, Section 2.7 concludes the chapter with some observations.

2.1 INTERNET SECURITY

The Internet communication is based on TCP/IP. However, there is an inherent lack of security services in the TCP/IP protocol suite [Bellovin89] that exposes the Internet to attacks of various kinds, such as that of the "Berferd" incident [Cheswick91] and the "IP Spoofing" attack [Shimomura96]. Therefore, there are no guarantees of the end-to-end service between two Internet-connected computers. In such a scenario, communication compromise, or subversion of the communication line between two communicating computers, is a distinct possibility. In general, there are two classes of compromising attacks [Oppliger96]:

1. A *passive attack* threatens the confidentiality of the data that is being transmitted. The intruder can wiretap the communication line to capture and interpret the information that the transmitted data may represent.
2. An *active attack* threatens the integrity and/or availability of data that are being transmitted. Here, the intruder can observe and control the data that is flowing by, whether it be to modify, extend, delete, or replay data units. Often, passive and active attacks are used in tandem to effectively cause a communication compromise.

With the advent of executable contents like Java applets, ActiveX controls and Netscape plug-ins that can be downloaded from arbitrary servers on the Internet and executed on the local machine, new security issues arise. Here, the primary concern is that the downloaded content may be malicious and potentially modify or steal confidential data on the user's machine if it has unrestricted access to the local file system and network hosts.

With the issues discussed above in mind, it is easy to see that commercial use of the Internet for transmitting and sharing data, and allowing access to downloaded software, without any security measure installed, is not advisable. Thus, for a network-centric infrastructure for accessing remote application to be useful, it is imperative that the security concerns of the potential users be addressed.

2.2 NETWORK SECURITY SERVICES

To protect against communication subversion and compromise, a system based on internetworking with TCP/IP should, ideally, have the five classes of security services: authentication, access control, data confidentiality, data integrity, and non-repudiation [Oppliger96].

1. **Authentication services** are needed for: (1) authentication of communicating peer entities to verify that a peer entity in an association is the one that it claims to be; and (2) authentication of data origins to corroborates that the data received originated from the intended source.
2. **Access control services** are needed for protection of system resources against unauthorized use or access.
3. **Data confidentiality services** provide protection against unauthorized disclosure of data. Specifically, a third party should not be able to extract the information encoded in the data being communicated between two parties.
4. **Data integrity services** provide means of protecting the data from unauthorized modification.
5. **Non-repudiation services** provide some means whereby the sender of cannot falsely deny that a specific data was not sent, or in the case of the receiver, that specific data was not received.

2.3 CRYPTOGRAPHIC CONCEPTS

Most of the security services built for Internet communication are based on cryptography. Therefore, it is important to discuss some of the basic concepts of cryptography, like cryptographic algorithms, one-way hash functions, digital signature and public-key certificates. The following is a brief and relevant account of these concepts. A detailed discussion of cryptography can be found in [Schneier96].

2.3.1 Cryptographic Algorithms

There are two general types of key-based cryptographic algorithms: symmetric and public-key. Furthermore, these can be combined into hybrid cryptography system.

Symmetric algorithms (Figure 2.1), also called secret-key algorithms, are algorithms where the encryption key can be calculated from the decryption key and vice versa. In most symmetric algorithms, the encryption and decryption keys are the same. These algorithms require that the sender and the receiver agree on a key before they can communicate securely. Data Encryption Standard (DES) is an example of a symmetric algorithm.

Public-key algorithms (Figure 2.2), also called asymmetric algorithms, use different keys for encryption and decryption. The encryption and decryption key is often called the *public* and *private key* respectively. The decryption key cannot in any reasonable amount of time be calculated from the encryption key. The encryption key can therefore be made public. This way, anyone can use the encryption key to encrypt a message, but only a specific person having the corresponding decryption key can decrypt the message; hence the name “public-key”. Public-key algorithms are significantly slower than symmetric algorithms. Rivest-Shamir-Alderman (RSA) is an example of a public-key algorithm.

Hybrid cryptography system uses public-key cryptography to securely distribute *session keys*. Session keys are used with symmetric algorithms for relatively fast encryption/decryption. Most practical applications use hybrid cryptography because it solves a very important key-management problem, in that a session key is used only once per session and need not be stored, hence reducing the risk of its compromise. Pretty Good Privacy (PGP) is a popular example of a hybrid cryptography algorithm.

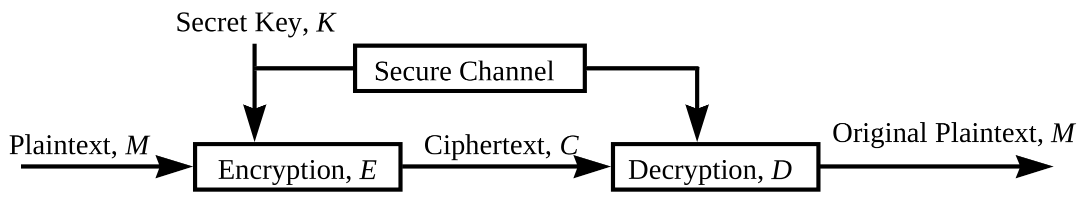

Figure 2.1 Symmetric cryptography.

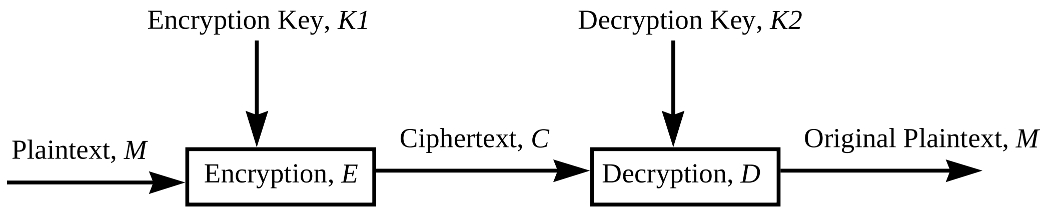

Figure 2.2 Public-key cryptography.

2.3.2 One-way Hash Functions

A hash function is a mathematical function that takes a variable-length input string, called a *pre-image*, and converts it into a fixed-length output string, called a *hash value* or *message digest*. A one-way hash function is a hash function that works in one direction; it is easy to compute a hash value from pre-image, but it is hard to generate a pre-image that hashes to a particular value. A good one-way hash function is collision-free; that is, it is hard to generate two pre-images with the same hash value.

One-way hash function can be thought of as a way of fingerprinting files. If two files produce the same hash values using a particular one-way hash function, then the two files are almost certainly identical. Normally, one-way hash functions are used to check the integrity of a message. Some popular one-way hash functions are SHA (Secure Hashing Algorithm), MD2, MD4, and MD5.

2.3.3 Digital Signature

Digital signatures (Figure 2.3) are used for authentication and non-repudiation of a message exchange. In modern cryptography, public-key cryptography is used for digital signatures. *Encrypting* a document using one's private key results in one's secure digital signature. The signature on a digitally signed document can be verified by *decrypting* the document with the entity's public key and comparing the resultant with the original document. In practice, the message digest of a document, rather than the document itself, is digitally signed for fast encryption and decryption. Digital signatures frequently include timestamps. The date and the time of signature are attached to the message and signed along with the rest of the message. This helps in countering a *replay attack* [Schneier96]. Some popular digital signature algorithms are RSA and DSA (Digital Signature Algorithm). While RSA can be used for both digital signature and encryption, DSA can only be used for digital signature.

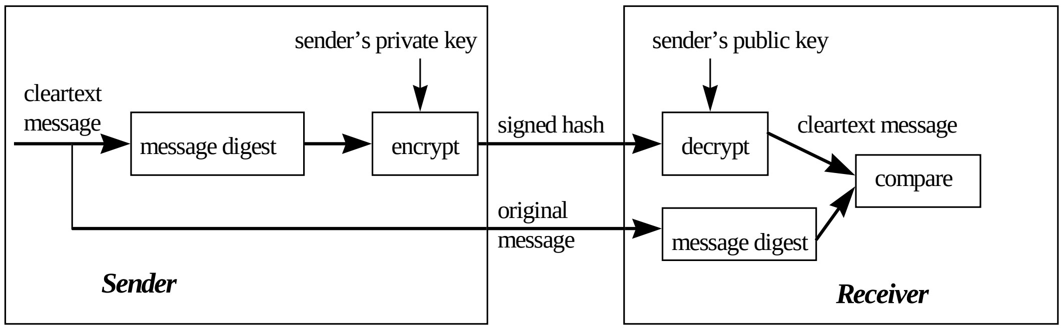

Figure 2.3 Digital signature protocol using public-key cryptography.

2.3.4 Public-key Certificates and Certification Authority

Open distribution of one's public key is susceptible to a *man-in-the-middle attack* [Schneier96], where a malicious entity substitutes his/her public-key for the communicating entities' public-key during the key-exchange, and thus is able to play foul. To guard against such attacks, an authentication framework is required that binds users' public keys and their identities. A *public-key certificate* is a certified proof of such binding vouched by a trusted third-party, called a *certification authority* (CA). In other words, a public-key certificate is someone's public key, signed by a trustworthy party. A user's certificate contains not only his public key but also information about his identity, and it is signed by a CA.

There is a complicated non-cryptographic issue associated with the public-key certification system. The system works in a "chain of trust"--a public-key certificate may bear the signatures of a number of entities, each "transferring" the trust to another entity. However, several questions arise. What does certification mean? Who is trusted to issue certificates to whom? What level of trust in someone's identity is implied by his/her certificate? It is argued that the CA paradigm relies on an authentication chain that ends in a CA that eventually certifies itself [Gerck97]. However, the public-key certificate and

CA paradigm has become *de facto* method of authentication of identity on the Internet. Today, it is widely used to authenticate signed Java applets in order to grant them access to certain local resources of the client's computer. The Java applet designed in support of this thesis follows this approach, and is signed by the author's Digital ID™ (public key signature certificate) obtained from Verisign, Inc.

2.4 NETWORK-CENTRIC APPLICATION ACCESS

Advances in computer networks are rapidly changing the way engineering is performed. Increasingly, the Internet is used to interconnect software, information and human resources. The World-Wide Web (WWW) has made a variety of information easily accessible for most users, both through static Hyper Text Mark-up Language (HTML) pages, and more recently, through live access to remote application programs.

Accessing software and applications in a networked environment, however, can be traced back to the 1950s and 1960s when multiple users could access software running on a mainframe from terminals connected to the mainframe [Krantz95]. The 1970s and 1980s brought workstations based on the UNIX operating system and the Ethernet local area network (LAN) [Goralski95], which shifted much of the computational load from the central mainframes to the local workstations. With workstations and LAN came the ability to distribute the computational load among several computers using remote procedure calls (RPC). RPC lets a process on a local system invoke a procedure on a remote system while hiding the details of sockets, network byte order, and the like from the client and server application programs [Stevens90]. It is typically used for running the application interface on a local workstation and the computationally intense part on a remote computing server.

Grieshaber et al. [Grieshaber91, Grieshaber92] describe a client-server model for integrating applications across the network. This model differs from RPC in that the applications running on different computers communicate with each other indirectly through an integration server. The server is responsible for translating the data between the formats of the communicating applications, using the transformation functions each application has registered with the server. The communication is through TCP/IP and UNIX sockets.

The above computing models are not without restrictions. The mainframe and workstation models typically limit the software to one operating system within a LAN. The direct (RPC) and indirect (Grieshaber et al.) integration models do not, in principle, have this restriction, but they require the maintenance of data exchange interfaces distributed across the network at the users' sites. Hence, while the applications can be maintained centrally, the customized associated application interfaces can not.

To enable easy access to remote applications, with centralized maintenance of both client (application interface) and server (application program) software, the client software must be distributed from the server to the client when it is about to be used, and the client software should not be tied to a particular operating system. Recent advances in the WWW technologies facilitate such access. Java applets are downloaded when they are needed and run within the JVM of a browser, in principle, independent of the operating system. Together, the applet and the browser can effectively form a system-independent application interface that is maintained centrally on a server, and that can access centrally maintained application programs running on the server. The following Section will review these WWW technologies in more detail

2.5 TECHNOLOGIES FOR WEB-BASED APPLICATION ACCESS

Several Web technologies have emerged that can be used for providing access to distributed applications. The most important technologies in this regard are (1) HTTP and CGI, (2) Java Sockets, and (3) CORBA/IIOP and RMI.

2.5.1 HTTP and CGI

The Common Gateway Interface (CGI) has enabled a class of specialized Web applications to exploit the Internet's interactive nature. CGI programs are often scripts which run on the server machine and produce output, usually in the form of HTML pages, that is displayed on the client's browser. The Hypertext Transfer Protocol (HTTP) and CGI are the protocols that govern the interactions between the clients, the server and the script. Under these protocols, a client sends a request to the server to retrieve a document or execute a script. The server responds and sends back a response containing the requested data.

HTML supports several ways of interacting with the user. These include HTML forms (which are fill-out forms for requesting general keyboard information), buttons, checkboxes, pull-down menus, etc.; and ISMAP, which is a feature of HTML that facilitates a graphical interface for accepting a user's mouse inputs. The data gathered from these user interface component are communicated to the server by using the encoding type "*application/x-www-form-urlencoded*," or in the case of arbitrary file upload, "*multipart/form-data*" [Saha97].

The HTTP/CGI method has been used in several Web-enabled engineering applications. Two examples are as follows:

(1) Wagner et al. [Wagner95, Wagner97] describe a system in which the user defines a polygonal part by filling out forms or by drawing a shape on a ISMAP graphical interface on a Web page, and this geometry is communicated to the server. The server runs a custom CGI program to send the request to another machine which runs a modular fixturing algorithm and generates the results as images in the graphics interchange format (GIF). The server then calls another CGI program to format the result and returns the solutions as GIF images in a dynamically created HTML page.

(2) Bailey [Bailey95] describes a system where users can submit .STL files to the server from a Web page. Encoding type of "*multipart/form-data*" method is used for file-upload via HTTP. The server runs a CGI program to checks the incoming files for robustness and sends them to a RP queue for fabrication on a RP device.

The HTTP/CGI method of application access has several shortcomings These are discussed as follows:

1. In this method, the result of one user interaction is often another HTML page that presents an intermediate result and prompt the user for the next interaction. However, a problem with HTTP is that it does not maintain a session between the client and the server. Each transaction comes in as a new connection, and the connection is closed once the reply is sent. The applications that need to maintain state information beyond a single interaction depend on storing this information in the client (i.e., the

browser) as a set of hidden input elements that are not displayed but that are retransmitted to the server with the next interaction. The need for repeated opening and closing of connections is rather costly and puts a heavy load on the server.

2. The stateless nature of HTTP can be a hindrance for some categories of Web based applications access which require a persistent connection. For example, an application that requires clients and server to communicate securely using cryptographic keys, would be difficult to manage using CGI/HTTP because of the lack of state.
3. Since the result of invoking a CGI script is returned as an HTML page, the HTTP/CGI method is mostly limited to applications where the result is textual or graphical.
4. Systems using these technologies do not take communication security into consideration, because, inherently, these technologies do not have network-security support in their framework.
5. HTTP/CGI method offers no client-side processing, which is discussed in the next sub-section.

2.5.2 Java Sockets

Java provides Web-based applications with two important access features: client-side processing and distributed computing. Client-side processing allows a server to delegate some of the responsibility for processing data to the client. Distributed computing helps in linking programs running on a number of computers distributed throughout a network, thus making it possible to access computation-intensive software, running on remote high-end machines, from low-end computers.

Java facilitates complex client-side processing in a safe environment by means of applets [JavaSoft98a]. Therefore, applications that do not need extensive computational power could be processed locally within a Java applet, while computationally-intensive tasks are processed on a remote server communicating with the applet through a TCP/IP socket connection using classes in the *java.net* package [Chan98]. This combined approach to computing opens up new possibilities for distributed applications access.

The distributed computing and client-side processing capabilities of Java have been used to develop several systems for Web-based distributed engineering. Smith et al. [Smith96] describe a Web-based design to fabrication system. Their system exists in a network of inter-operable software agents that are based on the Java Agent Template [Frost96]. Through these agents, interconnected CAD/CAPP/CAM components can be accessed over the Internet.

Puliafito et al. [Puliafito98] describe a network computing platform that lets Web users share applications not specifically devised for network use, including those that are computationally intensive. A Java applet is used to present a graphical user interface through which the users communicate the inputs to the application running on a remote server. The Java applet communicates with its server through Java sockets, in both sending out the input to and receiving the output from the server. The server, on receiving the input, invokes the application code to process the input. The output is returned to the client applet via the persistent TCP/IP socket connection. Signed applets [Javasoft98b, Javasoft98c] provide authentication to access local resources. The system uses server-side Secure Socket Layer (SSL) [Frier96] to securely distribute digital ID's to its users who must present their ID's to the applet in order to access the application on the server (see also Section 2.6).

This approach is in essence the same as the one taken in this thesis with the following two exceptions. In this thesis, the client software (Java applet) that runs on the users' end is subjected to fine-grained access control and the data transmission is secure. Puliafito et al. grant the applet full access to the local machine resources after evaluating the developer's signature certificate. In the NetCAD model, the applet can only access those local resources that the user specifically grants access to. Puliafito et al. also do not secure the data transmission but transfer the data in "plaintext". In the NetCAD model, the data transfer is secured through hybrid cryptography and a message digest. These differences are summarized in Table 2.1.

Table 2.1 A Comparison of different features of the NetCAD model and the Puliafito et al. model.

Features	NetCAD Model	Puliafito et al. Model
Method of communication	Java Sockets	Java Sockets
Method of application access	Java wrapper around application code	Java wrapper around application code
Applet authentication	yes (code signing)	yes (code signing)
Client-side access control	fine-grained (Capabilities model)	coarse-grained
Data encryption	yes (hybrid cryptography)	no
Data integrity checking	yes (message digest)	no

Another example of distributed application access using Java sockets is described by Becker et al. [Becker97]. Their Java based client-server system provides access to code that is distributed on different client computers for finding sensitivity derivatives of a decomposed complex system. These clients connect to the server to receive inputs and return intermediate results to the server which then processes the results to produce new inputs.

Other examples of Java based distributed application access are: (1) the NexusJava library [Foster97] that provides access to supercomputer resources from low-end PCs over the Internet; (2) the Javelin architecture [Bernd97] that runs parallel applications on numerous, anonymous computers communicating with each other through the Internet to create a multiprocessing environment; and (3) the Kansas Engineering and Science Interface (KESI) [Driggers97] for accessing different chemical engineering software tools, to support distributed collaboration for engineering applications using the Java-based environment called Habanero [Chabert98].

The Java sockets method for communicating information is better than CGI method because state is maintained and each request does not create a new process. Since the server remains connected to the client, the need for costly shutdown and initialization operations are obviated.

2.5.3 CORBA/IIOP and RMI

The Common Object Request Broker Architecture (CORBA) [OMG98] is a standard for distributed computing that has gained widespread acceptance. CORBA provides an infrastructure which enables invocations of operations on objects located anywhere in the network as if they were local to the applications using them. Together with the Interface Definition Language (IDL) and Application Programming Interface (API) that are defined by the Object Management Group (OMG), CORBA 2.x enables client/server object interaction and interoperability by specifying how Object Request Brokers (ORB) from different vendors can inter-operate. Simply put, CORBA provides an interface by which objects written in different programming languages and distributed on the network can invoke each other's methods in the same manner that they invoke their local methods.

The Internet Inter-ORB Protocol (IIOP) is based on CORBA and is implemented using TCP/IP. Using IIOP for Web application development has several advantages:

1. CORBA-based IIOP provides language transparency which allows development of server-side software in the language of choice. It therefore facilitates the integration of legacy code in the server-side software.
2. CORBA and IIOP provide location transparency. The network-level programming details are hidden from the user. Objects are invoked by other objects in exactly the same way, whether they are running locally or on remote host. This enables tighter integration of client and server software.

3. The roles of client and server can be interchanged. This provides a peer-to-peer architecture that scales well to support the development of distributed applications.

Remote Method Invocation (RMI) [Javasoft96] is an alternative approach to enable remote access to Java objects. Though, RMI is a Java-only solution, RMI and CORBA can be integrated since Java provides IIOP support for RMI. An example of CORBA/RMI-based application access is described in [Fan98]. In this, Fan et al. describe a distributed agent system where agents can invoke each other's method and pass messages interactively.

2.6 SECURE APPLICATION ACCESS

The previous section described different existing technologies for accessing remote applications through the medium of the Internet. However, to be useful in real-world applications, a secure environment that provides the network security services described in Section 2.2 is needed. Today, there are two technologies that are gaining wide-spread acceptance and that can be used to provide secure application access. These are Secure Socket Layer and Java Cryptography Architecture.

2.6.1 Secure Socket Layer

Originally introduced by Netscape, and now an Internet Draft of the Internet Engineering Task Force (IETF), Secure Socket Layer (SSL) defines a general security protocol for private, authenticated communication [Frier96]. SSL provides security services just above the TCP layer, but below application layer, using a combination of public-key and symmetric cryptography systems. The protocol can negotiate an encryption algorithm (from a suite of cryptographic algorithms) and session key, as well as authenticate a server (and optionally a client) before the application protocol transmits or receives the first byte of data. The SSL protocol provides a "secure communication channel" with the following three basic properties:

- The channel is private. Encryption is used for all messages after a simple handshake to define a secret key.
- The channel is authenticated. The server endpoint of the conversation is always authenticated, while the client endpoint is optionally authenticated. Certificates based on X.509 [CCITT87] are used for authentication.

- The channel is reliable. The message transport includes a message integrity check based on a message-digest.

SSL can be used in a number of ways. For example HTTPS, an implementation of HTTP over SSL, provides security service to Internet communication by HTTP. The OMG's document [OMG98] contains specification of the IIOP over SSL. The specification does not define an extension to the ORB which is needed for using IIOP over SSL, and it is left to the ORB vendors to develop products compliant with the specification. Besides these, there are freely available SSL implementations called SSLref from Netscape Communications and SSLeay which can be used to incorporate SSL functionality into arbitrary TCP/IP applications.

2.6.2 Java Cryptography Architecture

The Java Cryptography Architecture (JCA) [JCA97] refers to the framework for accessing and developing cryptographic functionality for the Java Platform. It encompasses the parts of the JDK 1.1 Java Security API related to cryptography, as well as a set of conventions and specifications provided in the JCA specification. It introduces a "provider" architecture that allows for multiple and inter-operable cryptography implementations. The Java Cryptography Extension (JCE) extends the JCA API to include encryption and key exchange. Together, it and the JCA provide a complete, platform-independent cryptography API.

The Java Cryptography Architecture (JCA) was designed around these principles: (1) implementation independence and interoperability; and (2) algorithm independence and extensibility. Implementation independence and algorithm independence are complementary: their aim is to let users of the API utilize cryptographic concepts, such as digital signatures and message digests, without concern for the implementations or even the algorithms being used to implement these concepts. Implementation independence is achieved using a "provider"-based architecture. The term Cryptography Package Provider ("provider" for short) refers to a package or set of packages that implement specific algorithms, such as the DSA or the RSA algorithm. A program may simply request a particular type of object implementing a particular algorithm and get an implementation from one of the installed providers. An example of JCA/JCE-compliant provider is the Cryptix-Java library from Systemics, Limited which has been used to develop the software in support of this thesis.

JCA is intended to be a general-purpose security provider architecture. In conjunction with JCA/JCE-compliant providers, it can be used to implement the different network security services for protocols that need to communicate over the network. It gives the developer the freedom to selectively incorporate only those services and algorithms that are required by the application, thus providing a great degree of flexibility in developing custom protocols.

2.6.3 Signed Applets and Access Control

Java applets are Java programs that travel across the network as part of a web page and run inside of the end-user’s browser. An untrusted applet, or an applet whose code does not bear a verifiable digital signature, is restricted to run in “sandbox” environment [Javasoft98c] within the browser. This means that the applet’s activity is very limited: it cannot access the local file system, it can open a network connection only with the machine from where it was downloaded, it does not have access to any hardware devices connected to the computer on which it runs, and so on. The sandbox model runs untrusted code in a trusted environment so that if a user accidentally imports a hostile applet, the applet cannot cause any damage.

However, this model restricts the applets severely and debars a number of useful applications where accessing local computer resources is necessary. To circumvent this problem, a *signed applet*, also called a *trusted applet*, is used [Javasoft98b]. A signed applet can be authenticated for its source and its integrity, and it can be subject to fine-grained access control of local resources.

To create a signed applet, the developer writes a Java applet and digitally signs the applet's bytecode files using the signature certificate he or she obtained from a CA, such as VeriSign, Inc. The digitally signed code is then bundled into a Java ARchive (JAR) file, thus creating a "digital shrink-wrapped" code. This JAR file consists of the applet code, the message digest of the code digitally signed with the developer's private key and the developer's signature certificate. The signature certificate, which is included in the applet's JAR file, can be used by browsers which recognize the signature certificates from the certificate's issuer (CA) to verify the integrity of the code, by checking its digital signature and the identity of the entity who signed the code.

Netscape Communicator 4.01 and later releases have defined several extensions to the Java security model, by which the developer can exercise fine-grained control over a signed applet's activities beyond the "sandbox". These extensions are called Capabilities classes™ [NCC97] and they are based on the Capabilities model of access-control. All access-control decisions boil down to who is allowed to do what. In the Capabilities model, a *principal* represents “who,” a *target* represents “what,” and the *privileges* associated with a principal represent the authorization, or denial of authorization, for a principal to access a specific target. A principal is typically represented by a signature certificate and a target is typically a system resource. Thus, the developer designs the applet to “request” additional access privileges on developer’s behalf (through his signature certificate), while the user who downloads the applet has the ultimate veto power; he or she can grant or deny the applet’s request, depending upon his or her level of trust of the developer.

The Capabilities model for access-control is based on Java stack-inspection. In this approach, the Java runtime system exports an interface to allow the security-checking code to examine the runtime stack for frames executing untrusted code, to enable security decisions to be made at runtime based on the state of the stack [Felten98]. The Java Development Kit™ 1.2 (JDK 1.2) also uses the Capabilities model of access control [Gong98]. However, at the time of this writing, JDK 1.2 was still in beta-testing phase and the Netscape’s Capabilities Classes gave a more complete implementation. Wallach et al. [Felten97] describe some other approaches to software-based access-control to achieve secure execution of downloaded code.

2.7 OBSERVATIONS

Significant progress have been made in the areas of network-based application integration, remote procedure calls, and WWW-based application access. There are several similarities between this work and that presented in this thesis. For example, like Grieshaber et al. [Grieshaber92] and Puliafito et al. [Puliafito98], the NetCAD system presented in this thesis provides network-based access to application program using sockets.

However, past efforts in this area have not fully addressed secure network-based application access. This has been due to the absence of a definitive standard for providing secure Internet communication, and that most WWW-based application access technologies have not been developed with security in mind.

The emerging open standards for WWW-based application access (CGI, Java sockets, and IIOP/RMI) and for network security (SSL, JCA, and code signing) now provide the means for developing secure application-access models, an example of which is presented by this thesis.