The Gateway Security Model in the Java Electronic Commerce Framework

Theodore Goldstein

Chief Java Commerce Officer

Sun Microsystems Laboratories / JavaSoft

ted.goldstein@eng.sun.com

This paper describes an extension to the current Java security model called the "Gateway" and why it was necessary to create it. This model allows secure applications, such as those used in electronic commerce, to safely exchange data and interoperate without compromising each individual application's security. The Gateway uses digital signatures to enable application programming interfaces to authenticate their caller. JavaSoft is using the Gateway to create a new integrated open platform for financial applications called Java Electronic Commerce Framework. The JECF will be the foundation for electronic wallets, point of sale terminals, electronic merchant servers and other financial software. The Gateway model can also be used for access control in many multiple application environments that require trusted interaction between applications from multiple vendors. These applications include browsers, servers, operating systems, medical systems and smartcards.

1. Electronic Commerce in Java

Sun Microsystems Inc.'s Java programming language and APIs provide a reliable, secure platform to deliver applications on Internet and Intranet networks. Without Java, users who downloaded applications from these networks could be vulnerable to dangerous software viruses. All of the popular personal computer operating systems are vulnerable to such attacks. Of course, Java executing on conventional personal computers cannot totally prevent virus attacks. But if users execute care and only load Java software and software from totally reliable sources, then they lower their chances of software virus infection. The ultimate solution is an entirely Java operating system running directly on a processor. JavaSoft is currently developing JavaOS and Java chip devices that will ultimately eliminate viruses on personal computers.

Java provides safety and security that by restricting the use of potentially dangerous operations. Unfortunately, many beneficial applications, such as electronic commerce applications, are also prevented from executing these operations. The limitations of Java motivate a new security model called the Gateway that extends the current Java security model. This new model is the core of an entire application framework for financial applications called the Java Electronic Commerce Framework (JECF). This paper reviews the requirements for electronic commerce and the existing Java security model. The paper then describes how the JECF and the Gateway model meet the requirements of electronic commerce and improve the Java security model by providing:

- a common graphical user interface

- a secure encrypted database

- an interface to strong cryptography

- a structure for purchasing applications

2. Internet Electronic Commerce Requirements

Internet electronic commerce has many demanding security requirements:

- Transactions must be private. Only the intended participants in the transaction may know the details of the transaction.

- Transactions must be authentic. The identities of the buyer and the seller have to be true and accurate. Both participants in the transaction must agree to the terms.

- Transactions must be auditable. Auditing detects fraud and mistakes. Auditability must be managed against the needs of users' right to privacy. Auditing also assists in tuning efficiency.

- Financial transactions must obey the law. It is very difficult to define the venue of Internet electronic commerce transactions. A transaction that is legal at the seller's location may be illegal at the buyer's location. Taxes must be paid at one or both ends of a transaction. Cryptographic control laws must be obeyed. Consumers will want privacy. All of these requirements are in conflict. World governments will be spending many years learning to cope with the conflicting demands of the Internet.

- Existing systems must be extended to accommodate new applications. The financial industry and the computer industry continually develop new payment instruments, financial services and technologies. Users can only take advantage of these new applications if the infrastructure can accept change.

- Payment services must cooperate. Just as with physical world commerce, Internet customers want to use cash, checks, credit, debit and discounts to buy things. Therefore multiple payment mechanisms must work together.

- Financial services must cooperate. Few customers use just one bank or broker, but instead have relationships with many financial institutions and many merchants. Financial goal analysis and government taxation require a total and complete integration of a customer's entire financial picture. Financial applications must communicate to integrate a financial portfolio. Trusted third party financial applications can greatly assist users with managing their finances and reaching their financial objectives. Therefore, even home banking and broker systems must work together.

Current software protocols and conventional financial applications do not yet satisfy these requirements. Financial technology companies and institutions have made great strides in electronic commerce protocol and application design, but there is still need for significant improvement to allow the Internet to meet users' requirements for purchasing and financial services.

By employing cryptography and digital signatures, protocols such as Secure Electronic Transactions (SET)[MV96], Secure Key Internet Protocol (SKIP) and Secure Socket Layer (SSL) enable privacy and authenticity of network

communication. These protocols, however, do not enable inter-application communication, cooperation or auditability.

The only interesting development in the area of inter-application cooperation is the Joint Electronic Payment Initiative (JEPI)[JEPI97]. The JEPI protocol negotiates payment protocol selection. JEPI does not specify, however, how the user is to acquire new payment protocols. Though CommerceNet's first implementation of JEPI is written in Java, JEPI itself does not use Java's dynamic download ability to install new protocols and applications.

Today, when users want to add functionality to a consumer financial application, they have to discard the existing application. As more and diverse applications appear on the market, discarding applications and reentering data ceases to be a viable technology migration strategy. End users need a system that enables data sharing between applications from different vendors.

Soon, conventional home banking software, home broker software, and electronic wallets will cooperate. To cooperate these packages must share data with each other. These packages should not try lock the user into a single application because customers will choose home banking packages that communicate with their home broker software and their electronic wallets.

2.1 Limited Trust

To understand the need for the Gateway model, consider the trust relationships in business. All business relationships have defined limited trust. By definition, businesses in a relationship do not have complete access to all aspects of each party's affairs (Such devotion is reserved for relationships of the heart). You trust your lawyer for some procedures, and your doctors for others. Every business-to-business contract specifies roles, responsibilities, rights and deliverables among the participants. Sometimes rights are transferable, but often they are not.

The business community uses a broad variety of control mechanisms to enforce contracts. Employees, security guards, access badges, locks, and laws provide infrastructure for the physical world to enforce and limit contractual rights. These mechanisms are not infallible. It is still possible to break these mechanisms and violate the contract. When these physical world mechanisms fail, government and the court system mediate the disputes. To quote an old saying, "Locks keep honest people honest; laws discourage dishonest people from being dishonest."

The electronic commerce community needs an equivalent set of software locks and infrastructure to keep honest people honest, and to discourage dishonest people. Implementing electronic commerce requires a comprehensive suite of functions such as public key operations, database transactions, smartcard exchanges, and elaborate protocol communication. Without software protection mechanisms, rogue software can steal money from users. Even legitimate software may inadvertently violate the contractually defined privacy of a user's financial portfolio or other records. The Gateway model allows for such a lock to be placed at the point of call of an application programming interface. This means that

software from different vendors can cooperate in the same address space using an API calling mechanism and without compromising each application's security integrity.

2.2 Example: Home Broker and Tax Reporting Software

Consider the communication between tax reporting software and home broker or banking software. It would be useful for a tax program to import be able to capital gains information from a broker software's database. The problem is that the broker's database contains information that is none of the tax program's business to know, such as a portfolio information advisor. There are many separate roles involved in having access to the client's investment database: a transaction broker role, a tax report role, a financial advisor role, etc.   The software that implements each role should have access to only the data it needs to fulfill that role. The software should not have access to data used by roles it does not hold. This principle is common in the physical world. For example, unless your accountant is also your doctor, your accountant does not have access to your medical records without your explicit permission.

None of the current common brokerage software allows any tax software to read its database or any part of its database. There are both business and technical reasons for the lack of this feature. The business reason is that the broker software must protect the privacy of the customer's portfolio. The technical reason is that the broker software's authors cannot limit the tax software authors to having access to only the capital gains information. Hence the need for Java and the JECF.

Before Java, it was impossible to have reliable separation between applications in the same address space. Developers had to keep code for application components that do not trust each other in separate address spaces. Many programmers think the word application is synonymous with separate address spaces. Unfortunately, separate address spaces limit the ability of applications to cooperate. Inter-address communication mechanisms are never as efficient or as convenient as intra-address space communication. Applications in the same address space cooperate by sharing data and objects. Applications in different address spaces can communicate only by copying or by using one of the new distributed object model systems such as CORBA[OMG95]. Distributed objects may be an appropriate technology for some commercial Intranet systems, but CORBA systems require a large infrastructure. Small CORBA systems tend to occupy many tens of megabytes of size for small applications and hundreds of megabytes for large ones. In contract Java's design is much more compact.

Java provides the appropriate technology for deploying tight integration between applications while maintaining application data integrity. The JECF builds on Java's inherent extensibility and adds the ability to safely share some information while keeping other information secret. Applications can precisely define the limits of their trust of the data and behavior that they extend to other applications.

3. Review of the Java Security Model

The Java language[J1][Gosling 96] is a portable, safe, object-oriented language. The safety properties of the Java language and the runtime security model of the Java environment, called the Java Sandbox security model, provide safeguards that inhibit inappropriate activity. The Java application environment has an extensive application framework that includes a set of common utility classes, a portable window system, database interfaces, and many other useful reusable classes that enable programs.

Many web browsers contain Java implementations. Java-enabled browsers support a unique application type called an applet. It is easy for web page authors to embed applets into web pages. When the browser loads the web page, a Java-enabled browser also loads the implementation of the applet into the browser and executes the applet's instructions. Without appropriate safeguards, viruses could use the dynamic loading mechanism to damage the host system. Java's safeguards allow applications from unknown sources to download and execute safely.

3.1 Safety Properties of Java Programming Language

Java is similar to other strongly typed object-oriented languages such as C++. C++ is designed for systems programmers. Unlike C++, Java prohibits certain language loopholes that might damage type safety. System programmers use C++ because it provides them with necessary unsafe features to implement device drivers, memory management systems and other low-level features. The unsafe features of C++ increase the expressive power and efficiency of the language, but sacrifice reliability and portability. For example, both C++ and its predecessor C, have a feature called a cast. A cast allows a programmer to write an expression that tells the compiler to treat an integer as a pointer, or treat a pointer to one type as though it were a pointer to another. This feature breaks type safety, but low-level system programmers often need this feature to efficiently implement many systems. In contrast, Java is designed for application programmers and does not allow such unsafe designs.

3.2 Isolation Properties of Java

The Java language sacrifices some of C++'s expressive power and efficiency to get greater reliability, safety and tighter inter-application coupling. There are no unsafe casts in Java. All references to objects conform to the type system of the language. Type safety can never be broken. This allows multiple Java applications to run in the same address space. Java applications running in one address space may actually run faster than those C++ applications that must be run in two address spaces.

The Java virtual machine enforces safety, privacy and isolation rules. Instead of using address space separation between applications, Java access modes protect unauthorized access and isolate one application from another. The Java type system has explicit visibility declaration rules that define three access modes: public, private, and protected.

The idea behind Java's access comes from C++ access modes. But the underlying object model of C ++ does not enforce access modes. Unlike Java objects, C++ objects are simply bits in memory. Adroit programmers exploit the memory model of C++ and break the type safety of the language both for good and malicious purposes. For example, to falsely create as an object, a programmer merely sets an array of characters to suitable value and casts it into the object's type.

Unlike C++, a Java object must emerge only from a new operation. The new operation invokes a special method called a constructor. The constructor's access mode controls object allocation. The constructor also ensures that the object is initialized to a known value. Thus if the constructor is not public or protected, another module cannot create the object. Unforgeable objects provide the basis of security in Java.

Another safety loophole of C++ is that pointers may point anywhere in memory, even into the interior of objects. C++ arithmetic pointer operations allow a program to move the target of a pointer to any location in memory. Additionally, there are no bounds checking on arrays or checks against referencing outside the boundaries of an object. This obliterates any of the security properties of C++ access modes.

It is also trivial in C++ to cast an integer into a pointer and access any location in memory. Even the dynamically-typed language Smalltalk-80[GR83] has a loophole operation called asOop that allows a programmer to transform an integer into an object.

In Java programming language has no operations that allow pointers to break access mode safety. All accesses to objects are bounds checked. There is no language mechanism for forging an integer into an object reference. Array dereferencing is either statically or dynamically checked. Object dereferencing must occur only in the boundaries of the object. It is not possible in Java to turn an integer value into a pointer.

Using the safety model of the language as a basis, it is now possible to have multiple isolated applications residing in the same address space. The applications may communicate as specified by the type system of the Java language. Java makes it possible to isolate one application from another. Isolation allows Java to have the next tier of security called the Java Sandbox security model.

3.3 The Java Sandbox Security Model

The Java Sandbox security model [Mueller96] controls dangerous operations such as writing to users' disks or network communication. The Sandbox security

model is intended to evoke the notion of putting a child in a sandbox. The child cannot get hurt, nor can they wander outside of the sandbox and cause any damage.

The Sandbox security model divides applets into untrusted applets and trusted applets. Untrusted applets may access limited system resources. Trusted applets have greater access to system resources, but still must obey the language safety rules as described above. Trusted applets are digitally signed. The browser has a list of digital signatures of trust authorities. The Java implementation consults the list at applet load time to determine the correct security manager to use for the applet.

The security manager is an object that is a subclass of class SecurityManager. When an applet performs a dangerous system operation, the implementation of the operation consults the applet's security manager. Potentially dangerous operations include opening a file or a network socket, changing a system variable, and setting the security manager. Each method in the security manager either allows the operation or throws a Java language exception to prohibit it.

In many browsers, the security manager for untrusted applets limits the effect of applets by denying all dangerous operations. The only exceptions are that an untrusted applet may make a socket connection to the applet's originating network site.

There are untrusted applets and trusted applets, but no partially trusted applets possible. So, the Java Sandbox supports single applets, but it does not support interaction between applets that do not completely trust each other. Since limited trust applets are required for electronic commerce, the new Gateway model was a needed component of the Java Electronic Commerce Framework (JECF).

4. Framework and Responsibilities

Many object-oriented practitioners use the term "framework" to mean an "application programming interface." In this paper, the term "framework" refers to a set of APIs that impose responsibilities amongst participating software packages. The JECF defines application responsibilities for merchants and financial institutions. JECF provides services, such as database services, to merchant and financial institution applications. These layers are depicted in Figure 1.
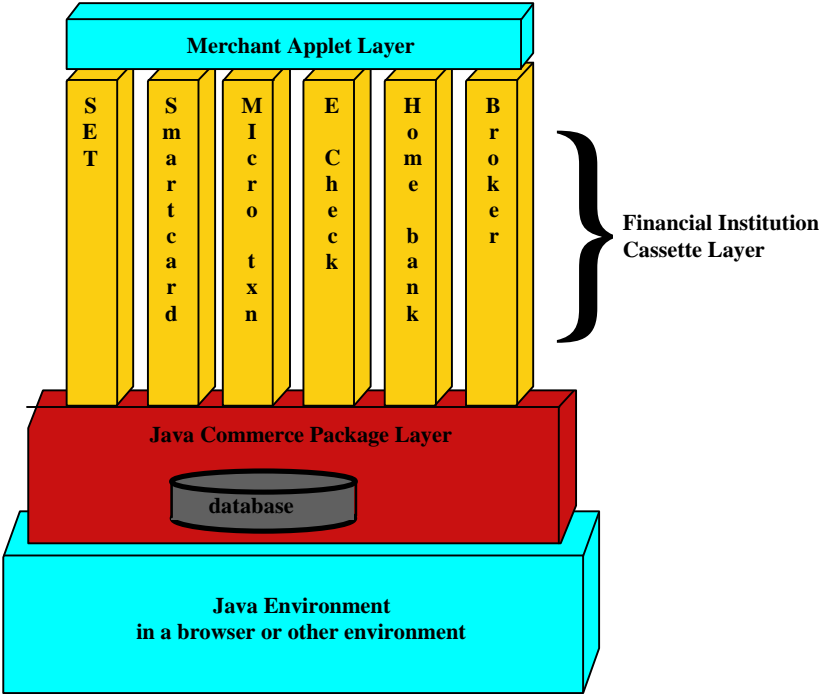

Figure 1. Simple Schema of JECF

he Merchant Applet Layer uses Java applets to enhance the shopping experience. Applets are an appropriate way to implement short term customer relationships such as the shopping experience. Example consumer applets are shopping cart applets and content charging applets. Merchant Applet Layer code does not require a long term customer to merchant relationship. Examples of bank applets include loan questionnaire applets and CD investment selling tool applets.

The Cassette Layer implements long term customer relationships such as credit cards, home banking and brokerages. Cassettes are a new feature that JECF adds to Java. Similar to applets, cassettes are downloaded from servers to client computers. Unlike applets which disappear when users quit the browser, cassettes are retained on the customer's system. Cassettes store information in a database provided by the JECF. Cassettes may safely store valuable information such as

public key certificates and transaction records since the entire database is encrypted. Cassettes provide long term customer-to-institution relationships. Examples of cassettes include SET certificates and protocols, home banking, brokerage accounts, financial analysis, and planning software. Cassettes contain code, digital certificates, GIF images and other resources. Financial institutions will use cassettes to deliver customer service features. Smartcard application developers can put smartcard reader device drivers and application user interfaces in cassettes.

The Java Commerce Package Layer implements the infrastructure needed by the merchant and the cassette layers. Features at this layer include a user interface, an application model, a database, and access to strong cryptography.

The Java Environment Layer is the underlying browser or operating system.

4.1 The Services in the JECF

Applets and cassettes use several layers of service in the Java commerce package. Figure 2 details the layers.

The GUI Services Layer provides a graphical metaphor of a wallet. The wallet depicts credit cards, ATM cards, membership cards and other commonly found documents. The Wallet user interface plays a central function in user interaction. GIF images implement the simplest card. Alternatively, a cassette may use all of Java's user interface components, such as JavaBeans [Hamilton 96], to create elaborate animations and interfaces. As with all user interfaces, developers should use good taste and discretion to avoid creating excessively elaborate cards.

The Application Services Layer implements common application metaphors. Initially, the Java commerce package supports metaphors most appropriate to purchasing. But future metaphors will include other financial services. The central classes in this layer are Instrument, Action and ActionBuilder. There is a one-to-one mapping between Instruments and the card objects. An Action is an individual payment transaction. A MasterCard or Visa SET transaction would be one Action. A coupon redemption is another Action. ActionBuilders assemble actions to create one business transaction. For example, the PurchaseActionBuilder is responsible for understanding the business rules for merchants. Merchants who do not like the PurchaseActionBuilder provided by the JECF may define their own.

The Foundation Service Layer includes the database classes, access to strong cryptography, smartcard device access and various common utility classes such as Money. The Java commerce database provides a subset of the functionality of a relational database. The database is reliable and uses very little memory and disk resources. Although designed for computers with local or remote disks, the database is pluggable to meet the needs of diskless network computers. Third party database vendors will create scaleable commercial databases to provide software safe deposit box services.
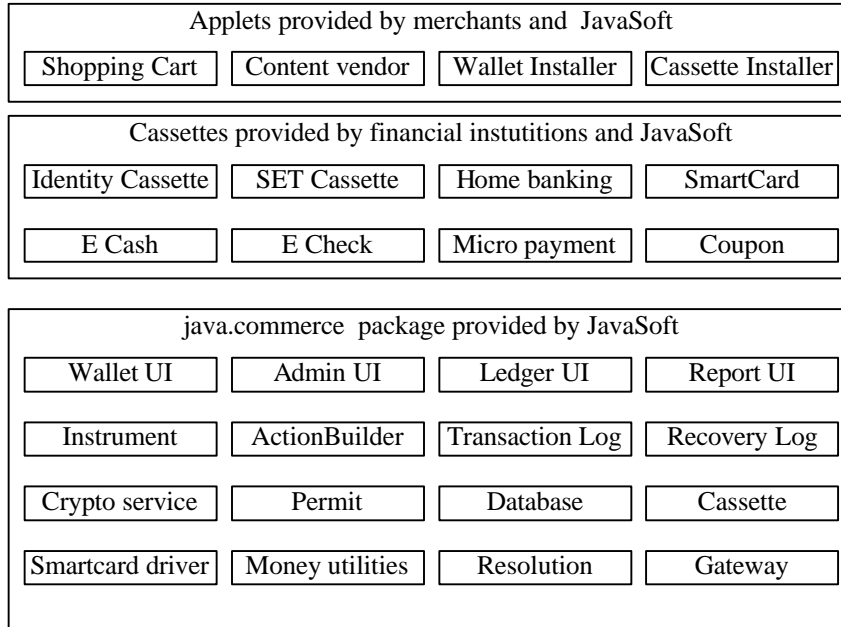
Applets provided by merchants and JavaSoft			
Shopping Cart	Content vendor	Wallet Installer	Cassette Installer

Cassettes provided by financial instutitions and JavaSoft			
Identity Cassette	SET Cassette	Home banking	SmartCard
E Cash	E Check	Micro payment	Coupon

java.commerce package provided by JavaSoft			
Wallet UI	Admin UI	Ledger UI	Report UI
Instrument	ActionBuilder	Transaction Log	Recovery Log
Crypto service	Permit	Database	Cassette
Smartcard driver	Money utilities	Resolution	Gateway

Figure 2. Detailed Schema of packages in the JECF

5. The Gateway Security Model

The Java Sandbox security model's virtue is that it easy to understand. Code is either trusted or it is not. The Sandbox security model defines only the security relationship between an applet and system services. The model makes no statement about non-system services. Commercial relationships are far more complicated. Commercial entities (businesses, individuals) will trust each other for some purposes and not others. The JECF provides a complementary model to the Sandbox security model called the Gateway security model. The Gateway security model provides the means to implement contractual trust relationships. This model uses the safety features of the Java language as well as the infrastructure of the Java Sandbox security model. It also provides the ability for developers to create arbitrary trust relationships.

5.1 Principals and Rights

To completely understand what is occurring, it is necessary to define a few terms: rights and principals. In this paper, a right is an abstract privilege. A principal uses a right. A principal can be a person, a corporation, a program, or a body of code.

The Java sandbox is all-or-nothing because the security principal is the applet. The principal is really the applet's thread of execution that runs the applet. Any thread that the applet creates has to have the same security manager as the applet itself.. The right conferred can only be one of the basic system operations that the Java developers programmed into the security manager object. Electronic commerce requires more rights than just the built-in rights.

Electronic commerce requires that every merchant, financial institution and software developer be entitled to create new rights. These rights represent application-specific activities, such as conducting a purchase or acquiring capital gains information. A single Java electronic commerce thread of control will require collaboration among multiple principals to purchase goods or analyze a portfolio.

Electronic commerce requires an extensible set of rights. Electronic commerce needs to identify a principal more precisely than the applet thread. Possible candidates include threads and packages. The Sandbox security manager can only control the rights invented by the Java system developers on a per thread basis. Electronic commerce also needs the ability to delegate rights from one principal to another as long as the delegation follows the contract-specified business. This cannot be done by threads alone.

In addition to threads, the Java language provides a construct called a package. Packages provide namespaces in the Java language. All classes in a package have access to package-private data members and methods. Package trees are primarily an organizational convention. Packages within a package tree have no special access. Packages are a natural choice for creating a security principal. Perhaps in a

future version of Java, another security domain may emerge. Packages are a natural source of protection.

5.2 Java Objects are Rights

In the Java Sandbox security model, granting an individual right and exercising the right is the same. There is a test in the implementation method code itself to determine whether to grant the right (perform the method) or not. This design has unimportant performance implications, but important model implications. It is not possible to transitively confer a right from one principal to another with the Sandbox security model.

Electronic commerce needs an object-oriented security model where rights can be transferred from one principal to another. The Capabilities model, originally defined by Dennis and Van Horn [DH66], provides such a model. Although originally based on hardware protection, Wulf and a team at Carnegie-Mellon University[WCCJRPP74] adapted Capabilities to object-oriented technology. The object-oriented definition of Capabilities is very simple. The Capabilities model means that possession of an object confers the right to use it. Capabilities need a gatekeeper service to decide whether to grant a right to the object. Figure 3 shows the order of operations between the client code, the gatekeeper, a credential checker and the actual Capability object.
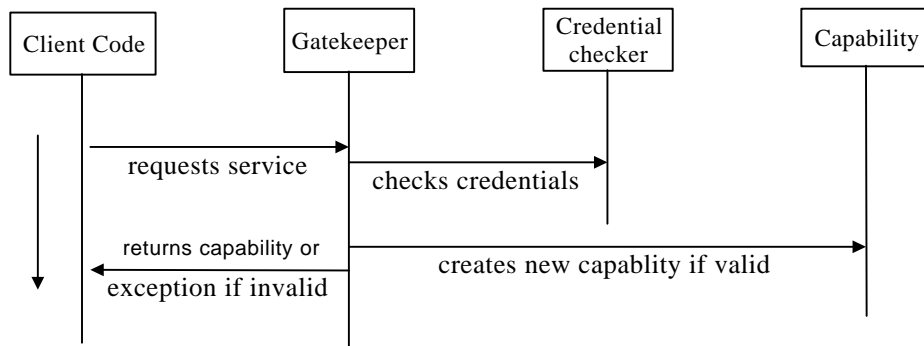

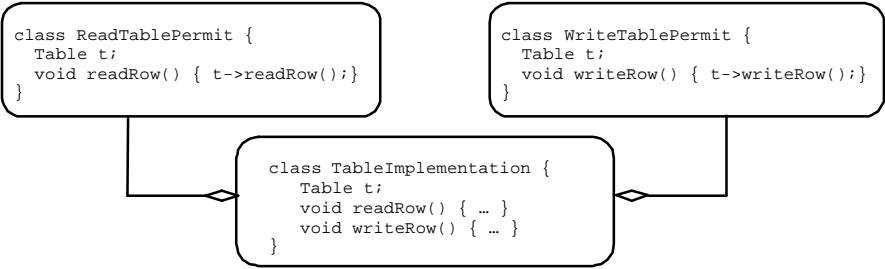Figure 3. Capability creation interaction diagram.

Security systems frequently combine these components. In the Java Commerce Sandbox security model, the implementation immediately performs the requested operation or throws an exception. The implementation never returns the Capability. In most operating systems, the gatekeeper and the credential checker are the same operation. In the JECF the four components are independent.

The Capabilities model is at the core of many important operating systems. The list includes Hydra[WCCJRPP74], IBM System/38 [Bertis 80], KeyKos [Agorics96], and the Spring [HPM93] operating system. In these systems, the

principal was an entire address space. The operating system in supervisor mode is responsible for preventing unauthorized access to Capabilities. In these systems, address-space isolation prevents unauthorized access.

Adding the Capabilities model to Java has proven to be simple because Java already has many of the necessary features. Java objects are unforgeable. Java's language visibility rules for private and package-private scope provide address space-like protection against unauthorized access. The only needed component is some form of authorization mechanism to allow or deny access. In the JECF, individual Capabilities are called permits. Permit objects require careful handling. Permit objects should never be stored in a public or protected variable. Permits are rarely the actual implementation of some right, but are typically implemented by another object. The permit forwards the call to the actual implementation. For example, in the JECF database, the ability to read a row in a table and the ability to write a row in the table   are both implemented in a private class named TableImplemenation. Access to the implementation for TableImplemenation from outside the implementation   is granted by a   pair of objects named ReadTablePermit and WriteTablePermit. Figure 4 illustrates the relationship between permits and the permit's underlying implementation in the Java language in the Java language.

Figure 4. Relationship of permit objects to implementation object

```
class ReadTablePermit {
  Table t;
  void readRow() { t->readRow();}
}
```

```
class WriteTablePermit {
  Table t;
  void writeRow() { t->writeRow();}
}
```

```
class TableImplementation {
    Table t;
    void readRow() { … }
    void writeRow() { … }
}
```

A permit is a delegater pattern (as described in Design Patterns Methodology [GHJV95]) to the actual implementation object. Both the permits and the implementation are in the same package. The implementation however will have only package private constructors. The permit objects need to be obtained by a client package to be useful.

Gaining access to a permit is done via a pattern called a Gate. A Gate is a specialized form of factory pattern method [GHJV95]. A Gate decides whether to grant an instance of a Permit to a caller or deny access by throwing an exception. The JECF uses the Java Sandbox digital signature mechanism for authentication. The general form of a gate method in Java is:

```
class Database {
    TablePermit createReadTablePermit() throws SecurityException {
        if ( caller is digitally signed with the ReadTableRole )
          return new TablePermit();
        else
          throw new SecurityException();
    }
}
```

A Java security manager class can look up the call stack. But often the immediate caller of the method is not the code that should be tested. The implementation itself will frequently invoke such a factory method. Instead of validating the eventual consumer method, the validation check is checking the implementation module itself. The JECF needs a token that represents the actual consumer of the permit. Figure 5 describes in more detail the interaction and call structure of creating the objects so the proper cassette will be validated. This token is passed from the client code into the gate method. In the JECF, this token is of class Ticket. Thus the final form of the Gateway check code is:

```
class Database {
 TablePermit createReadTablePermit(Ticket t) throws SecurityException
 {
if ( t.stamp(ReadTableRole) )
    return new TablePermit();
 else
    throw new SecurityException();
   }
 }
```

Figure 5. Interaction diagram for Table permits

5.3 Roles: Representing relationships

The new element in this example is class Role. The class ReadTableRole provides a representation of the business relationship for this object. Roles reify the trust relationship between two business entities. The JECF uses digital signatures to represent roles. This is the same mechanism that the Java sandbox security model uses. But instead of installing the digital signature into the Sandbox database, the digital signature is embedded in the code as a constant. A Ticket is instantiated with a specific role. The gate function in the client code calls the stamp() method with the Role object as an argument to test and invalidate the Ticket. This technique limits the possibility of a Ticket being used for another possibly malicious purpose.

Anyone (with the appropriate tool) can create a Role. It is essentially a public and private key pair. Every Role creator is the top of its own certificate authority. All the usual procedures about keeping the private key a secret apply. The Role's public key is embedded in the home banking, brokerage software or payment cassette. The financial institution's cassette is then distributed over the Internet, via floppy disks, etc.

When third party companies want to gain access to the financial institution's cassette, they first sign a contract with the financial institution. The institution will typically impose certain usage rules about the data and Capabilities of the cassette that are specified in the contract. The institution then signs the third party cassette which is then distributed over the Internet, via floppy disks, or any other means desired. This process of code signing is illustrated in Figure 6.
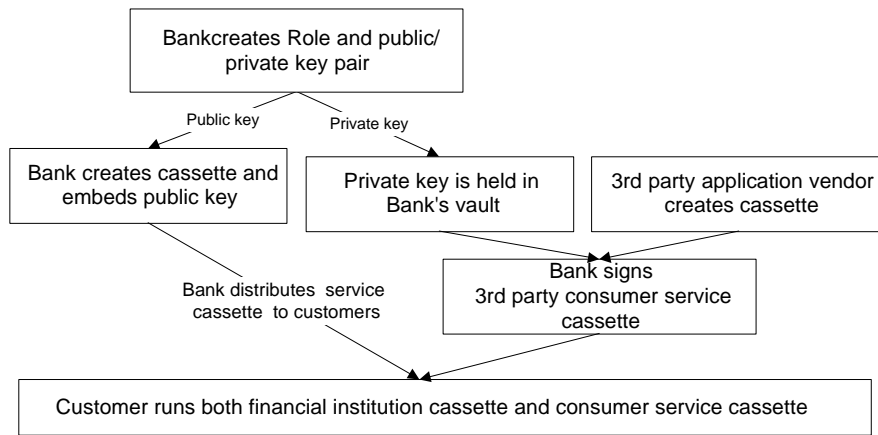

Figure 6. Distribution of signatures among cassettes

Some financial institutions will not want to do their own signing. These institutions will delegate Role signing to trust authority companies. These companies will make a business of validating the design and conformance of a cassette to the financial institutions specification.

Installation of cassettes is initiated by a Java applet class named CassetteLocator. CassetteLocator applets are dynamically loaded from financial institutions and other software suppliers. CassetteLocators communicate with a CassetteInstallation object in the JECF to negotiate loading of cassettes. In some environments, the CassetteInstallation class may impose additional requirements that cassettes be signed with a digital signature of a known trusted authority. Further details are available in the JECF architecture specification[JECF96].

6. Summary and Future Work

This paper explained the need for the Gateway security model extension to the Java Sandbox security model and how the Gateway works to make electronic commerce possible in Java. Java supports electronic commerce by providing tight integration between applications, while maintaining application integrity and isolation. But electronic commerce also requires applications that trust each other to cooperate and the Gateway security model provides that access control using Roles and Permits. Roles are implemented using digital signatures. Permits are Capabilities implemented with Java objects. To summarize the terms of Capabilities used in the JECF:

- Gates are authentication methods
- Tickets are use-once tokens of Roles that are passed into Gates
- Permits are Capabilities objects returned by the Gates
- Roles represent the signature and are used to check Tickets

The full specification for the JECF is available from http://java.sun.com/commerce. Trial deployments of the JECF in browser environments are under way. JavaSoft and Sun Microsystems Laboratories is currently investigating the use of JECF and the Gateway in other limited trust environments. Currently under investigation are electronic commerce servers, medical information systems and the Java Card smartcard interface.

References

[Agorics96], Agorics Inc. http://www.agorics.com/allkey.html, September 15 1996.

[Bertis 80] V. Bertis. Security and Protection of Data in the IBM System/38 in Proceedings of the 7th Symposium on Computer Architecture,. May 1980.

[DH66] J. B. Dennis and E. C. Van Horn, Programming Semantics for Multiprogrammed Computations. Communications of the ACM 9(3), March 1966.

[GHJV95] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns, Elements of Object-Oriented Software, Addison Wesley 1995.

[Gosling 96] J. Gosling, W. N. Joy, F. Yellin; The Java Programming Language Addison Wesley 1996.

[GR83] A Goldberg, D Robson, The Smalltalk-80 Language, Addison Wesley 1983.

[HPM93] G. Hamilton, M. Powell, J. Mitchell Subcontract: A Flexible Base for Distributed Computing. Symposium on Operating Systems Principles

[Hamilton 96] K.G. Hamilton, The Java Beans Specification, http://java.sun.com/.

[JECF96] http://java.sun.com/commerce.

[MV96] MasterCard/Visa Secure Electronic Transaction Protocol Specification (S.E.T.), http://www.mastercard.com/ and http://www.visa.com/.

[Mueller 96] M. Mueller, The Java Security Model, http://java.sun.com/

[OMG95] Object Management Group, The Common Object Request Broker , http://www.omg.org/.

[WCCJRPP74] W. Wulf, E. Cohen, W. Corwin, A. Jones, R. Levin, C. Pierson, and F. Pollack. Hydra: The Kernel of a Multiprocessor Operating System, Communications of the ACM 17(6), June 1974.