THE JAVA CARD™ 3 PLATFORM

White Paper
August 2008

Table of Contents

Table of Contents

Chapter 1

One Technology, Two Editions

Java Card™ technology enables smart cards and other devices with very limited memory to run small applications that employ Java™ technology. It provides smart-card manufacturers with a secure and interoperable execution platform that can store and update multiple applications on a single device. Java Card technology is the most pervasive open platform for secure devices, with more than 3 billion Java technology-powered smart cards deployed worldwide. Sun Microsystems is now announcing the release of a new generation of its market-leading Java Card technology, and industry experts believe it will revolutionize the way smart-card services are conceived and deployed.

Why a new version?

Since the introduction of Java Card technology in 1997, Sun and its licensees have been working closely to make sure the specifications remain current and continue to reflect the latest requirements from the smart-card industry. The Java Card 2.2.2 platform, released in 2006, was the sixth update based on the original architecture for Java Card technology.

Smart-card hardware has progressed tremendously over the past 10 years. The latest advances in silicon technology can support diverse means of smart-card communication. On the one hand, powerful smart cards can free organizations from the limitations usually associated with the deployment of security services.

At the same time, there is an ever-growing demand for high-volume Java Card technology that can be implemented on more resource-constrained chips, while still supporting the latest smart-card and cryptography standards.

To address opportunities on both ends of the smart card hardware spectrum, Sun is introducing the first new Java Card technology architecture in 10 years — and one that will continue to serve the existing markets and deployed applications. How? With the introduction of two editions of the new generation of Java Card technology.

Why two editions?

Java Card 3 technology will be available in two separate, yet coherent editions: Classic Edition and Connected Edition. Java Card Platform, Classic Edition technology is based on an evolution of the Java Card 2.2.2 platform and targets more resource-constrained devices that support traditional applet-based applications. It introduces several incremental changes to the previous version to ensure alignment with smart-card and security standards.

Java Card Platform, Connected Edition technology features a significantly enhanced execution environment and a new virtual machine. It includes new network-oriented features, such as support for Web applications and for applets with extended and advanced capabilities. It targets the high-end smart-card hardware coming from the latest advances in silicon technology.

Both editions are compatible with applications written for previous versions. They also share the key security features and build on the trust and expertise derived from 10 years of deploying secure Java Card technology products.

Java Card Platform, Classic Edition Architecture

Java Card Platform, Classic Edition technology is an evolution of the Java Card 2.2.2 platform architecture. As with previous versions of the Java Card platform, it relies on a split virtual-machine technology that allows for off-card preprocessing of the applications that will be loaded onto the card. This virtual-machine technology ensures that the Java Card platform can be implemented on cards with minimal memory and CPU requirements.

Compared to Java Card 2.2.2 technology, Java Card Platform, Classic Edition technology introduces several incremental changes designed to make sure the technology remains current and continues to address the needs of the majority of smart-card deployments. These include support for the latest cryptography algorithms, including 4096-bit RSA and NSA Suite B, and alignment with the latest contactless standards.

Java Card Platform, Connected Edition Architecture

Java Card Platform, Connected Edition technology provides high-end smart cards with improved connectivity and integration into all-IP networks. A high-end, Java Card 3 technology-enabled smart card can act as a secure network node, capable of providing security services to the network or requesting access to network resources. It also allows for the convergence of smart-card services by handling multiple, concurrent communications through contact interfaces, using IP or ISO 7816-4 protocols, and through contactless interfaces, using the ISO 14443 protocol.

The Java Card Platform, Connected Edition virtual machine is based on the Connected Limited Device Configuration (CLDC) virtual machine widely used in mobile phones, so it’s able to support some of the richer features of the Java programming language. The CLDC-based virtual machine has also been enhanced to meet the requirements of a security device environment: it has been reduced in size, and support for smart-card protocols and security has been added. It’s the cornerstone of an architecture that can provide a new level of functionality and convenience for developers.

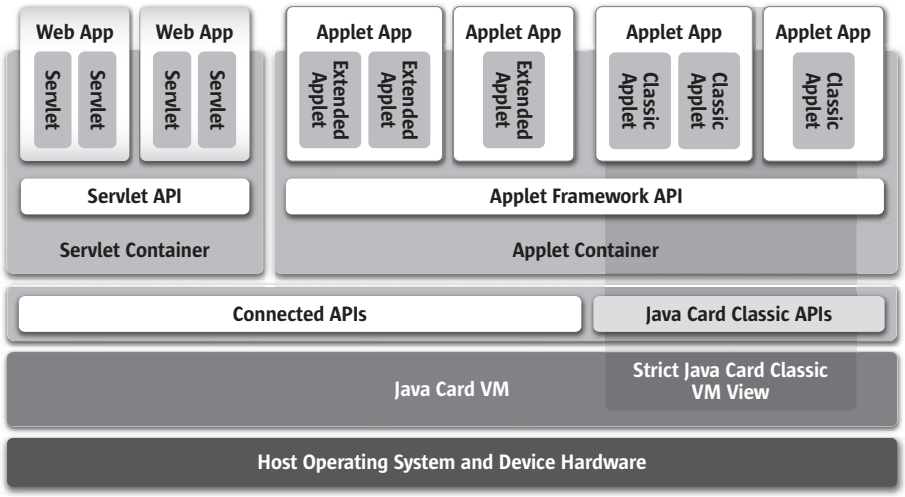

Figure 1. High-level architecture of Java Card Platform, Connected Edition technology

The Java Card Platform, Connected Edition architecture enables the simultaneous deployment of Web applications and traditional smart-card applications. On-card Web applications provide services to entities on an IP network. They're accessible using standard Internet protocols, such as HTTP/HTTPS, so they can be integrated easily into existing Internet services infrastructures. Traditional smart-card applications leverage standard card protocols such as ISO 7816 or contactless protocols. They are required for integration into existing smart-card infrastructures.

This white paper focuses on Java Card Platform, Connected Edition technology, comparing it to the Java Card Platform, Classic Edition only when relevant.

Chapter 2

Enhanced Virtual Machine for Advanced Hardware

The Virtual Machine Specification for the Java Card Platform, Connected Edition platform is based on the Connected Limited Device Configuration Specification, Version 1.1 standard defined by the Java Platform, Micro Edition (Java ME) platform. The technology defined in the Java Card specification is suitable for an advanced smart-card device, which are typically much more resource constrained than Java ME platform devices.

The technology supports class file loading from a Java Archive (JAR) file application distribution format and supports on-card class file verification. It supports multithreading, concurrent execution of applications, and automatic garbage collection (GC), while providing a framework for end-to-end connectivity.

Typical hardware configuration and connectivity

Java Card Platform, Connected Edition technology is intended to run on a wide variety of smart-card and secure token devices with constrained resources. The typical hardware configuration for Java Card Platform, Connected Edition technology corresponds to high-end hardware when compared to devices targeted by Java Card Platform, Classic Edition. It typically has a faster processor and more volatile RAM and persistent EEPROM and ROM. A key characteristic of this high-end hardware is the support of a full-duplex, high-speed interface with its hosting device (such as a phone) and connectivity to some kind of network, typically through its hosting device.

The target devices for Java Card Platform, Connected Edition technology typically have a high-speed contacted physical interface, such as a USB interface. They may have additional I/O interfaces, including ISO 14443 compliant contactless physical interfaces. The Java Card platform provides its applications with a logical network interface supporting IP-based protocols such as TCP, TLS, HTTP, and HTTPS.

This table compares the smart-card hardware targeted by the Java Card Platform, Connected Edition technology with the traditional hardware targeted by previous releases of the Java Card technology and by Classic Edition technology.

Table 1. Evolution of Java Card Technology Targeted Device Configurations.

Traditional Smart Card Hardware	**High-End Smart Card Hardware**
8/16-bit CPU	32-bit CPU
~2-Kb RAM	24-Kb RAM
48-Kb - 64-Kb ROM	>256-Kb ROM
8-32Kb EEPROM	>128Kb EEPROM
Serial I/O interface	High-speed interfaces
9.6 Kb/sec - 30 Kb/sec	1.5 Mb/sec - 12-Mb/sec
Full duplex	Half duplex

Virtual machine technology

As was previously stated, the Java Card Platform, Connected Edition virtual machine is based on the CLDC virtual machine, version 1.1. It has been enhanced to meet the requirements of a security device environment, with reduced size, support for smart-card protocols, and added security. It has also been extended to follow some of the recent advancements of Java Platform, Standard Edition (Java SE technology) for ease of use and development.

The virtual machine technology in the Java Card Platform, Connected Edition is intended for a 32-bit CPU on a high-end smart card device, while the virtual machine technology in the Classic Edition is suitable for an 8- or 16-bit CPU on a more resource-constrained, traditional smart-card device. The Connected Edition virtual machine is able to load class files directly, whereas the Classic Edition uses the traditional split-VM technique, whereby the loading, linking, and verification functions of the Java virtual machine[1] are performed by the off-card converter tool, which then produces an optimized CAP file format suitable for use on the card by the on-card virtual machine.

Both editions feature persistent virtual machines with persistent objects. Multiple applications securely execute on the virtual machine while their objects are protected from intrusion by a firewall-based context-isolation mechanism. The transaction facility supported by the virtual machines allows an application to maintain a consistent state across card tears and power losses.

Java Card Platform, Connected Edition includes the following Java technology features not found in the Classic Edition:

- **Multithreading**
  Multiple application threads may execute concurrently to process off-card messages. Applications may start background threads.
- **On-card bytecode verification**
  The application's class files are verified for type safety by the virtual machine on the card. The Connected Edition leverages the stack map attribute information in the class files generated by the Java SE Development Kit (JDK software) 1.6 compiler to efficiently perform the application code verification, using the limited amount of volatile memory available on the card.
- **Automatic garbage collection**
  Temporary session data is automatically garbage collected when no longer in use.The core platform libraries in the Connected Edition include the system classes that support these new VM features and its networking capabilities. In addition, they have been extended to support collection and other utility classes to provide a more developer- friendly programming environment. These core platform libraries include the following:

1. The terms "Java Virtual Machine" and "JVM" mean a Virtual Machine for the Java platform

- **Data types**: Char, Long, and String
- **Multidimensional array data type**
- **Primitive wrapper classes**: Boolean , Integer, and more
- **String manipulation classes**: StringTokenizer, StringBuffer, StringBuilder
- **Multithreading support classes**: Thread and more
- **I/O classes**: Reader, Writer, and Stream
- **Networking classes from the Generic Connection Framework**: Connector, Connection, and more
- **Collection classes**: Vector, Hashtable, Stack , Iterator, and more
- **Date and time utility classes**: Calendar, Date, TimeZone
- **Localization and Internationalization support classes**: Locale, ResourceBundle, and more

The Java programming language support in the Connected Edition has been enhanced to provide advanced features of Java SE technology, including:

- **Generics**: Allows code to be reused with varying data types with full type safety
- **Metadata (Annotation)**: Allows programmers to insert metadata information in the code. The transaction annotation demarcates transaction-protected zones within the code (uses the SOURCE and CLASS annotation retention policy)
- **Assertions**: Enables programmers to test their assumptions about the runtime program state
- **Enhanced for loop**: Allows a program to iterate conveniently over elements of an aggregator object
- **Autoboxing**: Allows a program to use a primitive type as a first-class object
- **Typesafe enumerations (enums)**: Provides a type safe mechanism to define a set of named values for a type
- **Varargs**: Allows a programmer to define methods that accept a variable number of arguments
- **Static import**: Helps the programmer avoid the Constant Interface Antipattern problem

Chapter 3

Enhanced Programming Model for Advanced Use Cases

Java Card Platform, Connected Edition technology brings the application programming experience closer to that of mainstream Java programming environment. It not only supports a virtual machine with a richer subset of the Java virtual machine features, such as multithreading and richer data types, but also provides a Web application programming environment. Moreover, the applet application model has been extended to benefit from newly added features, such as multithreading and network connectivity, thereby providing a migration path from classic applet application programming to Web application programming. A Java Card platform application developer may also use many of the new or enhanced facilities provided by the platform, such as persistence, transactions, interapplication communication, and restartable tasks facilities.

Web applications

The Java Card Platform, Connected Edition supports a subset of the Java Servlet API Specification 2.4 Web application model.

Web applications are applications that interact with off-card Web clients via HTTP or HTTPS requests and responses. The lifecycle of Web applications, the network services over which requests and responses are sent, and the security of access to these applications and their resources are managed by the Java Card platform's Web application container.

A Java Card technology developer typically implements the following components of a Web application:

- **Servlets, request and response filters, lifecycle event listeners, and other business logic and utility classes**
  Servlets and filters are the components of a Web application that generate dynamic content, defined as content that's computed upon requests from clients and sent back to the clients as part of the response.
- **Static resources such as HTML documents and embedded images or objects**
  Static resources constitute static content, which is content that resides in files or the equivalent.
- **Web application deployment descriptor**
  The deployment descriptor describes the Web application's components, how they are mapped to client requests, and their security requirements (which include user authentication and authorization, as well as secure communication requirements).

Each Web application is deployed into the Web container and is uniquely identified by the specific path to which it is rooted in the Web container namespace. That path is used both as the application's unique identifier (application URI) on the platform, for such tasks as application management and on-card interapplication communications, and for dispatching HTTP requests submitted by off-card Web clients, based on the queried URL.

The Web container is multithreaded and is able to dispatch multiple HTTP requests concurrently.

Beyond allowing for integration of Java Card technology-based solutions into Web service architectures, supporting a subset of the Java Servlet Specification 2.4 Web application model has several benefits for application providers:

- It leverages the skills of Web application developers already familiar with the Java Platform, Enterprise Edition (Java EE) environment
- It takes advantage of existing tool chains and allows for using a Web application development environment in off-the-shelf IDEs

Applet applications

The Java Card Platform, Connected Edition supports both the classic and extended applet application models.

Applet applications interact with off-card applet clients via ISO 7816-4 APDU commands and responses. The lifecycle of applet applications is managed by the Java Card platform's applet container[2]. Extended applet applications differ in the following respects from classic applet applications as they are supported in the Classic Edition:

- Extended applet application code may comprise multiple packages, while classic applet application code comprises only one package.
- Extended applet application code may use all facilities and libraries of the Connected Edition, while classic applet applications are restricted to those defined in the Classic Edition.
- Extended applet applications may execute concurrently on different threads to process APDU commands received over different I/O interfaces, while classic applet applications execute in a single threaded environment and are not thread-aware.

A Java Card platform developer typically implements the following components of an applet application:

- **Applets and other business logic and utility classes**
  Applets are the components of an applet application that process incoming APDU commands from clients and send APDU responses back to the clients.
- **An applet application deployment descriptor**
  The deployment descriptor contains information to uniquely identify each concrete applet class in the application.

2. The concept of application container, which is implicit in the Classic Edition, has been formalized in the Connected Edition to better support the different application models.

Each applet application is deployed into the applet container and is uniquely identified on the card by its applet instance application identifier (AID). An applet instance AID has an equivalent application URI form that is used as the application's unique identifier on the platform for application management, on-card interapplication communications, and applet selection and dispatching of APDU commands submitted by off-card clients.

Multithreading

The Java Card Platform, Connected Edition virtual machine supports multithreading and concurrent execution of applications. The Java Card API includes a subset of the Java SE technology thread API, which allows an application to create and handle threads of control.

As described in the previous sections, the Web application environment and the extended applet application environment are multithreaded application programming environments. Other facilities of the Java Card Runtime Environment (Java Card RE) may also use different threads to concurrently invoke application-entry-point methods, such as to asynchronously notify an application of an event. A developer of such applications must account explicitly for multithreading and may even rely on application-managed threads to implement applications.

Threads are not persistent and cannot be resumed after a platform reset — and application-managed threads must be manually recreated after each one. To alleviate this issue, the Connected Edition provides a facility for an application to register tasks that are automatically restarted upon a platform reset.

The Connected Edition supports a classic applet application programming environment for backward compatibility with the Classic Edition, which does not support multithreading, but still operates concurrently with the other two application environments.

Persistence

Code and data persistence across card-restart events such as card tear (or reset) followed by power up is key to the special nature of smart-card devices and the Java Card platform. The Java Card virtual machine and application code persist across card-restart events. Objects may be made persistent as well, under certain conditions.

On the Java Card platform, the memory is divided into volatile memory and nonvolatile memory:

- **Volatile memory** (typically DRAM) does not retain its contents across card-restart events on the smart-card device. Volatile objects are objects stored in volatile store and are typically intended to be short-lived or to require frequent updates. Volatile objects are garbage collected on card-restart events.

- **Nonvolatile memory** (typically ROM, EEPROM, and flash memory) retains its contents across card-restart events on the smart-card device. Persistent objects are objects stored in nonvolatile store and are intended to be long-lived objects, and they retain their contents across card restart events.

While the persistence of an application's code (its classes) and of some of its data (its objects), such as is required for the management of the application, is ensured by the platform, application-created objects are all initially volatile and may only be made persistent under certain conditions of reachability. The Java Card platform implements a strategy called *persistence by reachability* for promoting volatile objects to become persistent objects. A newly created object remains volatile as long as it is not referenced by any other persistent object. If referenced, it becomes persistent.

All objects are garbage collected when no longer referenced by other objects. Garbage collection on volatile objects is typically initiated automatically. Garbage collection on all objects, especially persistent objects, may be initiated on demand by application code.

Transactions

On the Java Card platform, an application may complete a single logical operation on application data atomically, consistently, and durably within a transaction. Atomicity ensures that updates to data within the scope of a transaction either all occur, or none occur. Consistency allows the application to establish a consistent state before the start and after the end of the transaction. Durability ensures that when the transaction is successfully completed, the updates are committed.

The transaction facility of Java Card Platform, Connected Edition technology extends the transactions and atomicity subsystem of the Classic Edition to provide the following additional features:

- **Support for multiple concurrent transactions**: More than one transaction may be ongoing at the same time.
- **Support for nested transactions**: A subtransaction within an ongoing transaction may be initiated and may complete independently before the original transaction.
- **Better programmer control and program audit of transaction durations**: A method is annotated to explicitly declare its transactional behavior.

The Connected Edition uses annotations (a subset of Java SE technology annotations) to mark classes or methods for transaction demarcation.

Interapplication communications

In Java Card Platform, Connected Edition technology, interapplication communications have been enhanced beyond the classic shareable interface mechanism to allow for an application to communicate with another application using the following two facilities:

- **Services**

  An application can publish a service it wants to provide to other applications in a central registry. This facility extends the classic shareable interface mechanism and allows for all application models (applet and Web applications) to interact through shareable interface objects in a unified way. In addition to services that may be defined by applications, the specifications for the Connected Edition include a set of predefined standard services, such as for user authentication.

- **Events**

  Through a central registry, the platform or an application can also notify other applications of a particular condition. When the condition occurs, it is encapsulated in a shareable interface object called an event and is passed for handling to an object called an *event listener* that has been registered for notification of the condition. This facility allows for Web and applet applications to communicate asynchronously with each other through events. In addition to application-defined events, the specifications for the Connected Edition include a set of predefined platform and standard events, such as clock resynchronization and application-lifecycle and resource-lifecycle management.

The use of central registries for services and events allows for a loose coupling between server and client applications. In conjunction with support for application lifecycle events, central registries enables provisioning of client and server applications independently of each other throughout the lifetime of a card.

Network communications

In addition to the networking capability intrinsically offered by the Web application container (container-managed connections), the Java Card platform provides applications with a means for managing network communications by themselves. An application can open server communication endpoints and also initiate client communications with off-card entities. Thus, applications on the Java Card platform are no longer limited to providing services to off-card clients, but may also themselves be clients of off-card services across a network.

Network connections, both client and server, can be managed on the Java Card platform using the Generic Connection Framework. The Generic Connection Framework classes provide a set of related abstractions to request and manage network or I/O connections using various protocols, including both secure and nonsecure protocols such as TCP, TLS, HTTP, HTTPS and, optionally, UDP.

File access

The Java Card platform has optional support for hierarchical file systems of directories and files. An application may use the Generic Connection Framework to access file system objects.

Each application can have a dedicated file system to which it has exclusive access. An application may nevertheless be implemented to act as a file server to other applications. A typical file server application stores the file contents in its own private file system and may implement some access-control policy using the security mechanism provided by the platform, such as user authentication and client authentication, to permit or deny access by client applications to certain files.

Chapter 4

Secure Applications, Secure End-to-End Communication

Beyond the low-level Java bytecode verification implemented by the Java Card virtual machine, Java Card Platform, Connected Edition technology implements a variety of security mechanisms that provide application-level *and* end-to-end communication security.

Secure containment of applications

Java Card Platform, Connected Edition technology provides several complementary mechanisms that enforce the security containment of application code and data.

Code isolation and Java programming language package access control

The Java Card platform supports a *code isolation* mechanism. Code isolation ensures that code loaded from one application does not interfere with the code of other applications. Code loaded in this manner cannot override or directly access the code of other applications. This is implemented by defining and enforcing different class namespaces for each loaded application's code.

On the Java Card platform, code isolation is implemented with a *class loader delegation hierarchy*, which enforces isolation of application code by default and allows for explicitly sharing code, such as libraries and public interfaces, among cooperating applications. The class loader delegation hierarchy is not exposed to application developers and is strictly defined by the platform.

Additionally, the Java Card platform includes mechanisms to prevent loaded code, applications, or libraries from overriding or extending the set of system classes. It also supports the standard *JAR file package sealing* mechanism, which prevents the classes in a sealed package from being overridden or extended once that package has been loaded.

These mechanisms cater to the secure containment of application code.

Context isolation and object sharing

The Java Card platform supports isolation of contexts and applications. *Context isolation* ensures that objects created, and therefore owned, by applications running in the same context cannot be accessed by applications from another context unless the applications owning these objects explicitly provide interfaces for access. Such interfaces are called *shareable interfaces* and objects implementing these interfaces, called *shareable interface objects*, constitute legal entry points to these applications.

Similarly, the Java Card RE executes in a separate context and provides well-defined entry points, called *Java Card RE entry-point objects*, that can be used by applications to request system services.

Context isolation is enforced by an application firewall and protects against any unauthorized access, such as those that may result from developer mistakes and design oversights.

The firewall-enforced context isolation mechanism originally defined in the Classic Edition has been enhanced in the Connected Edition to account for the extended bytecode set and to allow for more effective interapplication communication through shareable interfaces. The Connected Edition introduces a new mechanism, called the *object ownership transfer mechanism*, which allows for an application to transfer the ownership of an object to another application, thereby transferring exclusive access to the object to that other application.

Context isolation guarantees the secure containment of application execution.

Dedicated application namespaces

Java Card Platform, Connected Edition technology supports a *unified naming* scheme that allows applications of all types and their respective resources to be uniquely named and uniformly addressed on the platform. Each application is assigned a dedicated resource namespace rooted under its unique name. The name of an application is represented as a URI. An application's resources, including Web resources, services, events, and files, are named relative to that application's URI.

The Java Card platform makes certain that the application has exclusive use of its namespace. For example, an application cannot register a service under a URI that is not rooted in its own namespace.

Assignment of application URIs and namespaces is a key responsibility of card management applications and is essential to the deployment of multiapplication-provider cards that support post-issuance provisioning.

Dedicated application namespace enforcement is yet another mechanism that contributes to the security containment of applications.

Access control

Java Card Platform, Connected Edition technology supports complementary access control mechanisms that allow for the definition, configuration, and enforcement of platformwide as well as per-application security policies.

Permission-based security

Permission-based security allows a security authority for the card to restrict access to protected system, library, and application resources based on some of the characteristics of the application requesting the access, such as the type of application model implemented (Web application or extended or classic applet application) and the credentials of the application's code signer.

A *protection domain* is associated with each application group (group context). A protection domain corresponds to the set of permissions granted to an application or group of applications as per the security policy that applies to that group of applications. Each permission in a protection domain represents access to specific protected resources, such as security-sensitive system resources, or application resources, such as services provided by other applications.

On the Java Card platform, the permission-based security policy is enforced by two types of permission checks:

- Context-switch-triggered permission checks, which are automatically initiated when an application in one context attempts to access an entry point object (for example, a shareable interface object, or SIO) in another context
- Programmatic permission checks, which are programmatically initiated by the system or extension libraries

Each permission check determines if the permission requested is granted by the protection domain of the application requesting the permission.

Permission-based security is used to restrict access to resources to only those applications that have been specifically granted the permission to access the resource.

Permission-based security provides a powerful means for a card security authority to control, with fine granularity, the resources accessed by an application or group of applications.

Role-based security

Role-based security allows an application's security policy to restrict access to protected application resources, including Web resources, SIO-based services, and events. The restrictions are based on some of the characteristics of the application requesting the access, such as its identity, and the identity of the user on whose behalf the access is requested.

An application developer may express the logical security requirements of the application either declaratively in the deployment descriptor of a Web application or programmatically through the implementation of programmatic security checks within the code that manages access to certain resources. These declarative security constraints and programmatic security checks name the client or user security roles permitted to access the protected resources.

A user or client security role is a logical grouping of users or client applications defined by the application developer or assembler. When the application is deployed, user and client roles are mapped by a deployer (for example, the application provider) to actual user identities and client application identities or characteristics on the targeted platform.

Role-based security is used to restrict access to resources to only those users and client applications that have been granted a particular security role.

Role-based security provides a flexible means for an application developer to define and implement the security requirements of his application while allowing for the actual security policy to be configured only upon deployment.

User authentication and authorization

User authentication is the process by which a user proves his or her identity to the card. This authenticated identity is then used by the platform to perform authorization decisions, such as those required by role-based security, for accessing protected resources, including SIO-based and event-based services, as well as Web resources.

On the Java Card platform, users are categorized as either a cardholder, the primary user of the card, or as "other users," such as a remote card administrator. Each type of user has a different user identity on the card.

User authentication is implemented by *authenticators*, specialized authentication services that can use a variety of schemes to authenticate a user, such as a password, a PIN, or a biometric template. These services can be invoked both by an application or by the Web container.

Because several conversational sessions can be established simultaneously between on-card Web applications and Web clients, Web-user authentication is tracked on a per-session basis. Session-scoped *authentication* contrasts with the global authentication typically implemented on a classic platform in that it prevents a user authenticated in a conversational session under one identity to gain unauthorized access to protected resources authorized to another, simultaneously authenticated, identity.

Additionally, the Java Card platform distinguishes applications that are accessible locally and safely (through a cardholder-facing client) from applications that may be accessed remotely and which have stronger security requirements. Such a remotely accessible application may be an administrative user application or a card management application. To be accessed, such applications require the explicit authorization of the cardholder.

On-card client application authentication and authorization

Client application authentication is the process by which a client application proves its identity to a server application. This authenticated identity is then used by the server application to perform authorization decisions for accessing protected resources, such as SIO-based and event-based services it exposes.

A server application, when accessed through one of the services it exposes, may initiate and check authentication of a client application programmatically by naming the client security role permitted to access the service. That client security role is associated to the credentials of authorized application clients to which it has been mapped during deployment. These credentials may be symmetric or asymmetric cryptographic key or certificate materials.

A client application may have to authenticate with several different server applications. The authentication of the client application with each of these server applications is tracked independently.

On-card client authentication provides a flexible and robust mechanism for managing the security policy of a server application. Instead of relying solely on the identification of an application, such as based on its identifier on the platform to grant access to its services, a server application can use credentials to authenticate its trusted clients.

Network communication security

With the Java Card platform, applications may interact with off-card peers (for example, the mobile phone the card is embedded in or a server located on the Internet) through secure network communications over Secure Sockets Layer (SSL) or Transport Layer Security (TLS). These communications may be established over Web container-managed HTTPS server connections, such as between an off-card client and a Web application, or over application-managed client or server SSL/TLS or HTTPS connections.

The security of such network communications, also referred to as end-to-end security, relies on secure network protocols and cryptography services that ensure the confidentiality and the integrity of the data transmitted, as well as the authentication of the peers.

Key and trust management

Application developers may configure the security requirements and characteristics for secure communications with peers either programmatically in the application's code or declaratively in the deployment descriptor of a Web application. The security requirements for a secure communication include peer authentication, integrity, and confidentiality of the data transmitted.

Additionally, an application developer may use a c*redential manager object* to manage the credentials used to establish secure communication with peers. A credential manager is used both for managing the key material that is used to authenticate with peers and for managing the trust material that is used when making trust decisions, such as deciding whether credentials presented by a peer should be accepted.

The Java Card platform allows an application to delegate to a card manager application the management of the keys and trust decisions for the secure connections that it specifically opens or that are opened on its behalf by the Web container. Such a delegated model of credential management is essential for supporting card management frameworks, such as GlobalPlatform.

Secure hosting of Web applications

A Web application developer may declare requirements for content integrity and confidentiality in the deployment descriptor of a Web application. The application developer or provider can also require that the application be hosted on a dedicated secure port. The Java Card platform's Web container enforces a Web application's requirements for content integrity and confidentiality by only accepting requests for protected resources of that application, over HTTPS connections open on that application's dedicated secure port.

As in the case of application-managed secure communications, the security characteristics of connections from Web clients on that application-dedicated HTTPS port are negotiated using the credential manager that applies to that application.

Therefore, each Web application that requires protection is securely hosted on its own dedicated secure port and uses its own security requirements and credentials to establish secure communications with Web clients.

Extensible cryptography framework

Java Card Platform, Connected Edition technology supports an extensible cryptography framework that allows platform implementers to include for each cryptography service a variety of algorithms and implementations. An application developer or provider can discover all the various algorithms and implementations a card supports. Then the developer can select the cryptography algorithm and implementation that best suits the needs of his application and the constraints of the operating environment the application is deployed to. Such a selection can be made based, for example, on the provider of a particular cryptographic algorithm implementation.

Chapter 5

Developing and Deploying New Services

The development lifecycle of applications for Java Card Platform, Connected Edition technology differentiates the roles and responsibilities of the participants in the process, from the developer of an application to its assembler and distributor to the one who deploys the application to the card.

- **Application developer**
  The application developer creates the application. The output of a developer is a set of application classes and resources and supporting libraries. The developer is typically an application domain expert and is aware of the application environment and its consequences when programming, including concurrency considerations.
- **Application assembler**
  The application assembler takes the output of the developer and ensures that it is a deployable unit. The output of the application assembler is an application archive conforming to one of the distribution formats supported by the Java Card platform.
- **Application deployer**
  The application deployer takes one or more application archive files provided by an application developer and deploys the application onto a card in a specific operational environment. The operational environment includes other installed applications and libraries, as well as frameworks defined by standards bodies. The deployer must resolve all the external dependencies declared by the developer and is an expert in a specific operational environment. For example, the deployer is responsible for mapping the security roles defined by the application developer to the users that exist in the operational environment where the application is deployed. (The application deployer is often the application provider as well.)

Note that these roles are only logical roles. An actual participant, such as an application provider, may perform several of these roles. For example, a Java Card technology application provider may buy an application or application components from one or several application developers, assemble his application and deploy to cards in the field, all on his own.

Application descriptors

The application's descriptors are central to its development lifecyle. These descriptors are documents that describe the structure, configuration, and deployment information of an application, and convey this information to application developers, assemblers, and deployers.

In Java Card Platform, Connected Edition technology, there are three types of descriptors:

- **Application model-specific deployment descriptors**
  The Web application deployment descriptor and the applet application deployment descriptor describe the application's elements and configuration information that are dependent on the application model.
- **Java Card platform-specific application descriptor**
  A Java Card platform-specific application descriptor describes the application's elements and configuration information that are specific to the Java Card platform.
- **Runtime descriptor**
  A runtime descriptor describes the application's configuration and deployment information that are specific to an operating environment to which the application will be deployed.

The use of descriptors, especially to convey the security configuration requirements of applications, alleviates the need for hard-coding such configurations and better leverages the security mechanisms in the platform. Therefore, the security evaluation of applications for their reuse and deployment in different operating environments is simplified.

The Java Card Platform, Connected Edition leverages a model for development, distribution, and deployment of applications that is very familiar to enterprise Web application developers. Moreover, developers can create Web applications targeted at the Java Card platform using the same tools used for enterprise Web applications, such as a standard, off-the-shelf IDEs.

Distribution and deployment units

The Java Card Platform, Connected Edition supports three types of distribution and deployment units:

- The application module distribution format JAR file encapsulates one application, either a Web application or an applet application.
  The Web application distribution format is a Web archive (.war) file with a runtime descriptor and a Java Card platform-specific application descriptor as additional metadata files.
- The applet application distribution format contains the classes for the applet application module and the applet application model deployment descriptor file. The runtime descriptor and Java Card platform-specific application descriptor files are also included as additional metadata files.

The classic applet application distribution format also contains the applet CAP file components (*.cap) inside the JAR file.

- The extension library JAR file is a standard library JAR file containing Java class files. Extension library classes are accessible to all applications on the card.
- The classic library JAR file is a standard JAR library format containing Java class files as well as the library CAP file components (*.cap). Classic library classes are only accessible to the classic applications on the card.

The inclusion of classic applet and library CAP file components in classic deployment units ensures that these deployment units can be deployed on platforms implementing the Classic Edition without any changes. Backward compatibility is ensured for these applications, and the interoperability of both the Connected Edition and the Classic Edition is guaranteed.

Application deployment and card management

The card manager in Java Card Platform, Connected Edition technology supports the following post-issuance functions after proper authorization of the requesting off-card client:

- Loading of deployment units (application modules and libraries)
- Creation of application instances
- Deletion of application instances
- Unloading of deployment units

The Connected Edition defines a card management SPI for these post-issuance functions to enable industry-defined card management applications to be integrated with the platform. The GlobalPlatform card management layers are currently being developed for the Connected Edition.

Application development tools

The application developer can use an off-the-shelf IDE to develop an application for Java Card Platform, Connected Edition technology. Convenient tools are available with the development kit for packaging the application binary file into the distribution unit format and for validating and deploying onto the platform's reference implementation.

A Web application that uses the Connected Edition subset of the Java EE programming environment may be developed and run on a standard IDE. Both the servlet API and the binary Web archive format are compatible with that of the Java EE platform.

An early prototype implementation of a plugin module for the NetBeans™ IDE that makes it easy to develop applications for the Connected Edition was demonstrated at the 2008 JavaOne℠ Conference. The plugin makes it simpler for the application developer to:

- Create a project
- Configure the target platform for resources settings
- Generate prototype code
- Suggest code-completion choices for the API
- Create static HTML pages
- Create descriptors
- Deploy, run, and debug the application against the configured target platform

Chapter 6

Smart Card, Smart Choice

While the Classic Edition of the Java Card 3 platform is an evolution of the Java Card platform that supports previously deployed applications and integrates into legacy infrastructures, the Connected Edition constitutes a major step forward. The Connected Edition brings Java Card technology closer to the mainstream Java environment in terms of programming experience and enabling smart card-based security services to integrate into all-IP networks — while continuing to support existing card services.

A Java Card 3 technology-based smart card could enable a wide range of new services and user experiences, such as:

- Services that leverage the capability to handle multiple, concurrent communications through contact and contactless interfaces. Examples of such convergence are:
  - **Mobile ticketing**: Contactless punching of on-card transit tickets while simultaneously querying the balance of the on-card ticket book or viewing DRM-protected media content.
  - **Contactless and mobile payment**: Performing online network authentication and authorization, and contactless payment simultaneously.
- Services that provide end-to-end secure communications with services on the network without the need for ad hoc communication proxy on terminals or hosting devices. Examples of end-to-end secure communications are:
  - **Mobile banking**: Performing a complete end-to-end financial transaction
  - **Post-issuance secure provisioning of applications**
- Services that allow for controlled access to both on-card and off-card protected content, such as:
  - **Streaming DRM-protected content**: Content going from a SIM card to a phone or from a media server through the card to the phone
- Applications that provide secure access to on-card information through rich Web interfaces without the need of any ad hoc client application on the terminal or hosting device. Examples are:
  - **Securely managing private information from a Web browser**
  - **Browsing content like phone book entries in a SIM card**
  - **Remotely browsing card content (known as “home-page on SIM”)**
- Applications that combine data from both on-card and off-card sources into a single integrated user experience (Web mashup)

Java Card 3 technology is the smart choice for securely managing personal information and data in the all-connected era.

Chapter 7

Glossary

AID (APPLICATION IDENTIFIER)

Defined by ISO 7816, a string used to uniquely identify card applet applications and certain types of files in card file systems.

APDU

An abbreviation for Application Protocol Data Unit as defined in ISO 7816-4.

APPLET

Within the context of this document, a Java Card applet, which is the basic component of applet-based applications and which runs in the APDU application environment. Each applet is identified uniquely by an AID or its equivalent URI.

APPLET APPLICATION

An application that consists of one or more applets.

APPLET CONTAINER

Contains applet-based applications and manages their lifecycles through the applet framework API. Also provides the communication services over which APDU commands and responses are sent.

APPLICATION URI

A URI uniquely identifying an application instance on the platform.

AUTHENTICATION

The process of establishing or confirming an application or a user as authentic using some sort of credentials. On the Java Card platform, user authentications are performed by dedicated authentication services, named authenticators.

CARDHOLDER

The primary user of a smart card.

CARD MANAGER

The on-card application to download and install applications and libraries.

CARD MANAGEMENT FACILITY

The Java Card platform layer responsible for securely adding and removing application code and instances onto the platform.

CLASSIC APPLET

Applets with the same capabilities as those in previous versions of the Java Card platform and in the Classic Edition.

CLASSIC EDITION

One of the two editions in the Java Card 3 platform. The Classic Edition is based on an evolution of the Java Card platform, Version 2.2.2 and is backward compatible with it, targeting resource-constrained devices that solely support applet-based applications.

CLASS LOADER

A Java Card RE component that defines and enforces a different class namespace for the classes it loads. Class loaders are assembled into a class loader delegation hierarchy that enforces code isolation among applications while allowing for sharing of system and library code.

CONNECTED EDITION

One of the two editions in the Java Card 3 platform. The Connected Edition has a significantly enhanced runtime environment and a new virtual machine. It supports both applet-based and Web applications.

CREDENTIAL

Material that can be used to ascertain the identity of a party (authenticate) in order to control access by that party to information or other resources and/or to protect the integrity or confidentiality of information exchanges with that party. Examples of credentials are password, PIN, or public-key certificates.

CREDENTIAL MANAGER

An object that manages the key and trust material of an application when a secure communication is being established by either that application or by the Web container on behalf of that Web application.

DECLARATIVE SECURITY

A means of expressing an application's security structure, including roles, access control, and authentication requirements in a form external to the application, such as in the deployment descriptor of a Web application.

DESCRIPTOR (APPLICATION, DEPLOYMENT, RUNTIME)

A document that describes the configuration and deployment information of an application.

DISTRIBUTION FORMAT

Structure and encoding of a distribution or deployment unit intended for public distribution.

EVENT

An object that encapsulates some occurring condition or situation. In the context of the event notification facility, an event is a shareable interface object that an event-producing application uses to notify its clients (event-consuming applications) of an occurring condition. Each event type is identified with a unique event URI.

EVENT NOTIFICATION FACILITY

A Java Card RE facility (or subsystem) that is used for event-driven inter-application communications. The core component of the event notification facility is the event registry, which is used for registering for event notification and for notifying events.

EXTENDED APPLET

An applet with extended and advanced capabilities (compared to a classic applet) such as the capabilities to manipulate String objects and open network connections.

FIREWALL

The mechanism that prevents unauthorized accesses to objects in one application group context from another application group context.

GARBAGE COLLECTION

The process by which dynamically allocated storage is automatically reclaimed during the execution of a program.

GROUP CONTEXT

Protected object space associated with each application group and Java Card RE. All objects owned by an application belong to the context of the application group.

JAVA CARD RUNTIME ENVIRONMENT (JAVA CARD RE)

Consists of the Java Card virtual machine and the associated native methods.

JAVA CARD VIRTUAL MACHINE (JAVA CARD VM)

A subset of the Java virtual machine, which is designed to be run on smart cards and other resource-constrained devices.

JAVA CARD RE CONTEXT

The context of the Java Card RE, with special system privileges so that it can perform operations that are denied to contexts of applications.

MODULE (APPLICATION)

The logical unit of assembly of Web or applet-based application. The components of a Web application are assembled into a Web application module. The components of an applet application are assembled into a applet application module.

PERMISSION

An object that represents access to specific protected resources, such as security-sensitive system resources, or application resources, such as services provided by applications.

PERMISSION-BASED SECURITY

Measures defined by a permission-based security policy that restrict access to protected system and library resources to those client applications that are in a protection domain that grants the permissions to access the protected resources.

PERSISTENT OBJECT

Persistent objects and their values persist from one card session to the next, indefinitely; that is, they are not lost when the card loses power. Persistent object values are typically updated atomically using transactions.

PROGRAMMATIC SECURITY

A means for a security-aware application to express the security model of the application when declarative security alone is not sufficient.

PROTECTION DOMAIN

A set of permissions granted to an application or group of applications.

ROLE (SECURITY)

An abstract notion used by an application developer in an application that can be mapped by the deployer to a user or a an on-card client, or a group thereof, in a security policy domain.

ROLE-BASED SECURITY

Measures defined by a role-based security policy that restrict access to protected application resources to those users and clients that are in roles permitted to access the protected resources.

SERVICE

A shareable interface object that a server application uses to provide a set of well-defined functionality to its clients (client applications). Every service is uniquely identified by a service URI.

SERVICE FACILITY

A Java Card RE facility (or subsystem) that is used for inter-application communications. The core component of the service facility is the service registry which is used for registering and looking up services.

SERVLET

A Web application component, managed by a container, that generates dynamic Web content and that runs in the Web application environment.

SHAREABLE INTERFACE

An interface that defines a set of shared methods. These interface methods can be invoked from an application in one group context when the object implementing them is owned by an application in another group context.

SHAREABLE INTERFACE OBJECT (SIO)

An object that implements the shareable interface.

RESTARTABLE TASK REGISTRY

A Java Card RE facility that is used for registering tasks for recurrent execution over card sessions.

THREAD

The basic unit of program execution. On the Java Card platform, several threads may be running concurrently each performing a different job, such as waiting for I/O or events or performing a time-consuming job.

TRANSACTION

An atomic operation in which the developer defines the extent of the operation by indicating in the program code the beginning and end of the transaction. Atomicity of data updates guarantee that data are not corrupted in case of power loss or card removal.

TRANSACTION FACILITY

A Java Card RE facility that enables an application to complete a single logical operation on application data atomically, consistently and durably within a transaction. transfer of ownership

TRANSFER OF OWNERSHIP

A Java Card RE facility that allows for an application to transfer the ownership of objects to an other application. Only instances of transferable classes can have their ownership transferred.

UNIFORM RESOURCE IDENTIFIER (URI)

A compact string of characters used to identify or name an abstract or physical resource. See RFC 2396 for more information.

VOLATILE OBJECT

Volatile objects and their values do not persist across card session; that is, they are lost when the card loses power. A volatile object is garbage collected on card tear (or reset).

WEB APPLICATION

A collection of servlets, HTML documents, and other Web resources that might include image files, compressed archives, and other data. A Web application is packaged into a Web application archive — a Web application module.

WEB APPLICATION CONTAINER

Contains and manages Web applications and their components (for example, servlets) through their lifecycle. Also provides the network services over which HTTP requests and responses are sent and manages security of Web applications.

Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 USA **Phone** 1-650-960-1300 or 1-800-555-9SUN (9786) **Web** sun.com

SunWIN #541097 Lit. #SWWP14463-0 08/08