The E Extensions to Java

An White Paper

Electric Communities has developed a set of extensions to Java, called E, to help developers write distributed applications. E combines Sun's Java programming language with some powerful enhancements: communications extensions for the development of distributed applications; optimistic computation, a powerful method for reducing the effect of communications latency in distributed systems; and an improved security model based on capability semantics and public-key cryptography, permitting fine-grained control over access to system resources without sacrificing security.

E is intended for programmers writing network applications, including databases, groupware, publishing systems, and multimedia.

What is E?

E is a programming language designed for developers who write distributed applications. It builds on the strength of Sun's Java language, an open standard that already provides some flexibility for developers writing World Wide Web applets. However, developers are finding that Java is not well suited to more ambitious distributed applications because Java's incomplete security model requires unacceptable trade-offs between expressive power and safety.

The E language improves on Java's security model and provides other powerful communications-oriented features. These include:

- Distributed communication: E features a built-in model for remote object communications, allowing an object to send messages directly to objects that exist in other machines.
- Capability semantics: a sophisticated security model that allows for extremely detailed control over sensitive functions within a single machine or across a network.
- Optimistic computation: a powerful method for reducing the effect of communications latency in distributed systems.

Distributed Communication

Distributed communication provides a single messaging facility that allows an object to send a message to another object, even if the other object is in a different machine on the network. The E language lets a programmer automatically establish connections between objects without having to deal directly with the underlying network protocols.

Capability Semantics

E’s capability semantics provides a security model based on the use of capabilities, which grant permissions to send particular messages to particular objects in a distributed system.

E’s security model is quite different from Java’s. Java’s basic assumption is simple: all code that originated on the local hard disk drive is trusted; all code that originated from the network is not trusted. Based on this model, Java provides system security by checking each time a "potentially dangerous" method is invoked; if no untrusted class is on the stack, the method is allowed to execute.

This all-or-nothing approach can be secure, but it has some basic weaknesses. A Java applet that establishes network connections, reads a configuration file, or manipulates a database, must first be downloaded to the user’s local hard disk. Once it is local, that code is automatically assumed to be completely trustworthy. It is entrusted with resources that it might not need, which can help foster viruses and trojan horses. It is not possible in Java to assign access selectively, on an as-needed basis. In order to grant some power to an applet, all powers must be granted.

In contrast to Java’s all-or-nothing model, E’s approach to system protection is flexible and general. E’s security model is built on its distributed communications and capabilities semantics features. Security is maintained by restricting the ability of objects to send specific messages to specific objects. It is impossible for an object to send an unauthorized message. Once the capability to reference an object or send a message has been granted, no further run-time check is required.

Capabilities are granted to E objects at load time. A cryptographic signature and a set of capability requirements are attached to every E object. The E object can have several kinds of "trustedness," independent of whether it exists on the local hard disk or whether it originated on the network. Based on the trustedness of the signature, the set of capabilities is granted or refused.

For example, you would not want to give a relatively untrusted application the ability to write anywhere in the file system. But you might want to give that application the ability to add a limited number of records to a database, and nothing else. By creating a trusted object that implements limited capabilities, and giving the untrusted application the ability to send messages to this new object, you can obtain more interesting functionality from the untrusted code without compromising security.

Optimistic Computation

Optimistic computation helps solve the troublesome problems of deadlock and race conditions which plague distributed systems. E’s implementation of optimistic computation lets your program send a message to an object requesting a value, and still continue executing the rest of the code "optimistically" as if it already has the answer it needs.

Optimistic computation simplifies design and increases reliability while decreasing network latency. E implements this by adding channels to its distributed communications model.

E classes rely on sending messages rather than on the traditional call/return semantics of Java. Messages do not have return results; they are merely sent. When an object actually requires a value as the result of sending a message to another object, it includes a channel as one of the message parameters. The channel acts as a container into which the result will be placed. The object providing the channel uses it as a reference to the return value and can begin sending messages to it immediately, even before the actual value has been determined. Thus, the sender of the message can continue executing without stopping to wait for the result. E also includes an exception handling mechanism to deal with bad messages or return values that never materialize.

By allowing computation to continue, more messages can be sent over the network in a given period of time, allowing for better utilization of the available bandwidth. Because many message roundtrips can now overlap, total latency is effectively reduced.

Another advantage of optimistic computation is the elimination of unnecessary network roundtrips. As a protocol is carried out, channels can be nested within channels so that the anticipated result can be altered "through the pipeline," as it were, before it is actually received. A series of messages with commands such as "multiply these," "add one to the result," "send your result to the following object" can be sent through a series of channels without ever requiring a packet to be received back across the network. Thus, latency is reduced because fewer network round trips are made.

E’s message passing semantics eliminates deadlocks and the need for locking. Only one method at a time may be executed by a single instance of an E object, if that object contains mutable state information.

By operating under these assumptions, the E environment guarantees that object state information will remain consistent and that processes can execute to completion.

E Classes and Java Classes

E classes can interoperate with standard Java classes. The coding of E methods is similar to Java method coding, with some syntactic extensions. E classes can contain ordinary Java methods internally.

The E environment still utilizes Java’s call and return semantics for communication between Java objects. Between E objects, however, optimistic computation is used. E classes can encapsulate Java code, enabling it to interact with other E objects optimistically.

Product Features

The E language is implemented as a compiler that maps E source code directly into Java byte codes. Java and E code can coexist in the same program--you can write Java classes in an E program, and implement private Java methods within E classes. Since E is built on Java, you can take full advantage of E features and yet easily integrate it with your Java code.

The E product includes a runtime library that supports E features such as security management, object persistence, communications, and access-controlled objects. There is also a unique distributed garbage collector that works with local collection systems to identify and collect obsolete cycles of objects on the network.

E also is implemented with a capability-based security infrastructure for certificates, called the Electric Communities Trust Manager (ECTM), that goes far beyond simple digital signatures. Implemented with the Java class loader, the ECTM lets users specify and enforce highly complex trust relationships governing the load-time behavior of Java classes--class importing, subclassing of other classes, and package membership. This Trust Manager is part of the E runtime, but is also a stand-alone product that can be added to any licensed Java implementation.

The E product also includes an example suite, which demonstrate E's unique feature set. The product documentation includes example source code and conceptual explanations.

Summary

E adds this set of powerful features to the popular Java language, transforming it from an applet language to a language for distributed applications:

- Transparent object-to-object communications
- Non-blocking message send
- Automatic connection management
- Access controlled messages and objects
- Cryptographic capability management
- Deadlock and race avoidance
- Latency compensation
- Interoperability with Java
- Orthogonal persistence
- Distributed garbage collection

Remember that E allows developers to create secure applications, but it cannot guarantee security on its own. The features that E provides must be utilized with care and common sense on the part of developers.

About Electric Communities

Founded in 1993, Electric Communities is a software development company based in the Silicon Valley. The founders of Electric Communities are leaders in the technology and sociology of human interaction in computer-mediated environments. We have a history of developing and delivering first-of-their-kind products, including Lucasfilm's Habitat, the first graphical online community, and AMiX, the American Information Exchange, the first online information marketplace. Our goal is to create comfortable, extensible environments for online entertainment and commerce.

Electric Communities

10101 N. De Anza Boulevard, Suite 100
Cupertino, CA 95014
408.342.9500
408.777.9200 fax
electric@communities.com
http://www.communities.com/