JAIN: A New Approach to Services in Communication Networks

John de Keijzer, Douglas Tait, and Rob Goedman, Sun Microsystems, Inc.

ABSTRACT

JAIN™, a set of integrated network APIs for the Java™ platform, provides a framework to build and integrate solutions (or "services") that span across packet (e.g., IP or ATM), wireless, and PSTN networks. The objective of JAIN is to provide service portability, convergence, and secure access (by services residing outside of the network) to such integrated networks. JAIN is defined and specified by a large number of participating communication companies (the JAIN Community), and according to a well-documented process (the Java Community Process or JCP). The objective of the JAIN Community is to create an open market for services across integrated networks using Java technology. This article is the first of a series in this issue of *IEEE Communications Magazine* and serves as an introduction to the other articles. The next section provides the JAIN business case. The article then summarizes how the JAIN Community works, and briefly introduces how the JAIN Community is organized. The next two sections are of a more technical nature and explain how separate JAIN work items fit together. In particular, we provide the rationale for the currently supported levels of abstraction (in terms of session/call signaling models) in JAIN. We look at possible implementation scenarios. Conclusions are contained in the last section. For more details on several of the JAIN Community work items, please refer to the other JAIN-related articles [1–3] in this issue of *IEEE Communications Magazine.*

THE JAIN BUSINESS CASE

Java computing starts with a highly structured, strongly typed, object-oriented computing language, which is compiled not to native machine code, but rather to a virtual machine "byte code." When ready for execution, the byte code is dynamically linked and then translated in a virtual machine to the native machine code. The dynamic linking and byte code translation ensures computer programming portability where a program or sections of a program can be written once on one platform and then run over any other platform that contains a virtual machine. This requires no recoding, recompilation, or relinking.

The high-portability feature of Java is balanced against performance and runtime semantic interface error checking. For performance, the byte code or source may be compiled to native machine code. For semantic interface checks, the dynamic linker may be replaced with a static linker. In either case, it is a trade-off between portability, performance, and runtime error checking. For the majority of applications this sacrificing of portability is not needed, and for the few exceptions the transition is simple.

JAIN builds on Java portability by standardizing the signaling layer of the communications networks into the Java language, and defines a communications framework for services to be created, tested, and deployed. The strengths of JAIN are in service portability, network convergence, and secure network access:

- Service portability: Technology and application development are currently constrained by proprietary interfaces. Portability of applications is almost nonexistent. This increases application development cost, time to market, and maintenance requirements. The JAIN approach is to reshape proprietary interfaces into uniform Java interfaces delivering portable applications.
- Network convergence: Call or session legs for most of today's applications and services typically span only a single type of network — public switched telephone network (PSTN), packet, or wireless — although clearly gateways between these networks do exist. The higher-level (discussed later) JAIN call models include facilities for observing, initiating, answering, processing, and manipulating calls, where a call is understood to include a multimedia, multiparty, multiprotocol session over the underlying integrated network.
- Secure network access: Communication applications and services run either inside the operator's trusted network or completely outside this network. The JAIN Parlay interface enables untrusted services, residing outside the network, to directly access network resources to carry out specific actions or functions inside the integrated network.

JAIN creates a new environment for developers and users of communication services to build systems on a set of standards guaranteed to run on conformant networks. The market opportunity for new services in such an environment has Internet-like growth potential. Merging the rapid service creation and deployment model in the Internet space and the proven quality of service of the PSTN creates a new market scenario.

JAIN further enhances the convergence of the Internet and PSTN by providing controlled access of untrusted services to the available

0163-6804/00/$10.00 © 2000 IEEE

functionality and intelligence inside the networks. Parlay (and JAIN Parlay) services become network-operator-independent, making it attractive to develop interesting services by service providers and third parties. (A detailed description of JAIN Parlay can be found in [3]).

The focus of the JAIN effort is to take the telecommunications market from many proprietary systems to a single open, distributed environment, not unlike the Internet today, able to host a large variety of services while still maintaining quality and reliability of service. By opening the network to Java applications, an opportunity is created to deliver thousands of portable, integrated services rather than the dozens currently available.

JAIN technology is based on Java component (Bean) software: components can be added, taken away, enhanced, assembled, shared, or redistributed (even geographically) in a dynamic running system. This allows services and features to be added, updated, and deleted in a live environment.

The removal of proprietary interfaces opens markets where network equipment providers (NEPs), independent software vendors (ISVs), protocol stack vendors, service providers, and carriers will be able to build and sell a variety of Java-technology-adapted components. Service providers and NEPs will then be able to select "best of breed" JAIN-conformant products from different vendors on the basis of functionality and value.

This *open value chain* market model stimulates the reuse of existing components and the development of additional or missing functionality — maximizing efficiency as well as innovation. It also opens the market to innovative new players. The next-generation architecture provided by JAIN creates a level playing field for deploying new services. This model is best served when all levels of the communications industry participate: hardware companies, stack providers, NEPs, and network and service providers. This is exactly what is happening in the JAIN Community.

How the JAIN Community Works

The Java Community Process (JCP) is a formal process for developing Java extensions. The JCP has shown to produce high-quality specifications in "Internet time" using an inclusive, consensus-building process that not only delivers the specification, but also the reference implementation and its associated suite of compatibility tests.

The best way to develop a specification is to start with a handful of industry experts who have a deep understanding of the technology in question, and then have a strong technical lead work with them to create a first draft. Consensus is then built using an iterative review process that allows an ever-widening audience to participate, and to see their comments and suggestions incorporated into successive draft versions of the specification prior to final release.

This formal process was designed to be fast, flexible, and adaptable to the wide variety of work styles present in the community today. An independent auditing firm may audit the process. All participants are required to sign a Java Specification Participation Agreement (JSPA) with Sun. The JSPA fundamentally assigns the copyright of the specification to Sun in order to enable Sun to enforce a single version of the specification at any time. Steps in specification development follow the guidelines set forth in the JCP. A summary is provided here:

- Propose a new specification through a Java Specification Request
- Form the Expert Group to write the draft specification
- Review and refine the Participant Draft by all JSPA Participants
- Public review
- Public release
- Maintenance

In building on the JCP, JAIN leverages the requirements for participants and the procedures for driving community approval on specifications. More information on the JCP can be found at http://java.sun.com/aboutJava/communityprocess.

The JAIN Community Organizational Structure

JAIN is a set of integrated network APIs for the Java platform and an environment to build and integrate JAIN components into services or applications that work across PSTN, packet (e.g., IP or asynchronous transfer mode, ATM), and wireless networks.

The JAIN approach integrates wireline, wireless, and packet-based networks by separating service-based logic from network-based logic. Thus, from this point of view, JAIN consists of two layers, application and protocol. This is illustrated in Fig. 1.

Based on the JAIN layered approach illustrated in Fig. 1, the JAIN program consists of two expert groups and several work groups:

- On the protocol layer, the Protocol Expert Group (PEG) standardizes interfaces to IP, wireline, and wireless signaling protocols. These protocols include TCAP, ISUP, INAP, MAP, SIP, MGCP, and H.323.
- On the application layer, the Application Expert Group (AEG) addresses recommendations and specifications for secure network access (JAIN Parlay), connectivity management, JAIN session/call control (JCC/JCAT), and a JAIN service creation and carrier grade service logic execution environment (JSC/SLEE).
- Work groups are engaged in either developing prototype implementations (e.g., of services or subspecifications) and/or feeding the insights gained back into the Expert Groups. Examples of active work groups are AT&T, KPN, CMG, Ericsson, and NTT Comware.

The JAIN Community structure is shown in Fig. 2.

Today, there are more than 30 companies actively participating at various levels in the JAIN Community.

JAIN Architecture

The protocol layer in JAIN is based on Java standardization of specific protocols (SIP, MGCP, H.323, TCAP, ISUP, INAP/AIN, MAP, etc.). By

JAIN builds on Java portability by standardizing the signaling layer of the communications networks into the Java language, and defines a communications framework for services to be created, tested, and deployed.

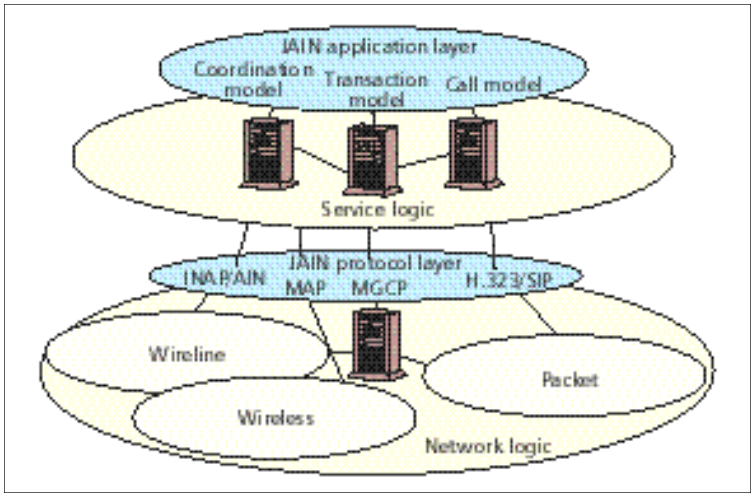

Figure 1. *The JAIN layered approach.*

providing standardized protocol interfaces in a Java object model, applications and protocol stacks can be dynamically interchanged and, at the same time, provide a high degree of portability to the applications in the application layer using protocol stacks from different vendors. A more detailed explanation of the ongoing work in the JAIN Protocol Expert Group is given in [1].

The application layer provides a single call (or session) model across all supported protocols in the protocol layer. The fundamental idea is to provide a single state machine for multiparty, multimedia, and multiprotocol sessions for service components in the application layer. This state machine is accessible by trusted applications that execute in the application layer through the JCC/JCAT API. The current proposal in the Edit Group is to use the core part of the Java Telephony API (JTAPI) as JCC. JCAT then augments JTAPI/JCC to provide a richer signaling model. This is explained in much more detail in [2]. The basic approach taken in JTAPI, with a core package and extension packages that extend the core package, will be followed by the JCC/JCAT Edit Group, and provides a structured and modular way to introduce enhancements to the call model.

The application layer also supports secure access to network resources accessible through JAIN APIs (i.e., the application layer call model or the individual protocols). This JAIN work item, JAIN Parlay, is the topic of [3].

Figure 3 illustrates the three basic abstractions supported by JAIN.

An application or service at the protocol level can talk directly to the JAIN adapters. These are Java class methods, callbacks, events, or Java interfaces that encapsulate the underlying resources. The resources may be implemented in Java, C, C++, and so on, but a JAIN-conformant protocol product does provide at least the relevant JAIN Java API. This lowest level of the JAIN abstraction does not provide any features to the application for dealing with different kind of protocols; for example, an application that needs a session spanning INAP and SIP will have to handle both protocols. But it does provide for the same application to run on top of protocol products from different vendors.

A service or application at the next level of

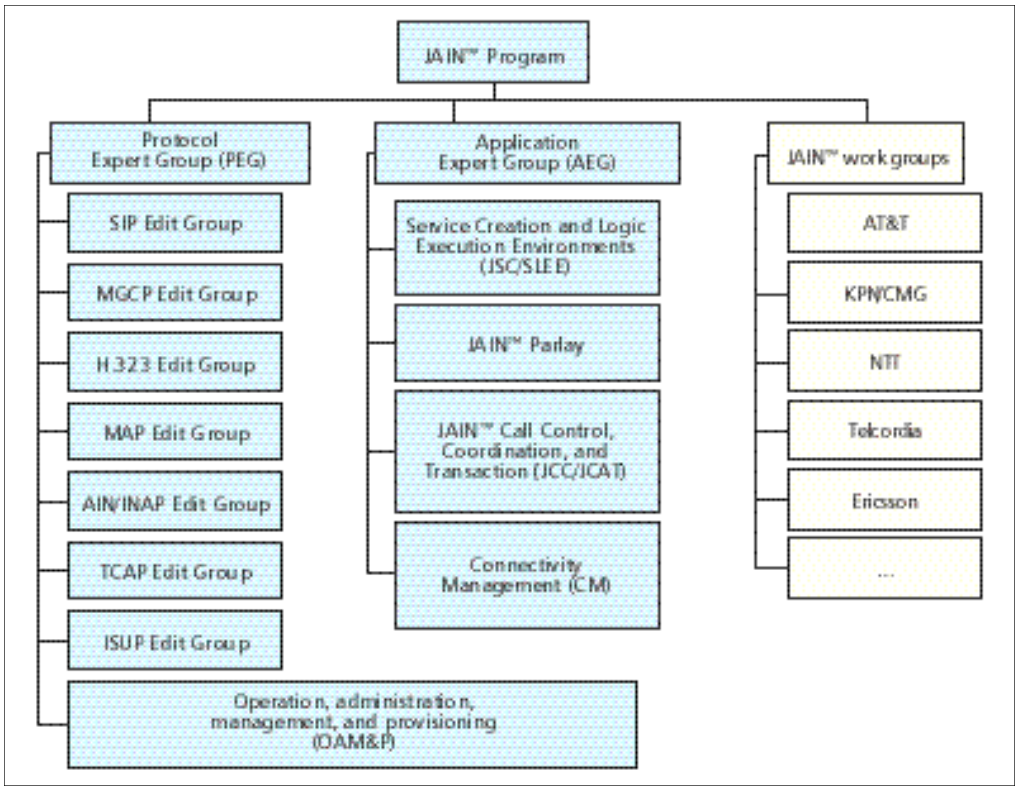

Figure 2. *JAIN Community organization.*

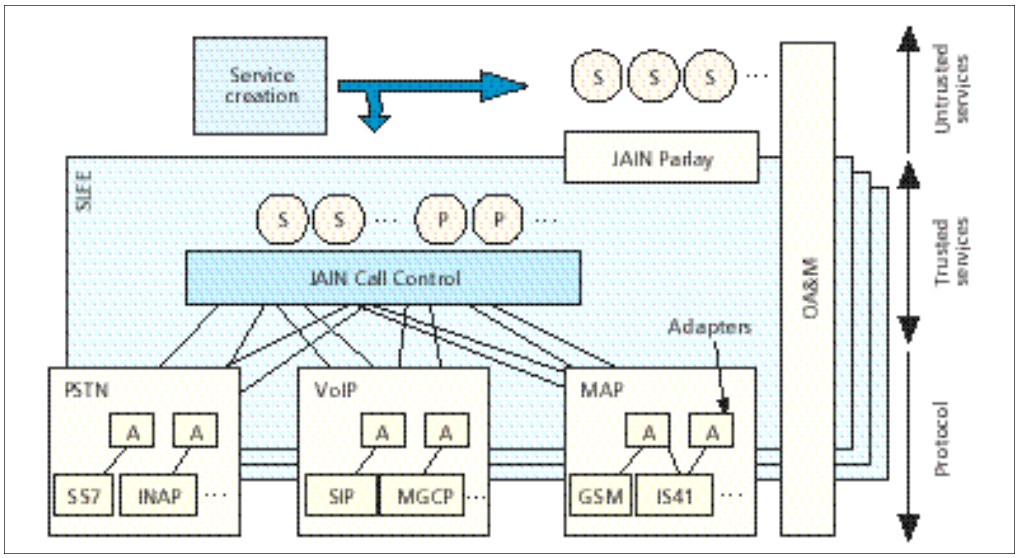

Figure 3. *JAIN abstractions.*

JAIN has chosen an evolutionary approach that over time will allow completely new approaches to be incorporated. Wherever possible the approach is decoupled and modularized to make this easier.

JAIN abstraction, the JAIN call control or trusted services level in Fig. 3, does not have to be aware that some of its session or call legs are using a different protocol. Furthermore, this is, from the point of view of a call or session, the richest call model available in JAIN [2]. A vendor providing a JCC-conformant product will have to provide a mapping from JAIN call control to one or more protocol adapters.

In Fig. 3, the little disks inside the SLEE labeled with either an S (for service) or a P (for policy) are the core JAIN components. They are constructed according to a component model defined by the JAIN Service Creation (JSC) and Service Logic Execution Environment (SLEE) Edit Groups. Instantiations of component assemblies run inside the JAIN SLEE, which is schematically illustrated in Fig. 3. This topic is further discussed in the last section.

JAIN provides a third level of abstraction through the JAIN Parlay interface. In Fig. 3, the little disks outside the SLEE rectangle represent JAIN-Parlay-based services. Earlier in this document these were referred to as untrusted services. JAIN Parlay acts as a firewall to protect the security and integrity of the integrated network. Some operators might opt to have all services, both inside and outside their integrated network domain, use the JAIN Parlay interface. The JAIN Parlay interface exports some or all capabilities available inside the integrated network to services running inside a different security domain. A detailed description of JAIN Parlay can be found in [3] (see also http://www. parlay.org).

Examples of capabilities addressed by JAIN Parlay are security, a generic call control service (GCCS), a short messaging service (SMS), mobility features, a discovery service, and so

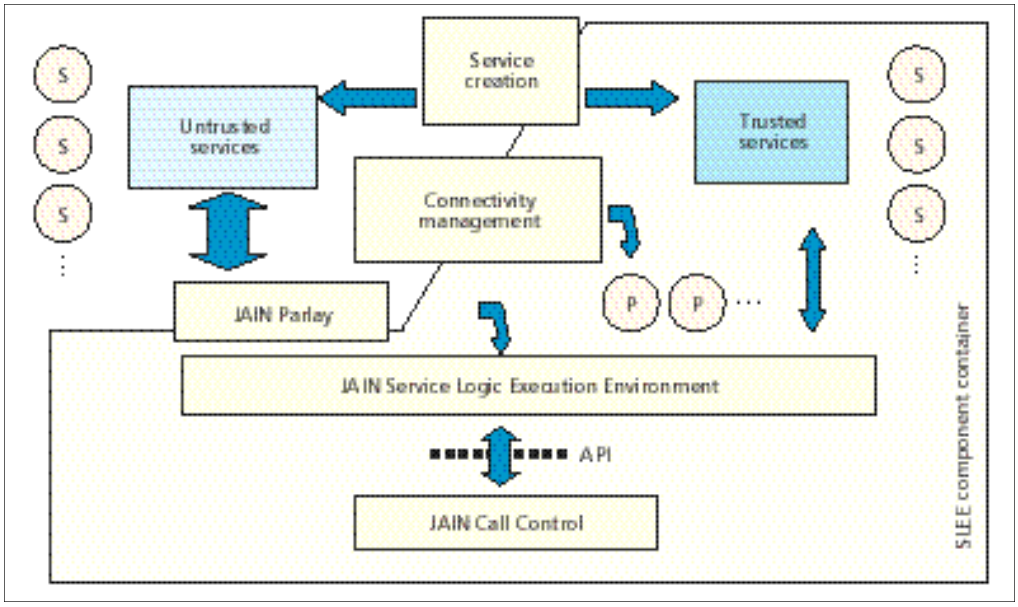

Figure 4. *JAIN Service Logic Execution Environment.*

WHAT IS JMX?

JMX is an effort under the JCP to define appropriate management extensions for the Java platform. JMX provides for the instrumentation level of elements (both software and hardware), an agent layer to group instrumentation-level management entities, a manager level to ease and consolidate distributed agents, and interfaces to management applications (including SNMP and TMN-based approaches).

A JMX manageable resource is one that has been instrumented in accordance with the JMX Instrumentation Level Specification. It can be a business application, a device, or the software implementation of a service or policy. A Managed Bean, Mbean for short, is a Java object that represents a JMX manageable resource. MBeans follow the JavaBean™ component model (see the box "What Is a Component Model?" below) closely.

A JMX agent is an Mbean Server/Container. It will have at least one connecter or adaptor to communicate (e.g., using HTTP, RMI, or SNMP) and may contain 1 or more management services, also represented as MBeans. The Mbean Server is a registry for MBeans in the agent and also allows for manipulation of those MBeans.

The JMX Manager provides one or more interfaces for management applications to interact with the agent, distribute or consolidate management information, and provide security. A JMX Manager can control any number of agents, thereby simplifying highly distributed and complex management structures.

WHAT IS A COMPONENT MODEL

So what is a (software) component model? It is a set of principles that define how solutions can be built from smaller (software) entities. In Java these smaller entities are Beans, and examples of currently available beans are JavaBeans, Enterprise JavaBeans, Federated Beans, and MBeans. Here I will refer to them as *-Beans. The principles cover many aspects of the entire lifecycle of the *-Beans, such as how individual beans can be assembled into larger entities and ultimately into complete solutions, how entities are visible in development tools, how *-Beans are deployed when they are needed in a running system, how many instances are needed to obtain the right level of performance, if the *-Beans are location-independent (in a local network or even in a wide-area network), and if the *-Beans handle concurrent requests, transactions, and persistence.

forth. A vendor of JAIN Parlay provides a mapping of the GCCS capabilities onto JCC and a mapping of the JAIN Parlay security features onto the security mechanisms supported by the JAIN SLEE (see the last section). A full-blown JAIN SLEE product also supports the other capabilities exported through the JAIN Parlay interface. From a JAIN Parlay service point of view, JAIN Parlay executes on the same host machine and inside the same Java Virtual Machine (JVM) as that service.

JAIN Parlay's GCCS is roughly based on the JTAPI core functionality [4]. JCC is significantly enhanced over JTAPI through the Java Coordination and Transaction (JCAT) extension. Figure 3 also shows the vertical bar labeled OA&M. These are a set of APIs that will provide for operational, administrative, and maintenance aspects of a JAIN environment [5] JAIN relies on the JMX effort in this area (see the box "What Is JMX?").

THE JAIN SERVICE LOGIC EXECUTION ENVIRONMENT

The relationship between the four AEG Edit Group work items and the JAIN SLEE is depicted in Fig. 4.

Services can be written directly on top of the JAIN protocol layer adapter APIs, JCC API, or JAIN Parlay API. This will provide portability of these services. Services will require and use much more than just a call model, though; for example, they might want to use Java's JDBC, text-to-speech, and JNDI APIs, to mention a few.

Furthermore, JAIN's SLEE provides portable support for transactions, persistence, load balancing, security, object and connection instance pooling, and so on. This is similar to Enterprise JavaBeans in the EJB specification [6]. The JSC

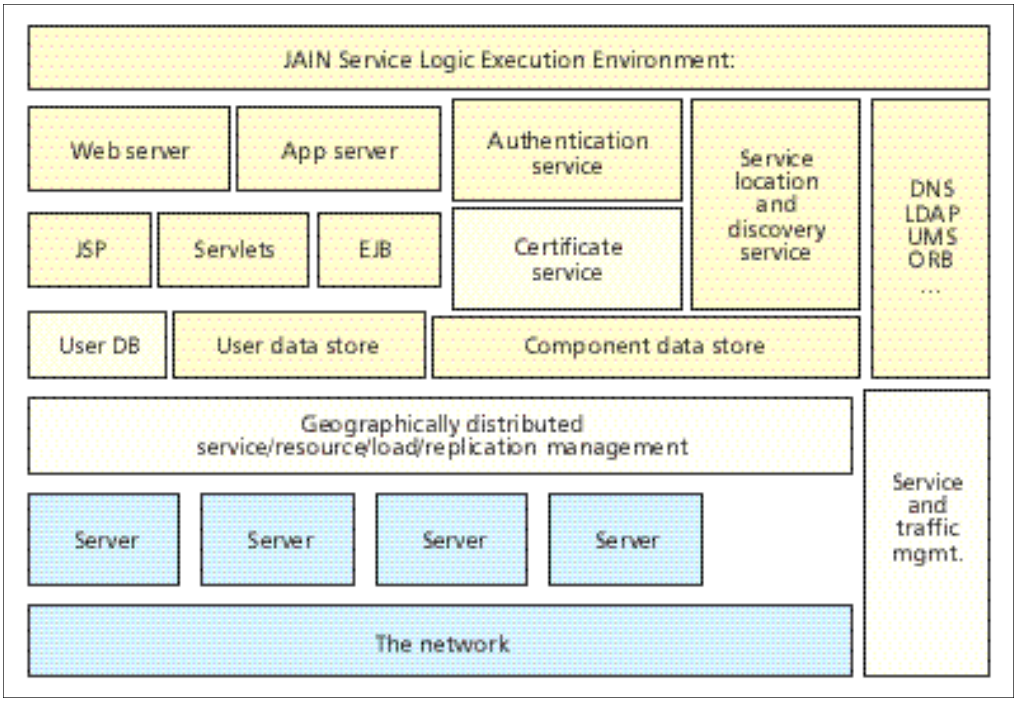

Figure 5. *An example of an EJB-based JAIN implementation.*

and SLEE focus on a component model (see the box titled "What Is a Component Model?") over specific implementations. EJBs (see the box on "The EJB Component Model") are an excellent candidate for such a component model, but it is certainly possible to use other component models, such as Jini™ [7], Jini JavaSpaces [8], Jiro™ [9] or JES [10]. Most of these container structures, extended with the right features, are viable implementation platforms for JAIN.

Figure 5 gives an example of a possible environment for an EJB-based JAIN implementation. In the figure the application server provides one or more types of containers for different types of EJBs. The EJBs are the building blocks for services. An implementation of the JAIN SLEE can be considered an integrated network application server.

The component model as defined for the SLEE also supports distributed implementations where portions of the SLEE are migrated to the edge of the network, say, to run on a residential gateway or even inside a consumer device.

Conclusion

The JAIN network topology provides carriers with the ability to deploy next-generation network services on devices inside or at the edge of the integrated network, including any Java-enabled end-user device. Furthermore, support for all the necessary telephony protocols that are used between the different network elements in intelligent networks, advanced intelligent networks (AIN), and IP-based (telephony) networks is mandatory. JAIN has chosen an evolutionary approach that over time will allow completely new approaches to be incorporated. Wherever possible the approach is decoupled and modularized to make this easier. A concrete example is the adoption of the JTAPI core/extension package structure which does not make all call models depend on a single state machine.

JAIN-compliant components do not reside on a single server; rather, functionality is implemented as a multitier, distributed application on all signaling network elements. Such an approach provides significant advantages for scalability, performance, reliability, manageability, reusability, and flexibility. The JAIN architecture leverages the platform independence of Java. However, it is also designed to be distributed and independent of any specific protocol or middleware infrastructure.

JAIN technology is changing the communications market from many proprietary closed systems to a single network architecture where services can rapidly be created and deployed.

For more information, including availability of completed portions of the specifications, please consult the JAIN Web site: http://java.sun.com/products/jain.

The EJB Component Model

The best known *-Beans are JavaBeans and Enterprise JavaBeans (EJBs). In the management domain MBeans have been around for several years. JavaBeans was the first software component model added to the Java world, and its primary focus was on reusable components, components that can be visually manipulated and customized in (GUI) development environments to assemble applications using introspection and a runtime event model to provide notification and callbacks.

Enterprise JavaBeans are focused on the enterprise problem domain. It delivers a server side software component model that supports the typical requirements of a database/transaction environment, such as access control, CORBA interoperability, object transaction monitors, availability, and load-balancing. Each EJB does not provide all these facilities by itself; it relies on a so-called container for these services. This is exactly the strength of a *- Bean/container model: The developer of the *-Bean can focus on expressing the service logic without having to worry about usually complicated system-level issues (e.g., related to high availability and load balancing).

Thus, one of the differences between JavaBeans and EJBs is that JavaBeans can execute directly on the Java Virtual Machine. EJBs need a container to live in. In many cases it is attractive, though, to provide a container or agent for JavaBeans as well (e.g., to handle lifecycle issues or translating JavaBean events into some other event mechanism such as SNMP traps). This is the case in JMX, and the currently available reference implementation of JMX (see the box "What Is JMX").

References

[1] R. R. Bhat and R. Gupta, "JAIN Protocol APIs," *IEEE Commun. Mag.*, this issue.
[2] R. Jain, F. M. Anjum, P. Missier, and S. Shastry, "Java Call Control, Coordination, and Transactions," *IEEE Commun. Mag.*, this issue.
[3] S. Beddus, G. Bruce, and S. Davis, "Opening Up Networks with JAIN Parlay," to appear, *IEEE Commun. Mag.*, Apr. 2000.
[4] Call Control Interoperability Working Group, Enterprise Computer Telephony Forum: http://www.ectf.org/ectf/tech/ccwg.htm
[5] Java Management Extensions: http://java.sun.com/products/JavaManagement
[6] Enterprise JavaBeans: http://java.sun.com/products/ejb
[7] Jini: http://java.sun.com/products/jini
[8] JavaSpaces: http://java.sun.com/products/JavaSpaces
[9] Jiro: http://www.jiro.org
[10] JES: http://www.sun.com/software/embeddedserver/index.html

Additional Reading

[1] JTAPI: http://java.sun.com/products/jtapi

Biographies

JOHN DE KEIJZER (john.dekeijzer@holland.sun.com) is the Integrated Networks strategist and JAIN program manager for Sun Microsystems. Working at Vicorp, Tandem, and now Sun Microsystems, he was the chief architect for intelligent network and enhanced networked service solutions in Europe, the United States, and Asia/Pacific. At Sun, he is responsible for implementing a unified strategy for open systems for next-generation telephone and data networks.

DOUGLAS TAIT (douglas.tait@East.Sun.com) received his B.S. in computer sciences from Temple University and his M.S. in computer architecture and network design from the University of Pennsylvania. His computer experience includes companies such as Unisys, Telesciences, General Electric, and Martin Marietta. He spent several years developing device drivers for SS7 protocol stacks and eventually managed AIN projects at MCI and Sprint. One of the original JAIN architects, he is driving the standardization of Java interfaces in the communications industry and providing solutions on Sun platforms.

ROB GOEDMAN joined the Dutch subsidiary of Sun in 1986 and transferred to the United States in 1990. He worked at Sun's FirstPerson on the blending of graphical user interfaces and digital video for TV based on the Oak language (Oak later became Java). He was responsible in the Sun Thomson Alliance for a team that investigated incorporating Java into set-top boxes. He subsequently led several engineering teams, one of which delivered Sun's implementation of a proxy cache appliance with high scalability and failover. Since mid-1998 he has been a member of the JAIN team, initially responsible for bringing the JAIN initiative in line with Sun's Java Community Process.