

## An Introduction to JAIN SLEE

A Service Logic Execution Environment (SLEE) is a high throughput, low latency event processing application environment.

JAIN SLEE is the Java open standard for a SLEE and is designed to allow implementations of the standard to meet the stringent requirements of communications network signalling applications. The JAIN SLEE specification is designed so that implementations can achieve scalability and availability through clustering architectures.

JAIN SLEE is the only industry standard aimed at portable communications applications, i.e. a communications application can be written once and run on many different implementations of JAIN SLEE. Applications can access resources and protocols across multiple networks from within the JAIN SLEE environment. The JAIN SLEE specification allows developers to write robust components as it integrates the well known ACID properties of transactions into the programming model. Components can be composed to solve more complex problems.



### FUNDAMENTAL CONCEPTS OF JAIN SLEE

**JAIN SLEE is a standard produced using the Java Community Process.**

The JAIN SLEE specification includes a component model for structuring the application logic of communications applications as a collection of reusable object-orientated components, and for composing these components into higher level and more sophisticated services.

The SLEE architecture also defines the contract between the components and the container that will host these components at run-time. Applications may be written once, and then deployed on any application environment that implements the SLEE specification.

The SLEE specification also defines the management interfaces used to administer the application environment and also defines a set of standard Facilities (such as the Timer Facility, Trace Facility, and Alarm Facility).

The SLEE architecture also defines an extension framework to allow new external protocols and systems (such as MSCs, MMSCs, SMSCs, Softswitchs, CSCFs, HLRs) to be integrated quickly for use by applications in the SLEE.

#### Service

A service in JAIN SLEE terminology is a managed field replaceable unit. The system administrator of a JAIN SLEE controls the life cycle (including deployment, undeployment and on-line upgrade) of a service. The program code can include Java classes, Profiles, and Service Building Blocks.

#### Profile

A JAIN SLEE Profile contains provisioned service or subscriber data. Service Building Blocks running inside the JAIN SLEE may access profiles as part of their application logic.

#### Service Building Block

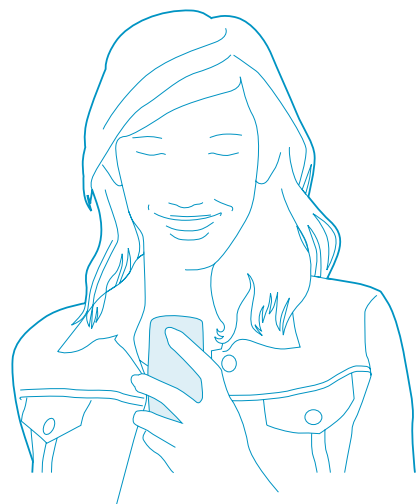
The element of re-use defined by JAIN SLEE is the Service Building Block (SBB). An SBB is a software component that sends and receives events and performs computational logic based on the receipt of events and its current state. SBBs are stateful. The program code for an SBB is comprised of Java classes.

#### Event

An event represents an occurrence that may require application processing. An event may originate from a number of different sources, for example, an external resource such as a communications protocol stack, from the SLEE itself, or from application components within the SLEE.

#### Resources and Resource Adaptors

Resources are external entities that interact with other systems outside of the SLEE, such as network elements (HLR, MSC, etc), protocol stacks, directories and databases. A Resource Adaptor implements the interfacing of a Resource into the JAIN SLEE environment.



“OpenCloud’s leadership of JAIN SLEE with Sun Microsystems is seen by Gartner as “a visionary contribution to the software industry, advancing event-driven application servers (EDASs) and event-driven architecture (EDA).” 2006

## FACTORS INFLUENCING THE DESIGN OF JAIN SLEE

The following sections discuss some of the drivers that influenced the design of the JAIN SLEE specification for **telecommunications applications with demanding performance, availability and operational requirements.**

### Importance of Reliable Software

Continuous availability is a primary goal of communication services and can only be achieved if the time to detect and recover from a failure is low. The computer industry has historically been concerned about hardware failures. But software failures are a significant cause of downtime (40%), indicating a clear requirement for software architectures that address software failures.

### Next Generation Services

The need to introduce new services across telephony networks more rapidly has been driving significant change over the recent years. Significant problems exist in the development of Next Generation services:

1. Service Portability
2. Network Convergence
3. Secure Network Access

### Application Development

In order to satisfy performance and availability requirements, the application programmer must have access to APIs that ensure their application logic:

- Is simple
- Functions through host and process failures
- Executes independent of particular computing nodes
- Executes concurrently

Implications on Application (software) components in Software architecture:

- Rely on a defined failure model in order to simplify application behaviour through process and host failure
- Do not need to explicitly manage concurrency
- Can have their state replicated
- Have clearly defined state synchronization points

Implications on the Application Server:

- Migrates application components between processing nodes in the system as particular processes and nodes may fail
- Manages concurrent execution of application components
- Prevents faulty application components from affecting other independent application components
- Allows application components to be up

Implications on the SLEE Specification:

- Event based model, asynchronous, support for composition
- Container manages component state
- Container manages garbage collection of components
- Transaction boundaries for demarcation and semantics of state replication
- Strongly typed event handling signatures
- 3rd party event driven components
- Management of lifecycle of Server, Services, Provisioned state
- Versioned services, upgrade of services, existing activities stay on existing service instances, new activities are directed to instances of upgraded services
- Independent of network technology/ protocols/elements through resource adaptor architecture

### Use a Best of Breed Programming Model

- Simple, Object Orientated, programming model
- Support for 3rd party application components and application development
- Application server manages application state
- Transactional
- Support independent units of work
- Possible to naturally model asynchronous systems (applications)
- Network independent

### Standards Based

*“Adoption of a low-cost infrastructure for telecommunications will likely open opportunities for new services (applications), forbiddingly expensive today. The impact of the new services in a standards-based converged network will be broader and faster than it can be today, ...” [Gartner, March 2006].*

*“Open, standard APIs hide the complexity of networks from the application layer, and the use of standard signalling and call control protocols are the keys to providing flexibility and creativity for the next-generation networks enhanced services.” [Yankee Group].*

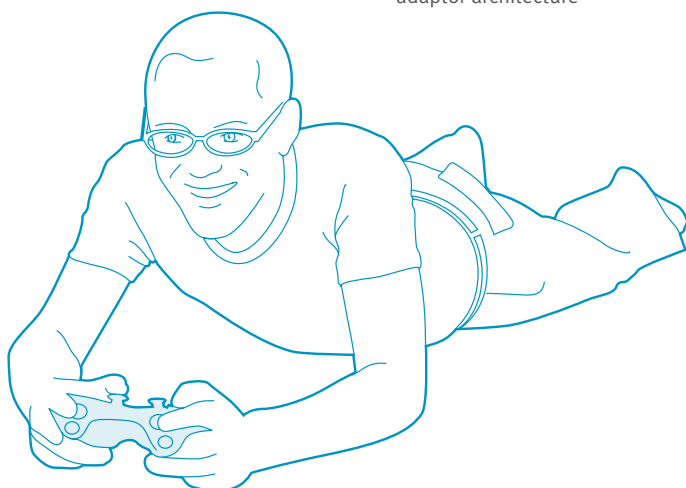
### Network Independence

Existing legacy equipment constitutes a significant investment by Network Operators. Transitional network architectures are likely to be adopted as Network Operators embrace new technology. Therefore, it must be possible to deploy applications in the SLEE application environment that use diverse network resources and signalling protocols.

The integration of a new type of network element, signalling protocol, or external system must not require changes to the core software architecture of the application server. This requirement is satisfied by a Resource Adapter Framework that supports integration of network resources.

Example resources include:

- SIP/ISC
- Diameter Base, CCA, Sh, Ro, Rf ...
- SMPP, MM7, HTTP, SOAP ...
- CAMEL (CAP), TCAP, MAP, INAP, AIN, WIN ...
- OSA/Parlay



## Portability

Network Equipment Providers and Network Operators have preferred hardware and Operating System platforms. Applications running in the JAIN SLEE application environment must have the following characteristics:

- Applications must be able to execute on different compliant platforms without change.
- Application source code must not require modification when moving between compliant platforms.
- Applications must be isolated from hardware architecture and systems level APIs (for example POSIX, WIN32, etc)

## APPLICATION CHARACTERISTICS DRIVE JAIN SLEE DESIGN

**Different implementations of JAIN SLEE provide different availability guarantees, have different cluster architectures, provide different ACID semantics, and exhibit different latency and throughput performance.**

In the following comparison we use the typical characteristics of ‘Event-Driven Applications’ and ‘Enterprise Applications’ as an example to outline the characteristics that different containers provide.

Event-Driven Applications	Enterprise Applications
<b>Invocations</b>	
<ul style="list-style-type: none"><li>– Typically asynchronous</li><li>– Events such as protocol triggers</li><li>– Events occurrences mapped to method invocations</li></ul>	<ul style="list-style-type: none"><li>– Typically synchronous</li><li>– Database, EAI systems</li><li>– RPC Calls</li></ul>
<b>Event Granularity</b>	
<ul style="list-style-type: none"><li>– Fine-grained events</li><li>– High Frequency</li></ul>	<ul style="list-style-type: none"><li>– Course-grained events</li><li>– Low Frequency</li></ul>
<b>Components</b>	
<ul style="list-style-type: none"><li>– Light-weight fine-grained objects</li><li>– Short transient lifetimes</li><li>– Rapid creation, deletion</li></ul>	<ul style="list-style-type: none"><li>– Heavy weight data access objects</li><li>– Long persistent lifetimes</li></ul>
<b>Data Sources</b>	
<ul style="list-style-type: none"><li>– Multiple data sources</li><li>– Location, context information</li><li>– Provisioned data, cached from master copy</li></ul>	<ul style="list-style-type: none"><li>– Database servers</li><li>– Definitive master copy</li><li>– Back-end systems</li></ul>
<b>Transactions</b>	
<ul style="list-style-type: none"><li>– Light-weight transactions</li><li>– For state replication demarcation</li><li>– Faster completion and more frequent</li></ul>	<ul style="list-style-type: none"><li>– Database transactions</li><li>– Slower completion and less frequent</li></ul>
<b>Computation</b>	
<ul style="list-style-type: none"><li>– Compute-intensive</li><li>– Processing is resource invocations and events</li></ul>	<ul style="list-style-type: none"><li>– Database access intensive</li><li>– Database, EAI systems</li><li>– RPC Calls</li></ul>

## JAIN SLEE AND THE JAVA COMMUNITY PROCESS

**The Java Community Process (JCP) is the mechanism by which JAIN SLEE became a recognised standard. This iterative process involved regular review of the proposed technology both by members of an expert group and by interested members of the public.**

Standards produced by way of the JCP are required to contain the following components:

### Specification Documents

The specification documents can be downloaded from the Java Community Process website

### Technology Compatibility Kit (TCK)

The TCK is a collection of tests that a SLEE implementation must pass in order to be certified JAIN SLEE compliant. The tests cover a wide range of behaviours from transactional correctness to verifying that API methods throw the specified exceptions when given specific parameters. The tests are designed to test both every day use cases, as well as unusual (but covered by the specification) use cases.

JAIN SLEE implementations are required to pass these tests. JAIN SLEE application developers can therefore write and test applications on one JAIN SLEE implementation with full confidence that the application will execute correctly on another JAIN SLEE implementation, without requiring modifications to the application code.

### Reference Implementation (RI)

The JAIN SLEE Reference Implementation is a fully functioning implementation of the JAIN SLEE that can be used to build and deploy services. It was implemented using J2EE technology and may be useful to JAIN SLEE developers. Services that deploy on the reference implementation will deploy on any other JAIN SLEE compliant implementation.

### JAIN SLEE Standards Compliance

In order to claim compliance to JAIN SLEE, vendors must exercise their implementation against the JAIN SLEE Technology Compatibility Kit, pass all tests and publish their results.

## Further Information related to JAIN SLEE

### Specification Documents

The specification documents are downloaded from the Java Community Process website.

<http://jcp.org/en/jsr/detail?id=240>

### Technology Compatibility Kit (TCK)

The SLEE TCK is licensed under the **OPENCLOUD COMMUNITY SOURCE LICENSE**. The current JAIN SLEE TCK 1.0. can be downloaded from the OpenCloud website at <http://www.opencloud.com>

### Reference Implementation (RI)

The JAIN SLEE Reference Implementation was built by OpenCloud and is available from Sun Microsystems at <http://java.sun.com/products/jslee/1.0/download.html>

### JAIN SLEE Tutorial

This is presentation material of a JAIN SLEE tutorial given by JAIN SLEE experts (including the expert group leads David Ferry from OpenCloud and Swee Lim from Sun Microsystems). It presents an overview of JAIN SLEE concepts and the JAIN SLEE programming model. It can be downloaded from the OpenCloud website at <http://www.opencloud.com>

## About OpenCloud

OpenCloud was formed in New Zealand in 2000 to create open standard software technology that would revolutionise the portability and interoperability of services in telecommunications specifically in the evolution to IP and 3G IMS. OpenCloud works with partners to deliver, integrate and support end-to-end solutions incorporating OpenCloud products to network operators and service providers worldwide. OpenCloud has offices in: UK, New Zealand, Madrid, Tokyo, and San Francisco.

## More Information

### General Enquiries

Email us at [info@opencloud.com](mailto:info@opencloud.com)  
or visit us at [www.opencloud.com](http://www.opencloud.com)

### OpenCloud HQ (UK)

140 Cambridge Science Park  
Milton Road  
Cambridge CB4 0GF  
United Kingdom

**TEL:** +44 1223 228490

**FAX:** +44 1223 228491

### The JAIN SLEE Standard

<http://jainslee.org/>

<http://jcp.org/en/jsr/detail?id=22>

<http://jcp.org/en/jsr/detail?id=240>

<http://archives.java.sun.com/jain-slee-interest.html>