SIP Specifications and the Java™ Platforms

The look and feel of SIP!

Author: Phelim O'Doherty – Sun Microsystems, Inc.
Contributors: Anders Kristensen – dynamicsoft
Chris Bouret – Nokia Corporation
Michael O'Doherty – Nortel Networks

For more information on Java API specifications for SIP, please contact JAIN-Interest@sun.com.

1 Introduction

The goal of this whitepaper is to identify the specifications to the Session Initiation Protocol (SIP) [1] defined through the Java Community Process^{SM} (JCP^{SM}) [2]. This document inspects each Java SIP specification describing the functionality, architecture, platform supported and how each interface relates to the other Java interfaces to SIP. This is not a tutorial on SIP or the Java language [3] and it is assumed that the reader has a reasonable understanding of both the Java language and the SIP protocol and their potential functions within an IP Communications network.

The Java language defines three platforms, namely Java 2 Platform, Standard Edition (J2SE™), Java 2 Platform, Enterprise Edition (J2EE™) and Java 2 Platform, Micro Edition (J2ME™). Targeting the three Java platforms are four Java Specification Requests (JSRs) for SIP. These are:

- JAIN™ SIP [4]: JSR 32 defines the JAIN SIP specification. Sun Microsystems is the specification lead along with the following expert companies and individuals: British Telecom, Cisco Systems, dynamicsoft, Ericsson, Open Cloud, Telcordia Technologies, Ranganathan Mudumbai, and Ulticom.

- SIP Lite [5]: JSR 125 defines the SIP Lite specification. Ubiquity is the specification lead along with the following expert companies and individuals: British Telecom, dynamicsoft, Hughes Software Systems, Nokia Corporation, Nortel Networks, Panasonic, PingTel, Ranganathan Mudumbai, Sun Microsystems and Tata Consultancy Services.

- SIP Servlet [6]: JSR 116 defines the SIP Servlet specification. dynamicsoft is specification lead along with the following expert companies, 8x8, Ericsson, IBM, Nokia Networks, Nortel Networks, Siemens AG, Sun Microsystems and Ubiquity.

- SIP for J2ME [7]: JSR 180 defines the SIP for J2ME specification. Nokia Corporation is specification lead along with the following expert companies and individuals: AGEA Corporation, Aplix Corporation, AromaSoft Corporation, Cingular Wireless, Cisco Systems, dynamicsoft, Anupama Hegde, Hutchinson 3G UK, Leapstone Systems, Motorola, Panasonic, Siemens AG, Sun Microsystems, Symbian, Tata Consultancy Services, Tira Wireless and Ubiquity.

The substantial overlap in the experts and companies defining the SIP specifications through the JCP helps ensure consistent API specifications across the various expert groups.

The collection of Java SIP specifications does not redefine or modify the SIP protocol. They simply define standardized API specifications specific to the Java environment, with the goal of simplified application development and application portability across different implementations of the SIP protocol.

Within the JCP there are also efforts to define Java interfaces to the SIMPLE protocol [8] and a generic protocol agnostic API specification [9] that will layer on any presence and instant messaging protocol. These Java interfaces will be covered in a separate whitepaper. Additionally Java interfaces also exist for SDP [10], MGCP [11], Megaco [12] and ENUM [13].

2 The Java SIP Specifications

2.1 JAIN SIP

The JAIN SIP specification was the first SIP specification standardized through the JCP. The JAIN SIP specification is a general purpose transaction based Java interface to the SIP protocol. It is rich both semantically and in definition to the SIP protocol. The motivation behind JAIN SIP is to develop a standard interface to the SIP protocol that can be used independently or by higher level programming entities and environments. JAIN SIP can be used in multiple ways:

- As a specification for the J2SE platform that enables the development of stand alone user agent, proxy and registrar applications.
- A base SIP implementation for a SIP Servlet container that enables the development of user agent, proxy and registrar applications in a Servlet based environment.
- A base SIP implementation for an Enterprise JavaBeans™ (EJB™) container that enables the development of user agent, proxy or registrar applications in an EJB environment.

JAIN SIP provides a standardized interface that can be used by communications developers as a minimum to support SIP in their applications. The JAIN SIP reference implementation provides a fully functional SIP implementation that can be used by developers to talk SIP from the Java environment. The target developer community for JAIN SIP is developers that are familiar with the SIP protocol and require transactional control over the SIP implementation.

2.1.1 Architecture Overview

JAIN SIP supports RFC 3261 functionality and the following SIP extensions; the INFO method (RFC 2976), Reliability of provisional responses (RFC 3262), Event Notification Framework (RFC 3265), the UPDATE method (RFC 3311), the Reason Header (RFC 3326) and the Message method (RFC 3428) defined for instant messaging [1].

JAIN SIP standardizes the interface to the generic transactional model defined by the SIP protocol, providing access to dialog functionality from the transaction interface. The architecture is developed for the J2SE environment therefore is event based utilizing the Listener/Provider event model. The specification is asynchronous in nature using transactional identifiers to correlate messages. It defines various factory classes for creating Request and Response messages and SIP headers. JAIN SIP defines an interface for each Header supported, which can be added to Request or Response messages respectively. These messages are passed to the SipProvider with a transaction to be sent onto the network, while the SipListener listens for incoming Events that encapsulate messages that may be responses to initiated dialogs or new incoming dialogs.

JAIN SIP is extensible by design. It defines a generic extension header interface that can be used by applications that utilize headers that are not supported directly by JAIN SIP. The design also defines a mechanism to support future dialog creation methods in a JAIN SIP environment via the use of Java Properties. JAIN SIP can be managed statically with regards to IP addresses and router function, and dynamically specific to ports and transports.

The default handling of message retransmissions in JAIN SIP is dependent on the application. Stateful proxy applications need not be concerned with retransmissions as these are handled by JAIN SIP. Typically User Agent applications must handle retransmissions of ACK's and 2xx Responses, however JAIN SIP provides a convenience function that ensures all retransmissions are handled by the JAIN SIP implementation, reducing the complexity for applications acting as user agents. This function may also be useful if JAIN SIP is used as a base implementation for a SIP Servlet container or an EJB implementation.

2.2 SIP Lite

The SIP Lite specification is an abstracted view of the SIP protocol that provides a SIP programming environment for developers who are not SIP literate. The API specification is primarily developed for the J2SE platform, however as the specification is quite small it can also be implemented on the J2ME platform. The motivation behind SIP Lite on the J2ME platform is to provide a rich object model that may be suitable for midsize devices with more processing power and memory than mobile handsets, i.e. PDA's and SIP phones. SIP Lite is most common as a J2SE platform based user agent or a programming interface to a SIP Phone.

2.2.1 Architecture Overview

SIP Lite supports the functionality defined in RFC 3261. SIP Lite can be implemented both on the J2SE platform and the J2ME platform, therefore it does not mandate the Listener/Provider event model like JAIN SIP; however, SIP Lite utilizes the Listener/Provider naming convention. SIP Lite defines a three-tier architecture, where the concept of a Listener exists for a Dialog, a Call and a CallProvider. These three interfaces in essence listen for incoming messages, dialogs and calls respectively.

SIP Lite does not define specific Request and Response interfaces; rather, they are combined using a single Message interface. Messages are identified based on Request and Response constants defined within the API specification. Messages are created from and sent via a specific Dialog, where a Response is created based on a specific Request message. A Request is created specific to a method, content type and content body. A generic interface is also defined for SIP Headers, containing generic SIP values and parameters.

The SIP Lite architecture defines the concept of a Call and Dialog interface within which a Call may contain multiple Dialogs. There is no model of transactions within a Dialog; transactions are handled in the underlying implementation hidden from the application developer. SIP Lite defines a single factory class which is used to create addresses, users, headers and parameters, while a CallProvider may be viewed as a factory class for Calls and a Call may be viewed as a factory class for Dialogs.

SIP Lite is an API specification designed specifically for User Agent applications. It does not define or support any proxying capabilities and will always behave statefully. All retransmission semantics will be handled by the implementation, further simplifying the programming environment for the application developer.

2.3 SIP Servlet

The SIP Servlet API specification defines an environment for execution of network based SIP applications. It is implemented on application servers that support SIP and optionally also HTTP and the J2EE platform. It builds on the HTTP Servlet API specification [14] and like HTTP Servlet defines both an API and a file format for application packaging. SIP Servlet supports baseline SIP as defined in RFC 3261 and also supports the following SIP extensions; the Event Notification framework (RFC 3265) and the Message method (RFC 3428) for instant messaging.

At the heart of SIP Servlet lies the ability of applications to perform SIP signaling, either as an endpoint (user agent) or as a proxy. The API specification aims to allow applications complete control over SIP signaling while at the same time hiding much of the non-essential complexity of SIP, which is not relevant to application developers.

The main difference between a non-application server based SIP API specification and an application server based SIP API specification is that an application server itself creates and manages resources whereas the former generally speaking does not. SIP Servlet containers manage resources like listening points, threads, transactions and dialogs, session state, and application components.

2.3.1 Architecture Overview

Applications can act as User Agents and as proxies. Proxying can be done in a transaction stateless or stateful manner. The SIP Servlet container maintains the underlying SIP transactions as well as established dialogs. There is no attempt to provide a SIP independent signaling API; the philosophy is that applications will frequently need access to SIP specific headers. SIP Servlet defines a SIP deployment descriptor and can be used for converged applications, which is very similar to the deployment descriptor defined by the HTTP Servlet API specification. Its most important role is to declare Servlets and to define a set of rules governing when the application may be invoked. Rules are essentially predicates over SIP requests. If a rule matches an incoming request, the container may invoke the corresponding Servlet, i.e. initial incoming requests are dispatched to applications whose rules match the request.

SIP Servlet allows multiple applications to be invoked for the same request. The choice of application composition functionality and the choice of multiple matching rules are left to the application. However, if application composition is implemented, the specification requires the cascaded services model, which implies that applications are independent and each follow standard SIP rules.

A Servlet application may contain both SIP and HTTP components (and components specific to other protocols as they are defined). Converged containers deploy both SIP and HTTP Servlets and allow state to be shared between them. SIP Servlet also includes declarative and programmatic security features along the lines of the HTTP Servlet API specification. State maintenance in the SIP Servlet API specification builds on and generalizes the session model of the HTTP Servlet API specification. Session objects hold application state and are automatically persisted and/or replicated by the container. The SipSession interface corresponds to SIP dialogs and plays a similar role as the HttpSession interface in HTTP Servlet as both represent point-to-point "signaling" relationships. In addition to these "protocol sessions", there is a notion of an application session which binds together multiple protocol sessions.

2.4 SIP for J2ME

The SIP for J2ME API specification defines a SIP interface for small platforms. SIP for J2ME is still quite early in the definition process, therefore all technical information provided within this document is subject to change, however the high level concepts presented here should remain true. SIP's acceptance as the protocol of choice by the IP Multimedia Subsystem (IMS) architecture [15] within the Third Generation Partnership Project (3GPP™), highlights the value of mobile devices understanding and communicating SIP. The goal of the SIP for J2ME API specification is to address this need.

The SIP for J2ME specification is based on the Connected Limited Device Configuration (CLDC) [16] framework within the J2ME platform and will typically be used in conjunction with the Mobile Information Device Profile (MIDP) [17] however is not limited to this profile.

2.4.1 Architecture Overview

CLDC based terminals have stringent memory requirements; therefore attention is required in the design of this specification to ensure a small memory footprint. The implementation of the SIP for J2ME API specification will be integrated in the mobile phone and will make the link between the terminal's native SIP implementation and the MIDP environment.

As more and more small devices are supporting the J2ME platform, it is essential that any specification targeting that domain extends the Generic Connection Framework (GCF) pattern to integrate easily with that platform. This means every new SIP Connection can be obtained through the unique Connector factory. SIP for J2ME also follows the simple and lightweight structure that all the other protocol frameworks standardized in MIDP do, i.e. HttpConnection, SocketConnection. This ensures a very flat class structure that inherently simplifies usage.

Similar to HttpConnection in the J2ME platform, SIP for J2ME is defined at the transaction level. This choice makes the API specification multipurpose and does not limit its use by including any assumptions of its intended usage, e.g. Voice over IP. As a consequence, a MIDlet that is implemented at the transaction level must handle the flow of messages. The HTTP like functionality has been extended to support the receiving of Requests that exist in SIP and HTTP, like blocking calls are extended with an event mechanism that allow application developers to choose the optimal programming style. For the sake of reducing the MIDlet code size some helper functions are provided to assist in the basic SIP tasks. Note that a MIDlet is free to ignore these helper functions to make previous flexibility requirements possible. Examples of helper functions include:

- Automatic initialization of mandatory headers in requests
- Creation of pre-initialized responses and acknowledgments
- Support for subsequent SIP dialogs
- Support for automatic request refreshes

Like SIP Servlet, SIP for J2ME is concentrating on building of the momentum of existing HTTP interfaces that exist in the Java platform, with the focus of application convergence and developer adoption.

3 Relationships among the various SIP API specifications

The SIP APIs being standardized for the Java platform may be viewed as overlapping in functionality with regards to the type of applications that can be implemented on each specification i.e. one can build a user agent on each specification, however they differ dramatically in the developer niche they target. SIP Servlet targets web tier application developers for the J2EE platform building of the momentum of HTTP Servlet. SIP for J2ME targets mobile phone application developers for the J2ME platform building of the momentum of HTTP Connection. JAIN SIP targets application developers for the J2SE platform and business tier application developers for the J2EE platform with the help of the Java Connector Architecture. Similar to JAIN SIP, SIP Lite targets application developers for the J2SE platform, however also targets midsize device application developers.

3.1 JAIN SIP and SIP Lite

JAIN SIP and SIP Lite are overlapping technologies as they are primarily designed for the J2SE platform; however they are distinctly different in programming model and scope. SIP Lite is focused solely on User Agent functionality, while JAIN SIP also supports proxy capabilities. SIP Lite defines a call centric architecture, while JAIN SIP defines a transaction centric architecture which provides more powerful control over the protocol. SIP Lite has an abstract simple look and feel i.e. SIP Lite has no concept of a Request or Response or specific SIP Headers.

SIP Lite may be easier to understand compared to JAIN SIP for enterprise application developers, however for SIP application developers SIP Lite may not provide the required control over the protocol. JAIN SIP targets SIP application developers that have a reasonable understanding of the SIP protocol, i.e. message components, headers and the transaction model. SIP Lite on the contrary is focused on enabling communication capabilities to non telecom developers, such as enterprise developers with minimal SIP knowledge.

3.2 JAIN SIP and SIP Servlet

JAIN SIP and SIP Servlet are complimentary technologies in the sense that a JAIN SIP implementation could be the standardized base SIP implementation of a SIP Servlet container. This enables different base SIP implementations to be plugged in and out of a SIP Servlet implementation.

3.3 SIP Servlet and SIP Lite

SIP Lite and SIP Servlet both focus on enabling communication capabilities to non-telecom developers, such as the enterprise developers, with generic message definition and no transactional semantics. The key difference between these two technologies is that SIP Servlet is container based and targets J2EE platform developers, as opposed to SIP Lite which targets J2SE platform developers. Additionally, SIP Servlet supports proxying capabilities and application packaging.

3.4 SIP Lite and SIP for J2ME

As SIP Lite has a small number of interfaces, it is possible to implement it on the J2ME platform. Unlike SIP for J2ME, SIP Lite does not explicitly adopt the MIDP architecture. SIP for J2ME is the recommended specification of choice for mobile handsets, however SIP Lite may be implemented on PDAs or SIP Phones.

4 Possible End to End architectures using the SIP APIs

Network architectures are often unique unto themselves and network requirements often determine architecture choices. However it is foreseen that the Java SIP specifications will be used as follows in end-to-end network architectures:

- SIP for J2ME will be implemented in mobile handsets.
- SIP Lite will be implemented in midsize devices, i.e. PDAs, SIP phones, and desktops.
- JAIN SIP will be implemented on desktops and in business tier enterprise application servers. It may also be the base SIP implementation for SIP Servlets.
- SIP Servlet will be implemented in web tier enterprise application servers, bringing the benefit of converged SIP and HTTP applications.
- SIP Servlet, JAIN SIP or JAIN SIP wrapped in a J2EE connector will co-exist in the core telecom network as the backbone for SIP network signaling and custom based SIP servers.

The diagram below shows how existing Java specification interfaces to SIP map onto the IP Multimedia Subsystem (IMS) architecture as defined by 3GPP.

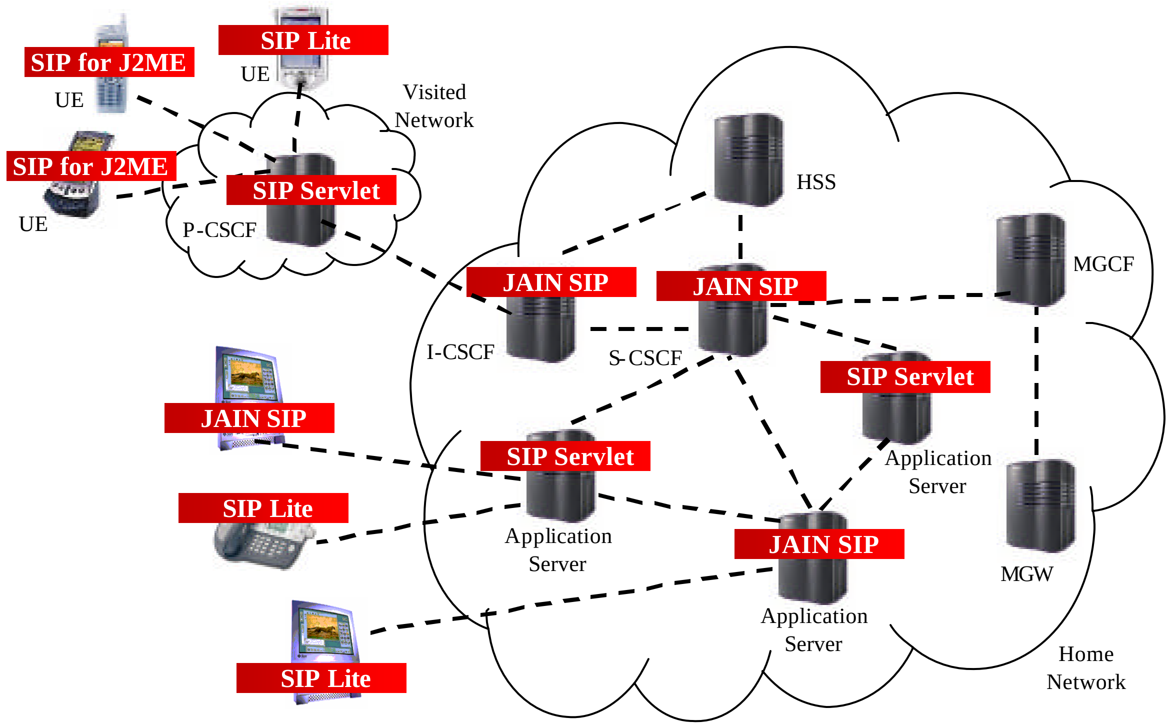

Figure 1: IMS architecture and the applicability of the Java SIP specifications

5 Considerations and Limitations

Education of developers is important; few enterprise developers have heard of SIP and the type of applications that can be developed with it. Time needs to be spent educating and marketing SIP to the Java enterprise development community.

Efforts must focus on establishing a unified developer community around SIP for the Java language. Today numerous specifications exist; however developers need additional resources that aid adoption of these specifications, such as sample implementations, developer tools, applications that will entrench the SIP protocol into the enterprise development environment. This is an achievable goal, which will be worked on in the future by the JCP expert groups and others leading the industry adoption of SIP.

6 The Future of SIP and the Java platforms

The SIP protocol was designed with Internet thinking, which compliments the enterprise thinking of many Java developers. This commonality should help lower the barrier of convergence between the telephony and Internet worlds.

The communications industry is behind the success of both SIP and the Java platforms, highlighted by the numerous SIP specifications being standardized through the JCP. The acceptance of SIP as the universal signaling language for internet communications, coupled with the acceptance of the Java language as a preferred alternative to C or C++ as a development language for communication protocols demonstrates an inherent shift in thinking. The rapid growth of the Java language and the use of SIP for communications offer a great formula for communication network architectures.

The Java language has a number of features that make it very attractive for the SIP protocol. The dynamic loading features make it easy to deploy and update applications at runtime; errors can be caught and handled gracefully; the built in security framework allows containers to restrict applications in what actions they can perform. The Java platform also provides access to a large set of useful functionality, such as JDBC for database access, JNDI for directory lookups and JMF for handling streamed media. Furthermore integration of the SIP protocol with the J2EE platform ensures SIP applications can plug into backend infrastructure already in networks today.

The Java SIP standards are also important for the success of the SIP protocol. Communications protocols traditionally are standardized in text document or unified modeling language (UML) format only. This leaves scope for interpretation when realizing an API specification for a specific programming environment, which has been the downfall of many good protocols in the past. The Java SIP standards unify the SIP communications development community and drives SIP into the enterprise domain by providing standardized API specifications to the SIP protocol specific to each Java platform.

When the SIP protocol gains widespread adoption, a strong case will be presented to bundle SIP functionality into the J2SE platform. Acceptance of SIP in the J2SE platform may in turn lead to SIP being adopted by both the J2EE platform and the J2ME platform (similar to HTTP) in the future.

It is expected in the coming years that the J2EE platform will experience the biggest growth of SIP enabled applications; however acceptance of SIP in the J2EE platform indirectly means success for SIP in the J2SE platform. The production of SIP applications on the J2ME platform may take a little longer due to air interface bottlenecks and limited processing power on the mobile handset.

7 References

[1] Request for Comments: 2976, 3261, 3262, 3265, 3311, 3326, 3428. Internet Engineering Task Force (IETF) Network Working Group, 2002. See http://www.ietf.org/rfc.html

[2] Sun Microsystems, “Java Community Process Program - JSR 171”, 2002. See http://jcp.org/en/jsr/detail?id=171.

[3] Sun Microsystems, “The Java Programming Language”, 1996. See http://java.sun.com.

[4] Sun Microsystems, "JAIN SIP API Specification - JSR 32", 2002. See http://jcp.org/en/jsr/detail?id=32.

[5] Ubiquity, Sun Microsystems, "JAIN SIP Lite API Specification - JSR 125", 2002. See http://jcp.org/en/jsr/detail?id=125.

[6] dynamicsoft, Sun Microsystems, "SIP Servlet API Specification - JSR 116", 2002. See http://jcp.org/en/jsr/detail?id=116.

[7] Nokia Corporation, Sun Microsystems, "SIP for J2ME API Specification - JSR 180", 2002. See http://jcp.org/en/jsr/detail?id=180.

[8] Panasonic, Sun Microsystems, "JAIN SIMPLE Presence API Specification - JSR 164", 2002. See http://jcp.org/en/jsr/detail?id=164. Panasonic, Sun Microsystems, "JAIN SIMPLE Instant Messaging API Specification - JSR 165", 2002. See http://jcp.org/en/jsr/detail?id=165.

[9] Panasonic, Sun Microsystems, "JAIN Presence API Specification - JSR 186", 2002. See http://jcp.org/en/jsr/detail?id=186. Panasonic, Sun Microsystems, "JAIN Instant Messaging API Specification - JSR 187", 2002. See http://jcp.org/en/jsr/detail?id=187.

[10] dynamicsoft, Sun Microsystems, "SDP API Specification - JSR 141", 2002. See http://jcp.org/en/jsr/detail?id=141.

[11] Telcordia Technologies, Sun Microsystems, "JAIN MGCP API Specification - JSR 23", 2001. See http://jcp.org/en/jsr/detail?id=23.

[12] Hughes Software Systems, Sun Microsystems, "JAIN Megaco API Specification - JSR 79", 2002. See http://jcp.org/en/jsr/detail?id=79.

[13] NetNumber, Sun Microsystems, "JAIN ENUM API Specification - JSR 161", 2002. See http://jcp.org/en/jsr/detail?id=161.

[14] Sun Microsystems, "Java Servlet API Specification - JSR 154", 2002. See http://jcp.org/en/jsr/detail?id=154.

[15] 3GPP TS 23.228: “IP Multimedia Subsystem (IMS) - Stage 2” - http://www.3gpp.org.

[16] Sun Microsystems, "Connected Limited Device Configuration - JSR 139", 2002. See http://jcp.org/en/jsr/detail?id=139.

[17] Motorola, Sun Microsystems, "Mobile Information Device Profile - JSR 118", 2002. See http://jcp.org/en/jsr/detail?id=118.