JAIN™, a set of Java™ APIs for Integrated Networks

JAIN
JAVA CALL CONTROL (JCC)
APPLICATION PROGRAMMING INTERFACE (API)
Version 0.8.4

Overview of the API

September 18, 2000

Table of Contents

List of Figures

1. Executive Summary

The Java Call Control (JCC) Application Programming Interface (API) is a Java interface for creating, monitoring, controlling, manipulating and tearing down communications sessions in a converged PSTN, packet-switched, and wireless environment. It provides facilities for first-party as well as third-party applications, and is applicable to network elements (such as switches or Call Agents) both at the network periphery (e.g. Class 5 or end-office switches) and at the core (e.g. Class 4 or tandem switches).

JCC allows applications to be invoked or triggered during session set-up in a manner similar in spirit to the way in which Intelligent Network (IN) or Advanced Intelligent Network (AIN) services can be invoked. JCC thus allows programmers to develop applications that can execute on any platform that supports the API, increasing the market for their applications. It also allows service providers to rapidly and efficiently offer services to end users by developing the services themselves, by outsourcing development, purchasing services developed by third parties, or a combination thereof.

The API is not intended to open up telecommunications networks' signaling infrastructure for public usage. Rather, network capabilities are intended to be encapsulated and made visible using object technology in a secure, manageable, and billable manner. This approach allows independent service developers to develop applications supported by the network without compromising network security and reliability.

The API is specified in terms of a coherent collection of related and interacting objects that model different physical and logical elements involved in a session, and related functions. Applications interact with these objects via an object-oriented Listener paradigm. Note that the API is applicable to control of voice, data or multimedia sessions, and not just voice calls, but for convenience we often use the word "call" in the specification. Also note that "call" is understood to include multiparty multimedia sessions over the integrated (PSTN, packet, and/or wireless) network.

The API is structured into the following three functional areas; this document describes only the first two, while the third will be the output of a separate JAIN Edit Group:

- **Elementary Call Control: JCP.** The *Java Call Processing (JCP)* package includes the very basic facilities required for initiating and answering calls. It is likely that the facilities offered by this package will be too elementary for many if not most carrier-grade deployments. However, as explained later, it represents an important conceptual cornerstone for unifying the call control APIs developed by the Java Telephony API (JTAPI), JAIN and Parlay expert groups. It also represents a simple first step of software that can then be implemented, tested, reused and logically extended to a full implementation of the Java Call Control API.

- **Core Call Control: JCC.** The *Java Call Control (JCC)* package includes the facilities required for observing, initiating, answering, processing and manipulating calls, as well as for invoking applications and returning results during call processing. It is likely that the facilities offered by this package will suffice for implementing most, but not all, of the basic and value-added services offered by carriers.

- **Extended Call Control: JCAT.** The *Java Coordination and Transactions (JCAT)* package includes facilities similar to JCC, but extended to provide finer granularity of call control. In

particular, unlike JCC, JCAT enables all common AIN applications as well as other integrated voice/data and next-generation services.

For all the packages above, applications may be executing on the switching platform itself (e.g. a softswitch or Call Agent platform) or in a coordinated, distributed fashion across multiple general-purpose or special-purpose platforms.

The JCC and JCP APIs define four objects, which model the key call processing objects manipulated by most services. These are a Provider, Call, Connection, and Address. Several of these objects contain finite state machines that model the state of a call, and provide facilities for allowing applications to register and be invoked, on a per-user basis, when relevant points in call processing are reached.

The JCP and JCC APIs described in this document are intended to be consistent with the APIs issued by the JTAPI and Parlay groups. In the case of JTAPI, the JCP API represents an elementary call control package that forms the common base of both JCC and JTAPI; thus JCC and JTAPI are very consistent in this respect. In the case of Parlay, JCC is in fact the Java version of the Parlay API for call control as accepted by the JAIN Service Provider API (SPA) group that is standardizing Java instantiations of the Parlay API. However, as of this writing, the Java version of the Parlay call control API (i.e., JCC) is similar but, unfortunately, not functionally identical to the UML version of the Parlay call control API; it is hoped that future revisions to the JCC and Parlay call control APIs will close this gap.

This document is a high-level overview of the JCC API. It is not a substitute for the actual JCC specification, which is a companion to this document. An additional document provides UML diagrams ("call flows") describing how the API can used to implement example services, such as first and third-party originated and terminated calls and Virtual Private Network (VPN).

2. Introduction

This document provides a framework for Application Programming Interfaces (APIs) to support network services over integrated networks. The APIs described are the Java Call Processing (JCP) and Java Call Control (JCC) APIs.

2.1 Service Drivers

This API is intended to allow carriers to offer a wide variety of integrated voice/data services over a common PSTN/packet/wireless network infrastructure, even though most of the focus in the industry today is on offering telephony services over such a network.

The API definition was carried out with some example services in mind. These services are:

- First and Third-party originated and terminated calls
- Voice virtual private network (VPN),
- Toll-free number translation
- Voice-activated dialing,
- Click-to-dial,
- Meet-me conference

Needless to say, the API can be used to implement a wide variety of other integrated voice and data applications beyond the examples mentioned above. A companion document provides UML sequence diagrams as examples of how the first two applications above can be implemented with the API.

2.2 Network architecture and protocols

The API defines a programming interface to next-generation converged networks in terms of an abstract, object-oriented specification. As such it is designed to hide the details of the specifics of the underlying network architecture and protocols from the application programmer to the extent possible.

Thus the network may consist of the PSTN, a packet (IP or ATM) network, a wireless network, or a combination of these, without affecting the development of services using the API. The API is also independent of network signaling and transport protocols. Thus the network may be using various call control protocols and technologies, for example, SGCP, MGCP, SIP, H.323, ISUP, DSS1/Q.931, and DSS2/Q.2931, without the explicit knowledge of the application programmer. Indeed, different legs of a call may be using different signaling protocols and be on different underlying networks.

It is assumed that the network will be able to notify the platform implementing the API regarding events that have occurred (e.g. call arrival) and the platform will be able to process the event as necessary and inform the application using the API. In addition, the application will be able to initiate actions using the API (e.g. call origination) that the platform will translate into appropriate protocol signaling messages to the network. It is the job of the platform to interface to the underlying network(s) and translate API methods and events to and from underlying signaling protocols as it sees fit. We stress that this translation is vendor-specific and is not specified by the API; thus different platform vendors may differentiate and compete based on the attributes (e.g. performance) of their translation.

2.3 Call Control Package Hierarchy

The rest of this overview assumes some familiarity with object-oriented programming and Java concepts. In general, readers familiar with JTAPI will find that many of the basic concepts of the API bear similarity to JTAPI, and hence some knowledge of JTAPI will be helpful in reading this document and the specification.

The two packages described in this document are the Java Call Processing (JCP) package, and the Java Call Control (JCC) package that inherits from JCP using Java inheritance, as shown in Figure 1. In section 6 we will discuss the relation of these two packages to JTAPI, Parlay, and the JCAT package.

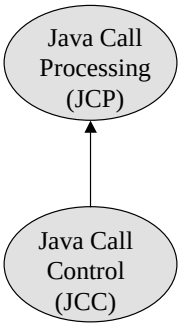

Figure 1: JCP and JCC inheritance relationship

3. Basic Components of the API

In this section we describes the basic objects of the API common to both JCP and JCC as well as the common design patterns to both packages.

For both JCC and JCP, the API components consist of a related set of interfaces, classes, operations, events, capabilities, and exceptions. The API provides four key objects, which are common to JCP, JCC and more advanced packages. We provide a very brief description of the API in this overview document; an overview of each object and its details can be found in the description of the object interface in the specification. The four key objects are:

- Provider: represents the "window" through which an application views the call processing.
- Call: represents a call and is a dynamic "collection of physical and logical entities" that bring two or more endpoints together.
- Address: represents a logical endpoint (e.g., directory number or IP address).
- Connection: represents the dynamic association between a Call and an Address.

The relationship among these objects is depicted pictorially in Figure 2 for a two-party call. Multiple parties are represented simply by additional Connections and Addresses associated with a Call; there is no inherent limitation in the model on the number of such parties. In traditional telephony parlance, this model is "symmetric" in that there is no fundamental distinction at the highest level between originating or terminating parties of a call, and is a "full" model in the sense that the application, in principle, has a view of all parties of the call and not simply the originating or terminating party.

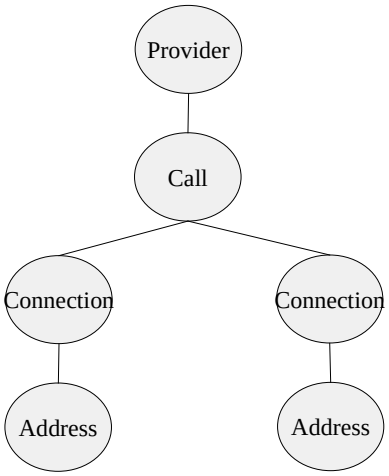

Figure 2: Object model of a two-party call

The purpose of a Connection object is to describe the relationship between a Call object and an Address object. A Connection object exists if the Address has been associated with the telephone call. Connection objects are immutable in terms of their Call and Address references. In other words, the Call

and Address object references do not change throughout the lifetime of the Connection object instance. The same Connection object may not be used in another telephone call.

3.1 Basic API patterns: Listeners and Factories

The basic programming pattern of the API is that applications (which reside "above" the API) make synchronous calls to API methods. The platform or network element implementing the API can inform the application of underlying events (e.g. the arrival of incoming calls) by means of Java events. The application provides Listener objects corresponding to the events that it is interested in obtaining. IN or AIN-style triggers are also implemented using this basic event and Listener paradigm; to control the events that an application receives, event filters (described in the specification) can be defined.

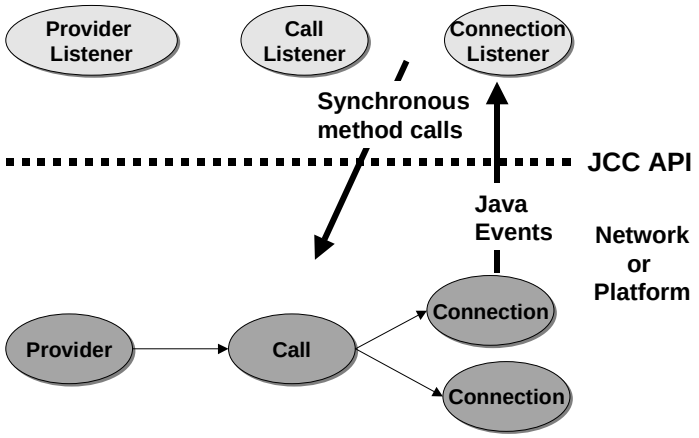

Figure 3: API programming pattern using Java Listeners

In addition, the API also uses the Factory design pattern commonly used (and recommended) in Java. In brief, applications use a PeerFactory to obtain a Peer, which in Java nomenclature is "a particular platform-specific implementation of a Java interface or API", i.e., a vendor's particular implementation of the API. Applications then use the Peer to obtain access to the Provider object.

3.2 Event and Listener inheritance diagrams

Since several objects in the API can generate events, which in turn can be trapped by different Listeners written by the application programmer, the Event and Listener objects are organized by inheritance. The inheritance diagrams are shown in Figure 4.

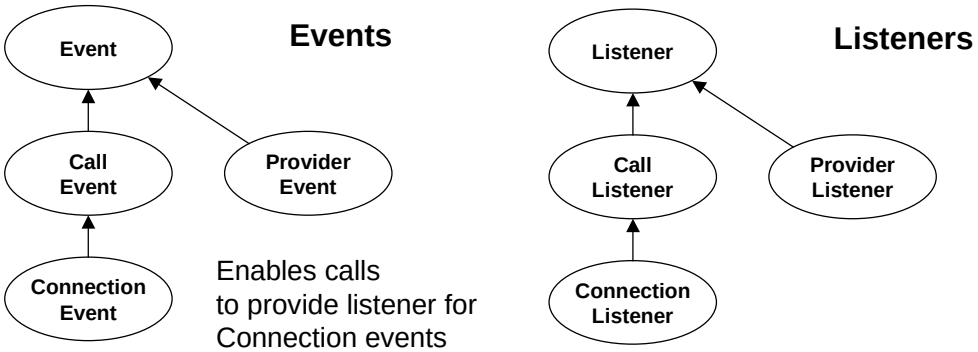

Figure 4: JCC and JCP Event and Listener inheritance diagrams

4. Java Call Processing (JCP)

In this section we describe the basic components of JCP. JCP is the elementary Java package from which more advanced call control packages inherit.

4.1 JCP Finite State Machines

The behavior of the JCP Provider, Call and Connection objects is specified in terms of Finite State Machines (FSMs), shown in Figure 5 - Figure 7 below. (Note that the Provider FSM is the same for JCP and JCC.)

For the Provider FSM, the meanings of the states are as follows.

- IN_SERVICE : This state indicates that the Provider is currently alive and available for use.
- OUT_OF_SERVICE: This state indicates that a Provider is temporarily not available for use. Many methods in this API are invalid when the Provider is in this state. Providers may come back in service at any time; however, the application can take no direct action to cause this change.
- SHUTDOWN: This state indicates that a Provider is permanently no longer available for use. Most methods in the API are invalid when the Provider is in this state. Applications have access to a method to cause a Provider to move into the SHUTDOWN state.

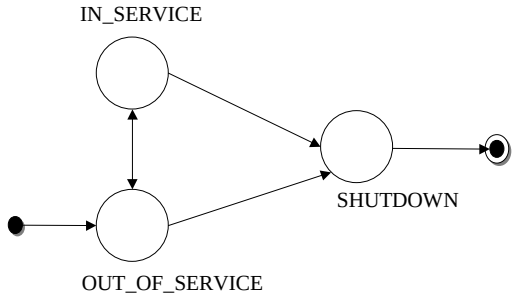

Figure 5: JCP and JCC Provider Finite State Machine

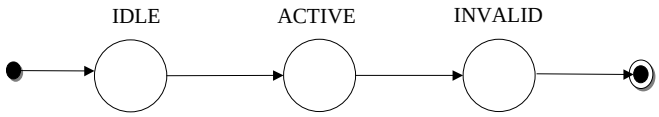

Figure 6: JCP Call object Finite State Machine

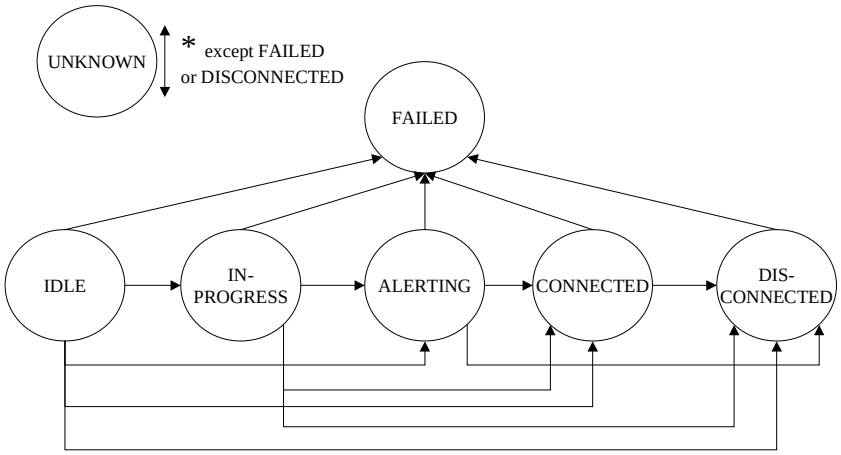

Figure 7: JCP Connection object Finite State Machine

JCP provides, among others, a few key methods to support its primary features of placing, answering and dropping calls. The JCP specification defines, for each method, the pre and post conditions of the method in terms of state transitions in these FSMs. We will not discuss the FSMs for the JCP Call and Connection objects in this overview document; an overview as well as details are provided in the specification of each interface (e.g. JcpConnection, JcpCall). Also note that the JCP Connection object FSM is identical to the JTAPI core package's Connection object FSM.

5. Java Call Control (JCC)

In this section we briefly summarize JCC. The JCC API has the same four key objects as JCP, namely Provider, Call, Connection and Address. Since JCC inherits from JCP, each object may contain additional methods beyond those in JCP. Each object's interface specification includes a brief overview of the object along with its details, so we provide only a very brief description here.

We note that as for JCP, JCC Connection objects are immutable in terms of their Call and Address references. In other words, the Call and Address object references do not change throughout the lifetime of the Connection object instance.

5.1 JCC Finite State Machines

5.1.1 JCC Provider

The JCC Provider has the same FSM as for JCP, described in Figure 5 and Section 4.1.

5.1.2 JCC Call

The JCC Call object FSM is shown in Figure 8.

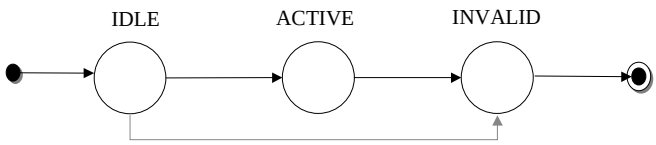

Figure 8: JCC Call object FSM

Thus the JCC Call FSM is similar to the JCP Call object, but has an additional transition, from the IDLE to the INVALID state. The descriptions of the states are:

- IDLE: This is the initial state for all Calls. In this state, the Call has zero Connections.
- ACTIVE: A Call with some current ongoing activity is in this state. Calls with one or more associated Connections must be in this state.
- INVALID: This is the final state for all Calls. Call objects which lose all of their Connections objects (via a transition of the Connection object into the DISCONNECTED state) move into this state. Calls in this state have zero Connections.

5.1.3 JCC Connection

The JCC Connection object has a different FSM from JCP. Note however, that the JCC Connection FSM is a refinement of the JCP Connection FSM (i.e., obtained by adding transitions or splitting states into multiple states). The FSM is shown in Figure 9. The states are described as:

- IDLE: This state is the initial state for all new Connections. Such Connections are not actively part of a telephone call, yet their references to the Call and Address objects are valid. Connections typically do not stay in the IDLE state for long, quickly transitioning to other states.

- AUTHORISE_CALL_ATTEMPT: This state implies that the originating or terminating terminal needs to be authorized for the Call.

- ADDRESS_COLLECT: In this state the initial information package is collected from the originating party and is examined according to the "dialing plan" to determine the end of collection of addressing information.

- ADDRESS_ANALYZE: This state is entered on the availability of complete initial information package/dialing string from the originating party. The information collected is analyzed and/or translated according to a dialing plan to determine routing address and call type.

- CALL_DELIVERY: On the originating side this state involves selecting of the route as well as sending an indication of the desire to set up a call to the specified called party. On the terminating side this state involves checking the busy/idle status of the terminating access point.

- ALERTING: This state implies that the Address is being notified of an incoming call.

- CONNECTED: This state implies that a Connection and its Address are actively part of a telephone call. In common terms, two parties talking to one another are represented by two Connections in the CONNECTED state.

- SUSPENDED: This state implies that this connection object is suspended from the call, although it's references to a Call and Address objects will still remain valid. This state is typically entered when an application above the API has been invoked (triggered) during an active call.

- DISCONNECTED: This state implies it is no longer part of the telephone call, although its references to Call and Address still remain valid. A Connection in this state is interpreted as once previously belonging to this telephone call.

- UNKNOWN: This state implies that the platform is unable to determine the current state of the Connection.

- FAILED : This state indicates that a Connection to that end of the call has failed for some reason (e.g. because the party was busy.

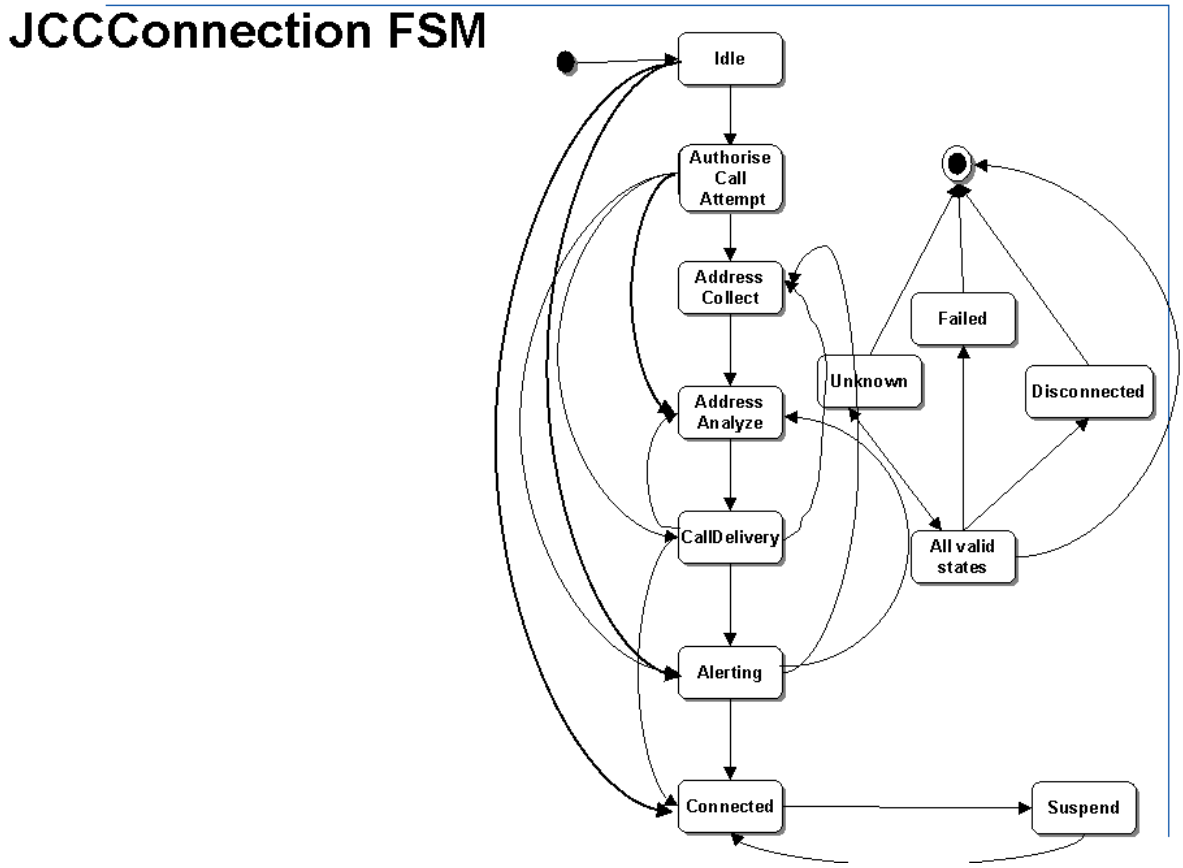

Figure 9: JCC Connection object FSM

5.2 API Methods and Usage

So far we have specified the key objects in JCC (and JCP) as well as their FSMs. To understand operationally how these objects are used and the methods they offer, we refer the user to the UML sequence diagram examples in the companion document.

6. Relationship of JCC and JCP to JTAPI and Parlay APIs

The relationship between JCP, JCC and the call control APIs of JTAPI and Parlay is depicted pictorially via the object inheritance diagram in Figure 10.

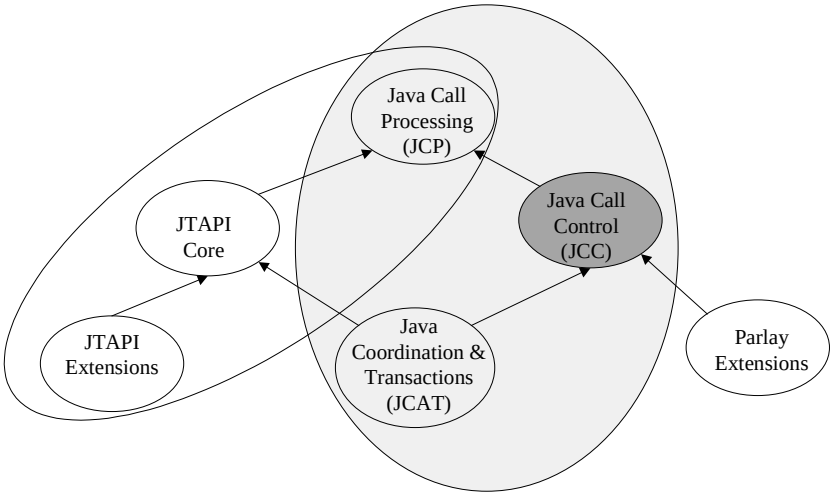

Figure 10: Relationship of JCC packages to JTAPI and Parlay

The JCP package is an elementary call control package from which all other packages inherit. The left ellipse in the Figure represents call control extensions for CTI type of applications, and consists of the JTAPI Core package as well as its extensions. The central shaded ellipse represents the domain of call control packages defined by the JAIN consortium. In addition to JCP and JCC, it includes the JCAT package to be defined to provide advanced call control and IN/AIN type of functionality.

Finally, the Parlay extension packages (e.g. multimedia or conferencing) are intended to extend from JCC along the right side of the Figure. JCC itself is based around the language-neutral Parlay 2.1 Enhanced Call Control Service (ECCS) specification in an effort to harmonize JAIN and Parlay call control. (It is hoped that the gap between the language-neutral Parlay specification and JCC call control will be diminished as the two groups work together.) Note that for Java however, JCC is the official Java instantiation of Parlay call control.

Note that with this structure a programmer using JCP and JCC is linked closely to the other Java specifications for call control, namely JTAPI and the Java instantiation of Parlay call control. In particular, the objects, methods and programming paradigms of the JCP, JCC and JTAPI packages are closely related and consistent, so that as a programmer develops expertise and code for one it is relevant to the others.