

# Java Call Control, Coordination and Transactions<sup>1</sup>

*Ravi Jain, Farooq Anjum, Paolo Missier and S. Shastry*

Applied Research

Telcordia Technologies Inc.

445 South St, Morristown, NJ 07960

{rjain,fanjum,paolo,sshastry}@telcordia.com

## Abstract

Future telecommunications networks will consist of integrated packet-switched (IP and/or ATM), circuit-switched (PSTN) and wireless networks. Service providers will offer a wide portfolio of innovative applications over these integrated networks. Doing so rapidly and efficiently requires open network APIs, with a key API being that for call control as well as coordination and transactions. The JAIN community is defining an API for Java Call Control (JCC) and Java Coordination and Transactions (JCAT). The JCC API defines the interface for applications to initiate and manipulate calls, while JCAT defines the facilities for applications to be invoked and return results before or during calls. Note that in this context a call refers to a multimedia, multiparty, multi-protocol communications session. The JCC/JCAT Edit Group of JAIN is in the process of defining the first version of the JCC/JCAT API specification, which is expected to be released in early 2000.

This introductory paper describes the background and motivation for the design of the JCC/JCAT API. We briefly describe the AIN and JTAPI call models, upon which the JCC/JCAT API is based. We then describe the scope of JCC/JCAT and its relationship to other JAIN Edit Groups defining facilities for enabling service creation. Finally we describe the requirements and example service drivers for JCC/JCAT, as well as the initial proposed design and structure for JCC and JCAT.

## 1. Introduction

Future telecommunications networks will be characterized by new and evolving architectures where packet-switched, circuit-switched, and wireless networks are integrated to offer subscribers an array of innovative multimedia, multi-party applications. Equally importantly, it is expected that the process by which telecommunications applications are developed will change, and will no longer solely be the domain of the telecommunications network or service provider. In fact, in order to provide a broad portfolio of novel, compelling applications rapidly, service providers will increasingly turn to third-party applications developers and software vendors. Thus application development in the telecommunications domain will become more similar to that in the software and information technology in general, with customers reaping the benefits of increased competition, reduced time-to-market, and rapid leveraging of new technology as it is developed .

To make this vision a reality it is necessary that future integrated networks offer application developers a set of standard, open Application Programming Interfaces (APIs) so that applications written for one vendor's system can run on a different vendor's system. This will enable the cost of applications development to be amortized, reducing the final cost to the customer. JAIN<sup>TM</sup> is a community of

---

<sup>1</sup> Copyright 1999 Telcordia Technologies, Inc. All Rights Reserved

companies led by Sun Microsystems under the Java Community Process that is developing standard, open, published Java<sup>TM</sup> APIs for next-generation systems consisting of integrated Internet Protocol (IP) or Asynchronous Transport Mode (ATM), Public Switched Telephone Network (PSTN) and wireless networks. These APIs include interfaces at the protocol level, for different protocols like MGCP<sup>2</sup>, SIP, and TCAP, as well as at higher layers of the telecommunications software stack.

One of the key APIs being developed at the higher layers of the telecommunications stack is the API for defining the *call model* that the network offers the applications developer. The call model can be regarded as a specialized virtual<sup>3</sup> machine for the development of telecommunications applications [1], with the API being the interface to that virtual machine. In this paper we describe such an API being developed by a subgroup of JAIN, called the Java Call Control (JCC) and Java Coordination and Transactions (JCAT) Edit Group. We stress here that the use of the phrase “call model”, which has traditionally been associated with the PSTN, does not imply that the work of the JCC/JCAT Edit Group is focused on the PSTN. The charter of JCC/JCAT is to develop an API that applies equally well to IP or ATM, PSTN, and wireless networks, as well as networks integrating these technologies.

The development of open network APIs using Java represents an important departure from traditional methods by which the PSTN was made more open. In the past, AIN defined models that allowed creation of services outside switches, but typically these services were written in specialized languages, using specialized service creation environments, by specialized personnel. The benefits and potential pitfalls of using Java as a language for implementing telephony APIs have been discussed in [2], and for implementing protocols in [3]. We will not repeat these discussions here; the reader is referred to [2,3]. (Also note that a Java API to ATM is being developed by the Service Aspects and Applications Working Group of the ATM Forum.) However, we point out that aside from the benefits of the Java language itself (such as portability across different execution platforms), using Java allows the arsenal of Java-based technologies (Java Beans, Enterprise JavaBeans, etc.) to be applied to service development for telecommunications services. In addition, the growing number of tools, support utilities, development environments, and experienced programmers and designers available for Java potentially opens up large economies of scale in the service creation process. Finally, we have previously implemented a prototype call-processing platform in 100% pure Java that completes basic calls, performs advanced services, and also allows dynamic service deployment [1].

This paper is organized as follows. In the following section we discuss background material on call models and APIs, providing the foundation and motivation for the design of the JCC/JCAT API. In section 3 we describe the relationship of JCC/JCAT to the other components of the service creation process being addressed by the other Edit Groups within JAIN. In section 4 we briefly describe the requirements and structure of the JCC/JCAT API, and finally in section 5 we end with concluding remarks.

## 2. Call models and APIs

Traditionally, the word “call” in the PSTN evokes associations with a two-party, point-to-point voice call. In contrast, in this paper and within the JAIN JCC/JCAT Edit Group, we use the word *call* to refer in general to a multimedia, multiparty, multi-protocol communications *session* over the underlying

---

<sup>2</sup> A list of acronyms is given at the end of this paper

<sup>3</sup> Not to be confused with the Java Virtual Machine (JVM). The virtual machine we refer to here is offered to the application by the software layer implementing the API. There may be a JVM below this layer if the application is written in Java.

integrated (IP, ATM, PSTN, wireless) network. By “multi-protocol” we mean here that different legs of the call, representing the logical connection to individual parties of the call, may be effected by different underlying communications protocols over different types of networks. Thus one leg of the call may be effected using the H.323 protocol [4], another via SIP [5], and a third via traditional PSTN signaling protocols like ISUP [6].

Several call models and associated APIs have been developed in the past, including Advanced Intelligent Network (AIN) [7], Java Telephony API (JTAPI) [8, 2], and Telephony API (TAPI) [9]. While there are important differences among these call models, reflecting the architecture or application for which they were intended, their overall goal is generally similar: to initiate, control and manipulate calls, and to facilitate the development of applications that execute before, during or after a call. Rather than select any one particular call model, we believe it is worthwhile to learn from the experience gathered by the different communities that have developed existing call models, and develop a generic call model suitable for integrated next-generation networks.

For example, the AIN call model was designed to allow applications to be developed for the PSTN. Thus the AIN call model implicitly assumes a specific distributed architecture where telephone switches perform the basic call processing functions. It is assumed that value-added services (e.g. toll-free number translation, time-of-day call routing etc.) are executed, before or during calls, by a specialized Service Logic Execution Environment (SLEE) like the Service Control Point (SCP).

In contrast, JTAPI focuses on call processing and applications for a Private Branch Exchange (PBX) or Call Center environment, where a much greater degree of centralized processing and control is the norm. Thus unlike AIN, JTAPI contains no facilities for suspending execution of call processing and invoking applications during call setup or mid-call. On the other hand, unlike AIN, JTAPI offers convenient, object-oriented abstractions for call manipulation, which facilitate the rapid development of object-oriented applications.

A survey of existing call models we have considered is outside the scope of this paper. In the following subsections we briefly review two existing call models and APIs that are especially relevant to our efforts, namely AIN and JTAPI.

## 2.1 Advanced Intelligent Network (AIN)

In terms of the service creation process, the AIN architecture represented an important advance when it was introduced. AIN separated service development from switching, allowing service logic to be developed more quickly and placed in specialized network elements attached to databases, e.g. the Service Control Point (SCP), while switches could be optimized for speed and efficiency.

To do this, AIN introduces a call model that consisted essentially of two major elements. The first element is a pair of finite state machines representing the progress of a call as it is processed at the originating and terminating switch respectively. The second is the concept of *triggers*. Triggers can be defined at specific states of the originating or terminating switch’s finite state machine (FSM). When call processing reaches a state in the FSM where a trigger is defined and enabled, processing is suspended and a program (called *service logic*) executing at a remote network element like the SCP to be invoked; call processing is resumed once the service logic completes execution.

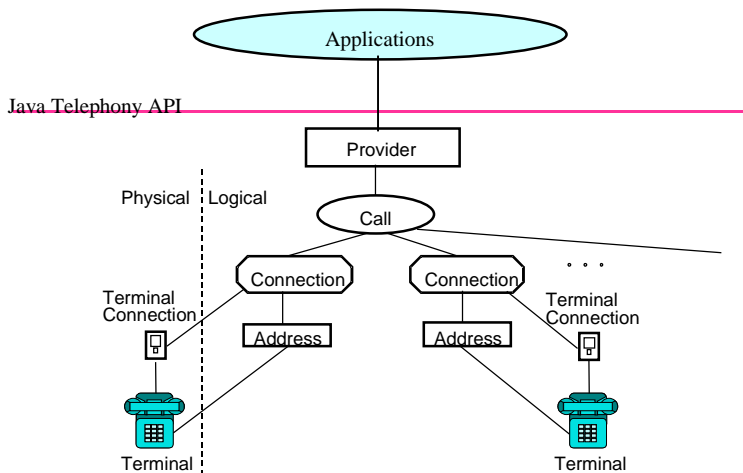
Note that the definition of the AIN call model can be regarded as *switch-centric*, in the sense that the fundamental activity is seen as call processing in the switch, while the service logic (which is not even dignified by being called an application program) is viewed as an ancillary activity. The application programmer must understand the details of the originating and terminating FSMs and interact with call

processing at pre-specified states in the FSMs. There is no explicit abstraction offered to allow the programmer to manipulate entire calls, or legs of a call, or the principal logical entities in the call (e.g. the calling or called party’s address or phone number), and certainly not in any object-oriented fashion. Enhancements built around AIN (e.g. ITU’s Connection View [10]) offer facilities for modeling and manipulating calls, but these are also quite limited; for instance, at present, multi-party calls of more than three parties cannot be handled. Nonetheless, the AIN FSMs do capture the critical stages of call processing and states where it would be useful for application programs to intervene.

## 2.2 Java Telephony API (JTAPI)

The Java Telephony API (JTAPI), published by JavaSoft [8], is a portable, object-oriented interface for Java-based computer-telephony applications. In the following, a “call” refers to a communications session among two or more parties; each party is informally said to be participating in one “leg” of the call. Thus a call has as many call legs (or connections) as the number of parties in the call. JTAPI is expressed in Java and defines a core call model to support basic call setup, and a number of extensions, mostly designed to model call center features, multi-party conference calls, call routing, etc. The core model consists of a few telephony classes and their relationships, as shown in Figure 1. Each object in the figure corresponds to a physical or a logical entity in the telephone world. The Provider is an abstraction of a telephony service provider. A Provider class manages Call objects, representing calls at various stage of progress.

A Provider maintains a collection of static Terminal and Address objects in its domain. Terminal objects represent the physical endpoint of a call, while Address objects are logical endpoints. Notice that each Address can be associated with multiple Terminals and vice versa, reflecting the standard configuration for a call center. The Call, Connection, and Terminal Connection objects are created dynamically, on a per-call basis. The Call object models the state and operations of the call as a whole, i.e., the communications session among the different parties. Each leg of the call is separately modeled by means of the Connection object. More precisely, the Connection object models the state and operations of the logical association between a Call and a particular Address. Finally, a Terminal Connection represents state and operations of the logical relationship between one Connection and one Terminal object.



**Figure 1: Objects in the JTAPI model**

The state of a telephone call is maintained by finite state machines associated with Call, Connection and Terminal Connection objects (e.g., when a call is answered by the called party, the originating

Connection object moves to the CONNECTED state). The complete definition of the state machines is part of the published JTAPI specifications [8].

It is clear from this brief description that JTAPI overcomes several of the limitations of AIN mentioned earlier. JTAPI offers the programmer clear, explicit abstractions for manipulating calls and the logical entities in a call. The API is object-oriented, and draws upon the advantages of Java by using inheritance for extensibility. The state of a call (maintained largely in the Connection object FSM) is encapsulated so that it can be manipulated only via accessor methods. JTAPI uses Java exceptions and the Java events model for reporting changes in state as well as other events of interest to the application.

Nonetheless, JTAPI also has some drawbacks. The first is that the FSM in the Connection object is not as rich and detailed as AIN's, and even with the call control extension package cannot represent all the states of call processing that AIN does. Thus not all the points in the call that may be of interest to applications are modeled. The second is that JTAPI does not contain any mechanism similar to AIN triggers, i.e., no mechanism to suspend call processing at a defined state in the FSM, invoke an application, and return results.

Finally, JTAPI seems to be oriented towards providing support for developing applications in two types of scenarios: (1) where applications run on a single platform (e.g. a PBX); and (2) where applications run on a platform that is "horizontally partitioned", i.e., the higher layers of software (the application and the JTAPI layer) communicate via Java Remote Method Invocation (RMI) [11] with the lower layers over a network. Also, in JTAPI a Provider is assumed to be in control of all the legs of a call (they all hang off the same Call object managed by the Provider). While this assumption may add to the convenience of managing a centralized call center, it is not realistic in the broader setting of integrated next-generation networks.

As we will describe in later sections, the JCC/JCAT API is attempting to build upon the best aspects of JTAPI and AIN while avoiding their drawbacks. This is possible because JCC/JCAT deployment is not necessarily switchbound. The main idea is to extend JTAPI-style call control beyond the traditional call center boundaries, while supporting AIN-style third-party service invocation.

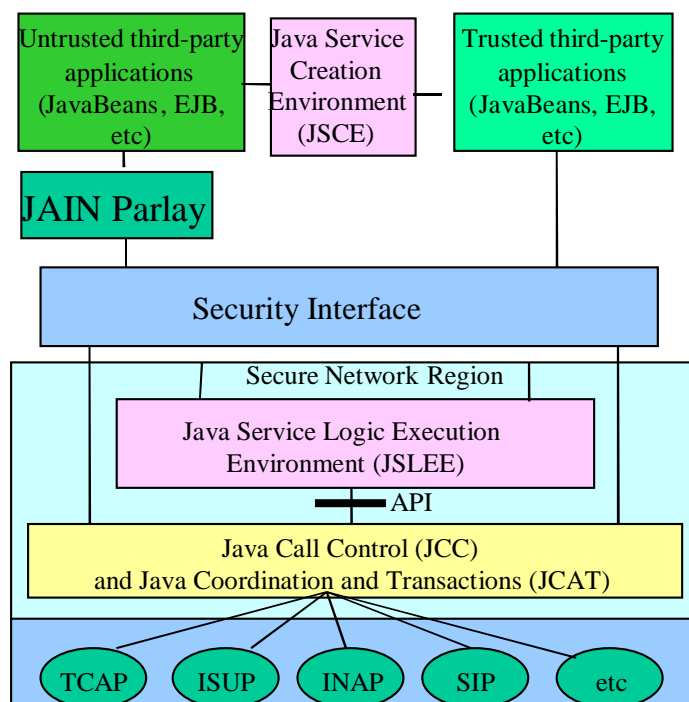
### **3. Scope and relationships of JCC/JCAT**

The JCC/JCAT Edit Group is developing an API that provides an interface to a generic call model that captures the essential aspects of existing call models. The API thus provides the applications programmer with a convenient and powerful abstraction for manipulating calls and managing the interaction between the application and calls. In addition, the API is extensible, so that as additional functions are required, they can be added incrementally and in a modular fashion to the API. To understand JCC/JCAT it is helpful to understand the scope and context within which it is being defined.

The JAIN standardization effort is organized in three broad areas: a Protocols Expert Group (PEG) standardizing interfaces to PSTN and IP signaling protocols; an Application Expert Group (AEG) dealing broadly with the APIs required for service creation within a Java framework; and a Working Group which is engaged in developing prototype implementations and feeding the insights gained into the other two groups. Each Expert Group is organized as a collection of Edit Groups dealing with specific protocols or APIs.

Telcordia has been active in several aspects of the JAIN effort, including providing critical input to the JAIN TCAP Edit Group within the PEG which is standardizing a Java interface to TCAP. In Phase II of the JAIN standardization effort, Telcordia has taken a lead position within the JAIN Phase II AEG

(JP2AEG); in particular, Telcordia is the Edit Group lead for the JAIN Edit Group standardizing interfaces for Java Call Control (JCC) and Java Coordination and Transactions (JCAT).



**Figure 2: Diagram depicting approximate relationships of JAIN Edit Groups**

The Java Call Control (JCC) and Java Coordination and Transactions (JCAT) API provides applications with a consistent mechanism for call control (or processing) and coordination and transactions (or interacting with calls.) Informally, *call control* includes the facilities required for observing, initiating, answering, processing and manipulating calls. Informally, *coordination and transaction* includes (but is not limited to) the facilities required for applications to be invoked and return results before, during or after calls; to process call parameters or subscriber-supplied information; and to engage in further call processing and control. Note that in this context applications may be executing in a coordinated, distributed fashion across multiple general-purpose or special-purpose platforms.

### 3.1 JCC/JCAT and signaling protocols

We first discuss the relationship of JCC/JCAT to the JAIN Protocols Expert Group. The JCC/JCAT API provides the applications programmer with convenient, powerful, object-oriented abstractions for manipulating calls and managing the interaction between applications and calls. As such, a primary purpose of JCC/JCAT is to hide the multiplicity of underlying signaling protocols used to set up, maintain, and tear down calls over heterogeneous networks. This is shown diagrammatically in Figure 2, where it is assumed that applications interact with the JCC/JCAT API in order to avoid interacting with specific protocols.

Nonetheless, note that JAIN is in fact developing, or has already issued, Java APIs to all the signaling protocols shown in the figure, and more as can be seen from other articles in this issue. An application that wishes to may access these underlying protocol APIs, bypassing the abstractions offered by JCC/JCAT. An application that does so will typically get finer-grained control, as it can select, for example, the precise sequence and content of messages sent by the protocol. Bypassing the JCC/JCAT layer may also have some performance advantages, in some cases, since the overhead of creating and manipulating the JCC/JCAT objects is avoided. On the other hand, the applications programmer will be

forced to deal with low-level details involved with each protocol (in terms of message and parameter types, etc.), and will not have the advantage of using the logical abstractions offered by JCC/JCAT. For example, to add a party to an existing call, the application programmer using JCC/JCAT can simply invoke a single method, called, say, **addParty()**, with the appropriate parameters. In contrast, using the protocol API, the application programmer would typically have to send and receive a sequence of protocol messages via the API. Thus the time to develop and test applications will typically be reduced significantly by using JCC/JCAT. In addition, using the JCC/JCAT abstraction means that the application is independent of which underlying protocols are used, and which type of network the application is running on (IP, PSTN, or wireless). In this sense, bypassing JCC/JCAT to develop applications directly using the protocol APIs is roughly analogous to developing applications in assembly language, for performance or other reasons, rather than using the abstractions offered by a high-level language like Java or C++.

### 3.2 JCC/JCAT and application-level facilities

In order to provide the reader with an understanding of the work of the JCC/JCAT Edit Group in the context of the overall service creation and execution process and the JAIN Applications Expert Group as a whole, we use the diagram shown in Figure 2. Note that this diagram is intended for illustration purposes and for this paper only. It is not intended to necessarily be a software layering diagram. There have been numerous discussions within the JAIN AEG attempting to define the scope of each Edit Group and their inter-relationships. These relationships continue to evolve as different Edit Groups continue to iteratively refine their specifications and work efforts. *The diagram of Figure 2 and discussion in this section is intended to be purely illustrative; it is not a final formal position of JAIN or any particular Edit Group within JAIN.*

#### 3.2.1 JCC/JCAT and service development and execution

The JAIN AEG architecture is designed to allow access to the integrated network both for untrusted third-party applications as well as trusted (service-provider created or third-party) applications. Applications would typically be written using components like JavaBeans (JB) [12] or Enterprise JavaBeans (EJB) [13] and would be created using a Java Service Creation Environment (JSCE) and execute within a Java Service Logic Execution Environment (JSLEE). It is possible that third-party Integrated Development Environments (IDE) will provide many of the facilities required in this area. The requirements and framework of the JSCE/JSLEE is within the scope of the JSCE/JSLEE Edit Group

Both untrusted and trusted applications must first undergo appropriate security checks before they can access the network resources made available via JCC/JCAT. Note that these security checks may be needed to not only authenticate and authorize the applications to the network but also vice versa. Pictorially we have depicted the region where applications may execute after undergoing security checks as the “secure network region”.

Note that in the diagram we show the JSLEE pictorially as residing between the applications and JCC/JCAT. Obviously trusted applications would actually execute using the facilities of the JSLEE, which could act as a “container” in the EJB sense. In this sense trusted applications execute “inside” the JSLEE, which could then be represented as the entire secure network region, and in fact JCC/JCAT as well as the underlying protocols reside inside the JSLEE. On the other hand, in the view of JCC/JCAT, some applications may execute on a third-party SLEE different from that defined by the JSCE/JSLEE Edit Group of JAIN. In the extreme case, the SLEE may simply be a minimalist environment consisting only of a Java Virtual Machine (JVM) . As the JSCE/JSLEE Edit Group evolves this precise relationship will be expanded and clarified.

### **3.2.2 Relationship of JCC/JCAT to JAIN Parlay**

We now discuss the position of untrusted applications and the JAIN Parlay API with respect to JCC/JCAT. As mentioned above, trusted applications would typically execute inside the JSLEE. Untrusted applications may execute on third-party SLEEs (e.g. on enterprise applications servers or PBXs) outside the secure network region. It is envisioned that untrusted applications would utilize the APIs developed by the JAIN Parlay Edit Group .

The JAIN Parlay 1.0 API is based on the Parlay API 1.2 specification issued by the Parlay community, and is designed to allow untrusted applications access to network resources in a controlled and limited manner. As such, it contains strong facilities for authenticating and authorizing untrusted applications. (Note that in the view of JCC/JCAT, untrusted applications must still go through a further layer of security, which is the same as for trusted applications, to access the JCC/JCAT abstractions).

The JAIN Parlay API specification has some features that could also be used for JCC/JCAT. In particular the Generic Call Control Service (GCCS) specified in JAIN Parlay 1.0 includes specifications of some entities (objects, events and callbacks) that are useful for call control and coordination and transactions in the sense of JCC/JCAT. JAIN Parlay's GCCS is a very useful set of functions but in our current view is not sufficient as a JCC/JCAT API as it does not include a complete specification of the key entities. In addition, the underlying state machines it describes for key objects are very close (if not practically identical) to JTAPI 1.2 [8], with corresponding limitations as discussed previously.

It has been decided within the JAIN AEG that future versions of the JAIN Parlay GCCS call model will evolve to become a true subset of the JAIN JCC call model. This will allow applications written for JCC to be migrated to JAIN Parlay's call control API and vice versa. This harmonization is in progress and will be developed as further releases of JAIN specifications are issued.

### **3.2.3 JCC/JCAT and Connectivity Management**

Another Edit Group in the AEG is the proposed Network Connectivity Edit Group. The scope of this Edit Group is policy management for network, connection and service attributes. Examples of such attributes include QoS, bandwidth management, data security, access control, etc. It is proposed that Connectivity Management will define interfaces by which management applications can access functions for managing both the network and service attributes, as well as the policies governing such attributes. Connectivity Management is not intended to replace or bypass control protocols (e.g. routing protocols) within the network. The role and objectives of this group need to be defined further before its relationship to the JCC/JCAT Edit Group can be determined.

## **4. JCC/JCAT requirements and structure**

Thus far the JCC/JCAT Edit Group has issued an introductory white paper as well as an internal requirements document. Currently, the JCC/JCAT specification document is in draft stage and is not yet approved for public review; a first draft is expected to be available in early 2000. In this section we describe the requirements for JCC/JCAT as well as a high-level introduction to the JCC/JCAT specification under development. *Note that the information in this paper is itself not a JCC/JCAT specification and is preliminary and subject to change.*

## 4.1 Service drivers and requirements

The design of JCC/JCAT is driven by a set of service drivers, which are specified explicitly in each release of the specification document. For the first release, it is required that the API support:

- *Voice virtual private network (VVPN)*: This is a corporate service that provides companies a way to link different sites with a uniform and private dialing plan, regardless of geographical boundaries. The main function of a VVPN application is to translate dialed (VPN) numbers into a routable directory number (e.g. phone number). Thus a user need only dial, say “1 2001” and the application will translate this into the phone number of a remote site (say, “1 973 829 2001”).
- *Voice-activated dialing (VAD)*: VAD allows users to initiate calls by speaking the name or number of the destination party rather than dialing DTMF digits or typing the number from a terminal. This type of service has numerous benefits such as: simpler handsets (no digit pad is required), fewer mis-dialed numbers, and faster dialing times.
- *Click-to-dial (CTD)*: CTD is a hybrid Internet/PSTN service that allows a terminal user browsing WWW pages to request a call setup by simply clicking a number or name displayed on the terminal. This service is particularly useful for providing catalogue shopping, banking or travel agent online services with the capability of letting a user speak to an agent using the telephone system.
- *Voice-recognition based User Agent (VRUA)*: The VRUA allows a user to access and customize their services through voice commands alone. For example, the Voice-activated dialing service described previously is an example of a service that could be provided by the VRUA.
- *Meet-me conference (MMC)*: This service allows users to participate in a pre-arranged conference by dialing into a conference bridge. Several participants can join, and leave the conference at any time.

These services can be mapped into capabilities that must be made available via the API, and typically these capabilities are also required for numerous other applications. For example, VVPN requires number lookup and translation capabilities during call setup; this facility is also required for toll-free calls. VAD allows voice recognition and user interaction facilities. CTD requires interaction between heterogeneous network types as well as voice/data integration. VRUA represents services that allow users to become self-serve customers, and finally, MMC represents the ability to perform true multi-party calling.

Additional requirements defined for JCC/JCAT include one that it support first-party as well as third-party call control. Here “first-party” control means that a call is initiated directly by a user, using appropriate terminal equipment, via underlying signaling protocols appropriate for the network the user is connected to. Third-party call control refers to the situation where an application program initiates a call to connect two (or more) users or end devices. An example is a hotel wake-up call service, where the application program rings the guest’s phone and connects him or her to an operator or automatic playback device.

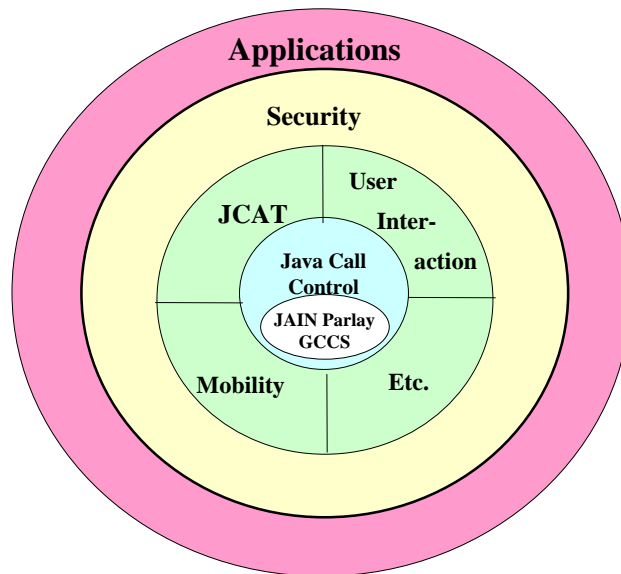
In general JCC/JCAT will also be required to support most of the features available as AIN or switch-based features in the PSTN, such as call forwarding, call waiting, etc. In addition, it is explicitly a requirement to keep the JCC/JCAT API in harmony with existing Java APIs for call control, in particular JTAPI (e.g. allowing third-party call control using mechanisms similar to JTAPI.)

## 4.2 JCC/JCAT structure

Based on the requirements above, the following structure has emerged for the JCC/JCAT API. The JAIN JCC/JCAT API specification shall consist of a core package (JCC) which supports the basic call model

for call processing and control, and an extension package (JCAT) to support the coordination and transaction related methods. In the rest of this section we describe the structure of the JCC/JCAT API. We stress again that this is not a final structure and may be (substantially) modified in the final specifications.

The proposed JCC/JCAT structure is shown diagrammatically in Figure 3. The core package, namely JCC, will be identical to that for JTAPI's core package (i.e., without any of the JTAPI extension packages), in particular the JTAPI 1.2 Core. Thus all the objects, methods, events, and exceptions of the JTAPI core package are included in JCC. In particular, the FSM of all the objects of JCC (including the Connection object) will be identical to the JTAPI 1.2 core.



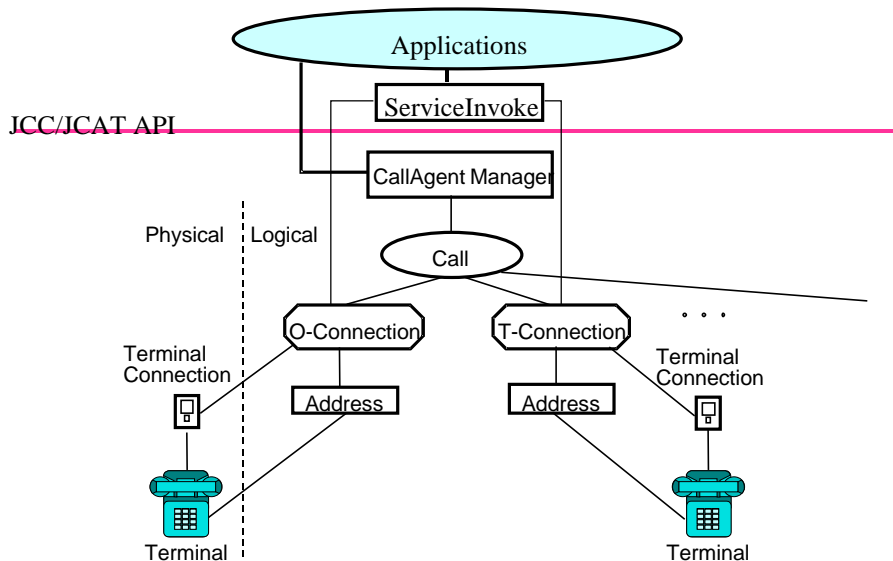
**Figure 3: Proposed JCC/JCAT API structure**

It is proposed that JCAT will be an extension package to JCC. JCAT will extend JCC in the following specific ways:

- Extend the Connection object FSM to become a richer FSM similar to that for AIN. It is proposed that in fact the JCAT Connection object essentially be a union of the originating and terminating FSMs of AIN.
- Add facilities for AIN-style triggers.
  - Allow applications to register, via the Address object, that the application should be invoked when a particular trigger in the Connection object FSM fires.
  - Allow every state transition in JTAPI's Connection object FSM to be treated as a trigger. When the state transition takes place, if an application is registered for the transition as a trigger, call processing is suspended, the application is invoked, and call processing resumes when the application so directs.
  - Introduce a callback object above the API to allow applications to be invoked in a uniform manner. Thus the application must implement a specific object, called *ServiceInvoke()*, and expose its interface. When the application registers for a trigger, it provides a reference to this object. When the trigger fires, a pre-specified method on the *ServiceInvoke* object is invoked, which in turn invokes the application. It is to be remarked here that problems in the form of conflicting instructions resulting from a trigger can arise at this point. This complex problem of feature interaction is not solved by the JCC/JCAT API and is outside the scope of the API. It is assumed that feature interaction is managed at the application level, or by means of provisioning and

management functions that may determine, for instance, which applications may register for the same trigger and their relative priority.

A schematic of the proposed API is shown in Figure 4. We omit further details of the proposed JCC/JCAT API structure as it is still far from being finalized. We observe that in future versions of JAIN Parlay, the JAIN Parlay GCCS call model will be a true subset of JCC, and may in fact become identical to JCC.



**Figure 4: Proposed objects in the JCC/JCAT API**

## 5. Concluding remarks

It is clear that future telecommunications networks will be integrated networks of packet-switched (ATM or IP), circuit-switched and it is also clear that to provide the large portfolio of innovative services that service providers desire to offer in these networks, open network APIs will be required. A central component of these APIs will be the API for call control (initiating and manipulating calls) and coordination and transactions (invoking and executing services before or during calls). The JCC/JCAT API will define the Java API for call control and coordination and transactions. Given the growing popularity of Java for applications development, it is expected that the JCC/JCAT API will be an important tool for rapid service development in future telecommunications networks.

We point out there are numerous issues that still need to be addressed in the implementation of a platform that supports the JCC/JCAT API. These include issues of performance and capacity planning; for instance how one would dimension such a call processing platform to meet desired performance objectives. Reliability and availability issues are also extremely important, and it is expected that testing would be a significant task. These issues are challenging for any high-volume, high-reliability call processing system, and addressing them adequately is similarly a challenge for any implementation of the JCC/JCAT API.

## References

---

- [1] F. Anjum, F. Caruso, R. Jain, P. Missier and A. Zordan, “*ChaiTime: A System for Rapid Creation of Portable Next-Generation Telephony Services Using Third-Party Software Components*,” *Proc. IEEE Conf. Open Arch. and Network Prog. (OPENARCH)*, Mar. 1999.
- [2] S. Roberts, *Essential JTAPI*, 555 pp., Prentice Hall, 1999.
- [3] Bobby Krupczak, Kenneth L. Calvert, Mostafa H. Ammar, Implementing Communication Protocols in Java, *IEEE Communications Magazine*, October 1998.
- [4] International Telecommunication Union, "Visual telephone systems and equipment for local area networks which provide a non-guaranteed quality of service," Recommendation H.323, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, May 1996.
- [5] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg, "SIP: session initiation protocol," Request for Comments (Proposed Standard) 2543, Internet Engineering Task Force, Mar. 1999.
- [6] T. Russell, *Signaling System 7*, 500 pp., McGraw-Hill, 1998.
- [7] U. Black, *The Intelligent Network*, 208 pp., Prentice-Hall, 1998.
- [8] Sun Microsystems, *Java Telephony API v. 1.2 Specification*, 1998. Available from <http://java.sun.com/products/jtapi>
- [9] Microsoft, *IP Telephony with TAPI 3.0*, white paper, Dec. 1998. Available from [http://www.microsoft.com/ISN/whitepapers/ip\\_telephony\\_with\\_ta.asp](http://www.microsoft.com/ISN/whitepapers/ip_telephony_with_ta.asp).
- [10] Maureen O'Reilly Roche, “Call Party Handling Using the Connection View State Approach: A Foundation for Intelligent Control of Multiparty Calls,” *IEEE Comm.*, June 1998.
- [11] J. Farley and M. Loukides (eds.), *Java Distributed Computing*, 384 pp., O’Reilly, 1998.
- [12] R. Englander, *Developing Java Beans*, 316 pp. O’Reily, 1997.
- [13] T. Valesky, *Enterprise JavaBeans*, 326pp., Addison-Wesley, 1999.

## ACRONYMS

AEG	Application expert Group
AIN	Advanced Intelligent Network
API	Application Programming Interface
ATM	Asynchronous Transfer Mode
DTMF	Dual-Tone Multifrequency
EJB	Enterprise JavaBeans
FSM	Finite State Machine
GCCS	Generic Call Control Service (in Parlay)
IDE	Integrated Development Environment
IN	Intelligent Network
INAP	Intelligent Network Application Protocol
IP	Internet Protocol
ISUP	Integrated Services Digital Network User Part
ITU	International Telecommunication Union
ITU-T	International Telecommunication Union – Telecommunication Standardization Sector
JAIN	Java APIs for Integrated Networks
JCAT	Java Coordination and Transactions
JCC	Java Call Control
JSCE	Java Service Creation Environment
JSLEE	Java Service Logic Execution Environment
JTAPI	Java Telephony API
MGCP	Multiple Gateway Control Protocol
PBX	Private Branch Exchange
PEG	Protocols Expert Group

---

PSTN	Public Switched Telecommunications Network
QoS	Quality of Service
SCE	Service Creation Environment
SLEE	Service Logic Execution Environment
SCP	Service Control Point
SIP	Session Initiation Protocol
TAPI	Telephony API
SS7	Signaling System No. 7
TCAP	Transaction Capabilities Application Part
WWW	World Wide Web