BULGARIAN ACADEMY OF SCIENCES

CYBERNETICS AND INFORMATION TECHNOLOGIES • Volume **6**, No 3

Sofia • 2006

A Mark-up Approach to Service Creation Based on Open Interfaces

Ivaylo Atanasov

Telecommunications Department, Technical University of Sofia,
E-mail: iia@tu-sofia.bg

Abstract: *This paper presents a new mark-up approach to service creation in Next Generation Networks. The approach allows deriving added value from network functions exposed by Parlay/OSA (Open Service Access) interfaces. To demonstrate approach applicability markup language constructions for data and method definitions, flow control, time measuring and supervision, and database access are developed. The comparison with existing standardized service creation technologies is performed.*

Keywords: *Markup languages, NGN service creation, Parlay/ OSA interfaces.*

I. Introduction

To survive in the competition for subscribers, network operators and service providers need to offer attractive, content rich, highly customized and ubiquitous applications and services.

The first step towards feature rich services is the concept of Intelligent network (IN). The idea of the IN has been to offer a set of generic service building blocks (SIBs) to create a multitude of value added services, similar to a distributed operation system that enables plenty of applications to be executed. This means that the combination of IT middleware and telecommunications system enlarged drastically the programmability of the telecommunications network. It has to be recognized, that the IN concept decoupled the service provision from the underlying network, which allows in principle to provide IN-based services on top of ohe different bearer networks. Thus the IN has been applied originally to the fixed networks and later to the mobile ones. The IN suits very well to circuit switched environment with centralized control. It is not applicable to packet switched communications with distributed control.

Applications have to be portable both across different network interfaces, as well as across a variety of end user interfaces and thus they have top be provided

through application programming interfaces (APIs). APIs at the service creation level should not only allow developers to work in a familiar programming environment, but to integrate the protocols required to handle calls with those that control other forms of communication on the web. Standardized APIs provide functional abstraction allowing applications to achieve portability across disparate systems and networks.

Parlay/ Open service access (OSA) APIs were inspired by the idea to provide multimedia services for converging networks and to open network interfaces for the 3rd party service developers. These APIs allow software developers to create new services of components. Although defined for UMTS networks, Parlay/OSA APIs are also applicable to next generation network domain providing a standard object oriented model for delivering services.

Programming languages like C++ and Java might be used for implementing APIs. However the declarative languages like eXtensible Markup Language (XML) are much easier for description of service logic in certain domain. XML is used as a basis for defining object parameters, as well as for the glue between low-level APIs and the application logic, which itself consists of these variously assembled objects. On the basis of this type of framework, it is possible to conceive of whole generalized applications (a voicemail program, for example) being defined as a set of standards, portable objects, which could then be made available as resources to a more specific application that might need to employ them.

Several XML-based languages such as CPL, CCXML, VoiceXML and SCML for service creation are developed [1, 2, 3]. These languages can be used to create applications combining network functions mainly for call control and user interaction. The existing markup languages can not derive added value from mobility, charging, terminal capability, messaging and others and none of them supports the whole variety of network functions accessed through Parlay/OSA interfaces.

In this paper we present in brief a new mark-up language called Service Logic Processing Language (SLPL) oriented towards Parlay/OSA APIs and that can be used for service creation. SLPL combines attractive features of XML-based languages, flexibility and expressiveness of programming languages. We compare SLPL service creation capabilities with those of the existing standardized markup languages. Further to illustrate the language applicability it is discussed how traditional intelligent network services can be implemented in OSA environment by the use of SLPL. Last, SLPL is evaluated with respect to criteria defined in [4] for classification and comparison of different service creation technologies.

II. Standardized mark-up languages for service creation and SLPL

Call Processing Language (CPL) is a language for call processing in Session Initiation Protocol (SIP) based telephony network. CPL tells a SIP server what to do with a call. On contrary, SLPL is oriented towards Parlay/OSA APIs and is protocol agnostic. Parlay/OSA APIs provide application developers with programmability of network resources such as telecommunication protocol stacks by defining these resources in terms of objects and methods, data types and parameters that operate on those objects hiding network specificity (Fig.1). SLPL provides means for data types and method definition. Service logic described by SLPL can access networks functions by invocations of Parlay/OSA interface methods.

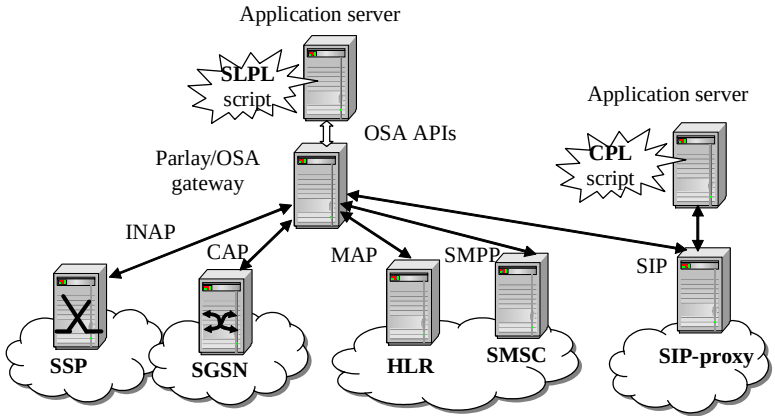

Fig. 1. Place of SLPL and CPL scripts in programmability architecture

Each CPL script installed in a SIP server has an owner, and is always associated with the address of this owner. CPL describes decision structures for routing incoming or outgoing calls. CPL does not provide means to handle multiparty or conference calls. Based on Parlay/OSA call control features, it is possible with SLPL to manipulate the individual parties in a call and their connections. Within the context of a call, SLPL service logic can add or drop call parties and connections, create multimedia call, delete multimedia call legs and define additional relationships between the parties. Further, SLPL supports the whole variety of Parlay/OSA service capability features. Besides from call control the language can derive added value from network functionalities as mobility, data session control, user interaction, messaging, presence and availability and others.

CPL is targeted at the end user who can upload his CPL script on a SIP server. CPL does not allow internal to the service provider actions such as charging. The intention of Parlay/OSA APIs is to give service provider full control over service capability features and thus SLPL is suitable for 3rd party development.

CPL can be used as a high-level service-creation language with restricted expressive power. It allows end users to define their call-processing preferences over the SIP-based network. On contrary, SLPL supports means for flow control, which draws the language near to programming languages. Flow control statements such as “if”-statement, “case”-statement and “while”-statement provide flexibility in service logic description. “If”-statement is used to check a logical condition, with “case”-statement decision may be done based on multiple choices and “while”-statement is used when a set of iterations has to be repeated when a predefined condition is true. “Try-catch” language construction makes possible to capture exceptions arising from method execution and to define exception specific processing.

In telecom applications the notion of time is important and the use of timers is frequent. The normal use of time and timers is when forming delays, supervising functions to be performed and measuring intervals. Time measuring and supervision means in CPL and SLPL are semantically overlapped i.e. both languages provide constructions handling with intervals and time. But while in CPL time handling capabilities are presented in a declarative form, in SLPL time constructions are rather procedure-oriented. Usually it is said, that the declarative form is easier to use while the procedure-oriented presentation provides more flexibility and expressive power.

In CPL time-switches allow a CPL script to make decisions based on the time and/or date the script is being executed. CPL supports the notion of timers without explicitly defining timer concept. Specific call routing may be done at repeating intervals secondly, minutely, hourly, daily, weekly, monthly, yearly. Time switching can be applied to a call during the set-up phase but there are no means to affect the call during the connection phase. For example, it is not possible to send the user a reminding message every 2 minutes during the active phase of the call.

In SLPL to handle with notion of “real” time predefined types “Date”, “Time”, “DateTime” and “Duration”, based on the type float are introduced. Two operations “CurrentDate” and “CurrentTime” are used to acquire the value of the current time. Also, the concept of timers is adopted as a predefined type. When a timer is set using operations “SetTime” or “SetDate”, a “Tme” or “Date” value is associated with the timer respectively. When a timer is reset by “Reset” operation, the associated value is lost. In many applications the decisions are made on the days of week so the enumerated type “Day” is defined and is used to denote the days of week.

In CPL locations for routing calls can be specified up through external means. CPL allows the SIP server to query an external database to retrieve the locations. CPL does not allow writing in a database, for example the CPL script can not modify data in subscriber database. SLPL provides service logic with means for database access. In SLPL the method “DB_modify” is used to update, insert or delete record(s) in an existing table and the result it returns corresponds to the number of modified records. The method “DB_retrieve” is used to retrieve data and the result returned consists of database response (for example the requested information). The database query is in a form of SQL statement and in method invocation this query is an argument of type string. As the method “DB_retrieve” returns a string, that string has to be converted in a reasonable SQL response. This conversion is done by another method “DB_conversion” which returns a set of structures representing records retrieved from the database. For each SQL statement a different "DB_conversion" method is defined that reflects the structure of logical records returned in database response.

CPL does not support user interaction. User interactions could be implemented in CPL by proxying calls to a special voice response server. To allow a user to interact with a web server through voice recognition technology, VoiceXML can be used. VoiceXML is also XML-based language designed for creating audio dialogs that feature synthesized speech, digitized audio, recognition of spoken and DTMF key input, recording of spoken input, telephony, and conversations with mixed initiative. SIP and VoiceXML can be used together to initiate or terminate not just signaling or control sessions but also content session. As SLPL supports user interaction service capability feature most of the services can benefit from this Parlay/OSA API. It is the responsibility of the service logic to conduct the user interactions. For example, when a user approaches a petrol station the service logic can determine his position using Parlay/OSA Mobility interface and initiating a call it can play the user a message using User Interaction interface.

More details on SLPL construction and usability might be found in [5, 6, 7, 8 and 9].

Table 1 shows how CPL elements can be translated to SLPL.

Table1. Mapping of CPL elements to SLPL

CPL element	SLPL equivalent
Address-switch	"Case"-statement with eventInfo parameter of the Call control method callEventNotify() as an identifier for comparison
String-switch	"Case"-statement with the corresponding string as an identifier for comparison
Language-switch	"Case"-statement with the corresponding string as an identifier for comparison
Priority-switch	"If-then else"-statement
Time-switch	"Case"-statement with time-related construction
Location	Corresponds to redirectingAddress parameter of the Call control method routeReq()
Lookup	Method DB_retrieve combined with method DB_conversion
Remove-location	"Case"-statement combined with method DB_modify
Proxy	Invocation of Call control method routeReq() with a specified redirectingAddress parameter
Reject	Invocation of Call control method release()
Redirect	Invocation of Call control method routeReq() with an update redirectingAddress parameter
Mail	Invocation of Multi-media messaging method sendMessageReq()
Log	Method DB_modify by the use of result returned by Call control method getCallInfoRes
Subaction	Method definition and invocation

III. Intelligent network services implemented in OSA environment

From subscriber's point of view network services are described as composed of service features. Service feature is a part of service functionality. A service feature is built of one or more Service independent building blocks (SIBs). Features are often combined by using a set of SIBs, but they are not themselves services, that is they should not be used directly by users. They are combined with other features, SIBs or both to create a service.

All IN services can be implemented by OSA APIs using SLPL, so these services can be available to users in IP-based networks.

Call related IN services implemented in OSA environment use Call Control API. The IN call-unrelated services allow user interactions outside the context of a call, for example in mobile networks user authentication and location updating. These services implemented by OSA APIs use Mobility API. It is possible call-related and call-unrelated user interactions to use out-channel signaling connections (for example short message). These IN services implemented by OSA APIs use Multimedia Messaging API or Data Session Control API. To allow service logic to interfere based on call-related or call-unrelated events it has to subscribe to receive notifications form the corresponding OSA APIs for those events.

Most of IN services may need originating or terminating user prompting and when implemented in OSA environment these services use User Interaction APIs. The IN services with special charging treatment are implemented in OSA using Charging API. IN service features for access control such as authentication, authorization and off-net access or restriction services like call screening and closed user group require service logic to consult with database. SLPL constructions for database access allow service logic to retrieve service-specific subscriber data from an external database.

A. SIBs and SLPL

Each SIB has one logical starting point and several logical ends. The logical start and ends are where the SIB is connected into the control flow. Multiple logical ends allow each SIB to be used as a decision point in the service script. The SIB acts on call instance data (CID). This is the call-dependent data, and includes the originator's address, the dialed number, and the destination routing address. Further the SIB may need service support data (SSD), which is independent on the call. Fig. 2 shows the structure of a SIB with its parameters.

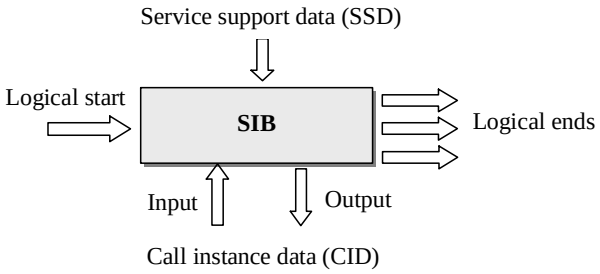

Fig. 2. SIB

Each of the SIBs, which functionality is used in service logic may be described by a method in SLPL. The most general SLPL description of a SIB is shown in Fig.3.

```
<method name="theSIB">
        <arguments>
                <argument name="SSD" type="sibSSDType"/>
                <argument name="inCID" type="sibInCIDType"/>
        </arguments>
        <returns type="theSIBResult"/>
        <raises>
                <exception name="theError"/>
                <!—description of all errors -->
        </raises>
        <body>
                <!—description of method functionality -->
        </body>
</method>
```

Fig. 3. General SLPL description of a SIB

Service support data are described as method parameters. Call instance data, on which SIB acts are also described as method parameters. Call instance data that are result of SIB execution are described as result of method invocation (i.e. data returned by method). Data returned by the method might be composed of several components organized as structure. Errors that can be raised during SIB execution are described as exceptions. SIB functionality is described in the method body.

Global functional plane of Capability Set 1 defines 14 SIBs. SIBs functionality is elementary although the complexity varies for different SIBs.

The SIB "Algorithm" applies a mathematical algorithm to data in order to produce a data result. It can be used to implement a simple arithmetic operation e.g. increment of a counter. In SLPL the method body is the placeholder of the arithmetic operation e.g. "increase" or "decrease", to be applied on data described as method parameter.

The SIB “Charge” determines the special charging treatment for the call. In SLPL the method body with the respective functionality involves setting values of parameters to the corresponding methods of OSA Charging API.

The SIB “Compare” performs a comparison of an identifier against a specified reference value. In SLPL its functionality is described by language construction “if-then-else”-statement.

The SIB “Distribution” distributes calls to different logical ends based on user-specified parameters. In SLPL its functionality is covered by language construction “case”-statement with multiple choices.

The SIB “Limit” limits the number of calls related to IN provided service feature. In SLPL the limiting functionality is achieved by applying time-related language construction when calls are passed for specific duration at specific intervals. When the functionality is concerning counting algorithm for call limiting then SLPL ‘increase’ operation is applied.

The SIB “Log call information” logs detailed information for each call into a file. In SLPL to log call-related information the method for database modification might be used.

The SIB “Queue” provides sequencing of IN calls to be completed to a called-party. In SLPL to organize queue a numbered list of data elements might be defined. Each data element presents the set of parameters for specific call waiting on the queue. The primitives for the queue i.e. pushing and popping are realized by simple iterations over the list.

The SIB “Screen” performs a comparison of an identifier against a list to determine whether the identifier has been found in the list. In SLPL to access the SSD list service logic uses method for database retrieval. Using the “while”-statement and if language constructions the equivalent functionality is achieved.

The SIB “Service data management” enables end-user specific data to be replaced, retrieved, incremented or decremented. In SLPL corresponding functionality is achieved by using of database retrieval and modification constructions.

The SIB “Status notification” provides the capability of inquiring about the status and/or status changes of network resources. This functionality in SLPL is achieved by straightforward invocation of the corresponding OSA methods of User Status service.

The SIB “Translate” determines output information from input information. In SLPL the functionality might be achieved by defining static lists or by database retrieval.

The SIB “User Interaction” allows information to be exchanged between the network and a call party. This functionality in SLPL is achieved by straightforward invocation of the corresponding OSA methods of User Interaction API.

The SIB “Verify” provides confirmation that information received is syntactically consistent with the expected form of such information. In SLPL the corresponding functionality is achieved by language constructions “case”-statement and/or “if-then-else”-statement.

The SIB “Basi Call process” is specialized SIB which provides the basic call capabilities. This functionality in SLPL is achieved by straightforward invocation of the corresponding OSA methods of Call Control API.

B. An example of SIB functionality described in SLPL

In order to illustrate the SLPL applicability to IN service description in this section we give an SLPL description of SIB "Distribution" in the context of a specific service. This SIB distributes calls to its different logical ends based on a user identified algorithm. Potential service applications of this SIB are Mass calling, Televoting and Freephone.

The service support data include the following:

- algorithm type – percentage, sequential, time of day and day of week;
- number of logical ends;
- algorithm parameters;
- CID field point that specifies where in output call instance data the error cause will be written.

There are no input call instance data. The SIB output enumerates logical ends. The output call instance data describe the error cause. Fig. 4 shows the graphical representation of SIB "Distribution".

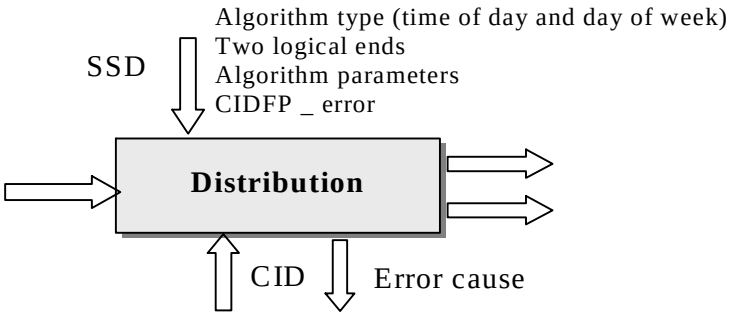

Fig. 4. Distribution SIB

Let us consider the SLPL description of the Distribution SIB in the context of service which distributes calls to different logical ends based on time of day and day of week. For example, the call distribution might be done based on the working hours of the day i.e. between 8 a.m. and 5 p.m. and on working days of the week To implement this distribution functionality we need two Distribution SIBs – one for time-dependant distribution and another one for day of week-dependant distribution as depicted on Fig. 5.

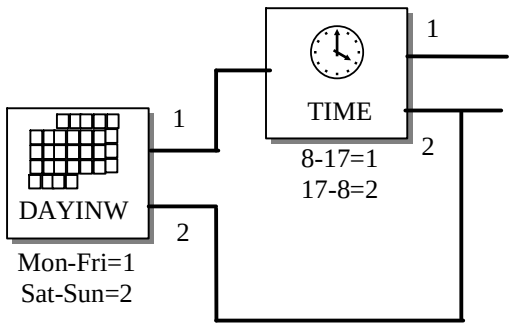

Fig. 5. Calls distribution based on time and day of week

In SLPL this kind of functionality is described by the use of one method that exploits time related language constructions as shown in Fig. 6. For the sake of simplicity just one of the method arguments is checked against whether invalid value is assigned.

```
<method name="theDistribution">
    <arguments>
        <argument name="beginTime" type="Time"/>
        <argument name="endTime" type="Time"/>
        <argument name="theDay" type="Day"/>
    </arguments>
    <returns type="Integer"/>
    <raises><exception name="invalidArgument"/></raises>
    <define>
        <variables>
            <id name="theMoment" type="Time"/>
        </variables>
    </define>
    <body>
        <if>
            <condition test="beginTime NOT BTW 00:00 AND 24:00"/>
            <then>
                <raise name="invalidArgument"/>
            </then>
        </if>
        <case refid="theDay">
            <on val="SU"><return value="2"/></on>
            <on val="SA"><return value="2"/></on>
        </case>
        <set refid="theMoment">
            <value> <CurrentTime/> </value>
        </set>
        <if>
            <condition test="theMoment BTW beginTime AND endTime"/>
            <then>
                <return value="2"/>
            </then>
        </if>
        <return value="1"/>
    </body>
</method>
```

Fig. 6. SPLP description of time and day of week based distribution of calls

IV. Assessment of SLPL using evaluation criteria

In [4] are defined evaluation criteria for classification of service creation technologies. In brief these criteria include the following: supported network capabilities, mapping towards reference architecture, interface abstraction, kind of interface and description language, suitability for 3rd party development and usability.

The suggested markup approach to service creation is based on Parlay/OSA interfaces and all network capabilities exposed by these APIs are accessible. SLPL provides application developers with functional abstraction of underlying network and allows adding value to call control, data session control, mobility, user status, charging, user interaction, messaging and others.

The second criterion defines the place of the SLPL in NGN service architecture. The SLPL interpreter is placed either at the application side of the programmability architecture or at the Parlay/ OSA gateway side. This means that both the operator and the 3rd party can support programmability through SLPL scripting.

The third criterion evaluates SLPL interfaces regarding interface abstraction and kind as well as the kind of interface description language. SLPL provides high level abstraction hiding technical details of network technology from application developers. The kind of interface describing communication method by which SLPL exposes network capability is XML-based. SLPL is scripting language and SLPL interpreter parses the script at runtime. XML schemes are used to define data types and methods in SLPL.

As the Parlay/OSA technology opens network interfaces for 3rd party SLPL attracts larger application developer community.

SLPL might be used for service creation in many different scenarios: using scripts created by end users or using scripts created by service providers. SLPL scripts are highly customized allowing service or end user specific data to be stored in an external database.

One of the factors that determine language usability depends on availability of supporting tools. For SLPL are developed the following tools:

- Java based Backus Naur Form (BNF) translator for syntactical grammar verification and generation of basic interpreter classes
- Java based SLPL interpreter
- Translator of Interface Description Language (IDL) descriptions into SLPL.

Another factor that influences on usability of the suggested service creation approach and accompanying language is measured in terms of the knowledge/ experience needed the supporting of the Parlay/OSA interfaces minimizes necessity of preliminary knowledge for network protocol and technologies.

V. Conclusion

In this paper a new markup approach to service creation is presented. Following this approach the service logic can derive added value from network functionalities accessible through APIs. The service logic is described by the use of an XML-based language SLPL that is platform independent, lightweight and suitable for 3rd party application development. As the language is intended to be used with OSA APIs, services described in SLPL can benefit from network functions hiding network protocol complexity. On contrary to the other markup languages SLPL allows control over call-unrelated events such as change of position, user status, TCP/IP session, presence and availability and so on. SLPL follows closely the architecture and APIs definitions developed by Parlay/OSA. Traditional value-added services provided in intelligent networks can be implemented by SLPL and thus accessible also in IP-based networks. These language features get telecommunication service development near to IT community and shorten time to market.

References

1. Bakker, J., R. Jain. Next Generation Service Creation Using XML Scripting Languages. **www.argreenhouse.com/ papers/jlbakker/ bakker-icc2002.pdf**
2. Bakker, J-L., R. Jain. SCML, Next Generation Service Creation Using XML Scripting Languages. **www.argreenhouse.com/papers/ jlbakker/bakker-icc2002.pdf** (accessed November 2004)
3. Bakker, J., D. Tweedie, M. Umnehopa. Evolving Service Creation; New developments in network Intelligence, Teletronikk. 2004.
4. Falcarin, P., C. A. Licciardi. Analysis of NGN Service Creation Technologies. – IEC Annual Review of Communications, Vol. **56**, 2003.
5. Atanasov, I. New XML-Based Language for OSA User Interaction Interface Description. – In: TELECOM'2005, Varna, Bulgaria, Proceedings, 2005, 195-200.
6. Pencheva, E., I. Atanasov. New XML-based Language for OSA Mobility Interface Description. – In: TELECOM'2005, Varna, Bulgaria, Proceedings, 2005, 201-207.
7. Atanasov, I., E. Pencheva. Service Creation Using a New Mark-up Language. – In: TELSIKS'2005, Nish, Serbia and Monte Negro, Proceedings, 2005, 575-578
8. Atanasov, I., E. Pencheva. A New Service Logic Processing Language. – In: Electronics'2005, Sozopol, Bulgaria, Proceedings, 2005, Book 1, 150-155.
9. Atanasov, I., E. Pencheva. A Mark-up Approach to Add Value. – Enformatika, IJIT, Vol. **3**, 2006, No 4, 267-276.
10. Lennox, J., X. Wu. CPL: A Language for User Control of Internet Telephony Services, RFC 3880, 2004. **WWW document** (accessed March 2004).
11. Notis, 2006. Translator and Interpreter Resources. **http://www.notis-net.org/links/tilinks.html** (accessed April 2006).
12. Parlay Group. Parlay APIs Overview, 2002. **http://www.parlay.org/docs/ Spec3_es_20191501v010101m.pdf** (accessed January 2004).
13. O'Doherty, P. SIP and Java platform. 2003. **http://java.sun.com/pro-ducts/jain/SIP-and-Java.html** (accessed January 2004).
14. Sun Microsystems, 2001. The JAIN APIs: Integrated Network APIs for the Java Platform. **http://java.sun. com/ products/ jain/WP2001.pdf** (accessed January 2004).
15. VoiceXML Forum, (2000) Voice eXtensible markup Language VoiceXML. **http://www.voicexml.org/specs/ VoiceXML** (accessed February 2004).