Preprint: Proc. IEEE Intl. Conf. on Comm. (ICC), Apr. 28 - May 2, 2002.

Next Generation Service Creation Using XML Scripting Languages

John-Luc Bakker and Ravi Jain
Telcordia Technologies, Applied Research
445 South Street
Morristown, NJ 07960

Abstract- **The next generation of scripting languages for creating value-added services in converged networks will be based upon XML. Industry fora like Parlay, JAIN and OSA have developed open standard Application Programming Interfaces (APIs) to enable service creation in converged Next Generation Networks (NGN). While services can be developed in traditional programming languages (e.g. Java or C++) using these APIs, XML-based scripting languages offer some advantages. While not as flexible or powerful as a programming language, scripting languages are typically easier to learn, and are language and platform independent.**

In this paper we describe the architecture and framework (creation, deployment and execution) of XML-based service scripts in NGN. We focus on the Service Control Markup Language (SCML) being developed by the JAIN forum for call control scripts that is closely tied to the JAIN Java Call Control (JCC) API. SCML is intended to be part of a family of NGN service scripting languages that include facilities for user interaction, mobility, and other open NGN API functions. We compare SCML to the Call Processing Language (CPL) defined by the IETF and note that SCML offers several advantages. We also briefly compare it to the requirements that are being developed by the W3C Voice Browser working group.

I. INTRODUCTION

The next generation of telecommunications networks, for both public and enterprise environments, will consist of converged networks, using fixed and wireless, as well as circuit-switched and packet-switched infrastructure. The key promise of these Next Generation Networks (NGN) lies not in the ability to interconnect this diverse technology, or even in the potential cost reductions obtained by doing so, but *the ability to develop and deploy innovative and lucrative services rapidly and efficiently*.

A critical ingredient for rapid service introduction is the development of open and standard Application Programming Interfaces (APIs) that span diverse NGNs, allowing 3rd party application developers to produce new services in a manner analogous to the development of software in the Information technology (IT) industry. In the past few years several industry efforts have emerged to develop such open APIs, including Parlay [8], JAIN [7], and the Open Services Architecture (OSA) [17]. These efforts are now reaching fruition, with the initial versions of specifications being released and several equipment vendor announcements of current or planned products implementing the APIs (e.g. see presentations at the recent Parlay meetings [8]).

The next step required to make the promise of NGN a reality is to streamline and rationalize the process of service creation itself. In the traditional Public Switched Telephone Network (PSTN), service creation was typically carried out by service providers with specialized personnel using specialized languages. The development of the Internet and the rise of the eXtensible Markup Language (XML) as a language standard have recently prompted proposals that XML-based scripting languages be used for telecommunications service creation. Among other advantages, XML offers a medium that is platform, network and technology neutral, independent of underlying programming languages, and readable by machines as well as humans. It is thus promising as a basis for enabling 3rd party application developers, particularly those who may not be programmers, to develop NGN services.

Within the area of service creation, a central concern is to specify and model *call control*, i.e., the creation, manipulation, termination and teardown of communications sessions. (Note that while we use term "call" for simplicity, throughout the paper it refers in general to a communications session, i.e., not only to two-party voice calls but more complex sessions like a multimedia multi-party communications session.) In this paper we describe the architecture and framework (creation, deployment and execution) for XML-based scripting of services in an NGN.

Several XML-based call control Markup Languages (ML) have been previously proposed, including CPML, TML, XTML CPL, and CallML [1]. A comprehensive survey of these call control ML proposals is not the purpose or within the scope of this paper. Instead, we observe that the more mature and interesting of these proposals are the Call Processing Language (CPL) [3, 4] being standardized by the Internet Engineering Task Force (IETF) and the call control ML requirements [2] being developed by the Voice Browser working group in the World Wide Web Consortium (W3C).

Our examination of the CPL and VoiceXML activities leads us to conclude that there are some deficiencies in both these approaches. In the latter half of this paper we describe an alternative approach and proposed draft standard language, namely the Service Creation Markup Language (SCML). SCML is an XML-based NGN service scripting language currently being developed within the Service Creation Environment (SCE) Expert Group of the JAIN industry forum [5]. While the SCE Group currently focuses on call control aspects of NGN services, our view is that SCML consists of a family of such languages that need to be developed to model other essential service aspects, such as mobility, user interaction, etc. Thus the SCML family contains a Call Control Markup Language (CCML), among others. (In this paper we will use the term SCML generically to encompass the markup language for call control as well as others; when discussing call control aspects of SCML it is to be understood that CCML is intended.)

Since at the moment the VoiceXML activity for a call control ML consists only of a requirements document, in this paper we will largely focus on comparing SCML with CPL. SCML and CPL have many goals in common and overlapping scope. Where they differ is the level of abstraction of the underlying telecommunication network and environment in which they operate.

One of the advantages of SCML over CPL is that SCML follows closely the architecture and API definitions for call control being developed cooperatively by the joint industry standards working group on call control, consisting of representatives and members of four bodies, namely Parlay, JAIN, ETSI and OSA. In addition, as

we discuss below, SCML is defined at a higher layer of abstraction than CPL, offers richer functionality, and offers a more modular, reusable and extensible base for further development of service creation ML standards, including those for mobility, charging, etc. We thus argue that SCML will evolve as a "best of breeds", satisfying most of the call control requirements being developed by the W3C Voice Browser working group while offering distinct technical benefits over CPL.

The rest of this paper is organized as follows,. In the next section an architecture for NGN is reviewed and a framework for execution of XML-based scripts is introduced. We briefly describe how the Parlay and JAIN industry bodies are developing open APIs for this architecture, focusing on the JAIN call control APIs and their relation to SCML. In Section III we discuss the use of XML, XML Schemas and XSL for call control ML scripts. In Section IV we focus on XML scripting and the creation, deployment and execution of XML scripts. In Section VI we compare CPL and SCML in some detail. To make this comparison concrete we consider two common example services: Voicemail on Busy, and a Wake-Up Call. In Section VII we conclude with some brief discussion and description of future work.

II. NGN AND CALL CONTROL ARCHITECTURE

We briefly review the NGN architecture assumed as a basis for SCML in this paper (see Fig. 1). At the lowest layer the architecture consists of transport technologies (PSTN, Internet, wireless) and interworking gateways and hubs (trunking gateways and residential hubs). The intelligence of the NGN is contained at the next layer in a softswitch or Call Agent that controls the gateways (using protocols like MGCP or Megaco) and interfaces to the (Advanced) Intelligent Network (A/IN) [15] elements such as Service Control Points (SCP) via signaling gateways. The Call Agent is primarily responsible for interworking and completing calls or communications sessions; for value-added NGN services it exposes an open standard NGN API (e.g. JAIN, Parlay, OSA). In particular the API provides call control facilities. The NGN services may execute on the Call Agent itself or on separate Application Servers (not shown) or the SCP.

Above this layer lies the domain of service creation, which is normally carried out offline as opposed to the online processing of calls or services. Services can be created in a SCE with a programming language like Java, e.g. using JavaBeans [18], or using a markup language like SCML. The SCE may provide a library of basic JavaBeans or SCML scripts that can be composed to create more advanced services, possibly in a hierarchical fashion. As we discuss later, the scripts can be downloaded and interpreted or executed either at the Call Agent or at the endpoints e.g. PCs or smart phones connected to residential hubs or IP networks.

Thus broadly speaking, the architecture (e.g. in JAIN and Parlay) assumes a layering consisting of signaling, functions or functional interfaces, and application (or service) logic (see Fig. 2; note that in Parlay the functional interfaces are called "service" interfaces). The signaling layer consists, for example, of SIP, MGCP, and ISUP protocols. These protocols are designed to communicate efficiently to their protocol peers, not for user friendliness or robustness. Programming in such a protocol layer involves issues like timing, authentication, error recovery, etc and is typically complex and tedious. The functional interface layer abstracts these lower level details to offer call control, mobility, user interaction and other functions driven by signaling protocol messages. Thus the call control function allows calls to be represented and manipulated at an abstract level in a common (object oriented) model across multiple network technologies and signaling protocols using a common API. Finally, applications reside in the applications layer making use of the various APIs in provided by the functional interfaces layer.

In general, there exist endpoints or end systems (such as phones or PCs) at the edge of the NGN and what we generically refer to as "call servers" (e.g. Call Agents, Application Servers or SIP servers) in the core of the network. End systems can originate calls, and accept, reject, or forward incoming calls [3]. On the other hand, call servers can perform of the following operations on a call. They can:

- *proxy it*: forward the call on to one or more other call servers or end systems, subsequently calculating what to do with any responses received.
- *redirect it*: informing the network of an alternate address for the incoming call.
- *reject it*: inform the sending system that the setup request could not be completed.
- *originate it*: create a new call through inviting two or more end user systems in a coordinated fashion.

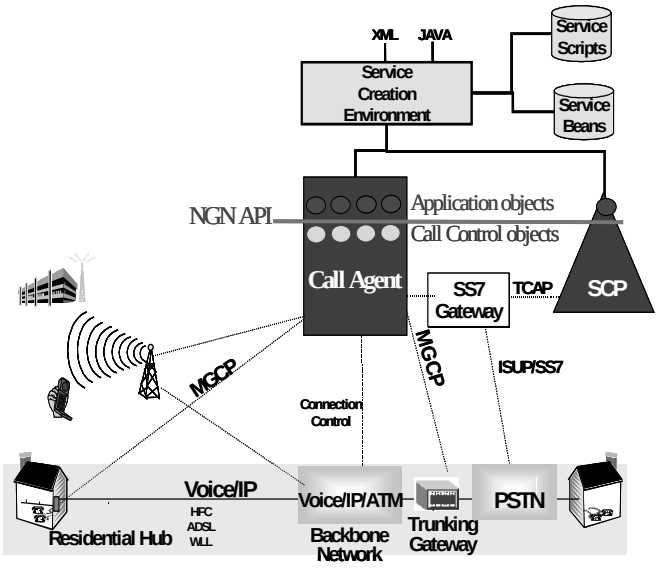

Fig. 1: NGN Architecture and SCML Scripts

Typically, call servers support applications that that may involve charging, user location, mobility, etc., and thus interact with other systems within the network, e.g. location databases, billing systems, etc. The following briefly discusses two industry initiatives that provide APIs to the functions provided by call servers.

A. Parlay

The Parlay group [8] provides 3rd party application developers with language-independent APIs that allow access to functions (or "services") including not only Call Control but Presence and Availability Management, Policy Management, User Interaction, Mobility Management, Content Based Charging, and others. Some of these are shown in Fig. 2. Access to these functions is managed through a Framework API that provides security and related

functions to expose the NGN infrastructure to 3rd party applications in a controlled and secure manner.

B. JAIN

The JAIN effort [6, 7] is similar in many respects to Parlay. Some of the differences are that JAIN (a) provides APIs not only at the functions layer but also at the protocols layer below it; (b) is not language independent since all the APIs are specified in Java; (c) explicitly defines a Service Logic Execution Environment (SLEE) and (d) defines a SCE. A number of APIs are provided for protocols such as SIP, MGCP, MAP, and INAP. On top of these protocols a number of protocol agnostic services are defined: Call Control, User Interaction, and Mobility Management are targeted to be essentially Java versions of the Parlay APIs. Similarly, the Parlay Framework was mapped to the JAIN environment to provide secured access to the services for third party applications. Applications running within SLEE, or created by the SCE, can exploit the JAIN suite of functional as well as lower-layer protocol APIs and also benefit from other Java APIs.

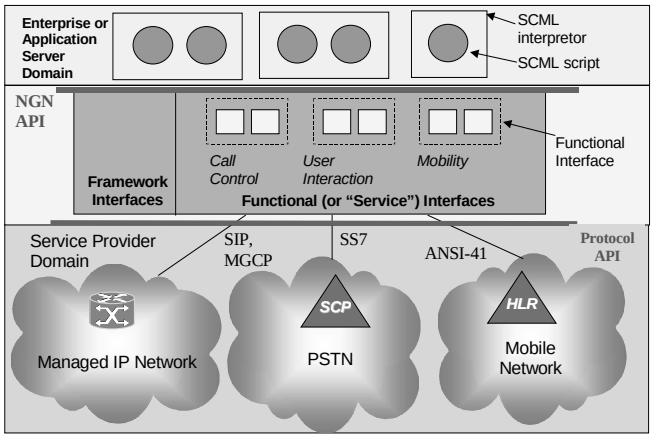

Fig. 2: NGN Layering and APIs

The basic call control service of JAIN is the Java Call Control service. Providing an API for call control that covers the needs of the industry, while remaining focused, is a daunting task. Early in the process within the JAIN group it was decided to define Java Call Control (JCC) as a simple API that supports today's main revenue generating applications. Java Coordination and Transaction (JCAT), on the other hand, inherits from JCC and supports more advanced applications. SCML is being defined as a scripting language tied closely to JCC. We briefly describe JCC and JCAT here.

JCC is a call control API that typically supports first-party and third-party creation and manipulation of multi-party calls, and common services such as FreePhone (or toll free) numbers, number translation, call forwarding, originating and terminating call screening, etc. JCC 1.0 is completed and is being implemented by several vendors [10]. It aligns with JTAPI [21] and is similar to Parlay's Multi-Party Call Control Service (MPCCS); future versions are expected to align closer with MPCCS.

JCAT [11] is under development within JAIN and is expected to support a richer class of call control applications than JCC. Examples of proposed JCAT capabilities are support for various call transfer flavors, call merging, modeling of terminal capabilities, and call 'navigation'. Through the call 'navigation' capability, an application can query the calls that a JCAT implementation knows of. If a call object is available that satisfies the query, the application can control this call. For example, it may "barge in" to the call or into a particular connection, e.g. for call waiting. The JCAT API is typically deployed on Class 5 switches.

It is expected that soon after completion of the JCAT standardization work, a call control ML will be derived from SCML for JCAT.

III. USAGE OF XML

A. XML application creation languages

Languages such as SCML and CPL create applications that make use of the functions provided by the functional interfaces layer. Today, XML [12] is commonly seen as the preferred vehicle to create such applications. Aside from its standardization and readability by both machines and humans, XML offers several benefits. XML supports restrictions to its expressiveness that enables easy validation and determinability. In general, XML schemas allow the design of languages that can be non-expressively complete, thereby guaranteeing that the XML interpreter while using a limited amount of time and resources can execute the script. Finally, many tools and libraries exist to create, interpret and validate XML documents.

B. XML Schema

CPL is defined using XML Data Type Definitions (DTD) while SCML is defined using an XML Schema. XML Schema [13] represents a more recent specification methodology developed in the W3C and have several distinct advantages over DTD. In this section we briefly discuss why they are particularly beneficial for SCML.

Like DTD, XML schemas are a language for specifying and constraining the content of XML documents. However, unlike DTD, XML Schemas are written in XML, thus avoiding the programmer having to learn yet another notation, and define a set of well-known basic document elements, thus saving programmer the effort of defining them. While these features are convenient, for our purposes schemas have two additional important attributes. Firstly, XML Schemas are type safe and come with a set of predefined simple types, where DTD only accepts one type: string. More importantly, XML schemas allow the programmer to define data types as well as to restrict, redefine and extend them in a manner similar to inheritance in object oriented programming languages. This latter attribute of schemas enables modularity, extensibility and reuse of SCML Schema. A feature called *extension* is useful in a manner similar to adding behavior by using inheritance in an object-oriented language. This is very important since most NGN APIs, such as JCC, make extensive use of inheritance. In particular, the extension package JCAT inherits from JCC and adds richer functionality.

C. XSLT

The eXtensible Stylesheet Language (XSL) and XSL Transformations (XSLT) are powerful tools for transforming one XML document into another. XSL consists of XSLT plus a description of formatting objects and properties. Through XSLT [14] higher-lever scripts can be supported to hide recurring XML sequences that represent common patterns when developing scripts.

One can imagine XSL rules for e.g. Wake Up Call applications. In this case many of the recurring aspects of specifying a service, and in particular a Wake Up Call service, are stored in the style sheet *WakeUp.xsl*. The higher-level script *WakeUp.xml* need only contain very specific information e.g. the name of the announcement server that reads the "wake up" message to the user, and in fact could be created by the script writer with the help of a Graphical User Interface (GUI). The XSLT processor is used to generate the actual SCML script that will be loaded onto the Application Server or Call Agent and executed. Thus instances of the XML that supports "wake up call" class of service allows for easy authoring of such scripts. This is particularly useful for provisioning and deployment of services. For instance, the higher-level script need only specify the parameters specific to a particular user, e.g. phone number, wake-up time, etc. (see Fig. 3).

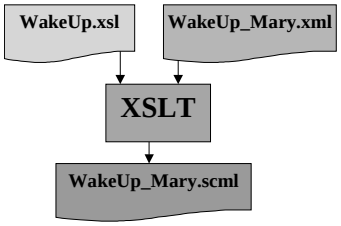

Fig. 3: Usage of XSLT to generate SCML

IV. Script Execution and Deployment

A script can run on a call server or (intelligent) endpoint system. It controls the server's or endpoint's functions: proxying (in case of a call server), redirecting, originating, or rejecting calls. The default behavior is in effect until processing in the call server completes, or a script takes over control (and disposition) of that call e.g. in a manner similar to triggers in A/IN. A script controls the call based on the information made available through other functional interfaces and based on settings controlled by subscriber (settings coded as rules in the script itself).

Typically, telecommunications scripts/applications are associated with an address, subscriber information, a point in call (e.g. in the originating and/or terminating portion of a call), and have access to auxiliary information such as location data through mobility management servers or an Interactive Voice Response (IVR) unit. Note that there may also exist applications that are not call-based. For example, an application that continuously tracks the whereabouts of end users for administration purposes is not necessarily activated through call-based conditions. For illustrative purposes, the remainder of this paper assumes call-based scripts. However, the concepts discussed in the remainder can easily be generalized and applied for activation of non-call based scripts.

Fig. 4 shows three phases in the lifetime of a script. In the SCE phase the script is created by a variety of possible means including XML tools (e.g. XML or XSL editors, XSLT), direct input using a general-purpose text editor, an XML add-on to a traditional PSTN SCE, or by converting Java (or JavaBeans) programs. In the deployment phase the script is validated for syntax and executability, and stored in a repository if valid. An offline service management program can query the repository and activate the script by downloading it for execution to a specialized XML processing engine (in this case, an SCML processor) that may have at its disposal a library of basic scripts. There are two possibilities here: the SCML processor could reside on the call server itself, or it could reside on a separate platform (e.g. an Application Server). In the first case, the SCML processor acts as an interpreter to convert the SCML instructions to the API exposed by the call server (in this case, JCC). In the second case the SCML processor can either make remote calls to the JCC interface (e.g. using Java RMI) or can send XML messages (via SOAP [22], for instance) to another XML processor located on the call server; this latter option involves more XML processing but makes the communication between the Application Server and the call server language-independent. The criteria for invoking the script are checked by a criteria checker. The criteria may consist of an A/IN style trigger or Java event in the call-processing platform (e.g. JCC supports such events) that is in turn caused by underlying signaling events. Alternatively, the criteria may consist of other activation conditions, such as time-based activation (e.g. Wake Up Call) or activation from other elements (e.g. Click-to-Dial scenarios or IVR interactions). As the script executes, it may in turn issue commands to the call server e.g. after performing a number translation or redirecting a call.

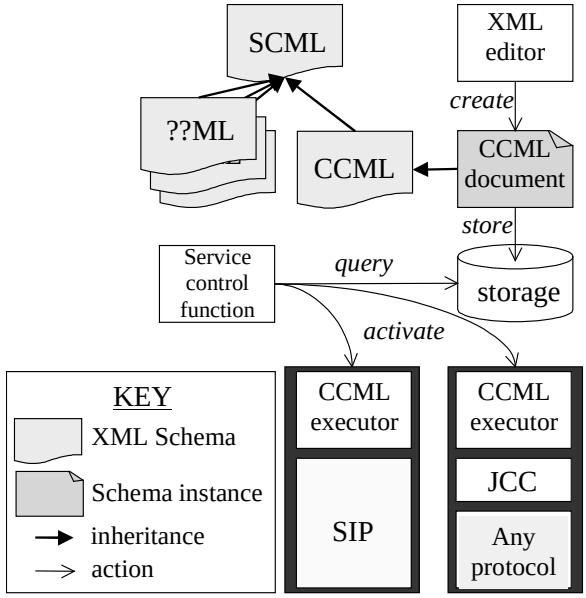

Fig. 4, Script creation, deployment and execution

Note that access by the scripts to the call-processing platform is expected to be restricted to authorized subscribers only. Subscribers are supposed to engage in a trusted relationship with the provider of the services. The Parlay Framework and JAIN Service Provider API (SPA) provide means to manage and control this access. The details of such authorization schemes may vary from administrative domain to administrative domain and are not in scope for this paper.

Scripts can have unwanted interactions within a system and between systems. Let us discuss the case in which scripts within a system interact. Assume a script *S1* with activation conditions *A* and a script *S2* with activation conditions *B*. Assume *S1* changes the conditions such that *B* is satisfied and, consequently, *S2* is invoked upon termination of *S1*. *S2*, however, changes the conditions such that *A* is satisfied and, consequently, *S1* is invoked again upon termination of *S2*. Obviously, the script execution support logic must monitor for this form of indefinite recursion and take appropriate corrective and informative actions. Note that it may very well be the case that the underlying protocols signal such loops.

In general this issue is completely analogous to feature interaction in A/IN systems and has been widely studied. We assume that resolution of feature interaction is in general outside the SCE itself, although an SCE could provide some support if so desired.

V. Analyzing open Call Control Scripting schemas

This section analyzes and compares CPL and SCML in the context of the architecture and framework given earlier in the paper and briefly considers the W3C Voice Browser working group requirements.

A. CPL

At the time of writing CPL is almost fully specified. Currently, a DTD and accompanying documentation is available as work in progress material ([3] and [4]). CPL is a non-expressively complete language; it cannot be used to create arbitrary complex scripts, thereby limiting the resources a script needs for successful execution. We now give a brief example of CPL use and discuss several of its deficiencies. CPL scripts are executed within our generic notion of call servers; CPL scripts process incoming and outgoing calls. The capabilities of the call server that executes CPL are limited; it can proxy, reject or redirect calls. CPL cannot originate calls towards two or more end users. The capability to originate a call within a call server is a significant service within A/IN networks.

The example CPL script in Fig. 5 is a simple script that redirects incoming calls to voicemail if the callee is busy (in the remainder we assume a basic familiarity with reading XML scripts). Note that, in CPL, upon receipt of the incoming call event, the callee's address is stored in a global variable. If the callee turns out to be busy, the location is cleared, changed into the user's voicemail address (`smith@voicemail.example.com`), and the call is redirected. We see that the global location variable can be manipulated through the `<location>` node.

```
<cpl>
 <incoming>
  <busy>
   <location
      url="sip:smith@voicemail.example.com"
      clear="yes">
    <redirect/>
   </location>
  </busy>
 </incoming>
</cpl>
```

Fig. 5, Example CPL Fragment: Voicemail on Busy

Typically, CPL scripts execute within the context of a user agent. A user agent is an entity that exists in SIP and H.323 networks. User agents do not exist in present day A/IN networks (in general the design of CPL is guided by the SIP & H.323 protocol). Therefore, CPL scripts are not protocol agnostic.

CPL is only activated through call events; it cannot be activated through non-call related events such as a timer. Hence, CPL cannot be used to write Click-to-Dial or Wake Up Call services.

CPL works with a call server's capability to provide location information of a registered user. It assumes a database that can be queried for the preferred location where the end user can handle the call. Additionally, the script can send mail through the `<mail>` node or log events through the `<log>` node. The last two nodes are particularly useful to give the subscriber feedback on script failures. CPL does not define how to interact with call servers that provide mobility management information (e.g. forward all calls to voicemail if the callee is driving in his car in the state New York), or with information provided by a user interaction system. Hence, CPL's architecture does not support all services provided by call servers as defined earlier.

B. SCML

We briefly discuss SCML here and compare it to CPL.

Note that SCML exists only as a work in progress material. The description here is based on SCML 0.2.2 and is not intended to represent the final specification. In the course of standardization within the JAIN SCE group and through interaction with other relevant bodies the SCML schema may change. The examples given in this document are for illustrative purposes only.

As motivated earlier, SCML is defined using an XML Schema that is derived from JCC. JCC provides an API to pure call control related capabilities and can support traditional A/IN services as well as NGN services such as Click-to-Dial, and is independent of the underlying network. JCC is truly protocol agnostic and can be mapped on top of SIP [16, 20], INAP, ISUP, and H.323 [16].

```
<scml>
 <terminating>
  <address-switch field="terminating">
   <address is="sip:smith@phone.example.com">
    <disconnected causeCode="CAUSE_BUSY">
     <routeCall connectionPtr="conC">
      <arguments>
       <targetAddress>sip:smith@voicemail.
          example.com</targetAddress>
      </arguments>
     </routeCall>
    </disconnected>
   </address>
  </address-switch>
 </terminating>
</scml>
```

Fig. 6, Example SCML Fragment: Voicemail on Busy

An example script in SCML is shown in Fig. 6 for the Voicemail on Busy service. In this script the activation criteria are more elaborate and are registered with the call server. The criteria are the callee's address, the fact that it concerns the terminating portion of the call, and the condition that call setup fails due to a busy callee. If these criteria apply, the scripts will be executed and the call will be redirected through specifying an alternative target address. The arguments XML node may contain more XML nodes, e.g. redirected address node. If such nodes are not specified, the call server is assumed to provide the appropriate values; in case such arguments are given the NIL value, the call server will clear existing values.

The example in Fig. 7 is included to demonstrate some more advanced features of SCML. It shows a Wake Up Call application; an end user is called each weekday morning at 6:00h and she can briefly recover from waking up while listing to some music. The script creates a call, routes it or notifies the end-user in case of failure by sending e-mail. As the script is not activated by an originating or terminating fragment of call, the scripts has to explicitly reference the call resources. The XML node `<createCall>` creates a call

resource identified as "`wakeupCall`" and the node `<routeCall>` routes the call, where the connection routed to `wakeup.com` is identified as "`wakeupMusicConnection`".

As mentioned before, a simpler script that is converted using XSL rules could generate the SCML in the Wake Up Call example. The examples given only demonstrate the call control service. As at the time or writing only the JAIN JCC API and Reference Implementation is publicly available [10], the SCML versions of other services, such as mobility management or user interaction, have not yet been attempted. Nevertheless, it is the intention to allow interworking between JCC SCML and other XML schemas within the telecommunication domains to enable scripting of feature-rich services.

```
<scml>
 <invocation>
  <time-switch>
   <time freq="weekly"
        byday="MO,TU,WE,TH,FR"
        dtstart="20010101T060000">
    <createCall callPtr="wakeupCall"/>
    <routeCall callPtr="wakeupCall"
       connectionPtr="wakeupMusicConnection">
     <arguments>
      <targetAddress>sip:jones@bedroom.
         phone.home.com</targetAddress>
      <originatingAddress>sip:jones@
         music.wakeup.com</originatingAddress>
      <originalCalledAddress
         xsi:nil="true"/>
      <redirectingAddress xsi:nil="true"/>
     </arguments>
     <failed>
      <mail url="mailto:jones@home.com?
         subject=wake%20up%20failed"/>
     </failed>
    </routeCall>
   </time>
  </time-switch>
 </invocation>
</scml>
```

Fig. 7, Example SCML Fragment: Wake Up Call

C. VoiceXML Call Control

VoiceXML call control only exists as a work in progress requirements document for a voice browser framework [2]. The scope of the language is not for building network-based call processing application. Rather, a voice browser is situated at the edge of the network executing in an environment that conforms to the notion of end user system. The document lists call initiation, VoiceXML interpreter context management, inter-session communication, conferencing capabilities, and call leg management requirements.

From the document we can conclude that most of the requirements are satisfied by the SCML framework and architecture proposed earlier, except for the requirements related to inter-session communication. As there exist no call server support for this capability, there is no SCML planned that satisfies these requirements.

We also question whether in general call control capabilities, even for end systems, should be built into a language intended for interacting with end users. In our view it is preferable to separate these concerns, and have VoiceXML scripts be able to interact seamlessly with SCML scripts to provide services.

VI. Concluding Remarks and Future Work

In this paper we have presented the methodology for service creation using XML-based scripting languages in NGN, covering an assumed NGN architecture as well as a framework for service creation, deployment and execution. We have also presented a brief initial overview of SCML, the XML-based language for call control currently being developed in the JAIN forum. We have compared SCML with the capabilities of an earlier language, CPL, as well as the requirements for call control being developed by the W3C Voice Browser working group.

We have argued that the SCML approach is superior to CPL in many respects. SCML is functionally richer, since it allows third-party call control while CPL does not. Also, since it is related to the JCC API standardized by the JAIN forum, it is truly protocol and network independent. CPL as currently defined interacts with call servers or other network servers that provide mobility or user interaction functions. In contrast, SCML is intended to work harmoniously with the suite of APIs defined by the Parlay and JAIN groups that include a variety of functional interfaces. Finally CPL's specification methodology is to use XML DTDs, while SCML is defined using XML Schemas. The latter provide not only language conveniences but are type safe and allow the programmer to define, restrict, redefine and extend data types in a manner similar to inheritance in object oriented programming languages. Since the Parlay and JAIN APIs make extensive use of inheritance this last capability is especially useful.

We have also considered the W3C Voice Browser working group's requirements for call control briefly and argue that call control features should not be added to a language intended for expressing voice user interaction control.

In further work we are refining the SCML schema and examining how current XML-based tools can be customized or extended to support the generation of SCML scripts.

Acknowledgment

We thank Alex Buckley, Philip Ber, Gary Levin, and Stefano Puglia of Telcordia for useful discussions and information.

References

[1] OASIS (Organization for the Advancement of Structured Information Standards), "The XML cover pages". See http://www.oasis-open.org/cover

[2] Porter, B., (ed.) "Call Control Requirements in a Voice Browser Framework", (work in progress) April 2001. See http://www.w3c.org/TR/call-control-reqs/

[3] Lennox, J. and H. Schulzrinne, "Call Processing Language Framework and Requirements", May 2000, See http://www.ietf.org/rfc/rfc2824.txt

[4] Lennox, J. and H. Schulzrinne, "CPL: A Language for User Control of Internet Telephony Services", (work in progress) November 2000. See http://www.ietf.org/internet-drafts/draft-ietf-iptel-cpl-04.txt

[5] Sun Microsystems, "JAIN Service Creation Environment (SCE) API Java Specification Request (JSR) 100", 2001. See http://jcp.org/jsr/detail/100.jsp

[6] Sun Microsystems, "The JAIN APIs: Integrated Network APIs for the Java Platform", June 2001. See http://java.sun.com/products/jain/WP2001.pdf

[7] Sun Microsystems, “The JAIN APIs”, August 2001. See http://java.sun.com/products/jain/

[8] The Parlay Group. See http://www.parlay.org

[9] Sun Microsystems, "JAIN Java Call Control (JCC) API Java Specification Request (JSR) 21", 2001. See http://jcp.org/jsr/detail/21.jsp

[10] Telcordia Technologies, Inc., "JAIN @ Telcordia", JAIN Reference Implementations download site, 2001. See http://www.argreenhouse.com/JAINRefCode/

[11] Sun Microsystems, "JAIN Java Coordination And Transaction (JCAT) API Java Specification Request (JSR) 122", 2001. See http://jcp.org/jsr/detail/122.jsp

[12] W3C, “Extensible Markup Language (XML) 1.0 (Second Edition)”, W3C Recommendation, October 2000. See http://www.w3.org/TR/REC-xml

[13] W3C, “XML Schema Part 0: Primer”, W3C Recommendation, May 2001. See http://www.w3.org/TR/xmlschema-0/

[14] W3C, “XSL Transformations (XSLT) Version 1.0”, W3C Recommendation, November 1999. See http://www.w3.org/TR/xslt

[15] IEC: International Engineering Consortium, “Intelligent Network (IN)”. See http://www.iec.org/online/tutorials/in

[16] Sasaki, H., J.-L. Bakker and P. O’Doherty, “Java Call Control v1.0 to Session Initiation Protocol Mapping,” http://java.sun.com/products/jain/wp_articles.html, Jan. 2002.

[17] The 3rd Generation Partnership Project (3GPP) Open Services Architecture (OSA). See http://www.3gpp.org.

[18] Leinecker, R., et al, *JavaBeans Unleashed*, Sams, 704pp, 1999.

[19] Jain, R., and F. Anjum, “Java Call Control,” in *Java in Telecommunications*, T. Jepsen (ed.), Wiley, 2001.

[20] Jain, R., J.-L. Bakker and F. Anjum, “Java Call Control (JCC) and Session Initiation Protocol (SIP),” *IEICE Trans. Comm.*, Vol.E84-B No.12, Dec. 2001.

[21] Roberts, S., *Essential JTAPI*, Prentice Hall, 672 pp., 1998.

[22] Box, D., et al, Simple Object Access Protocol (SOAP), World Wide Web Consortium (W3C), W3C Note, 8 May 2000.

[23] Subasinghe, C., and P. O’Doherty, “Java Call Control v1.0 to H.323 API Mapping,” http://java.sun.com/products/jain/wp_articles.html, Jan. 2002.