

# Evolving Service Creation; New Developments in Network Intelligence

JOHN-LUC BAKKER, DAVID TWEEDIE AND MUSA R. UNMEHOPA



*John-Luc Bakker (age?) holds an M.S. degree in programming aspects of distributed and parallel computing from the Delft University of Technology, The Netherlands, since 1996. Prior to his current position he worked with Lucent Technologies on component based service creation environments, advanced communication architectures, multimedia, and recent change code generation. He has published several papers in these areas. Bakker joined Telcordia in 2000 as research scientist and project manager in the Middleware and Mobile Applications Research Group. He is currently active in several research projects including participation in standardization fora such as JAIN, 3GPP and Parlay.*

*jbakker@telcordia.com*



*David Tweedie (age?) has been working as software designer for Nortel Networks in Ottawa, Canada since 1998. He joined Nortel after receiving a Bachelor of Computer Science with a major in Information Systems from the University of New Brunswick in Fredericton, Canada. While at Nortel, David has primarily been working on service enabling technologies. He started out as a member of an Advanced Intelligent Networks development team. After that, he progressed to next generation third-party programmability solutions which eventually led to investigations into the Parlay/OSA APIs.*

*dauidtw@nortelnetworks.com*

The application and deployment concept of programmable network capabilities have been well understood and embraced in the industry. To date, Parlay and JAIN have focused on unlocking existing (e.g. IN-like) network capabilities to a Java and web-enabled developer community. More recently, additional efforts to further appeal to both application developers and service providers have emerged. The advanced abstraction and simplification in the programming interfaces (Parlay X), the interest in alternative transport and middleware technologies (Parlay Web Services), and the emerging field of telecom-oriented, XML-based scripting languages are clear indications of this development. Furthermore, we see an attempt to unify the Parlay concepts with another service mediation technology, SIP, to combine programmatic objects and data types with session control capabilities, as witnessed in the IETF SPIRITS initiative. In this paper, the authors will give a compendious overview of these new developments in the area of programmability, and the applicability thereof within today's and the next generation of networks.

## 1 Introduction

In present day communications networks, connectivity and high quality voice have become commodities. The supplementary services and A/IN based applications are increasingly becoming commonplace. We have come to realize that it is the value added services and applications that are the moneymakers, providing the greatest revenue-generating potential. In the struggle to secure a piece of application market share, all the players have embarked on a quest for the killer application. **If anything has taught us however, it is that** one simply cannot predict what the killer application will be. Time to market is yet another key factor in the multifaceted equation. Killer applications are not tangible, sometimes subject to a disquieting amount of hype or fashion. Furthermore, the days of technology push have gone. Large service providers and telecommunication equipment vendors can no longer expect to introduce and rollout expensive dedicated network equipment and service platforms, supporting applications that are then forced upon the consumer community. Users of communications networks and services are educated and mature; there is no room for complacency. Hence, focus has shifted to finding and defining the killer environment, or the killer enabler, rather than the killer application. The killer environment will allow the network operator or service provider to quickly, easily, and without much disruption to ongoing network operations, introduce the killer application, or killer applications, once it is found.

It has generally been noticed in the industry that the killer enabler is provided by open access to, and programmability of, network service capabilities. There are two aspects to this killer environment. First, there is service mediation, i.e. providing application developers with a unified approach and consolidated architecture for open

but secure, easy but regulated, flexible but scalable, access to core network service capabilities. Second, to increase the likelihood that a killer application will emerge, one needs to provide a fertile greenhouse for as large a developer community as possible. This greenhouse should not be limited to the traditional R&D labs; the larger developer community, including the enterprise application developers and web developers, need to be engaged and inspired to partake in the process of devising the successful value added services and applications of tomorrow. This is envisaged to be achieved by abstracting from the sheer complexity of the application development process, and the technologies and protocols involved. The killer environment needs to appeal to the large crowd of application developers, by defining the open interfaces using technologies and methodologies that are close and familiar to them.

Figure 1 shows a layered architecture designed for open access to service capabilities [3]. It consists of a resources layer, a services layer and application servers. The SCFs (Service Capability Features) are implemented in the services layer and provide their capabilities to the application servers. In general, application servers and the programmable gateway are physically separated entities. Therefore, a distribution technology between the SCFs in the gateway and the application server is needed. Figure 1 also indicates an interface between the resources layer and the services layer.

Several papers have addressed the issue of open access, service programmability using third generation programming languages and its service mediation aspect (see e.g. [16] and [23]). The focal point of this paper however, is the second aspect of a successful killer environment, i.e. the appeal to the application developer crowd. This



Musa R. Unmehopa (age?) received his M.Sc. in computer science in 1996 from the University of Twente, The Netherlands, after which he joined Lucent Technologies, Bell Laboratories. He is currently a senior standards consultant engineer in the Wireless Advanced Technologies Lab within Lucent Technologies, The Netherlands. He is actively involved in the standardization of open network Application Programming Interfaces within 3GPP, 3GPP2, ETSI, the Parlay consortium, and more recently, the Open Mobile Alliance (OMA). Currently, Mr. Unmehopa holds the position of vice-chairman of 3GPP Technical Specification Group (TSG) Core Network Group 5 (Open Service Access).

unmehopa@lucent.com

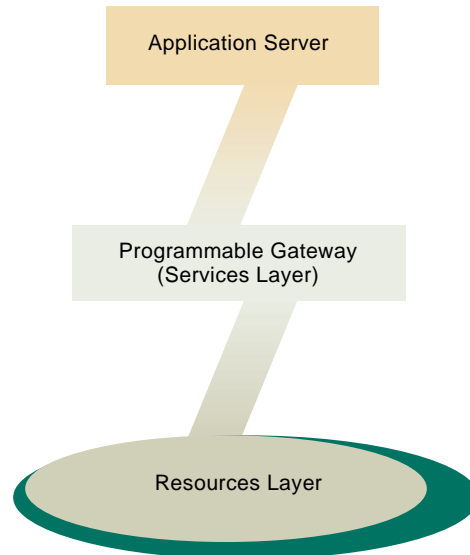


Figure 1 General Programmable Gateway Architecture

is achieved by simple access, using the technologies they like, such as scripting technologies, at a higher level of abstraction. The most promising technology to fulfill these requirements, currently under development in the industry, is that of Web Services. A Web Service is a capability that can be invoked via standard Internet protocols. As such, the prospect of an SCF (Service Capability Features), accessible via SOAP/XML ([25] and [24]), would perfectly fit the Web Services paradigm.

This paper will specifically address the aspect of engaging and involving the larger developer community. It is important to note that the work on service creation in next generation networks is in full progress. The approach that the authors adhere to, for the purpose of this paper, is to classify application programmability and next generation service creation into three main categories. These categories are generic Application Programming Interfaces (APIs), using many distribution technology realizations, comprehensive access to network capabilities using scripting, and simple and network specific interfaces deployed as Web Services (Parlay X). It is not the intention to provide an exhaustive survey; rather the authors will provide a concise synopsis by presenting one or more compelling and promising representative technologies out of each category. In addition, the paper will address the relation of Parlay with another service mediation technology proving to be of immense interest to the industry, i.e. the Session Initiation Protocol (SIP) [15]. The paper will conclude with a correlation of all these technologies and initiatives in a comprehensive overview, identifying how they link up into the global service architecture.

## 2 Programmability Through Programming Languages APIs

One of the first methodologies or design paradigms adopted by the telecommunications world from the information technology domain is the use of APIs. An API is a programmatic interface providing access to or programmability of software resources, such as database applications or telecommunication protocol stacks. An API provides application developers with programmability of software resources, by defining these resources in terms of objects and methods, data types and parameters that operate on those objects. Examples of resource layer APIs include the JAIN protocol APIs, such as JAIN SIP [17] or JAIN INAP [18]. At a higher abstraction layer we find the service layer or network capabilities APIs, for example the Parlay APIs [12] or JAIN JCC (Java Call Control) [20] and JCAT (JAIN Coordination and Transactions) [21]. The service layer APIs do not focus on individual resources, but rather provide application developers with access to service capabilities residing in a core telecommunications network. In the remaining we will focus on the Parlay APIs, their UML representation and the Parlay realizations.

Parlay APIs are independent of the actual resources; however, protocol mapping recommendations exist. Actual resources may reside either in the A/IN network, in Mobile networks, Managed IP networks, or in Next Generation Networks. Parlay Application is thus abstracted from resource implementation. As a consequence, the Parlay APIs expose only common aspects and generic functionality of communication networks.

The Parlay APIs are defined using UML (Unified Modeling Language) [12]. The UML modeling technology is realization technology independent. However, as the Parlay UML was reengineered from earlier Parlay OMG (Object Management Group) IDL (Interface Definition Language) [7] definitions, the Parlay UML is not fully technology independent. Distribution technologies other than OMG's CORBA may realize particular distribution aspects differently. This leads to mismatches when mapping the UML to other realization technologies. Also, the Parlay APIs specify supports for distribution aspects, such as authentication, on the application level even while they are essentially orthogonal to the problem domain of telecommunications. A full discussion of the Parlay UML and its deficiencies is not the scope of this paper.

Still, with some effort, a number of technology specific realizations can be generated from the Parlay UML model. Initially OMG IDL was the only generated and published realization.

Recently, a W3C WSDL [27] realization has been generated and published along with the rules that were established to enable automatic generation of future versions of the Parlay specification. In the immediate future a rulebook that specifies the mapping between the Parlay UML and the JAIN SPA (Service Provider APIs) will be made available. Figure 2 shows the relation between the Parlay UML, OMG's IDL, W3C WSDL, and JAIN SPA. The realization technologies will be further discussed in the remainder.

## 2.1 OMG IDL

The technology realization which was initially used with Parlay APIs was the OMG IDL. OMG IDL is used to specify the interfaces of remote objects. OMG IDL interface implementations (so-called remote objects) typically execute within a different process on a different host with respect to the implementation's client. As such, OMG IDL is well suited for defining the interface of the Parlay APIs. Additionally, OMG IDL is the definition language used to define a set of interfaces for use with the Common Object Request Broker Architecture (CORBA) [13]. CORBA is a standard, defined by the OMG, that supports distributed objects. CORBA provides a complete set of services to remote objects, including concurrency control, licensing, life-cycle management, security management, and persistence. Finally, another benefit of using OMG IDL as a specification language is that it is programming language independent; the OMG has provided standard mappings to many programming languages, including Java and C++.

## 2.2 Java

It is recognized that Java is a firmly established programming language in the enterprise application market, and emerging in other fields. For that reason, many middleware systems support language mappings to Java. The Java software development packages come with a vast number of libraries that implement telecom-related (i.e. JAIN) and other standards. The JAIN SPA working group felt that the Java communities experience in implementing as well as certifying conformance to standards could be beneficial to the Parlay community. Note, however, that the Java community does not focus on the distribution technique, but more on ease of use of the APIs and whether they follow the patterns and paradigms expected of good citizens of the population of Java API. It was felt that the OMG IDL mapping to Java inadequately conformed to these goals. Additionally, some CORBA ORB vendors inadequately implemented the (latest) OMG standards that map Java to IDL, thereby inhibiting the portability of Java applications.

To circumvent these hurdles and increase acceptance of the Parlay APIs within the large com-

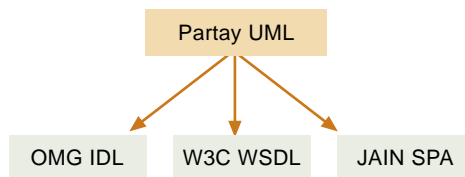
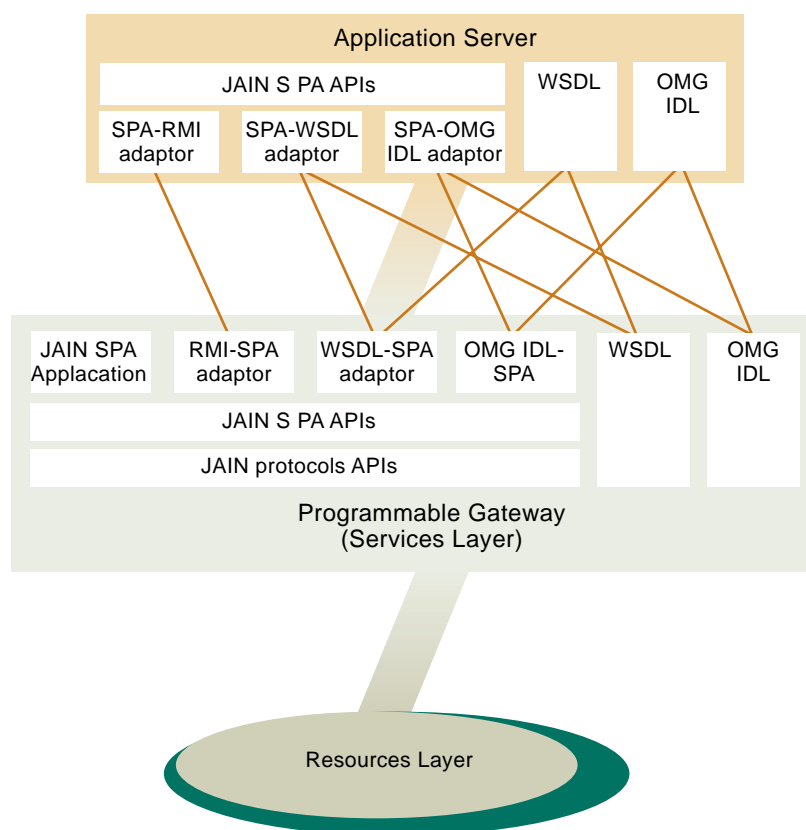


Figure 2 Parlay Technology Realizations

munity of Java developers the Parlay group is producing a rulebook which specifies the mappings between the Parlay UML model and Java. The Java realization activities underway within the Parlay group are part of SUN's JAIN SPA initiative.

As indicated in Figure 3, the JAIN SPA APIs are slightly different from the other technology realizations in that they specify a local API as opposed to a distribution technology API (such as IDL or WSDL). These JAIN SPA APIs can reside on either the Parlay application server or on the Parlay gateway. The JAIN SPA APIs are independent of the distribution technology used. It is up to the actual implementation of the JAIN SPA APIs to determine which type of distribution mechanism it wishes to use (i.e. IIOP, SOAP, RMI, etc). Figure 3 illustrates how the JAIN SPA APIs can be used to provide a layered approach to Parlay application and gateway servers.

Figure 3 Realization architecture



### 2.3 WSDL

The XML-based Web Services paradigm transforms the Web into an anthology of independent application components, each with a well-defined and published interface, which allow other Web applications to find them and use them. Web Services are built on top of existing, inexpensive and easy-to-encrypt Web protocols such as HTTP and based on open XML standards for data encoding. Web Services merely draw upon the omnipresent Internet infrastructure to discover and compile services into compelling value added applications.

The Parlay APIs are realized in WSDL (Web Services Description Language) [27]; WSDL is an XML format definition language which is used to describe the programmable interface for a service. This usage is similar to the OMG IDL use discussed earlier. WSDL is specified by the W3C organization and is defined with XML and XML Schemas [28]. The first transport supported by WSDL is SOAP [25] over HTTP. Note that other transport bindings may be supported while retaining the WSDL interface definitions. A WSDL document is defined by a series of XML Schema elements.

Within Parlay, a set of mapping rules from the Parlay UML model to WSDL has been written. These mapping rules provide a mapping from UML constructs into WSDL elements. The detailed mapping from UML to WSDL can be found in the Parlay Overview specification [14]. The strength of this approach lies in the strong association with web protocols and associated service creation and deployment technologies. Hence, the WSDL mapping is done to further appeal to the developer community.

Figure 3 shows the possible interplay of all Parlay realization technologies. The figure illustrates how the various Parlay realization technologies could be utilized within a Parlay solution. As illustrated, CORBA, SOAP, or even RMI plays a similar role as a distribution mechanism within a Parlay solution. The JAIN SPA APIs can provide a further abstraction from the distribution layer both on the Parlay Application Server and on the Parlay Gateway. This section has introduced the objective of the Parlay APIs to expose only common aspects and generic functionality of communication networks, in order to allow for application portability across network technologies. To be able to deploy these applications using this common and generic API in the various network technologies, Parlay defines several distribution technology realizations.

### 3 Simplified Programmability Through Web Services

The Parlay APIs expose capabilities of the telecommunications network in a network technology and programming language neutral way. In fact, the contributors to the APIs go to great lengths to ensure that the common aspects of (converged) mobile, fixed, and managed packet networks are made programmable for a large audience of developers of carrier grade systems. Yet, some more specific but highly valuable capabilities remain unsupported. As an example, the support for SMS (Short Messaging Service) is inadequate. Additionally, the developer who wishes to program simple applications, such as setting up a third party call (a.k.a. click to dial), using the Parlay APIs, needs to go through elaborate interactions with different components and interfaces.

These observations motivated the need for APIs that are predominantly simple and, consequently, restricted in their capabilities; developers that need access to advanced means of control would not be users of these simple APIs, but rather use the existing Parlay APIs. Additionally, the rigid dogma that favors exposure of common network capabilities over specific capabilities needed to be relaxed. Finally, the resulting APIs would predominantly be used in a multi portal or a web environment. The Parlay community proved to be open to these views and approved the establishment of a group, the Parlay X group, in late 2001, which was chartered to create APIs that incorporated the above views. It was felt that new markets are made available through Web Services with Parlay X application definitions. Given a set of high level interfaces that are oriented towards the skill levels and telecom knowledge levels of web developers, the Parlay X APIs open the accessibility of the network capabilities to a much wider audience.

Figure 4 shows where the Parlay X applications are situated with respect to the Parlay X gateway. Note that many of the Parlay X capabilities will be mediated by the Parlay Gateway. Some, however, are not supported because of Parlay's focus on common capabilities as opposed to network specific capabilities. Such capabilities cannot be mediated through the Parlay Gateway, rather they need to be made available through the resources layer.

Consider a web server that allows end users to charge for services they consume to their prepaid accounts, or a customer support page that creates, by the press of a button, a voice call between an end user and a customer service representative. Developers in this environment often use web services technology to communicate with different capability servers, i.e. they

use SOAP and WSDL. Web services provide a set of capabilities and technologies that result in a very compelling foundation for supporting converged telecom/IT applications.

As with the Parlay APIs discussed in the previous section, authorization, discovery, extensibility and activation are very important features that need to be adequately addressed. The Parlay APIs introduce the set of Framework APIs to address these issues on an application level. The web services environment is different *since several mature or existing or technologies exist today or are the industry norm that* can be leveraged to achieve the same. For example, UDDI (Universal Description, Discovery and Integration) or WSIL (Web Services Inspection Language, a.k.a. WS-Inspection) can be used to discover or retrieve references to specific services. WSIL files describe web services, possibly in a hierarchical manner, while UDDI serves a centralized registration and service publication solution. UDDI or WSIL allow for Parlay X applications to discover published services. The UDDI or WSIL-driven registry, finally, contains information using which the Parlay X application can bind and activate the Parlay X service. Note that authorized personnel can extend the registry with more Parlay X services.

Authentication required prior to accessing the private registry can be achieved through acquiring access using a VPN or through applying HTTPS. Many VPN solutions exist, supporting a variety of authentication methods. Based on the authentication information (e.g. user handle and password in the case of HTTPS) access to only the subscribed services can be enforced through dynamically generated WSIL files; effectively authenticating access to subscribed services only. Finally, various accounting schemes can be employed based on the invoked service, time of authentication, service agreement established a priori, etc.

Today, Parlay X participating companies have submitted contributions that target Parlay X compatible definitions of web services for UI (User Interaction), 3PCC (Third Party Call Control), Payment, TopUp, UserStatus/Presence/Location, and Messaging (i.e. SMS). 3PCC and UI web services are motivated by the observation that applications that interface with telecommunications resources often initiate and receive voice calls. 3PCC supports initiation of voice calls (e.g. click to dial) and recognizes user input (i.e. voice or DTMF). The TopUp API allows consumers to increase the value of their prepaid accounts and the payment API allows content providers to charge for certain types of content such that the billing is handled by the operator. Examples of content that can be charged for are

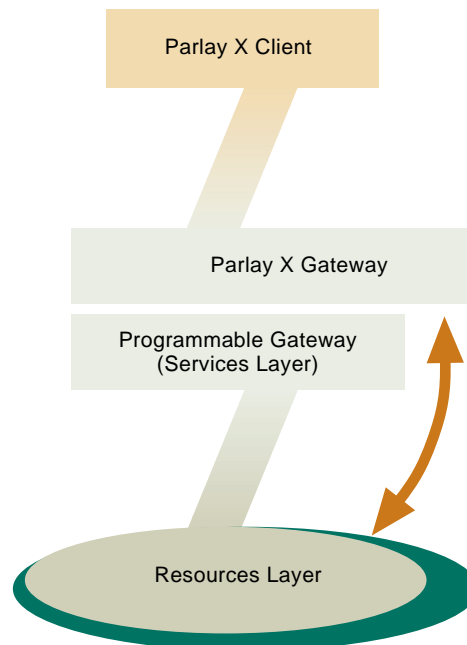


Figure 4 Parlay X Gateway accessing specific network capabilities directly

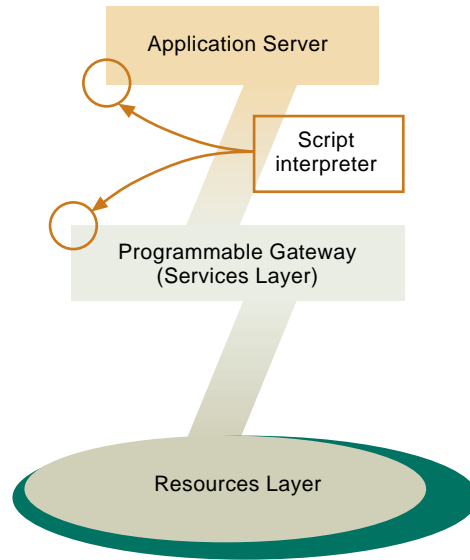
downloadable ring tones or downloadable voice mail announcements. Next, the UserStatus/Presence/Location contribution focuses on presenting User Status or Presence (such as online, offline, or engaged) and Location (fine grained as longitude and latitude *or as course as* within or not within an area). Finally, the Messaging API is intended for sending messages (most notably, SMSes) from web pages to devices that can accept such messages.

It is hard to measure simplicity. Parlay X focuses on exposing capabilities through accepted and largely applied technologies by the IT industry. Furthermore, the IT industry is currently furthering the Web Services architecture; emerging standards for transactions and integrated security will make the use of Web Services as a middleware solution even more attractive and will further reduce the complexity of creating telecommunications applications. Already, Parlay X is exploring the applicability of Web Services middleware in constructing a programmable gateway. As outlined above, a number of contributions have been suggested and are currently being consolidated and processed for inclusion in the first release of the Parlay X APIs.

## 4 Programmability Through Scripting

Scripting languages are lightweight, highly customizable, and typically interpreted languages, appropriate in the area of rapid application development, acting as glue to provide connections among existing components. These characteristics allow them to be used to code or modify

Figure 5 Script interpreters either integrated with the SCF or with the application server



applications at runtime, and interact with running programs. Such qualities and features make scripting languages suitable application enabler abstractions and highly applicable to the field of application programmability next to APIs.

XML is commonly seen as the preferred vehicle to create service creation languages. Aside from its standardization and readability by both machines and humans, XML offers several additional benefits. XML supports restrictions to its expressiveness. Such a restriction 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 easily execute the script. Finally, many tools and libraries are available to create, interpret and validate XML documents.

The next generation of scripting languages for creating value-added services in converged networks will be based upon XML. Industry fora like Parlay and JAIN have developed open standard APIs to enable service creation in today's and tomorrow's networks. While services can be developed in traditional (third generation) programming languages (e.g. Java or C++) using these APIs, the XML-based scripting languages offer some attractive advantages. While not as flexible or powerful as a programming language, scripting languages are typically easier to learn, and are platform independent.

Several XML-based call control markup languages have been previously proposed, including CPML, TML, XTML, CallXML [11], CPL [9] and [10], the Call Control XML (CCXML) [20], and Service Creation Markup Languages

(SCML) [17]. A comprehensive survey of these call control markup languages proposals is not the purpose or within the scope of this section. Instead, we observe the SCML developed by the Service Creation Environment Expert Group of the JAIN industry forum, and developed according to open processes by a number of actively participating companies. The rigorous process and the openness, along with SCML's close relation to Parlay and JAIN will contribute to an increased industry acceptance and therefore warrant a closer look.

Languages such as SCML, CCXML and CPL are used to create applications that make use of the functions provided by the services layer (as opposed to the resources layer), see Figure 5. The figure shows a scripting interpreter either at the application side of the programmability architecture or at the gateway side. This means that both the third party and the operator can support programmability through scripting. Scripts can be stored such that they can be retrieved from storage indicated by a URL or in subscriber databases like HLRs.

The remainder of this section will further discuss SCML and its features. Note that a comparison between CPL, CCXML and SCML was presented in [1]. Note also that CPL, CCXML and SCML are all in the *work in progress* state and could change in the course of further standardization. However, we expect that the overall concepts will continue to apply. Finally, the examples given in this section are for illustrative purposes only.

#### 4.1 SCML

The Service Creation Markup Language (SCML) is a scripting language that connects existing components such as JAIN SPA or Parlay APIs, enabling rapid prototyping, rapid application development, or easy end-user customization. We briefly discuss SCML here and compare it to CCXML.

SCML scripts are created, edited, and validated using regular editors or as a result of applying transformation techniques. Next, the scripts are deployed in a retrievable location (e.g. identified by a URL). SCML scripts follow a pattern of registering the static events and criteria that can be matched by events (e.g. those emitted by SCFs), followed by declaration of business logic to be executed in response to such an event. Therefore, activation initially encompasses provisioning the events and criteria and, subsequently, executing the business logic upon occurrence of the provisioned event. A script is deactivated through removing the provisioned criteria. The SCML specification only specifies the scripting language; no APIs are specified for

```

</scml>
  <register>
    <disconnected causeCode ="CAUSE_BUSY" destination ="sip:jdoe@home.com"
      connection="terminating" block="true">
      <routeCall targetAddress ="tel:2125552121"
        redirectingAddress ="sip:jdoe@home.com"/>
    </disconnected>
  </register>
</scml>

```

Figure 6 Example SCML Script: Call Forwarding on Busy

activation, deactivation, and other service life-cycle events.

SCML is intentionally extendible. It consists of a common core, defined in the scml-core package that should be extended per JAIN SPA or Parlay component. Currently, the core is extended with the jccml package (JAIN Java Call Control Markup (JCC) Language); the jccml package assumes a JCC 1.1 API [20] implementation. The jccml package is defined using an XML Schema that is derived from JAIN's JCC API. 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. Finally, JCC can be mapped on top of SIP [8], MGCP, and H.323 [8]. No public documents showing an informative mapping from JCC to ISUP, INAP and CAP may be available, but the JCC call model was designed to be protocol agnostic.

An example script in SCML is shown in Figure 6 for Call Forwarding on Busy. In this script the activation criteria (indicated by the <disconnected>-element) are registered with the gateway. The criteria specified by the registration element are the condition that call setup fails due to a busy callee, that callee's address, the fact that it concerns the terminating portion of the call, and an indication that the call processing must be suspended while the script executes. If these criteria apply, the scripts will be executed and the call will be redirected through specifying an alternative target address in the <routeCall> element. In this case, after forwarding the call,

processing, which was suspended, will automatically be resumed.

Note that SCML is not limited to support for the JCC 1.1 API only. Further extensions (e.g. adding Short Message Service (SMS) components) are intended. In fact, Figure 7 shows a script that exemplifies support for and usage of the <sendSMS> element. The <sendSMS> element causes an SMS to be sent to the RFC2806 compliant URI tel:2125556767 only if destination tel:2125551212 is busy. The differences with Figure 6 are subtle; the example in Figure 7 validates the document against a Schema that extends the Schema used in Figure 6 such that it accepts the element <sendSMS>. Additionally, processing of the call processing machinery is not blocked (as the attribute block of the <disconnected> is set to false); since handling of busy connection and sending the SMS are parallel activities.

Concluding this section we state that there are many XML-based scripting languages. We have further discussed SCML and found that SCML is promising as an application enabled abstraction as it makes use of the capabilities provided by the various SCFs. SCML is from the ground up designed to be protocol agnostic and extendible to other sources of events.

## 5 Service Mediation Through SIP

JAIN and Parlay have introduced the concept of third party access to service capabilities residing in the core network. Apart from Parlay, as

```

<scml>
  <register>
    <disconnected causeCode="CAUSE_BUSY" destination="tel:2125551212"
      connection="terminating" block="false">
      <getOriginatingAddress address="orig"/>
      <sendSMS to="tel:2125556767"
        content="'Call attempt by '+%orig;+' on 2125551212'"/>
    </disconnected>
  </register>
</scml>

```

Figure 7 Example SCML Scripts: SMS on Busy Subscriber

described in detail above, there is another popular service mediation technology that is receiving a lot of interest in the industry, i.e. SIP. This section discusses two recent standardization activities that incorporate the synergy of Parlay and SIP. The two activities are SPIRITS/PINT, taking place in the IETF, and OSA-to-ISC<sup>1)</sup>, taking place in 3GPP. In a nutshell, SPIRITS/PINT take conventional A/IN/CAMEL triggers/events from PSTN/ISDN networks and make those available in the Internet domain. OSA-to-ISC allows application developers to access the emerging 3G IP Multimedia Subsystem. Hence both activities target different underlying core network technologies, i.e. PSTN/ISDN, and 3G IP Multimedia. In the remainder we will elaborate on these two examples.

### 5.1 SPIRITS and PINT

Within the Internet Engineering Task Force (IETF) efforts have been ongoing for a while to define interworking between the traditional telephony networks and the Internet. PINT (PSTN/Internet Interworking Protocol) [5] addresses the requirement to invoke telephony services from the Internet, whereas SPIRITS [6] focuses on carrying A/IN triggers and events from the telephony network to the Internet. Combined, PINT and SPIRITS allow for A/IN-type service logic to be executed on IP hosts and to be delivered to traditional telephony subscribers. The remainder of this section will focus on SPIRITS.

The approach in SPIRITS has been to select those A/IN parameters of specific interest to application developers. The selection of relevant

A/IN parameters has been based on the Parlay specifications. Various considerations on automatically generating these XML parameters and the use of XML Schema can be found on [6]. Summarized, a process collocated on the A/IN SCP (called the SPIRITS Client) monitors for A/IN triggers and events of interest from the telephony network. Once received, the SPIRITS Client extracts the relevant A/IN parameters, according to the Parlay definitions, and parses them into XML. The XML-encoded parameters and data types are then placed in the payload of a SIP message. This message is transported to the SPIRITS Gateway in the Internet domain. The SPIRITS Gateway either passes this message on to the SPIRITS Server, located on an IP host, which terminates the telephony request and is responsible for executing the A/IN-type service logic. Or, as depicted in Figure 8, the SPIRITS Gateway parses the XML-encoded parameters and hands them to the appropriate SCF. The SCF would then use one of the Parlay realization technologies to contact the application server. For a more detailed description of the SPIRITS architecture the reader is also referred to [6]. This architecture and protocol definition allows the SPIRITS Server to execute a Parlay application, based on the XML encoded Parlay representation of the relevant A/IN parameters.

The SPIRITS protocol, between the SPIRITS Client and the SPIRITS Gateway, uses the SIP SUBSCRIBE and NOTIFY messages, as these are specifically suited for trigger and event reporting type functionality. The work on the SPIRITS protocol is currently in progress. Figure 9 depicts an example where the A/IN “Answer” event is represented in terms of Parlay parameters, and carried as an XML encoded body inside a SIP NOTIFY message. The SIP header portion of this message is simplified for the sake of brevity; again, the reader is referred to [6] for more detail.

### 5.2 OSA to ISC mapping

SIP has been adopted by 3GPP [1] as the control protocol for the wireless next generation core network. This core network is referred to as the IP Multimedia Core Network Subsystem (IM CN). In a nutshell, session setup, control, and teardown functionality are performed by SIP servers, referred to as CSCF (Call Session Control Functions). The CSCFs do not perform any service logic execution. Instead, these tasks are performed by application servers. One such application server, acting as a SIP application server, is the Parlay Gateway. The protocol

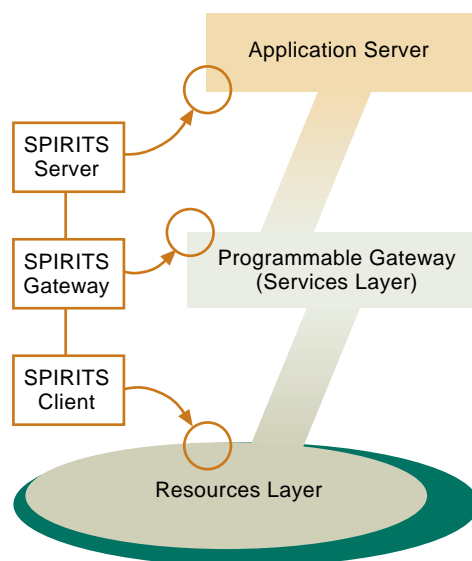


Figure 8 SPIRITS architecture in conjunction with programmability architecture

<sup>1)</sup> In this paper, OSA (Open Services Access) and Parlay are used interchangeably. Technically this is not correct, as the set of Parlay APIs is slightly larger than the set of OSA APIs. However, for the purpose of this paper this distinction is not essential.

defined between the CSCF and the application servers is the IM CN Service Control protocol (ISC). With the CSCF and application servers being SIP servers, the ISC protocol is in fact the SIP protocol. The IM CN and its functional entities, as well as the ISC protocol, are being specified by 3GPP [4].

As part of their specification set, 3GPP publishes API to protocol mapping recommendations for various underlying network protocols. For OSA to be deployable in IM CN networks, the 3GPP specification in [1] recommends mappings of OSA to the ISC protocol. The OSA-to-ISC mappings cover the entire scope of the SIP protocol (i.e. beyond just SUBSCRIBE/NOTIFY) and its supported headers in order to provide support for the full breadth and depth of the Parlay Call Control APIs.

Figure 10 and Figure 11 show an example of how an OSA API method maps to an ISC interface operation. The first part of the figure shows the IDL definition of the createAndRouteCallLeg method, which is used by an application to request the creation and routing of a new call leg. The second part of the figure shows the SIP INVITE message onto which the createAndRouteCallLeg method is mapped. In bold font it is indicated how the individual OSA method parameters and data types map onto the SIP headers and their contents. The appInfo OSA parameter is defined as a union data type (TpCallAppInfoSet) of which the alerting mechanism (CallAppAlertingMechanism) is one of the possible fields. In this example, the alerting mechanism is mapped onto the Alert-Info header. For the benefit of simplicity, in this example the application does not request for the arming of triggers with the routing of this call leg.

The bold font indicates how the OSA parameters are mapped onto their counterparts in the ISC operation.

When comparing SPIRITS with OSA to ISC, the different approaches are quite apparent. In the SPIRITS approach the Parlay data is carried as an XML body inside a SIP message, whereas in the OSA-to-ISC approach the Parlay data is mapped onto the SIP equivalent data (headers).

Concluding, SIP is an Internet protocol that operates well with Internet related technologies like HTTP and MIME. So the combination of Parlay with SIP allows third party telecom application development at Internet speed, tapping into Internet creativity.

```
NOTIFY sip:jones@iphost.home.com SIP/2.0
From: <sip:directory_nr@ptt.com>;tag=SPIRITS-ER_RES-directory_nr
To: <sip:jones@home.com>
Via: SIP/2.0/UDP gateway.ptt.com
...
<spirits-event>
  <DP Parlay=ER_RES/>
  <EVENT_REPORT_RESULT ver=1.0>
    <CALL_EVENT_TYPE>
      <P_CALL_EVENT_ANSWER />
    </CALL_EVENT_TYPE>
    <CALL_MONITOR_MODE>
      <P_CALL_MONITOR_MODE_NOTIFY />
    </CALL_MONITOR_MODE>
    <CALL_EVENT_TIME>
      1998-12-04 10:30
    </CALL_EVENT_TIME>
  </spirits-event>
```

Figure 9 Example SPIRITS NOTIFY message for the Answer event

```
TpCallLegIdentifier createAndRouteCallLegReq (
  in TpSessionID callSessionID,
  in TpCallEventRequestSet eventsRequested,
  in TpAddress targetAddress,
  in TpAddress originatingAddress,
  in TpCallAppInfoSet appInfo,
  in IpAppCallLeg appLegInterface
)
```

Figure 10 OSA API method

```
INVITE sip:targetAddress SIP/2.0
To: Bob <sip:targetAddress>
From: Alice <sip:originatingAddress>
Call-ID: callSessionID
CSeq: . . .
Alert-Info: <http://www.example.com/appInfo.CallAppAlertingMechanism.wav>
```

Figure 11 Corresponding ISC interface operation

## 6 Conclusion

In this paper the authors emphasize the thesis that the key success factor of a thriving killer environment consists of the potency to appeal to a large application developer community through the flexible programmability of service capabilities. The authors present an extensive survey of established and emerging service programmability technologies. Three main categories are put forward, i.e. programmability through the use of Application Programming Interfaces, programmability through Web Services, and programmability through scripting. Subsequently, the authors impart their view on the relationship of Parlay with another service

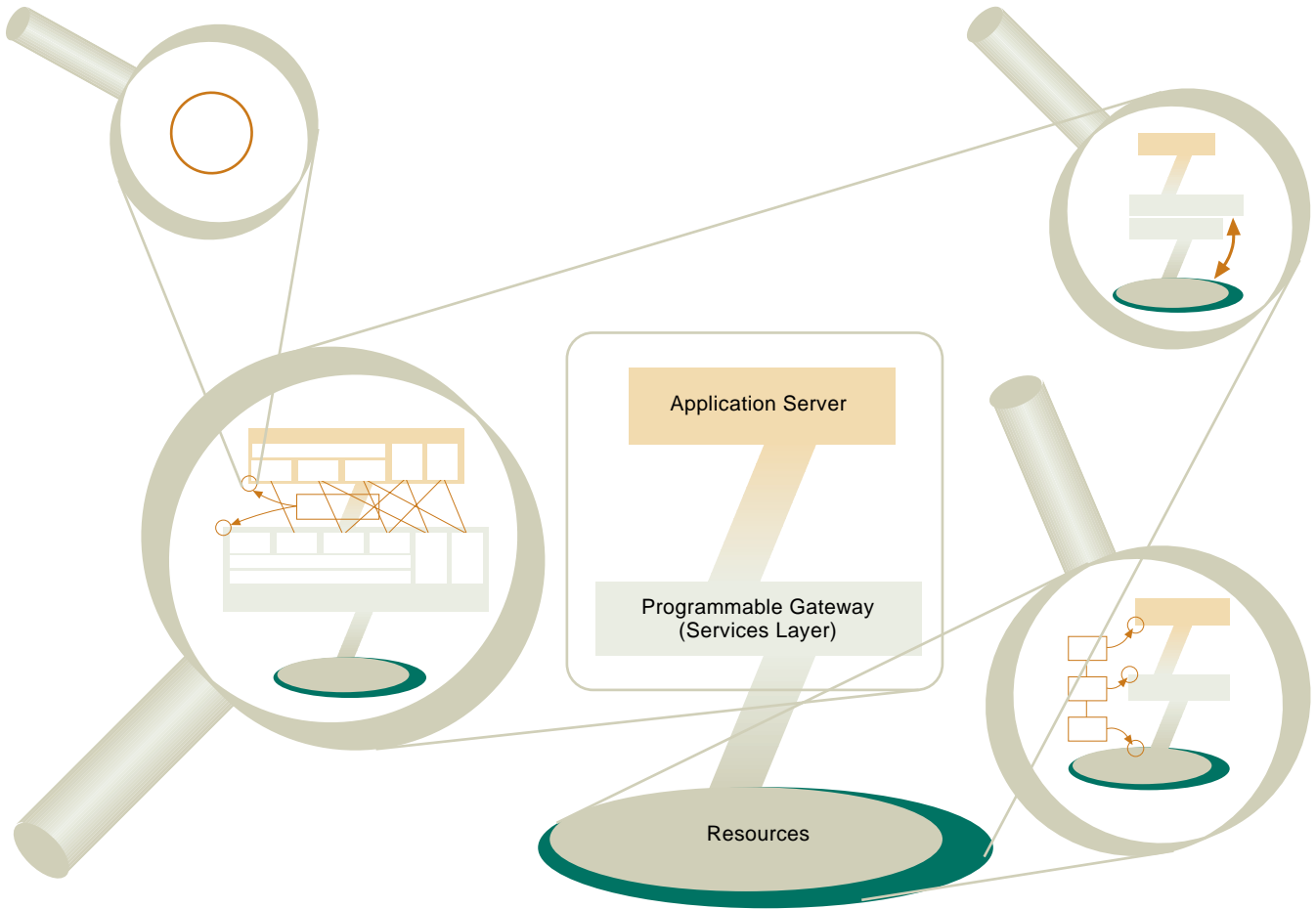


Figure 12 Overview of programmability efforts

mediation technology, SIP, by showing they are complementary, and synergy can be achieved. A recurring element throughout all these discussions is the use of XML. The authors then make an effort to compile and amass all the concepts and discussions and provide an all-encompassing helicopter view of all concepts and technologies and hence show the correlations and associations.

Figure 12 shows all initiatives aimed at the further evolution of service creation in next generation networks together. In the figure references to the original figures are made. The overall programmable architecture discussed in Figure 1 is at the center of Figure 12. The referenced Figure 3 discusses the Parlay APIs and their technology specific realization: WSDL, OMG IDL, and JAIN SPA. These realizations concern the services layer and application server. Figure 4 shows the Parlay X architecture in which functionality is mainly delegated to the Parlay APIs discussed in Figure 3. However, as Parlay X exposes specific network capabilities as opposed to the common capabilities exposed by the Parlay APIs, some Parlay X APIs can only be implemented through interaction with the resources layer. Therefore, the magnifying glass concerns the services layer, application server, and the resources layer. Figure 5 focuses on an alternative programming means: scripting lan-

guages. Scripting interpreters can be found in the services layer as well as in the application server. Finally, Figure 8 shows that programmability of network intelligence can also be achieved through designing protocols that interact with the resources layer.

## References

- 1 3rd Generation Partnership Project; Technical Specification Group Core Network; Open Service Access (OSA); Application Programming Interface (API) Mapping for Open Service Access; Part 4: Call Control Service Mapping; Subpart 4: Multiparty Call Control ISC (Release 5). 3G TR 29.998-04-4 v5.0.0. Publisher?, Place of publishing?, Date?
- 2 Bakker, J-L, Jain, R. *Next Generation Service Creation Using XML Scripting Languages*. ICC 2002, New York, NY (USA), April 28 – May 2, 2002.
- 3 Bakker, J-L et al. Rapid Development and Delivery of Converged Services Using APIs. *Bell Labs Technical Journal*, 5 (3), 12–29, 2000.
- 4 Grech, M, Torabi, M, Unmehopa, M R. *Service Control Architecture in the UMTS IP Multimedia Core Network Subsystem*. IEE

- 3G2002 Mobile Communications Technologies, London, UK, 8–10 May 2002.
- 5 IETF. *PSTN and Internet Internetworking (pint) Charter*. October 18, 2000 [online] – URL: <http://www.ietf.org/html.charters/pint-charter.html>
  - 6 IETF. *Service in the PSTN/IN Requesting InTernet Service (spirits) Charter*. October 18, 2000 [online] – URL: <http://www.ietf.org/html.charters/spirits-charter.html>
  - 7 ISO. *Interface Definition Language (IDL)*. March 1999. (ISO 14750)
  - 8 Jain, R, Bakker, J-L, Anjum, F. Java Call Control (JCC) and Session Initiation Protocol (SIP). Invited Paper. *IEICE Trans. Communications*, E84-B (12), 3096–3103, 2001.
  - 9 Lennox, J, Schulzrinne, H. *Call Processing Language Framework and Requirements*. May 2000. (URL: <http://www.ietf.org/rfc/rfc2824.txt>)
  - 10 Lennox, J, Schulzrinne, H. *CPL: A Language for User Control of Internet Telephony Services*. (Work in progress.) January 2002. (URL: <http://www.ietf.org/internet-drafts/draft-ietf-iptel-cpl-06.txt>) **Does this link work?**
  - 11 OASIS (Organization for the Advancement of Structured Information Standards). *The XML cover pages*. October 18, 2002 [online] – URL: <http://www.oasis-open.org/cover>
  - 12 OMG. *Unified Modeling Language (UML)*. Version 1.4, September 2001. October 18, 2002 [online] – URL: <http://www.omg.org/technology/documents/formal/uml.htm>
  - 13 OMG. *Common Object Request Broker Architecture (CORBA)*. Version 2.6, Dec 1, 2001. October 18, 2002 [online] – URL: [http://www.omg.org/technology/documents/formal/corba\\_iiop.htm](http://www.omg.org/technology/documents/formal/corba_iiop.htm)
  - 14 Parlay Group. *Parlay APIs Overview*. October 18, 2002 [online] – URL: [http://www.parlay.org/docs/Spec3\\_es\\_20191501v010101m.pdf](http://www.parlay.org/docs/Spec3_es_20191501v010101m.pdf)
  - 15 Rosenberg, J et al. *SIP: Session Initiation Protocol. June 2002*. October 18, 2002 [online] – URL: <http://www.ietf.org/rfc/rfc3261.txt>
  - 16 Stretch, R M. The OSA API and other related issues. *BT Technical Journal*, 19 (1), 80–87, 2001.
  - 17 Sun Microsystems. *JAIN SIP Specification 1.0, Java Specification Request (JSR) 32*. October 18, 2002 [online] – URL: <http://jcp.org/aboutJava/communityprocess/final/jsr032/>
  - 18 Sun Microsystems. *JAIN INAP API Specification 1.0, Java Specification Request (JSR) 35*. October 18, 2002 [online] – URL: <http://jcp.org/aboutJava/communityprocess/final/jsr035/>
  - 19 Sun Microsystems. *JAIN Service Creation Environment (SCE) API Java Specification Request (JSR) 100*. October 18, 2002 [online] – URL: <http://jcp.org/jsr/detail/100.jsp>
  - 20 Sun Microsystems. *JAIN Java Call Control (JCC) API Java Specification Request (JSR) 21*. October 18, 2002 [online] – URL: <http://www.jcp.org/jsr/detail/21.jsp>
  - 21 Sun Microsystems. *JAIN Coordination and Transactions (JCAT) API Java Specification Request (JSR) 122*. October 18, 2002 [online] – URL: <http://jcp.org/jsr/detail/122.jsp>
  - 22 Sun Microsystems. *The JAIN APIs*. October 18, 2002 [online] – URL: <http://java.sun.com/products/jain/>
  - 23 Unmehopa, M R et al. The Support of Mobile Internet Applications in UMTS Networks Through the Open Service Access. *Bell Labs Tech. Journal*, 6 (2), 47–64, 2002.
  - 24 W3C. *Extensible Markup Language (XML)*. October 18, 2002 [online] – URL: <http://www.w3.org/XML/>
  - 25 W3C. *Simple Object Access Protocol SOAP 1.1. W3C NOTE, 8 May 2002*. October 18, 2002 [online] – URL: <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>
  - 26 W3C. *Voice Browser Call Control: CCXML Version 1.0. W3C Working Draft, 21 February 2002*. October 18, 2002 [online] – URL: <http://www.w3.org/TR/ccxml/>
  - 27 W3C. *Web Services Description Language (WSDL) 1.1. W3C Note 15 March 2001*. October 18, 2002 [online] – URL: <http://www.w3.org/TR/2001/NOTE-wsdl-20010315>
  - 28 W3C. *XML Schema*. October 18, 2002 [online] – URL: <http://www.w3.org/XML/Schema>