Author(s): Licciardi, Carlo Alberto; Falcarin, Paolo
Article title: Analysis of NGN service creation technologies
Year of publication: 2003
Citation: Licciardi, C.A., Falcarin, P. (2003) 'Analysis of NGN service creation Technologies', in IEC Annual Review of Communications, vol. 56, 2003, pp. 537-551.

Analysis of NGN service creation technologies

Paolo Falcarin
Paolo.Falcarin@polito.it
Politecnico di Torino
Dipartimento di Automatica e Informatica
Corso Duca degli Abruzzi, 24
I-10129 Torino, Italy

Carlo Alberto Licciardi
Carlo.Licciardi@TILAB.COM
Telecom Italia Lab
Network Intelligence Dept.
Via R. Romoli, 274
I-10148 Torino, Italy

Abstract

Network Operators can see next Generation Networks (NGN) as new revenue stream, thanks to the potential they could have in increasing the service offering. Therefore it's important to understand how proposed technologies and solutions in NGN market can enable, flexible and easy service creation 3. This paper presents the result of the investigation of Eurescom P1109 project [1] in the area of advanced technologies that enable the introduction of new services in NGNs [5]. These technologies are evaluated with respect to some key evaluation criteria and then a comparison is provided.

1. Introduction

NGNs have been promoted to network operators as a way to decrease operational costs of existing infrastructure. Actually there is no clear business analysis that has proven this thesis. On the other hand NGN can be seen by network operators and service providers as a new revenue stream from their potential to increase service offerings. Therefore it is of paramount importance to understand how proposed solutions in NGN market can enable flexible and easy service creation both to service providers and 3rd party application developers.
EURESCOM P1109 Project "Next Generation Networks: the Service offering standpoint" [1] has addressed this issue by evaluating NGNs service platforms in terms of functionality, programmability, flexibility, openness, and inter-operability. In other words the objective has been to put to the test some of the major benefits promised by NGN, namely productivity, creativity and new revenues from new business opportunities, and to see how well current product offerings supported these capabilities, in terms of available tools for NGN service development; evaluating how much easy and efficient is to develop and deploy NGN services [5]; evaluate product maturity, standard compliance and interoperability. Among these issues this paper focuses on the application provider point of view, trying to extract useful guidelines for network operators that want to migrate to NGN in a profitable way [3]; moreover a description of different service creation approaches is given, in order to show which options are available to developers.
The paper is structured in different sections: first section introduces motivations and background needed to understand important issues for creation of NGN services; second section defines key evaluation criteria (openness to 3rd party developer, easiness to use, simplicity, supported network capabilities…) for service creation technologies; they will be detailed in third section, dedicated to the assessment of API, scripting languages and Service Creation Environments (SCE) that are key enablers of open service development; finally results of the evaluation are collected and compared in the last section.

2. Challenges and threats for Operators with NGN services

Operators are thinking about how to make NGN profitable in order to enable a new class of services like: any-to-any ubiquitous communication services, IP multimedia services, and audio/video conferencing.
All these services can be developed if NGNs address the main feature of service programmability. This means the ability of implementing new services following the customers' needs, and to differentiate the offer faster than competitors. Another important key point is the ability to offer to customers the same service everywhere, providing a seamless access from different terminals (mobile phones, soft-phones, UMTS phones…).
These goals can be obtained opening up levels of programmability to third parties or also to the users that want to personalize their own services. This means reducing costs and offer high quality services.

During the P1109 project, product selection and evaluation has shown that SIP [4] is the preferred technology to address NGN communications. The most of service creation environment (SCE) are designed on top of SIP based application servers. When compared to current PSTN networks, Next Generation Networks will be enriched by much more powerful terminals enabling the provision of new and innovative services. This remark may mean that massively used simple services with simple billing policies (e.g. flat rate) will demand much less resources from the network/application providers than PSTN services.

2.1 Benefits for NGN applications development

Application development in a NGN context is in many aspects very close to Internet application development. As a matter of fact, the main development skills required from NGN application developer are related to Java [13] and XML (eXtensible Mark-up Language) [22]. Thus NGN applications development will be accessible to a broader developer community, because it is more easy, productive and creative.
The easiness is due to the fact that need for knowledge that is specific to telecommunications is less than before and it demands for a rapid learning curve.
Productivity depends on the fact that most products don't provide a specific SCE: this allows using a standard IDE. This fact frees developers to choose the tools they are used to. Some systems provide several levels of APIs (abstract, medium and low level): this gives to the developers the flexibility of choosing the most appropriate level of abstraction for a given application (low level to control all protocol and network specific details, high level to hide network specificity). All these observations contribute for the developer productivity and, in average, a shorter time is needed for application development.
Creativity can increase because there is a move to use high-level application environments that can be used across different vendors. Having such modules can make the work of developers easier as they can concentrate in the programming aspects rather than the underlying technologies. On the other hand, the use of IT technologies makes the range of programmable features available to the developer quite wide, promoting the mix of IT functionalities (e.g.: email, instant messaging, presence, directories, web data) and telecommunications functionalities (e.g.: telephony, speech processing, quality of service, billing). These two facts free developers to focus on the creation of new types of applications that may bring new revenues to service providers.

2.2 Developer perspective

Developers have to overcome a set of challenges. They has to be adaptive in a very dynamic environment, because there will not be a single standard programming interface and developers need to select the most appropriate one from a wide range of choices.
Developers must become familiar with specific telecommunication problems such as billing, quality of service and security and they should be able to tackle these issues in a much richer but complex environment where the programmable features are very broad**.** However, the specific telecommunication aspects mentioned previously are not well supported by current programming interfaces.
A good understanding of communication technologies (like the SIP and Jabber [20] protocols) and IT technologies (like Java and XML) is required in order to build innovative services. Nevertheless, the simplicity and similarity of NGN communication technologies to other Internet protocols (e.g. HTTP and SMTP) facilitates the developer's job.
It is also important to follow the most relevant open source communities related to NGN to integrate the state-of-the-art third party components on his/her developments.
Service creation approaches in NGN can be therefore summarized in three categories: based on programmable APIs, scripting languages, or graphical SCE.

2.2.1 API

A whole variety and type of APIs are emerging in NGN products providing different levels of functional abstraction. Some are standards conformant (e.g. JAIN [12], SIP-servlet [8]) and others are not. This can present a very confusing picture to the developer community when having to choose which API to use for a particular service. On the other side developers can freely choose the tools they are used to, and linking components that offers APIs able to give connectivity with NGN protocols.

Some systems provide several levels of APIs (abstract, medium and low level): this gives the developers the flexibility of choosing the most appropriate level of abstraction for a given application (low level to control all protocol and network specific details, high level to hide network specificity).

2.2.2 Scripting languages

Scripting Languages are lightweight, highly customisable, 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 applications at runtime, and interact with running programs. These qualities and features make scripting languages applicable to the field of application programmability next to Application Programming Interfaces (APIs).
Scripting languages represent, in an XML-based file, the service behaviour that can be changed at run-time; they act like a dynamic reconfiguration of the script interpreter that follows a pattern of registering the static events and criteria that can be matched by events by the underlying network components, followed by declaration of service logic that should be executed in response to such an event. Typically, scripts are created, edited, and validated using regular editors or as a result of applying transformation techniques.
For example the Service Creation Mark-up Language (SCML) [7] and the Call Processing Language (CPL) are such scripting languages that connect existing components with a particular API, depending on the script file content. This approach enables rapid prototyping, rapid application development, and easy end-user customisation; if users are allowed editing their scripting files, they can directly personalize the service behaviour.

2.2.3 SCE

Using graphical Service Creation Environments (SCE) allows fast development of services with little knowledge of network protocols; service logic is designed using flow-charts of basic components offered by the SCE, like the one used to edit services with XTML (see figure 10). These components offer connectivity with different protocol stacks, linking to external code (usually java and C++), and connecting to databases. Of course, each component instantiated has to be programmed but the time to learn SCE's use is quick and it frees developers from some technical aspects. Depending on the chosen SCE, a developer can have more or less control on protocol message fields; in the first case a developer needs a deeper knowledge of technical issues to avoid risk of losing standard compliance; in the second case a service could be easily developed but it is not sure if this approach scales to more complex services where a team of developer must collaborate using the new approach based on graphical SCE.

3. Definition of evaluation criteria

In this section there is a definition of evaluation criteria used for classification and comparison of different service creation technologies, in short: supported network capabilities, mapping towards reference architecture, interface abstraction, kind of interface (and description language), suitability for 3rd party development, easiness to use, industry support, maturity, and future-proofness.
The first criterion is based on *Network capabilities*, i.e. the abstraction of underlying network infrastructure that can be used by application developers to exploit network functionalities; they can represent both functional (e.g. call control) and non-functional (e.g. authentication, logging…) aspects. Parlay/OSA [6] consortia have defined a set of capabilities, which have been considered as a basis for the following definitions of different network capabilities:

- Framework Functions is a part of the Open Service Access (OSA) API interface which provides management capabilities needed for accessing service interfaces in a secure and manageable fashion. It controls authenticated access to Service Capability Servers (SCSs) and also supports standard interfaces like service registration, service discovery, authentication, etc.
- Generic Call Control (GCC) feature: it is the set of interactive procedures required to establish, maintain, and release two party calls.
- Multi-Party Call Control (MPCC) feature is an enhancement of Generic Call Control functionalities with leg handling capabilities, in order to allows for multi-party calls to be established.

- Multi-media call control (MMCC) enhances the functionalities of the Multi-Party Call Control Service with multi media capabilities, allowing granular control of media stream (independent setup and tear down of media streams within the same call).
- Conferencing Call control (CCC) extends the Multi-Party Call Control with the ability to manage conference sessions. A conference can have zero or more members and includes the capability to have, via an appropriate conference bridge a multicast conference, sub-conferences.
- 3rd Party Call Control (TPCC) allows an application to set up and tear down a call between two users.
- Generic User Interaction (GUIN) is used to enable network resources such as voice response units to be connected to the caller to prompt them for information (typically digits) required by the application logic. It may also be used to record and delete messages applicable to a single call session.
- User location (UL) is the ability to determine the geographical location of a user: this may be undertaken using the cell identity, from which a mobile user is connected; otherwise it may also be provided using GPS information stored on the users terminal, or it can be ascertained for fixed PSTN terminals, or IP based terminals.
- User status (US) is about the availability of the user (also referenced as Presence information [19]): it defines two main events, subscription to and notification of changes in the communications state of a user.
- Data access session control (DASC) provides a means to control and manage data sessions and the establishment of a new data session. This has typically applicable to GPRS sessions within the underlying network infrastructure.
- Messaging is about accessing and managing mailbox folders: a messaging system is assumed to have the following entities: Mailboxes, Folders and Messages.
- Terminal capabilities (TC): is provided so that an Application provider my request from the network operator, what are the capabilities of the users terminal. The application may, need to send information such as pictures or text to the users terminal. The application provider is then able to tailor the information appropriately.
- User profile (UP): It contains information to present the personalized user interface within the capabilities of the terminal and serving network, and identification of subscriber services, their status and related service preferences.
- Matching to CAMEL/IN is an evaluation criterion that indicates how a technology can cover different aspects defined in CAMEL [21] specification for Intelligent Network and mobile network interoperability.

The second criterion defines which place a technology covers in the categorization proposed in P1109 project as reference architecture, depicted in figure 1.

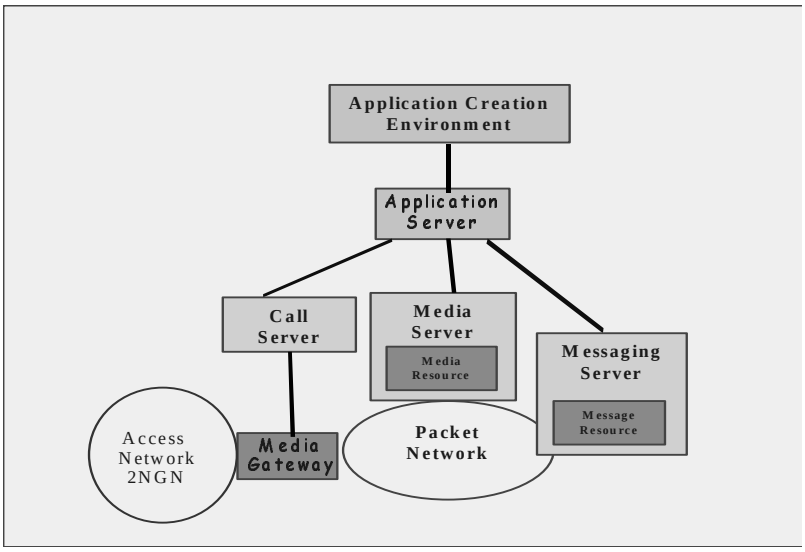

Figure 1. The P1109 Reference architecture

This layered architecture defines a distinction among technologies, depending on their characteristics: application server layer includes technologies used to execute services, programmed with tools, represented by the Application Creation Environment layer; call server

layer includes technologies handling routing and delivery of voice calls; media server layer represents technologies involved in multimedia communications, and messaging server stands for entities handling messaging and asynchronous communications. Media Gateway layer represents networks related technologies.

Third criterion is the evaluation of interfaces offered by technologies to developers; the interface evaluation defines the level of abstraction (AIL), the kind of interface (KOI), and its type of Interface Definition Language (IDL).

Regarding the abstraction level of an interface, an abstract interface hides technical details of the underlying technology to the developer, in order to gain more portability, easiness of use, concise programming; a mid level interface hides parts of the details of the underlying technology, but still requires some level of knowledge form developer, and also the ability to choose controlling low level details; a low level interface provides detailed access to the underlying technology (e.g. a network protocol stack), such that the application developer has to manage with less portable and more lengthy code, using technology specific API that are more difficult to learn.

The "kind of interface" should describe the communications method by which the technology in question is exposing network capability to external systems. This should include the following categories:

- Application Programming Interface: that can be Local, when the API is only resident on the local execution platform or Distributed, when it is accessible from distributed nodes in the network.
- Protocol based interface: if it is a direct interface to a protocol stack
- Scripting Language: if the information used to program a technology is passed using scripting languages 'interpreted' at runtime (e.g. XML-based and policy languages).

The type of interface defines the language used to define its API; we can classify them in Computing language based (Java, C++), middleware based (OMG IDL in CORBA, WSDL in Web Services), or data definition based (e.g. XML DTD for CPL or XML Schema).

Another criterion used in evaluation is programmability: that is suitability to 3rd party application development (TPAD), which describes the qualification of the technology in support of application development by 3rd party developers, and the suitability to 3rd party service provider (TPSP), which should describe the qualification of the technology in support of 3rd party service provider hosting of applications and services.

An important criterion is also Usability or Ease-of-use (EOU): this can be measured depending on:

- The background needed by the developer, i.e. how much knowledge/experience is required of the underlying technology
- Time-to-service, i.e. how quickly it is to develop and deploy applications using this technology
- Power: the scope of what may be accomplished by using the technology in question.

An important issue to be evaluated is also the industry and standard support (IS/SS), which measures technology's availability and maturity, showing how well this technology is supported in the industry and provide a general statement as to the level of its maturity in relation to approved standards.

Finally, the evaluation criterion of Roadmap technology (RT) should identify future publicly available plans for the technology and where possible describe time scales and details of releases (e.g. enhancements, features and functions), while the Future-proofness (FP) should describe how well a technology relates to emerging technologies in the industry and possible factors that promise a future for it.

Here follows a resuming table of all evaluation criteria and the possible values they can assume when applied to a particular technology.

Evaluation Criteria		Acronym	Possible Values
Network Capabilities	Framework Functions	FF	yes, partially,no
	Generic Call Control	CC	yes, partially,no
	Multi-party call control	MPCC	yes, partially,no
	Multi-media call control	MMCC	yes, partially,no
	Conferencing Call control	CCC	yes, partially,no
	3rd Party Call Control	TPCC	yes, partially,no
	Generic User Interaction	GUIN	yes, partially,no
	User Location	UL	yes, partially,no
	User Status - Presence	US	yes, partially,no
	Data access session control	DASC	yes, partially,no
	User Interaction-Messaging	UI	yes, partially,no
	Terminal capabilities	TC	yes, partially,no
	User Profile	UP	yes, partially,no
	Matching CAMEL-IN	CAMEL	yes, partially,no
Interface	Interface Abstraction Level	IAL	abstract, high, low
	Kind Of Interface	KOI	local,distributed, XML, language
	Interface Definition Language	IDL	C++, Java, IDL, WSDL,…
Programmability	for application development	TPAD	yes, partially,no
	for service provider	TPSP	yes, partially,no
Usability (Ease Of Use)		EOU	yes, partially,no
Support	Industry Support	IS	wide,significant, emerging
	Standard Support	SS	standard-based, proprietary
Product Maturity		PM	high,medium,low
Future Proofness		FP	wide,medium,low
Mapping towards Reference Architecture		MRA	Application, Call, Media, Messaging, Media gateway

Table 2: Definition of Evaluation Criteria

4. Assessment of service creation technologies

In this section we describe some of the more interesting technologies that can be used for service creation in NGN, using more relevant evaluation criteria defined in the previous section.

4.1 OSA/Parlay

The Open Service Access (OSA)/Parlay [6] defines an architecture that enables the inter-working between the IT applications and the telecommunications features in the mobile network through an open standardized interface, i.e. the OSA/Parlay API's. The network functionality is described as Service Capability Features (SCFs) and applications could be deployed in a third party administrative domain. SCFs implement groups of Parlay/OSA APIs (e.g. Call Control APIs, Mobility APIs, etc.) and provide access to the network capabilities that a Network Operator wants to export through OSA interface. They are provided/implemented by Service Capability Servers (SCSs) that are logical entities that implement one or more SCFs and interact with the network elements (e.g. SSP, HLR, Location server, etc.), as depicted in Figure 3.

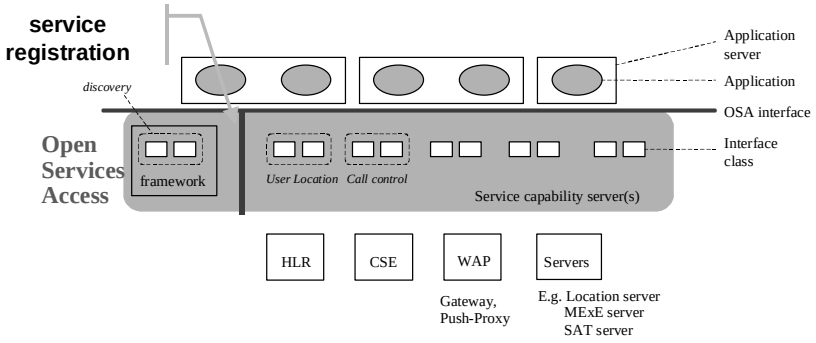

Figure 3: OSA architecture

The goal of OSA/Parlay is to identify and specify a Programming Network Interface in order to easily create applications using the network services provided by the Telco networks. The set of SCFs could be incrementally extended, because one of the aims of OSA/Parlay is to provide an extendible and scalable interface that allows for inclusion of new functionality in the network in future releases with a minimum impact on the applications using the OSA/Parlay interface. Main requirements and strong points of OSA/Parlay are:

- Hiding the complexity of the network, its protocols and specific implementation from the applications;
- Being suitable to 3rd party application development, but developers need a certain level of telecommunication expertise.
- Providing a secure, controlled access to network capability provided by a network operator to 3rd party service providers;
- Exposing almost all the network capabilities (defined in the previous section) provided by the corresponding network protocols and it eases the development of services combining several service capabilities and integrating IT applications.

OSA/Parlay maturity has become significant, mainly after creation of the joint working group, including 3GPP CN5, ETSI SPAN12, Parlay, and JAIN, with the goal to make these similar (but partially different) specifications converging in 2002 in OSA Release 4 and 5.
Referring to the P1109 Reference Architecture, OSA/Parlay can be seen as a complete effort to standardize interfaces among application server level and all underlying levels (call server, media server, messaging server).
Industry support is growing fast and Application Servers specialized for Parlay/OSA applications are proposed by several vendors proposing SDKs based on Web Services, J2EE development framework, or on JAIN SPA.
Parlay/OSA APIs provide a medium level of abstraction of the network capabilities. They provide an abstraction from different specific protocols, but the abstraction level of Parlay/OSA APIs is not judged oriented to traditional IT-developers and it could affects usability.
Several products are available, including Parlay/OSA Gateway, Parlay/OSA Framework, Application Servers, and applications. Most of the vendors' solutions are oriented to mobile network scenarios and several network trial have been already done, and some network operators is planning to deploy OSA/Parlay solutions in 2003.
OSA/Parlay APIs are provided as distributed APIs, through distributed processing mechanisms. Mapping on CORBA/IDL is already available, while mapping on Web Services technology (e.g., WSDL, SOAP) is under definition in Parlay-X initiative.

4.2 Web Services

The main goal of Web Services architecture is the realization of an interoperable network of services focused on service reuse and it is suitable both to interact with 3rd party applications and to export services by a network operator or a service provider.
The Web Services can be used to export network services by exposing its WSDL (Web Services Definition Language) [16] interfaces; these services communicate using SOAP [17] (Simple Object Access Protocol), a protocol used to transport data between web services; service discovery

and service registration are implemented accessing to the UDDI (Universal Discovery, Description and Integration) registry [15]; XML is used as data format for SOAP messages that rely on existing internet protocols like HTTP. Web Services implementations need that the language-dependent API must be translated in WSDL and the application server where web-services are deployed must translate incoming SOAP messages to the underlying interfaces (Java [13], CORBA...).
Different Web Services toolkits are available and some Application Creation Environments include them or offers a plug-in to handle Web Services. Toolkits can be used to translate in WSDL the existing applications' interfaces made with different languages. These toolkits also generates SOAP proxies used within the application server in order to translate SOAP messages in the underlying application language.
Network capabilities, as defined in OSA, are not present, because Web Services are a new middleware aiming to achieve real interoperability using SOAP as an application-level communication protocol, used to transport data between two web services, coding message parameters with XML. This message could be rely on existing Internet protocols but current toolkits implement SOAP mainly over HTTP, in order to bass through firewalls. XML encoding of big messages can obviously affect performances and introduce additional overhead.
With regards to P1109 reference architecture, services running on different Application servers could communicate using SOAP messages; another scenario can be made by a call server (e.g. a Parlay-X gateway) exposing WSDL interfaces (depicted with red boxes in Figure 4), that offers network connectivity to a 3rd party service running in an application server external to the call server domain; intra-domain connectivity among different components can be obtained migrating to Web Services, when different technologies must communicate.
A Web Services toolkit can then be seen as a plug-in of the service creation environment that can be used on code made with other IDE or it can be integrated within an IDE

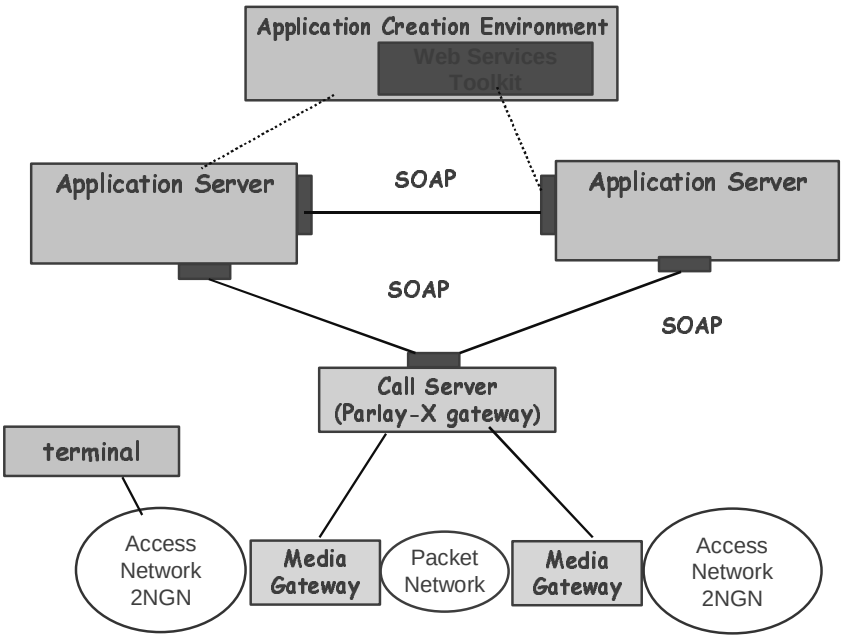

Figure 4: Web Services & P1109 reference architecture

Most important companies are investing a lot in Web-Services; in fact most used toolkits are provided by IBM, Microsoft (included in .NET framework), Sun (included in Sun One framework for J2EE) and also Apache is providing one. Web services are currently a good solution to integrate existing heterogeneous applications and a new way to access to the Web, but they are not mature yet for a widespread deployment because they still have some open problems like:

- The lack of a standard for transactions definition with XML, even if WSFL, XLANG and XAML are new specifications trying to cope these issues;
- The lack of a common standard framework for security, beyond reusing HTTPS;
- Absence of warranties about the contents of UDDI service repositories;

- Slow performances with current synchronous SOAP invocations provided by toolkits.
- No way to set up QoS parameters and to handle fault-tolerance and high-availability

4.3 SIP servlets

SIP servlets are a set of libraries that are used to create services on a SIP based network. The SIP Servlet API [8] is a Java API based on the previously existing Servlet API. SIP Servlets are also a programming model where the Servlets (the applications) are hosted by an infrastructure known as a Servlet container (see figure 5). The SIP Servlet specification has also the objective of standardizing the following aspects of a Servlet container: the rule based mapping between Servlets and SIP requests, the security model, the servlet deployment descriptor (as an XML DTD), a jar-based file format (similar to the WAR file format used by HTTP Servlets) for servlet deployment.
The SIP Servlet API allows application to initiate and to answer SIP requests. Therefore it simply exposes SIP capabilities (both User Agent and Proxy capabilities) to the application while hiding a few protocol details handled transparently by the SIP Servlet container.
SIP Servlet API is suitable for third party service development. It could be noted that third party service development is rather simple since they are seen as Java libraries.

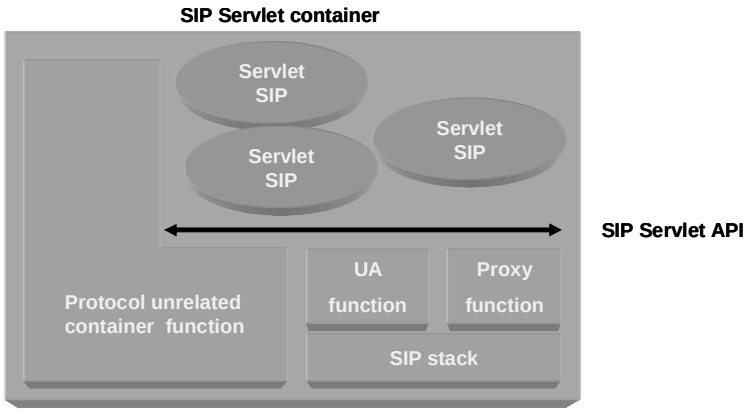

Figure 5 – SIP servlet and SIP servlet container

The SIP Servlet API allows application to initiate and to answer SIP requests. Therefore it simply exposes SIP capabilities (both User Agent and Proxy capabilities) to the application while hiding a few protocol details that are not considered useful to most of the applications. Protocol details handled transparently by the SIP Servlet container are: message retransmission, best response selection (e.g. when multiple responses are received when forking occurred), CSeq generation, Call-ID generation, Via header handling.
With regards to P1109 reference architecture the SIP Servlet container resides in the application server level, and as an emerging standard API, the SIP Servlet API is suitable for third party service development.
Looking at the JCP web site, the SIP Servlet API first specification version is still at community review stage (i.e. it is not yet publicly released). A second version is already foreseen and it should standardised non-API aspects of the SIP Servlet container.

4.4 JAIN SIP Lite

The JAIN SIP Lite API [12] is a Java API and it is only aimed at SIP User Agent type applications, which clearly define the kind of network capability exposed. Its methods expose SIP User Agent capabilities while hiding a few protocol details. The goal of the JAIN SIP Lite API is to enhance efforts made by JAIN-SIP and SIP servlets API: these ones can be seen as a wrapper on protocol stack and they requires an expert developer to be rightly programmed. JAIN-SIP Lite offers a higher-level API than is able to handle some SIP technical aspects, transparently to developer. The main differences compared to SIP servlets are that JAIN SIP Lite doesn't necessarily address application development within an application server and it doesn't mandate a SIP proxy function within its supporting platform.
With regards to P1109 reference architecture the JAIN SIP Lite API could resides in the application server or in desktop applications. As a standard API, the JAIN SIP Lite API is suitable

for third party service development. The JAIN SIP Lite API specification is currently standardised within the JCP. At the time this section is written the specification is still at community review stage, and then not publicly available.

4.5 VoiceXML (Voice Extensible Markup Language)

VoiceXML [14] has been defined as a technology that allows a user to interact with a web server through voice-recognition technology, which exploits Media Server capabilities. Using VoiceXML, the user interacts with voice browser by listening to audio output that is either pre-recorded or computer-synthesized and submitting audio input through the user's natural speaking voice or through a keypad, such as a telephone. VoiceXML can also be described as a phone markup language that can be used for voice applications that provide phone access to content and information, so it supports the network capability previously defined as GUI (Generic User Interaction).
However it should be noted that while there are many technologies concerned with speech recognition as a means of automated dictation, translation, and so on, VoiceXML is not one of them. Some elements of these technologies are present, but VoiceXML limits dialogs to enable voice interaction with arbitrary members of the public. Generic voice to text conversion, such as dictation or translation, requires some degree of training of the speech recognition engine in order to reliably recognise the speech pattern of a given user and is not a task suited to VoiceXML.
Other features are:

- XML Content can be accessed from any web server, providing a powerful solution for distributed content management;
- High Security: SSL can be used between voice browser and the web server thus making VoiceXML transactions highly secure, and not requiring human intervention;
- The code is processed in the “client side” on the voice browser, so any user can dial in the application;
- Each page acts as a standalone module that can be tested and debugged easily;
- Total abstraction of resource management.

Following the P1109 Reference Architecture, a VoiceXML interpreter can be seen as an enhanced feature of a media server, but its internal architecture (depicted in figure 7) is made of different parts:

- **The voice browser**: this is the software that renders the VoiceXML as a sequence of two dialogs between the system and the user. It is made of a core VoiceXML interpreter, integrated with software components for Text to Speech and audio file output and for speech recording and recognition (Automatic Speech Recognition). It is here that that the code is interpreted and “displayed”, and all the phones can be used to dialup the voice browser.
- **Web server**: where the application pages reside. The application pages can be VoiceXML files, ASP, JSP, or PHP to dynamically create VoiceXML pages.

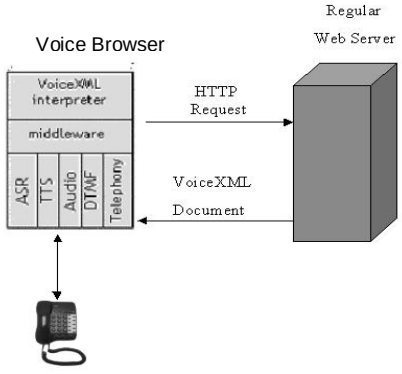

Figure 6. Basic Architecture for a Voice XML Service

The user dials in to a particular phone number or SIP URI corresponding to the voice browser.
The voice browser sends an HTTP request for the VoiceXML document to a server determined from the dialled number.

The voice browser renders the VoiceXML as a sequential dialog, consisting of prompts using either text to speech or pre-recorded audio. The input could be available through speech or touch tone key presses.
VoiceXML is a wide accepted standard with which voice applications are developed on the Internet. VoiceXML represents media server level in P1109 reference architecture and it should be noted that the interface from the application server to the media server has not been fully defined. It is possible that SIP and VoiceXML will be the interface between next generation application servers and media servers. SIP and VoiceXML can be used together for initiating and terminating sessions of all types, not just signalling and control sessions but also content sessions. These sessions could convey simple presence information such as, “I am in my office”, meaning that my presence is in the office or “I am at home” meaning don’t send me any documents or other media. The ability to establish these sessions means that a wide range of innovative services become possible.
VoiceXML is a high-level abstraction language and this means that developers with little training can use it. VoiceXML makes it easy to rapidly create new applications and shields developers from low level programming issues. VoiceXML also executes logic: main components of a VoiceXML-based speech service include tags, forms and rules that define the content and a speech browser for interpreting and presenting audio content. VoiceXML platforms are widely available and vendors are collected by the consortium VoiceXML Forum.

4.6 CCXML (Call-Control extensible Markup Language)

Call Control eXtensible Mark-up Language (CCXML) [11] is a call control language that aims to offer sophisticated call-handling capabilities, integration with call centre technology and multi-party conferencing (in particular to interact with media server). CCXML has been designed to complement and integrate with a VoiceXML system, because it cannot support some needed features. For example, support for multi-party conferencing, plus more advanced conference and audio control, the ability to give each active call leg its own dedicated VoiceXML interpreter. VoiceXML needs a more effective way of handling telephony resources and for richer and more asynchronous events. For example CCXML could be integrated with a more traditional IVR system and VoiceXML could be integrated with some other call control system.
Currently CCXML Specification is at a draft stage, publicly available on W3C in January 2002.
CCXML is designed to work with VoiceXML to provide voice dialog interaction with the user. However, a CCXML Interpreter can exist without a VoiceXML Interpreter.
Concerning the network capability exposed, CCXML aims to provide multimedia multiparty call control features, including:

- Outbound calling: Make and control an outbound call.
- Control of multiple calls: Independently control multiple outbound calls and optionally provide voice dialog interaction on each one of them.
- Whisper transfer: Provide a message to the recipient before connecting the call to the caller.
- Conferencing: Enable more than two people to converse with each other at the same time.
- Coaching: a supervisor whose purpose is to eavesdrop on a conversation between the user and the support agent in order to allow to coach to 'whisper' advice to the agent.
- Event handling: Handle asynchronous events that come from telephony infrastructure and the VoiceXML Interpreter.
- VoiceXML Interpreter session initiation and termination - Initiate a dialog session that is executed in a VoiceXML Interpreter and have the ability to start and stop a VoiceXML session at any time.
- CCXML Interpreter session initiation: During an application transaction, a new application can be launched (for example, a platinum card holder could be transferred to a different application after his PIN is entered, while a non-platinum card holder would continue through the original application).
- Conditional logic: Add conditional logic to your applications with elements such as <if>, <else>, and <elseif>.
- Post data to a web server: Interact with a web server using elements such as <goto> and <submit>.

The set of telephony-related events is derived from the JTAPI/JCP/JCC event.
The CCXML aims to reach mass deployments with programming switch from APIs to scripting language that could provide all the features needed by more than 90% of applications. The remaining applications would be programmed by traditional APIs technologies.

As depicted in figure 7, there are several identifiable objects in the CCXML universe: CCXML programs, Call legs, Conference objects, and Audio connections. These are media streams between two legs and/or conferences. Moreover voice dialogs objects representing voiceXML programs must interact with a two-way audio stream, and CCXML can send and/or receive the asynchronous events with these objects.

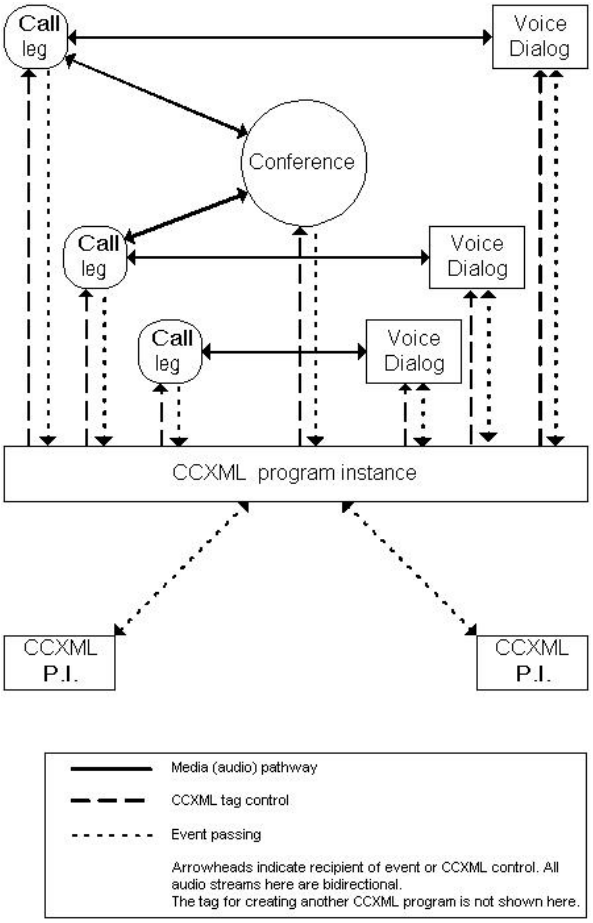

Figure 7 – CCXML information model

The CCXML aims to offer abstract interfaces and to be network agnostic and to be independent of the Call Control signalling but the specification draft seems to favour SIP network implementations.

In the P1109 reference architecture, the CCXML scripting interpreter fits well in the Call Server network element while the CCXML script may be hosted or dynamically generated by the Application Server. The CCXML Call Server would interface the Application Server via HTTP.

As mentioned above CCXML Call Server aims to be Call Control signalling independent meaning that it should be able to interface Media Gateways, Terminals and other non-CCXML Call Server by using any valid Call Control Signalling.

The interface between CCXML Call Servers for passing events between related Call Control threads is currently undetermined. HTTP and SIP are mentioned as two options as well as SOAP. This is a major drawback from current draft that should be fixed in future releases to guarantee interoperability between different CCXML Call Servers.

The same observation applies for the interface between CCXML Call Servers and voiceXML Media Servers.

Regarding third party programmability, CCXML follows very much the approach proposed by voiceXML, which is quite successful in terms of business models based on third party service providers and developers. CCXML third party provisioning may be achieved by solving issues, regarding trust relationship, and authentication.

The CCXML language was just proposed in the W3C as a draft in 2002. Its link to voiceXML success promises a bright future. There are still several issues to be resolved, like:

- Communication between CCXML instances on the same or different hosts;

- Communication between CCXML instances and VoiceXML instances on the same or different hosts
- Asynchronous communication of events from external entities to existing running CCXML instances
- Outbound Notification – CCXML script triggered by the application

4.7 JCC (Java Call Control)

JCC is a Java API that provides abstraction of call control capabilities, and it is standardized within the scope of JAIN. JCC allows applications to be invoked or triggered during session set-up in a manner similar in spirit to the way in which Intelligent Network (IN) or Advanced Intelligent Network (AIN) services can be invoked. JCC thus allows programmers to develop applications that can execute on any platform that supports the API. It also allows service providers to rapidly and efficiently offer services to end users by developing the services themselves, by outsourcing development, purchasing services developed by third parties, or a combination thereof. It has to be noted that a specific call model (including specific state machines) is defined within the API.
Regarding Network capability exposed, JCC includes the facilities required for observing, initiating, answering, processing and manipulating calls, as well as to invoke applications and return results during call processing. Here a call is understood to include (but is not necessarily limited to) a multimedia, multiparty session over the underlying integrated (PSTN, packet and/or wireless) network. It is likely that the facilities offered by this package will suffice for implementing most, but not all, of the basic and added-value services offered by carriers. In order to have the entire network capabilities, the programmer should refer to the JCAT (Java Coordination and Transaction) API.
The API can be used to implement a wide variety of other integrated voice and data applications like: First and Third-party originated and terminated calls, Voice virtual private network (VPN), Toll-free number translation, Voice-activated dialling, Click-to-dial, Meet-me conference.
JCC is explicitly aimed at converged network and is foreseen to be implemented using soft-switches/call agents/… thus it is assumed to provide an abstract interface to the application. Thus the network may consist of the PSTN, a packet (IP or ATM) network, a wireless network, or a combination of these, without affecting the development of services using the API.
With regards to P1109 reference architecture the JCC API could be implemented in two different ways: JCC implementation can be included as a library in applications in order to give network connectivity with the Call Server or directly with the Media Gateway level.
The API is not intended to open up telecommunications networks' signaling infrastructure for public usage. Rather, network capabilities are intended to give an approach that allows independent service developers to develop applications supported by the network without compromising network security and reliability. With regards to this third-party use case it is also worth mentioning the existence of a reference implementation and a compatibility test suite. A draft of the final version is publicly available since January 2002. JCC is the Java version of the Parlay call control API but, unfortunately, not functionally identical to the UML version of the Parlay call control API; it is hoped that future revisions to the JCC and Parlay call control APIs will close this gap.

4.8 CPL (Call Processing Language)

CPL is a scripting language defining how to handle outgoing and incoming calls in NGN networks. CPL was developed as an XML-based scripting language to be run on a SIP proxy server to implement services.
CPL is mainly intended for non-trusted end users to upload their services on SIP servers. CPL scripts created by end users can be uploaded to SIP servers for call set-up in a secure environment. CPL is lightweight, efficient, easy to implement, extensible because it is possible to add customized features in a way that existing scripts continue to work. CPL exposes a main network capability: the call control feature, then in the reference architecture CPL interpreter can be hosted by the Call Server and the CPL scripts could be retrieved from the Application Server via HTTP GET/POST (even, dynamically generated from Server Side components) following the mechanisms used by VoiceXML architecture.
CPL is a high-level abstraction language because also users can easily write and edit their applications. As CPL is XML-based, the kind of interface offered to the application level is the

scripting language itself. There is a range of commercially available CPL products and it should be noted that probably all Application servers deployed in SIP/H323 network have support for CPL.
CPL could be used for implementing services in a number of different scenarios: using scripts created by the end users and uploaded to a server, or using scripts created by the server administrators on behalf of the users, or using scripts created by web applications that translate it in CPL. Because CPL is a standardized language, it can be used to allow third parties to create or customize services for clients. These services can then be run on servers owned by the end user or the user's service provider. CPL is a quite mature language and it is almost fully specified. However, CPL cannot originate calls towards two or more users because it is activated only through call related events, and cannot be used to create complex scripts. This is because CPL structure is defined on DTD (Document Type Declaration) [23], that is one of the language used to define and validate XML files structure, but it is not so flexible like XML-Schema [24] that is more extensible and it is easier to learn because is similar to XML.

4.9 SCML (Service Creation Mark-up language)

SCML [7] is an XML-based scripting language useful to define services in NGNs and defined within the JAIN Initiative. The figure 8 describes the relationship between SCML language and JAIN/Parlay reference architecture.

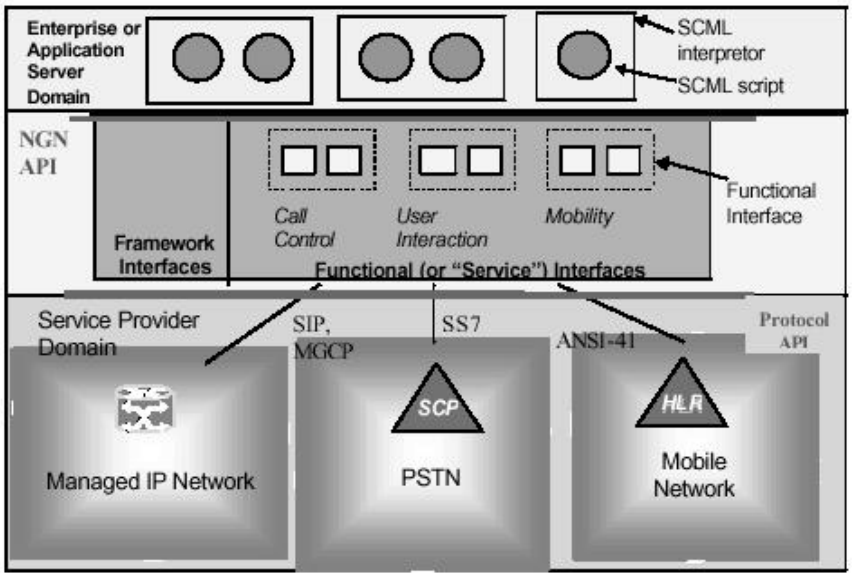

Figure 8: SCML approach in a Parlay/JAIN environment scenario

The Service Creation Mark-up Language (SCML) Service Creation environment assumes (but is not limited to) a JCC 1.0a API implementation.

SCML is supposed to handle multiple capabilities (from Terminal Capabilities to User Interaction, Multiparty, Multimedia, and Generic Call control), which could be exposed by Service Provider and used by Application developers. Currently the work has focused merely to Call Control Capabilities. In addition to that the scripting language allow through the register message to register event, which the application is interested to be notified to. The register element enables a script to register for a particular set of events. A registration remains valid until the script is deactivated. A script is activated upon execution and if the script registered for events, it is deactivated through de-registering the scripts registrations.
With respect to the reference architecture depicted in Figure 10, SCML scripts can run on a call server, on application server or on intelligent endpoint (not depicted here). A script can control the service logic and consequently the call depending on the information which is provided by other functional interfaces (i.e. JCC) and/or on settings controlled by the user. In the Service Creation Phase and in the application creation environment the script is created by using XML tools (e.g. XML or XSL editors, XSLT), text editors or by converting JAVA/C++ programs. The Application Creation environment takes also care of deployment phase, where the SCML script is validated, and upon syntax validation is stored in a repository. The script will then be activated by downloading it to the Application Server for execution.

An XML processing engine is supposed to run in the application server (SCML processor). The SCML processor acts as an interpreter to convert SCML instructions to the Call Server language (SIP Protocol, JCC API…)
In Figure 9 CASE a, alternatively the Application Server could send XML messages (i.e. via SOAP) to another XML processor, collocated in the Call Server making the interaction Application Server – Call Server language independent.
In Figure 9 CASE b, the former approach has better performance and it could be more suitable for inter-domain interactions, the latter would be more appropriate when Application Server and Call Server belong to two different domains (3rd Party Service Providers).

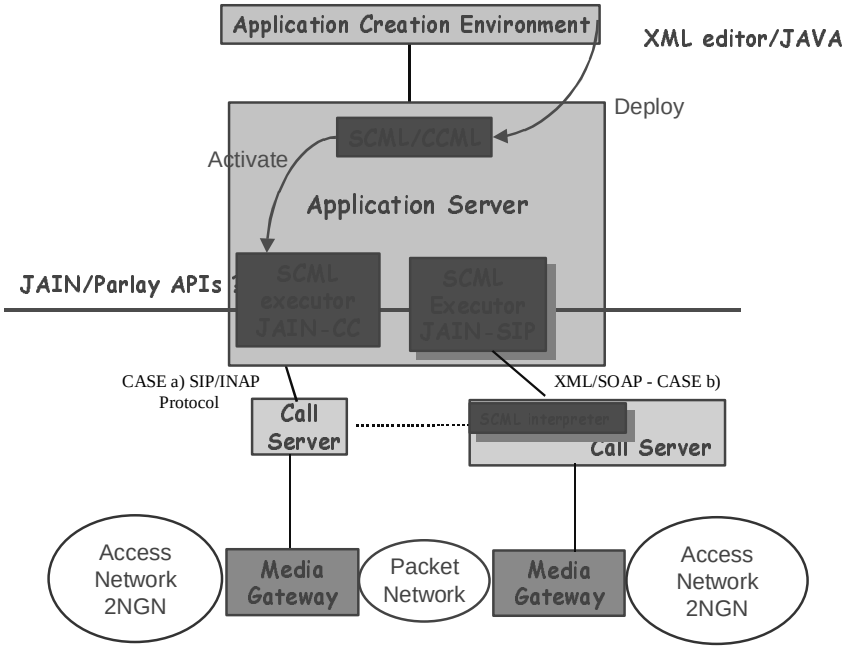

Figure 9: SCML in an NGN scenario

The interface abstraction can be considered high level API. It is based on JCC API standardised by JAIN and therefore it's truly protocol independent. It hides network complexity and it allows handling basic events to process a call. The element type can be easily mapped on any network protocol primitives (SIP, ISUP, Q931, INAP).
The Service Creation Mark-up Language has been designed to make life easier to application developers, in particular to the ones who are familiar with XML and web programming. While is using DTD (Data Type Definitions), SCML is using XML schema. The advantage of the latter approach is that the programmer does not have to learn a new notation. The application developers should be able to start developing services after a very short training period.
Regarding suitability to 3rd party service provider, SCML does not support explicitly mechanisms to support not-trusted relationships. However it could rely for that on the security mechanisms, which are provided by Parlay/OSA (Framework interfaces). In addition SCML scripts could be easily hosted on network provider domain and executed in a trusted environment.
The SCML looks quite easy to use (almost the same complexity of CPL). However with respect to CPL it is more powerful and flexible.
OSA/Parlay APIS are being specified by JAIN SCE Expert Group. The specification is also available at XML.org and they have been submitted as Internet Draft to IETF PINT Working Group (November 2001). Current version under Expert Group review is 0.5.1. The specification is in early stage and there is currently no product supporting it yet. The work is promising and inputs to Parlay web services and Parlay-X are likely to be done.

4.10 XTML (eXtensible Telephony Markup Language)

XTML [26] is an XML-based scripting language, which has been designed to provide a framework for telephony or multimedia application development without relying on a specific signaling protocol. It doesn't mean however that the application is independent of the signaling protocol, but merely that this technology is. In particular, an application can be very protocol-dependent if the support of the signaling protocol is offered at a low level. XTML is event-based: A XTML application is composed of a set of event handler, which responds to some given events. Events can be either protocol-independent (a timer expires, a session is started) or protocol-dependent (a SIP message have been received). An event-handler is made of a set of actions, which are linked together to reflect the application call-flow, as depicted in figure 11.

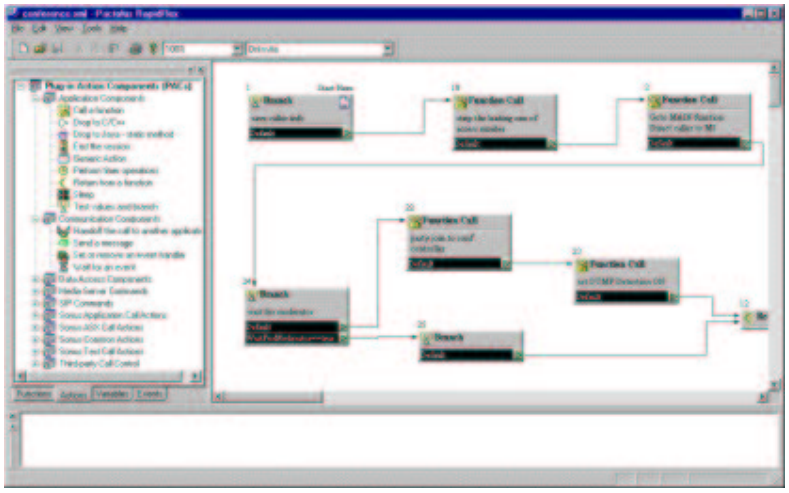

Figure 11. The Service Creation Environment for XTML

In addition, XTML introduces the notion of variables, which can be either local to a function (event handler), global or shared (i.e. shared among all the instances of an application). Scripts in JavaScript can be attached to any action to be executed to either before the action is executed, after the action is executed (but before the result is evaluated) or after the action is executed (and after the result is evaluated). Scripts could be used to do computation tasks or to update the value of variables.

The support for protocol-dependent aspects is introduced using a plug-in mechanism to define protocol-dependent actions. Each protocol-related action (e.g. the sending of a SIP message) is developed as a plug-in (with a specific name, specific parameters, and of course, a specific behavior). XTML is used by a proprietary technology, and it was proposed as a W3C standard. XTML specifications are public and open so that any vendor could build their own XTML-aware product. At the moment, only one vendor has developed products based on the XTML technology. Since XTML in itself is protocol agnostic, there is no network capability exposed.

The level of abstraction of the network level with regards to the application level depends on how the protocol is made available to the application: the only implementation available allows connecting with MGCP and SIP network elements. An XTML application is responsible for handling all of the SIP messages received (which are related to the current session), and to fully specify the SIP messages to send. However, it is the responsibility of the SIP stack to handle retransmissions and to check the validity of both the received and sent messages.

An XTML application is supposed to be run on an Application Server that can support one or more signaling protocols. So the exact position of XTML in the network depends on the signaling protocols used.

Since an XTML application server can be run on a very limited infrastructure, third party developers can develop services without making huge investments. However, since the protocol is handled at a rather low level (at least, for the current implementation of the SIP protocol), some specificities of the target network may need to be taken into account by the service developer. Since XTML is not standardized, the roadmap for that technology is not known. The only thing that can be noticed is that XTML seems to be in a pretty stable state.

5. Overall assessment of evaluated technologies

In the following table, we summarize different technologies, putting in evidence their evaluated features: network capabilities offered, kind of interface and supported languages, programmability, usability.

The following table is a summary of the evaluation of the technologies with respect to the selected criteria. Acronyms in the table can be found and are extensively described in the first section, while here is a brief resume of acronyms used: CC (Generic Call Control), UL (User Location), MP (Multi party CC), US (User Status or presence), MM (Multimedia), DASC (Data Access Session Control), TPCC (3rd Party CC), DTD (Document Type Definition), UI (User Interaction Instant Messaging).

	Network Capabilities	**Interface & Language**			**Programmability**		**Usability**	**Industry**
		Abstraction Level of Interface	Kind Of Interface	Interface Description Language	Applic. develop.	Service prov.		
OSA/Parlay	*Framework,* *CC (also MP, MM)* *UI* *UL/US* *DASC* *Messaging, (others)* *Good matching CAMEL/IN*	*Low level*	*C++ & Java*	*IDL, WSDL*	*yes*	*yes*	*No (unless a SCE is provided)*	*Wide* *Standard based*
Web-Services	*N/A* *Application-to Application middleware*	*abstract*	*XML Distributed*	*WSDL*	*Yes*	*Yes*	*Yes (with Toolkits)*	*Wide* *Standard based*
SIP Servlet	*CC* *IM & Presence* *Not matching IN*	*Low level*	*Java*	*N.R.*	*yes*	*no*	*Yes (with good SIP knowledge)*	*Significant* *Standard based*
JAIN-SIP Lite	*CC* *IM & Presence* *Not matching IN*	*Low level*	*Java*	*N.R.*	*yes*	*no*	*Yes (with good SIP knowledg)*	*Significant* *Standard based*
XTML	*No network capabilities (network capabilities are PAC dependant)* *Not matching IN*	*PAC dependant*	*XTML*	*N.R.*	*Yes*	*No*	*Yes with SCE / No without SCE*	*Proprietary*
VoiceXML	*GUIN*	*High Level*	*XML*	*DTD*	*Yes*	*Yes*	*Yes*	*Wide* *Standard based*
CPL	*CC*	*High Level*	*XML*	*DTD*	*Yes*	*Yes*	*Yes*	*Wide* *Standard based*
SCML	*CC (MCC)* *UI* *TPCC* *NO Camel/IN matching*	*High Level*	*XML (Java)*	*XML Schema*	*Yes*	*Yes*	*Yes (if SCE available) otherwise no*	*Proposed to JAIN-SCE standard & IETF*
CCXML	*CC* *yes*	*High Level*	*XML script*	*DTD*	*yes*	*yes*	*Yes*	*Emerging* *W3C Standard*
JAIN-CC	*CC* *UI* *Partly matching IN (some similarities in call model)*	*Low level*	*Java*	*N.R.*	*Yes*	*No*	*No*	*Significant* *Standard based*

Table 11. Overall assessment of service creation technologies

The overall assessment highlighted some key aspects that have to be considered when deploying NGN service platforms, which are summarised below.

SIP technology is the key

Product selection and evaluation has shown that SIP technology is key to the communications needs of an NGN deployment. There are however, still several major issues that SIP products must support before they may be considered mature enough for scalable, multi-service, managed communications networks. Functions in support of service selection, QoS, billing and security are four such areas of required attention.

Decreasing dependency on specialist Telco expertise

The dependency on specialist telecommunications expertise for service development is certainly declining. The convergence between the communication and Internet domains is a reality and brings with it the benefit of being able to apply more widely known IT technologies and methodologies that appeal to a broader developer community.

Balancing intelligence: edge and core

The location of intelligence is no longer restricted to the core-operating network. When compared to current PSTN networks, Next Generation Networks will be enriched by much more powerful terminals enabling the provision of new and innovative services. This remark may mean that massively used simple services with simple billing policies (e.g. flat rate) will demand much less resources from the network / application providers than PSTN services. Intelligence may be realised also at the edge of the network. Service providers should look to find their best synergy between edge and core offerings and accepting edge solutions as an opportunity rather than a threat. The terminals emerging support this edge model and will enable the provision of many new and innovative services.

Specific SCE vs. general purposes IDE

It has been observed that most NGN products don't provide a specific SCE as their programming interfaces (Java/C++ APIs, XML based scripts (e.g. voiceXML and CPL)) may be used with standard IDEs. This is an important SCE evolution as it frees developers to choose the tools they are familiar with, used to, better fit their needs and hence this enables faster service productivity and a more attractive model to application developers.

Wide availability of APIs

A whole variety and type of APIs are emerging in NGN products providing different levels of functional abstraction. Some are standards conformant (e.g. JAIN) and others are not. This can present a very confusing story to the developer community when having to choose which API to use for a particular service. However, at the same time it means that the same service may be implemented in many different ways allowing service providers better chance to differentiate themselves from each other.

Using open source software

Open source products have proved to at least generate the mindset that they could play an important role in NGN system solutions. It remains to be seen how successful this approach will be. However operators of NGN services will have to recognise this as a valid deployment model especially given the effective extensibility and tailor made adaptations that this approach offers. This could equate as another means for service provider differentiation.

Lightness vs. high availability and reliability

Network operators should become more and more aware that not all services need 5 nines availability and/or high quality (Internet and mobile telephony teach). High availability might not be a must for certain services and operators should carefully evaluate the best trade-off between quality, time and costs for providing it.

6. Conclusions

In conclusion the experiences of the project in service development phase has concluded that in general most vendors are adopting industry standard tools such as Java, XML, CPL and SIP servlets and in many cases in combination with SIP for their NGN products. SIP application

servers have matured as initial product offerings and are certainly capable of small-scale deployment scenarios today. However product maturity, system stability and generally all-around management capability might still be an issue. Functions in support of service selection, QoS, billing and security are four such important areas of required attention and further investigation. An important step forward achieved with SIP application servers is the integration between communication and Internet technologies. This has major implications for enabling the creation of many new innovative services for NGN networks. This evaluation has shown that as well as application servers, media servers are also a core component of NGN architecture XML technologies, for example VoiceXML, are also contributing to the integration of communication and Internet technologies. Concerning service creation, development of NGN services is made accessible to a broad public of application developers and in many ways is very close to the web and IT developer community approach, thus helping to enhance the productivity of application development, reducing the time to market of new services and on average only a couple of weeks, and even days, in some cases are required to develop new applications.

7. Acknowledgements

The information presented in this paper is based on the work done in the EURESCOM project P1109 “Next Generation Networks: the service offering standpoint” [1]. The authors would like to thank all EURESCOM P1109 participants, in particular Stephan Tuffin (France Télécom R&D), Paulo Chainho (Portugal Telecom Inovaçao), and Musa Halil (British Telecom – BT Exact).

8. References

[1] Eurescom Project P1109 Next Generation Networks: The services offering standpoint. On-line at http://www.eurescom.de/secure/projects/P1100-series/P1109/P1109.htm

[2] Lago, P., Licciardi, C.A., Canal, G., Andreetto, A., An architecture for IN-Internet hybrid services. In Computer Networks Journal, Special Issue on Intelligent Networks and Internet Convergence, T. Magedanz, H. Rudin, I. Akyildiz (Eds.), Elsevier, Vol. 35(5), April 2001, pp. 537-549

[3] Licciardi, C.A., Falcarin, P., Next Generation Networks: The services offering standpoint. In Comprehensive Report on IP services, Special Issue of the International Engineering Consortium, October 2002.

[4] Handley, M., Schulzrinne, H., Schooler, E. and Rosenberg, J., SIP: Session Initiation Protocol, RFC 2543, March 1999.

[5] Andreetto, A., Licciardi, C.A., Falcarin, P., Service opportunities for Next Generation Networks, In Proceedings of the Eurescom Summit 2001, Heidelberg, Germany, November 2001.

[6] The 3rd Generation Partnership Project (3GPP) Open Services Architecture (OSA). On-line at http://www.3gpp.org.

[7] Bakker, J.L., Jain, R., Next Generation Service Creation Using XML Scripting Languages, 2002. On-line at www.argreenhouse.com/papers/jlbakker/bakker-icc2002.pdf

[8] SIP Servlets specification. On-line at http://www.ietf.org/internet-drafts/draft-peterbauer-sip-servlet-ext-00.txt

[9] Lennox, J. and H. Schulzrinne, “CPL: A Language for User Control of Internet Telephony Services”. On-line at http://www.ietf.org/internet-drafts/draft-ietf-iptel-cpl-04.txt

[10] JAIN Java Call Control (JCC) API, Java Specification Request (JSR) 21", 2001. On-line at http://jcp.org/jsr/detail/21.jsp.

[11] Voice Browser Call Control: CCXML Version 1.0. W3C Working Draft 21st February 2002: http://www.w3.org/TR/2002/WD-ccxml-20020221/

[12] The JAIN APIs: Integrated Network APIs for the Java Platform. (White Paper), Jun. 2000. On-line at http://java.sun.com/products/jain

[13] Java™ 2 Platform, Enterprise Edition Specification Version 1.3. On-line at http://java.sun.com/j2ee/docs.html

[14] VoiceXML specification. On-line at http://www.voicexml.org

[15] UDDI Technical White Paper, http://www.uddi.org/

[16] Web Services Description Language (WSDL) 1.1, http://www.w3.org/TR/2001/NOTE-wsdl-20010315

[17] Simple Object Access Protocol (SOAP) 1.2 specification. On-line at http://www.w3.org/TR/SOAP

[18] Common Object Request Broker Architecture (CORBA) specification. On-line at www.corba.org

[19] A Model for Presence and Instant Messaging. On-line at http://www.ietf.org/rfc/rfc2778.txt

[20] Jabber Software Foundation, On-line at www.jabber.org/

[21] The CAMEL (customizable applications for mobile enhanced logic) standard. On-line at http://www.umtsworld.com/technology/camel.htm
[22] XML (eXtensible Mark-up Language) specification. On-line at http://www.w3.org/XML/
[23] DTD (document Type Description) language specification. On-line at http://xml.coverpages.org/XMLSpecDTD.html
[24] XML-Schema specification. On-line at http://www.w3.org/XML/Schema/
[25] JAIN SIP Lite specification. On-line at http://jcp.org/jsr/detail/125.jsp
[26] XTML (extended Telephony Mark-up Language). On-line at www.pactolus.org