International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:2, No:2, 2008

# A Mark-Up Approach to Add Value

Ivaylo I. Atanasov, and Evelina N.Pencheva

***Abstract*—** This paper presents a mark-up approach to service creation in Next Generation Networks. The approach allows deriving added value from network functions exposed by Parlay/OSA (Open Service Access) interfaces. With OSA interfaces service logic scripts might be executed both on call-related and call-unrelated events. To illustrate the approach XML-based language constructions for data and method definitions, flow control, time measuring and supervision and database access are given and an example of OSA application is considered.

***Keywords*—** Service creation, mark-up approach.

## I. Introduction

IN Next Generation Networks (NGN) the services and applications are the greatest revenue- generating potential. As a network of services, NGN makes clear separation between services and underlying transport technology. The services will be provided through a common, unified and flexible control environment over multiple types of transport.

It is difficult to predict which will be the most attractive application, so the focus is on service development tools making the process of service creation efficient, easy and productive one, and thus reducing time-to-market.

Evaluation criteria used for classification and comparison of different service creation technologies are shown in [1]. These criteria involve supported network capabilities, interface abstraction, kind of interface and description language, suitability for 3rd party development and industry support. A major prerequisite to rapid service development is open service access to network capabilities through standardized application programming interfaces (APIs). These APIs provide application developers with programmability of resources such as protocol stacks and data bases, by defining these resources in terms of objects and methods, data types and parameters that operate on those objects. Providing different levels of functional abstraction the APIs hide network specificity.

Several industry groups developed such open APIs, including Parlay/OSA (Open service access) and JAIN (Java applications for intelligent networks).

Manuscript received June 30, 2006. This work was supported in part by the research project 709 NI-7 Component development in next generation networks, Funded by TU-Sofia, Bulgaria, (2006).

I. I. Atanasov is with the technical University of Sofia, 8 bld Kl.. Ohridski, 1000 Sofia, BG; e-mail: iia@ tu-sofia.bg), tel+359 2 965 2050

E. N. Pencheva is with the technical University of Sofia, 8 bld Kl.. Ohridski, 1000 Sofia, BG; e-mail:enp@ tu-sofia.bg).

Although intended for UMTS networks, the principles of Parlay/OSA are applicable in the NGN domain as well. Parlay/OSA APIs provide a standard, object-oriented component model that enhances the traditional intelligent network approach of defining service building blocks with reusable units of functionality. Many exciting new services for NGN will derive added value from combining network functions such as call and data session control, mobility, user interaction, messaging and charging.

The JAIN initiative developed set of APIs for programming services in public networks. JAIN APIs do not support the whole variety of network capabilities like OSA APIs, and are mainly oriented to call control and user interaction.

While OSA APIs and JAIN APIs are protocol agnostic, there exist other service creation technologies based on Session Initiation Protocol (SIP) such as SIP Servlet and JAIN-SIP Lite. The network capabilities exposed by JAIN SIP Lite API and SIP Servlet API are call control, presence and availability and instant messaging.

While those APIs can be implemented by programming languages, eXtensible Markup Language (XML)-based languages possess many attractive features making them applicable to NGN service creation [2, 3]. XML-based scripting languages are easy to learn, platform and programming language independent. XML allows a user to define his or her own tags and can be easily adapted to specific application domain. Services scripts can be downloaded and interpreted during the run time.

Several XML-based languages such as CPL, CCXML, VoiceXML and SCML for service creation are proposed [4]. These languages can be used to create applications combining network functions mainly for call control and user interaction, but none of them do not support the full set of network capabilities exposed by Parlay/OSA APIs.

In this paper we present in brief an XML-derived approach to NGN service creation [5, 6, 7 and 8]. To illustrate how services might use the palette of network functionalities exposed by OSA, we define mark-up constructions within the framework of *Service Logic Processing Language* (SLPL). Service scripts can derive added value from network capabilities using nodes for invocation of method supported by OSA interfaces. In SLPL, nodes for flow control give the language expressing power of programming languages. As in telecom applications the notion of time is important and the use of timers is frequent, the markup approach offers means for time measuring and supervision. Further, NGN services will be more customized and they become increasingly data centric, so means for database access are provided

also. The next sections of the paper address the overall structure of SLPL service script, language construction for data type and method definitions, flow control and also means for time supervision and database access.

To illustrate the use of the mark-up approach an example of service logic script described in SLPL is given. Some aspects of the language translator concerning the usage of the SLPL are discussed.

## II. Service Logic Script Structure

The structure of a SLPL service script is modular and encompasses definition part and executive part. The node '*logic*' is used to denote logic beginning and logic end. In the definition part enclosed by '*definition*' node types' definition, variables' definition and definition of the methods supported by the application-side (callback) interfaces are given. The executive part enclosed by '*execute*' node is built of invocations of methods supported by the network-side interfaces, expectations of results and statements for flow control, assignment statements and others.

### *A. Definition of Types, Variables and Methods*

The types' definition involves simple or/and complex types. The variables' definition enlists variables where each variable is defined by name and type.

SLPL supports all simple and complex types defined in OSA and provides appropriate language constructions For example, Fig.1 shows the SLPL definition of a tagged choice of data elements.

```
<union name="TpAoCOrder"
    switch="TpCallAoCOrderCategory">
      <on val="P_CHARGE_ADVICE_INFO">
          <element name="ChargeAdviceInfo"
                  type=" TpChargeAdviceInfo"/>
      </on>
      <on val="P_CHARGE_PER_TIME">
          <element name="ChargePerTime"
                  type="TpChargePerTime"/>
      </on>
      <on val="P_CHARGE_NETWORK">
          <element name="NetworkCharge"
                   type="string"/>
      </on>
</union>
```

Fig. 1 SLPL description of a tagged choice

The declaration of variables of data types defined follows the similar approach. For example, Fig.2 presents a SLPL declaration of a variable of numbered set of data element type.

The methods' definition starts with the node '*methods*' followed by a list of definitions of methods which belong to the application i.e. callback interfaces in terms of OSA. Each OSA interface method is defined by its parameters,

```
<id name="userAddress" type="TpAddressSet">
  <sequence>
    <item index="0">
      <structure>
        <element name="plan"
                 value="P_ADDRESS_PLAN_IP"/>
        <element name="addrstring"
                 value = "164.23.7.3"/>
        <element name="name" value = ""/>
        <element name="presentation" value = "1"/>
        <element name="screening" value = "1"/>
        <element name="subaddrstring" value = ""/>
      </structure>
    </item>
    <item index="1">
      <structure>
        <element name="plan"
                 value="P_ADDRESS_PLAN_E164"/>
        <element name="addrstring"
                 value = "+359888010101"/>
        <element name="name" value = ""/>
        <element name="presentation" value = "2"/>
        <element name="screening" value = "2"/>
        <element name="subaddrstring" value = ""/>
      </structure>
    </item>
  </sequence>
</id>
```

Fig. 2 SLPL declaration of a variable of numbered set of data elements type

returned type and exceptions. The method body is used to get the results returned by network-side interfaces and to store them in local variables.

### *B. Flow Control*

The executive part denoted by '*execute*' is built of statements. There are different types of statements such as: statements for flow control, assignment statements, method invocations, statements for catching exceptions, synchronizing constructions and others. Before a method of network-side interface is invoked, values for the actual method arguments have to be assigned.

To request a service from network-side interfaces, the application logic has to invoke its methods (Fig.3). When a method is invoked its interface is specified and actual values of its argument are given.

The '*try-catch*' statement encompasses the method invocation and enlists exceptions that can arise and the statements processing them.

When the application has to wait for the result of method invocation (synchronous communications) the '*wait*'-statement is used. The 'wait'-statement enlists the expected events and the maximum time to wait. By invocation of application-side interface methods, the network-side interfaces return the result of the requested service. Flow control statements such as 'if'-statement, 'case'- statement and 'while'-statement are used to process results returned. '*If*'-statement is used to check a logical

```
<invoke>
   <interface name="IpUserLocation"
   < method name = "locationReportReq">
      <arguments>
         <argument name = "appLocation"
                   valref = "userLocationReference"/>
        <argument name = "users"
                   valref = "usersAddress"
      </arguments>
      <returns>
         <return type = "TpAssignmentID"/>
      </returns>
      <raises>
        <exceptions>
           <exception type =
             "TpCommonExceptions"/>
           <exception name =
             "P_APPLICATION_NOT_ACTIVATED"/>
           <exception name =
             "P_INFORMATION_NOT_AVAILABLE"/>
           <exception name =
             "P_INVALID_INTERFACE_TYPE"/>
        </exceptions>
      </raises>
   </method>
   </interface>
   <set refid="call_assignment"/>
</invoke>
```

Fig. 3 SLPL definition of method invocation

condition and with '*case*'-statement decision may be done based on multiple choices. '*While*'-statement is used when a set of iterations has to be repeated till a predefined condition is evaluated to be true.

### C. Time Measuring and Supervision

In telecom applications the notion of time is important and the use of timers is frequent. The normal use of time and timers is when modeling delays, supervising functions to be performed and measuring intervals. In SLPL, the concept of 'real' time is adopted. This notion of time exists and the service logic may obtain time measurements using time constructions of the language. The notion of the 'real' time acquires the value of the current time by two parameterless operations, '*CurrentDate*' and '*CurrentTime*'. On contrary with OSA in SLPL time related type are based on simple type Float that provides greater flexibility. Delays and supervisions are obtained by the use of '*Timer*', which also is a predefined type. Methods '*SetDate*', '*SetTime*' and "*Reset'* that support timer operation are provided.

### D. Database Access

In NGN services and data they require will be distributed among application servers. Service execution may be done on an external server and data related to the service may be stored on remote host in a database.

The SLPL provides service logic with means for database access. The method '*DB_modify'* is used to update, insert or delete record(s) in an existing table and the type it returns corresponds to the number of modified records. The method '*DB_retrieve'* is used to retrieve data and the type it returns consists of database response (for example, the requested information). The database query is in a form of SQL statement and in method invocation this query is an argument of type string. As the method '*DB_retrieve'* returns a string, this string has to be converted in a reasonable SQL response. This conversion is done by another method **'***DB_conversion'* which returns a set of structures representing records retrieved from the database. For each SQL statement a different '*DB_conversion'* method is defined that reflects the structure of logical records returned in database response.

In order to give an example for usage of SLPL language construction, in the next section we consider an application that exploits OSA APIs and illustrates 3rd party service control aspects of OSA.

## III. Service Logic Example

Let us consider an example of 'quick toll' service that exploits OSA interfaces. A logistic company wants to speed up the transition of the company's vehicles through the toll gates of overloaded highway. The speed-up effect is achieved by mobile payment of tolls. The 'subscribers' of the service are company vehicles which are equipped with mobile devices. The 'quick toll' service registers the passing through any of the toll gates of the highway and charges the respective mobile subscriber.

### A. Service Logic Implemented by OSA Interfaces

Let us suppose that the site of any of the toll gates is covered by a cell. When a company vehicle enters the cell covered by a toll gate the toll is reserved on the subscriber's account. When the company vehicle leaves the cell covering the next site of the toll the subscriber's account is charged with the amount due. The service exploits the following OSA interfaces:

- Mobility: to determine entering in and leaving from the highway;
- Charging: to charge the subscriber's account with the amount due.

Further, to illustrate the usage of SLPL time constructions let us assume that if a vehicle enters the highway in the middle of the week on Wednesday and Thursday or between 10 p.m. and 5 a.m. when the traffic is low, then the toll is less.

The complete cycle for using OSA service consists of three phases, authentication, service selection and service use. Here we consider the third phase only.

To mark entering and leaving the highway the user location interrogation triggered request is used. To charge the subscriber we have to define two trigger location reporting criteria – one for entering the highway and the other for leaving the highway. Having the assumption of two areas covering the toll gates and considering the two-way traffic through the gates we define both criteria

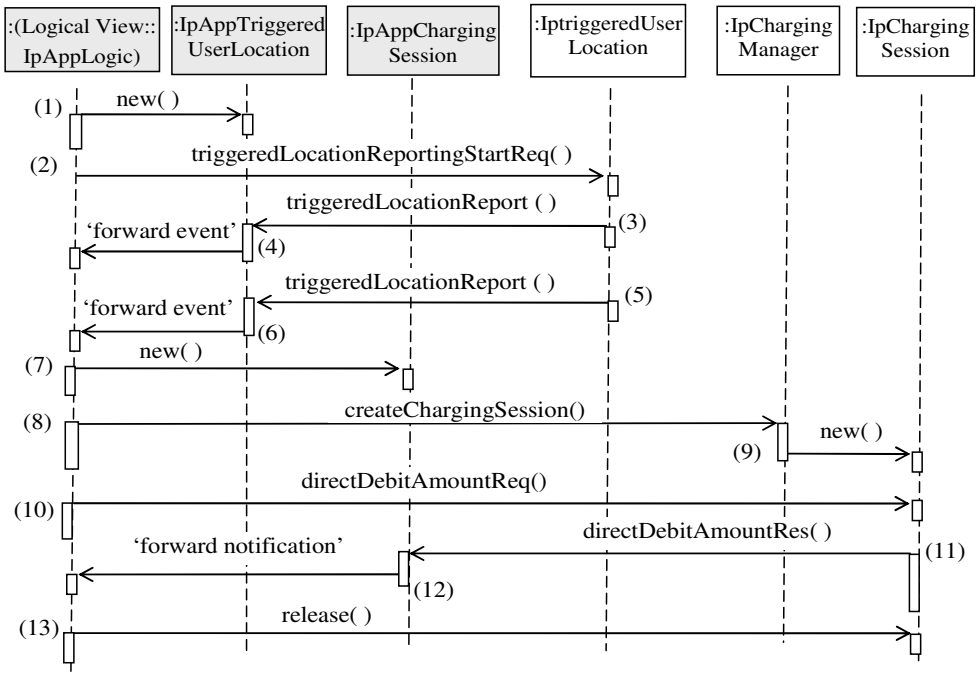


Fig. 3 Sequence diagram for 'Quick toll' service

triggering a location report are set as entering area in the beginning. When the subscriber passes through any of the toll gates, the other data element that defines the criteria triggering a location report is changed to leaving area.

The sequence diagram based on 3GPP TS 29.198-xx Technical Specification for 'quick toll' service is shown in Fig.3. Service scenario includes the following steps:

1. When the service is activated the application sets the criteria triggering a location report and creates a local object implementing the IpAppTriggeredUserLocation interface. This object will receive response messages form IpTriggeredUserLocation object.
2. The application starts triggered location reporting for the subscribers (the company vehicles).
3. When a subscriber enters the highway a triggered location report is passed to the corresponding callback object (the identity of the radio cell might be enough for the mobile payment).
4. The triggered location report is forwarded to the application. Having information for entering the highway, the application sets leaving criteria, checks if the day is Wednesday or it is Thursday or the time is between 10 p.m. and 5 a.m. and determines the toll amount. Then the application waits for a triggered location report for leaving the highway.
5. When the subscriber enters the area of leaving the highway, the triggered location report is sent.
6. The triggered location report is forwarded to the application.
7. The application sets both criteria that specify triggering a location report as entering the highway and creates a local object implementing the IpAppChargingSession interface. This object will receive response messages from the IpChargingSession object.
8. The application orders the creation of charging session.
9. A new object IpChargingSession is created.
10. Then the application requests to charge the user with the toll amount.
11. The payment is acknowledged.
12. The acknowledgment id forwarded to the application.

The application releases all the resources associated with the reservation and session.

Next we present the SLPL service logic script in brief.

### *B. SLPL Service Logic Script*

The framework of the service logic script in SLPL is shown in Fig. 4.

In the definition part variables and methods are defined. The variables include the following: variables which will be used for setting values for actual parameters of methods that belong to network-side interfaces; variables, in which the methods that belong to application-side interfaces will receive results; and local variables for service logic. The methods' definition includes definitions of methods that belong to application-side interfaces such as those depicted in Fig.4 ('triggeredLocationReport' and 'directDebit-AmountRes') and methods like database query handler and database error handler for interaction with an external application database storing data for geographical position of the toll gates (not presented shown in Fig.4).

The executive part consists mainly of method invocations, pending results and processing of data. The first method invoked is 'triggeredLocationReportStartReq' by which the application starts triggered location reporting.

Before method invocation the actual method parameters are set. The first 'wait' statement is used to receive a triggered location report for entering the highway. The criteria triggering location report for leaving the highway are set. The second 'wait' statement is used to receive a triggered location report for leaving the highway. The criteria for entering highway are set. The value of 'theToll' is set to 10 Euros (normal tariff). Then the method 'DayOfWeek' is invoked with an argument CurrentDate and the result is assigned to the variable 'Today'. If the value of 'Today' is equal to 'WE' or 'TH', then the toll is 5 Euros. The value of "theMoment" is set to CurrentTime and if it is between 10 p.m. and 5 a.m., then the toll is 5 Euro. Then the method 'createChargingSession' is invoked to create charging session. The method 'debitAmountReq' is invoked to charge the subscriber account with the toll. The third '*wait*' statement is used to await the result of the payment. The method 'release' is invoked to free all the resources associated with the charging session.

## IV. SLPL Usability Aspects

Language usability depends on availability of language supporting tools – concerning scripting languages this is the interpreter. To achieve portability SLPL interpreter is written in Java. Another reason for Java choice as implementation language is that we use Ericsson Network Resource Gateway SDK (version R5A02) [9] that simulates Parlay/OSA interfaces to verify the SLPL capabilities. The interface method calls of OSA interface simulator are form of Java code.

Data types and methods definitions in the definition part of SLPL service logic script is time consuming and tedious task if done manually. In OSA the structure of data types used in interfaces is composite and usually nest liked and there are multiple arguments in interface methods. Data types and methods in OSA are described in IDL (Interface Description Language). To reduce time and effort in service script development, a translator is developed that translates IDL descriptions in SLPL descriptions. The functionality of that translator is not restricted to OSA IDL descriptions only – whatever IDL description is passed as input, the translator generates the corresponding SLPL description.

As the idea behind the development of SLPL is to be extensible without reprogramming of the translator producing SLPL descriptions, a 'translator generator' is developed i.e. a translator that generates translators. This translator generator is supplied with language grammar rules described in Backus Naur Form (BNF) and as output a Java code is generated that 'understands' language syntax. For example, if we pass BNF rules of IDL and BNF rules of SLPL as input then we have a Java code that makes IDL to SLPL translation as output. Once generated the IDL descriptions of OSA interface data types and methods can be used as interfaces dependant parts in SLPL service logic scripts and they form the SLPL programming library. Further this translator generator produces the Java code of SLPL interpreter also. Thus adding new rules in SLPL grammar that extend or change language expressive power there is no need to rewrite the SLPL interpreter and the translator that translates an IDL description into corresponding SLPL description.

## V. Conclusion

A new mark-up approach to NGN service creation is suggested. The approach is based on OSA interfaces addressing the full range of network capabilities such as call and data session control, mobility, user interaction charging and so on. Language constructions that illustrated approach implementation are defined. OSA APIs provide medium level of abstraction and developers need some telecommunication knowledge when linking components that offer APIs, but as SLPL posses all attractive feature of scripting languages is enables rapid prototyping and rapid application development.

Unlike traditional scripting languages the expressive power of SLPL draws it near to programming languages. SLPL nodes for flow control provide developers with flexibility in logic processing. Special keywords are defined as to describe the time-related types and to facilitate the manipulation of current moment and current date values. SLPL constructions for database manipulation and query enable service logic to consult external database to retrieve service specific data.

Considering the necessity of provisioning easy-to-use and cost effective tools for value-added service creation the SLPL can be regarded as one of ways to NGN intelligence.

## References

[1] Falcarin, P. Licciardi, C.A. (2003) Analysis of NGN service creation technologies, *IEC Annual Review of Communications*, vol. 56.
[2] Bakker, J. Tweedie, D. and Umnehopa, M. (2004) Evolving Service Creation; New developments in network Intelligence, *Teletronikk*
[3] Bakker, J. Jain, R. Next Generation Service Creation Using XML Scripting Languages, www.argreenhouse.com/papers/jlbakker/ bakker-icc2002.pdf
[4] Bakker, J-L., R. Jain, SCML, Next Generation Service Creation Using XML Scripting Languages, www.argreenhouse.com/papers/ jlbakker/bakker-icc2002.pdf (accessed November 2004)
[5] Atanasov I., New XML-based Language for OSA User Interaction Interface Description (2005), *TELECOM'2005*, Varna, Bulgaria, Proceedings pp. 195-200.
[6] Pencheva, E. Atanasov, I. (2005) New XML-based Language for OSA Mobility Interface Description, *TELECOM'2005*, Varna, Bulgaria, Proceedings, pp 201-207.
[7] Atanasov, I. Pencheva, E. (2005) Service creation using a new mark-up language, *TELSIKS'2005,* Nish, Serbia and Monte Negro, Proceedings, pp 575-578
[8] Atanasov I., Pencheva, E. (2005) A new service logic processing language, *ELECTRONICS'2005,* Sozopol, Bulgaria, Proceedings, pp. 150-155, book 1
[9] Ericsson Network Resource Gateway SDK (version R5A0) http://www.ericsson.com/mobilityworld/sub/open/technologies/parlay/tools/parlay_sdk (accessed March 2005)
[10] 3GPP TS 29.198-x, (December 2004) Open Service Access (OAS) Application Programming Interface (API);.
[11] Lennox, J. Wu, X. (2004) CPL: A Language for User Control of Internet Telephony Services, RFC 3880, [WWW document](accessed March 2004)

```
<logic>
  <define>
    <types/>
    <variables>
      <id name="Today" type="Day" />
      <id name="theMoment" type="Time" />
      <id name="theToll" type="Float" />
      <!-- VARIABLES NEEDED FOR METHODS OF 'IpTriggeredUserLocation', 'IpIpChargingManager',
           'IpChargingSession' INTERFACES AND THEIR CALLBACKS-- >
    </variables>
    <methods>
      <!-- MOBILITY 'IpAppTriggeredUserLocation::triggeredLocationReport'  -->
      <!—CHARGING 'IpAppChargingSession::directDebitAmountRes'  -->
    </methods>
  </define>
  <execute>
    <!-- FRAMEWORK AUTHENTICATION -->
    <!--   1. SET PARAMETERS.
           2. INVOKE thiggeredLocationReportingStartReq -->
    <wait/>
    <!--  RECEIVE TRIGGERED LOCATION REPORT FOR ENTERING THE HIGHWAY -->
    <!--  SET LEAVING AREA CRITERIA -->
    <wait/>
    <!--  RECEIVE TRIGGERED LOCATION REPORT FOR LEAVING THE HIGHWAY-->
    <!--  SET ENTERING AREA CRITERIA -->
    <set refid=”theToll” val=”10”/>
    <invoke>
      <method name=”DayOfWeek”>
        <arguments>
          <argument name=”a_date”>
            <value>
              <CurrentDate/>
            </value>
          </argument>
        </arguments>
        <returns> <set refid=”Today”/> </returns>
      </method>
    </invoke>
    <case refid=”Today”>
      <on val=”WE”>
        <set refid=”theToll” val=”5”/>
      </on>
      <on val=”TH” >
        <set refid=”theToll” val=”5”/>
      </on>
    </case>
    <set refid=”theMoment”>
       <value>
         <CurrentTime/>
       </value>
    </set>
    <if>
       <condition test=”theMoment BTW 22:00 AND 5:00”/>
       <then>
          <set refid=”theToll” val=”5”/>
       </then>
    </if>
    <!-- SET PARAMETERS AND INVOKE 'createChargingSession' METHOD -->
    <wait/> <-- RECEIVE NOTIFICATION FOR SESSION CREATION -->
    <!-- SET PARAMETERS AND INVOKE 'directDebitAmountReq' METHOD -->
    <wait/> <-- RECEIVE ACKNOWLEDGEMENT FOR PAYMENT-->
    <!-- SET PARAMETERS AND INVOKE 'release' METHOD -->
    <wait/>
  </execute>
</logic>
```

Fig. 4 Simplified service logic script for 'quick toll'