COMPARATIVE ANALYSIS ON MARK-UP SERVICE CREATION LANGUAGES

Ivaylo Ivanov Atanasov, Evelina Nikolova Pencheva

Department of Telecommunications, Technical University of Sofia, 7 Kliment Ohridski st., 1000, phone: +359 2 965 2050, e-mail: iia@tu-sofia.bg, e-mail: enp@tu-sofia.bg

The paper presents a comparison between existing service creation mark-up languages and a new one that supports Parlay/Open Service Access (OSA) Application programming interfaces (APIs). The language is platform, network and technology independent, easy to use and possesses flexibility and expressiveness of programming languages. In addition to call control the language allows service logic to derive added value from network functions such as mobility, user interaction, data session control, presence and availability and others.

Keywords: Open Service Access, markup languages, CPL, VoiceXML

1. INTRODUCTION

Services in next generation networks (NGN) is expected to generate the greatest revenue, so it is important for network operators and application developers to have the right service creation environment. This environment will provide tools for easy and quick service development reflecting requirements for customised services. Scripting languages proved to be promising tool in NGN service creation. They are lightweight, customisable and typically interpreted languages.

Available scripting languages such as CPL and VoiceXML possess restricted processing power and are call-centric. None of the existing scripting languages supports the whole variety of network functions accessed through Application Programming Interfaces (APIs). In this paper we present in brief a new mark-up approach to NGN service creation based on Parlay /OSA (Open service access) APIs. Service Logic Processing Language (SLPL) combines attractive features of eXtensible Mark-up Language (XML)-based languages and flexibility and expressiveness of programming languages. We compare SLPL service creation features with those of the existing standardized markup languages. An example is presented to highlight the SLPL benefits.

2. STANDARDIZED MARK-UP LANGUAGES FOR SERVICE CREATION AND SLPL

CPL is a language for call processing in Session Initiation Protocol (SIP)-based telephony network. CPL tells a SIP server what to do with a call. SLPL is oriented towards OSA APIs and is protocol agnostic. OSA APIs provide application developers with programmability of network resources such as telecommunication protocol stacks by defining these resources in terms of objects and methods, data types and parameters that operate on those objects hiding network specificity.

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

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

CPL can be used as a high-level service-creation language with restricted expressive power. It allows end users to define their call-processing preferences over the SIP-based network. SLPL on contrary supports means for flow control which draw the language near to programming languages. Flow control statements such as 'if'-statement, 'case'-statement and 'while'-statement provide flexibility in service logic description. 'If'-statement is used to check a logical condition, with 'case'-statement decision may be done based on multiple choices and 'while'-statement is used when a set of iterations has to be repeated till a predefined condition is true.

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. Time measuring and supervision means in CPL and SLPL semantically are overlapped i.e. both languages provide constructions handling with intervals and time. But while in CPL time handling capabilities are presented in a declarative form, in SLPL time constructions are rather procedure-oriented. Usually it is said, that the declarative form is easier to use while the procedure-oriented presentation provides more flexibility and expressive power.

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

In SLPL to handle with notion of "real" time predefined types 'Date', 'Time' and 'Duration' based on the type float are introduced. Two operations 'CurrentDate' and

'CurrentTime' are used to acquire the value of the current time. Also, the concept of timer is adopted as a predefined type. A timer instance is an object that can be active or inactive. When an inactive timer is set using operations 'SetTime' or 'SetDate', a 'Time' or 'Date' value is associated with the timer respectively. When an active timer is reset by 'Reset' operation, the associated value is lost. Before the first setting of a timer instance it is inactive. In many applications the decisions are made on the days of week so the enumerated type 'Day' is defined and is used to denote the days of week.

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

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

3. AN EXAMPLE OF SERVICE DESCRIBED IN SLPL

Let us consider an application that sends greetings by phone on demand. Imagine "Happy birth day" company that receives requests by clients and sends greetings by

phone (e.g. musical or voice greetings on requested dates with the appropriate content for the occasion). On receiving a request a record is created in a database. The application looks up daily in the database and retrieves the set of records corresponding to the current date. If there are some records then the service logic creates a call to the particular number and plays the greeting.

The sequence diagram for 'greetings by phone' application is shown in Fig.1.

The example is included to demonstrate some more advanced features of SLPL in comparison with CPL and VoiceXML. We will not consider the part of the application that receives requests for greetings but just the script that initiates a call (steps 1-3), routes it to the end user (steps 4-6), plays the greeting (steps 7-12) and releases the call (steps 13, 14). As the script is not activated by an originating or terminating fragment of call, the script has to explicitly reference the OSA Call control API. To play the greeting the script uses OSA User interaction APIs.

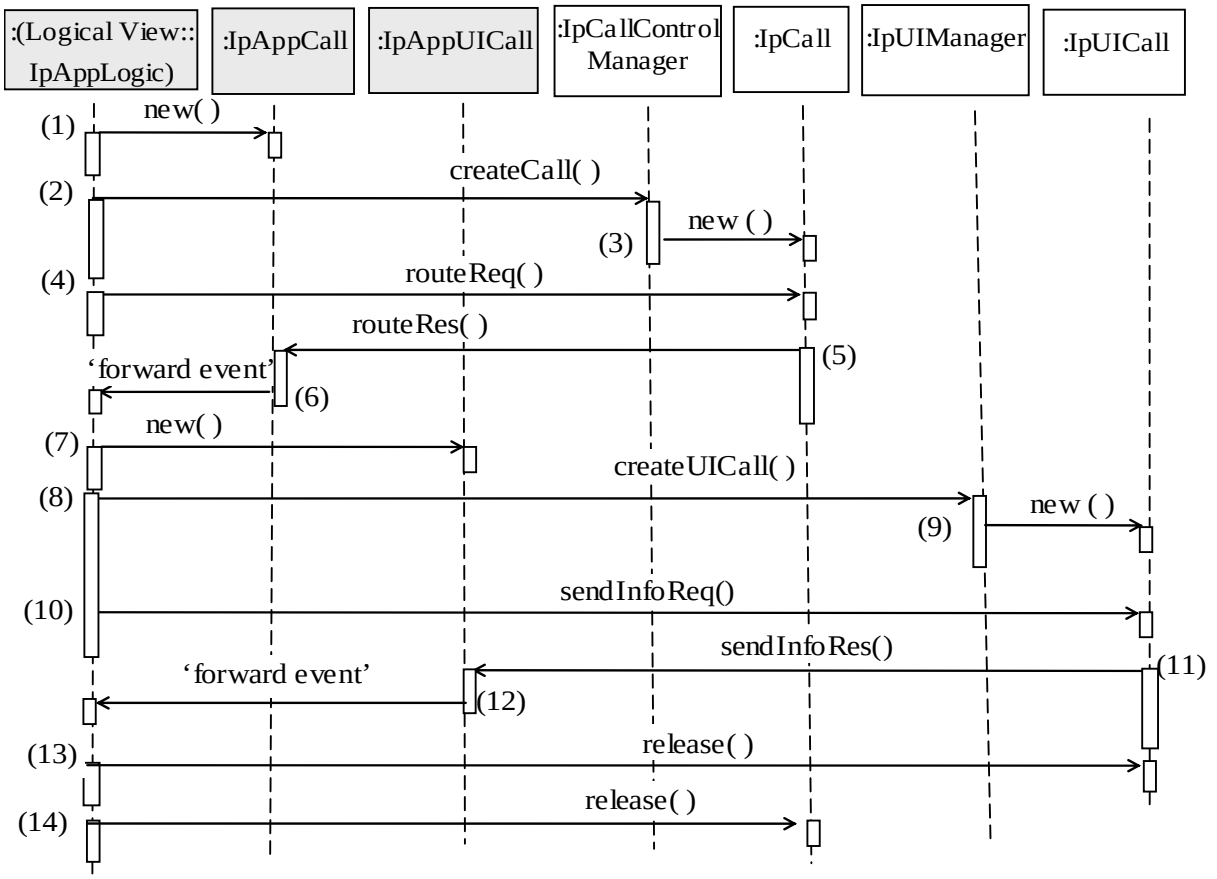

Figure 1 Sequence diagram for 'greetings by phone' application

The simplified service logic script in SLPL is shown in Figure 2. Local types, variables and methods are defined in the definition part of the script.

```
<logic>
  <define>
    <types>
      <structure name="DB_record"> <!--a record in query result-->
        <element name="The_date" type="Date"/> <!-- date in query result-->
        <element name="Phone_num" type="TpInt32"/> <!-- phone num in query result-->
        <element name="greetingID" type="TpString"/> <!-- ID of greeting-->
      </structure >
      <sequence name="DB_records" item_type="DB_record"/>
      <structure name="DB_result"> <!-- DB query result-->
        <element name="SetOfRecords" type="DB_records"/> <!-- records in query result-->
        <element name="numRec" type="TpInt32"/> <!-- records num in query result-->
      </structure>
    </types>
    <variables>
      <id name="a_DB_res" type="DB_result"/> <!-- a database query result -->
      <id name="daily" type="Duration" val="1" /> <!-- DB look up period -->
      <id name="T_greetings" type="Timer" /> <!-- timer for DB look up period-->
      <id name="theSQLquery" type="TpString" /> <!-- SQL statement-->
      <id name="theSQLresult" type="TpString" /> <!-- SQL result-->
      <id name="DB_address" type="TpSrting"/> <!-- address of database server-->
      <id name="aDate" type="Date"/> <!--current date-->
    </variables>
    <methods> <!-- 'IpAppLogic::DB_retrieve', 'IpAppLogic::DB_conversion,
     ''IpAppCall::routeRes''IpAppUICall::sendInfoRes','IpAppCall::callEnded'--></methods>
  </define>
  <execute>
    <!-- FRAMEWORK AUTHENTICATION -->
    <while test="true">
      <set refid="aDate"> <value> <CurrentDate/> </value> </set>
      <!-- 1.SET VALUE TO 'theSQLquery'   2.SET VALUE TO 'DB_address"-->
      <!—INVOKE 'DB_retrive' -->
      <wait/> <!-- SET VALUE TO theSQLresult -->
      <!—INVOKE 'DB_conversion' -->
      <wait/> <!-- SET VALUE TO a_DB_res -->
      <if> <condition test="numRec EQ 0" /> <!-- no retrieved records -->
        <then> <goto label="skip_day"/> /then></if>
      <while test="a_numRec GT 0"> <!-- while there are retrieved records -->
         <decrease refid="a_numRec " by="1" />
         <!-- 1. SET PARAMETERS. 2. INVOKE 'IpCall::createCall'
              3. INVOKE 'IpCall::routeReq' 4. WAIT 'IpAppCall::routeRes'
              5. INVOKE 'IpUICall::sendInfoReq' 6. WAIT 'IpAppUICall::sendInfoRes'
              7. INVOKE 'IpUICall::release' 8. INVIKE 'IpCall::release' -->
      </while>
      <label name="skip_day"/>
      <invoke><method name="SetDate">
                <arguments>
                   <argument name="a_date" valref="aDate"/>
                   <argument name="a_dur" valref="daily"/>
                   <argument name="a_timer" valref="T_greetings"/>
                </arguments> <returns/> </method>
      </invoke><wait/>
    </while>
  </execute>
</logic>
```

Fig.2 Skeleton of 'greetings by phone' service logic described in SLPL

The database record type is defined as a structure of 3 elements: the date of greeting, the phone number to be dialed and the greeting ID to be played. A type of set of database records is defined as a numbered set of data elements. The database query result type consists of the number of records in the query result and the set of database records. Variables of the defined types are declared. A variable of type Duration represents the database look up interval and this interval is supervised by the use of variable of type Timer. The local methods are defined also. The executive part of the service logic script is organized as an endless loop. First, the current date is yielded. Then a SQL statement which retrieves all database records with date of greeting equal to the current date is created and the method "DB_retrieve" is invoked. The returned result is used as an argument in method "DB_conversion" invocation to retrieve the number of retrieved records and the set of these records. For each of the retrieved records a call is created and routed to the phone number in the record, and the greeting is played. When all of the retrieved records are browsed, a timer is set with value equal to database look up period.

5. CONCLUSION

In this paper a new markup approach to service creation is presented. Following this approach the service logic can derive added value from network functionalities accessible through APIs. The service logic is described by the use of an XML-based language SLPL that is platform independent, lightweight and suitable for 3rd party application development. On contrary to the other markup languages SLPL allows control over call-unrelated events such as change of position, user status, TCP/IP session, presence and availability and so on. SLPL follows closely the architecture and APIs definitions developed by Parlay/OSA. In addition the language provides means for database access and time measuring and supervision. These SLPL features make it functionally richer and superior to the other markup languages proposed for service creation.

REFERENCES

[1] Bakker J-L, D.Tweedie and M. Umnehopa, *Evolving Service Creation; New developments in network Intelligence*, Teletronikk, 2004

[2]Bakker J, R. Jain, *Next Generation Service Creation Using XML Scripting Languages*, www.argreenhouse.com/papers/jlbakker/ bakker-icc2002.pdf

[3]I. Atanasov , *New XML-based Language for OSA User Interaction Interface Description*, TELECOM'2005, Varna, Bulgaria, Conference proceedings.

[4]Pencheva E., I. Atanasov, *New XML-based Language for OSA Mobility Interface Description*, TELECOM'2005, Varna, Bulgaria, Conference proceedings.

[5]Atanasov I., E. Pencheva, *Service creation using a new mark-up language*, TELSIKS'2005, Hish, Serbia and Monte Negro, Proceedings, pp 575-578

[6] Atanasov I., E. Pencheva, *A new service logic processing language*, ELECTRONICS'2005, Sozopol, Bulgaria, Proceedings.