Simulation of Next Generation Network Applications

Ivaylo I. Atanasov[1]

***Abstract* – The paper presents a way for functional validation of an approach to service creation. The essence of the approach is synthesis of a new mark-up language used to describe service logic. An application that uses network functions accessed through open programming interfaces is designed. The application logic is described using the proposed mark-up language and the application behaviour is simulated by the use of network resource gateway simulator.**

***Keywords* – Mark-up languages for service creation, Parlay/OSA interfaces, network resource gateway simulator**

I. INTRODUCTION

Next generation networks (NGN) are expected to provide content rich and customized services. To be competitive network operators need to have not only the right technology, but the right tools to create these services efficiently.

An important ingredient of NGN service architecture is the concept of Application Programming Interfaces (APIs) that allows third parties to get in on developing services for telecommunication networks, with transportable suppliers independent skills [1]. A promising technology providing access to network function through open API is Parlay/OSA (Open service access).

One of the ways of implementing Parlay/OSA is by the use of mark-up languages. There exist some markup languages for service creation [2], but none of them supports the full range of network functions exposed by Parlay/OSA.

Service Logic Processing Language (SLPL) is a new mark-up language developed to meet challenges of NGN service creation [3]. The language supports the whole palette of network functions exposed by Parlay/OSA interfaces. To allow language usability an SLPL interpreter is developed. The SLPL functionality is validated by simulation.

First in the paper, the network resource simulator used is presented. Then an example that uses network functions accessed through Parlay/OSA interfaces is considered. The suggested mark-up language SLPL is presented in the context of the application logic description. Last, the process of simulation of the application functional behaviour is explained.

II. NETWORK RESOURCE GATEWAY SIMULATOR

To validate the approach applicability the Ericsson Network Resource Gateway (NRG) simulator is used [4]. The NRG simulator emulates NRG node and its purpose is to test applications in absence of an actual NRG running on a live telecom network.

The NRG product is Service Capability Server, which provides network services to applications in a secure way and controls the network elements. Beneath the service capability server is the telecom network which uses a wide range of network protocols to implement the services in the network.

As it is shown in Fig. 1, the NRG provides a set of APIs independent from underlying network.

The NRG product includes Software Development Kit (SDK) also. The SDK offers software libraries to Java application developers, which simplify the development of applications that use NRG capabilities.

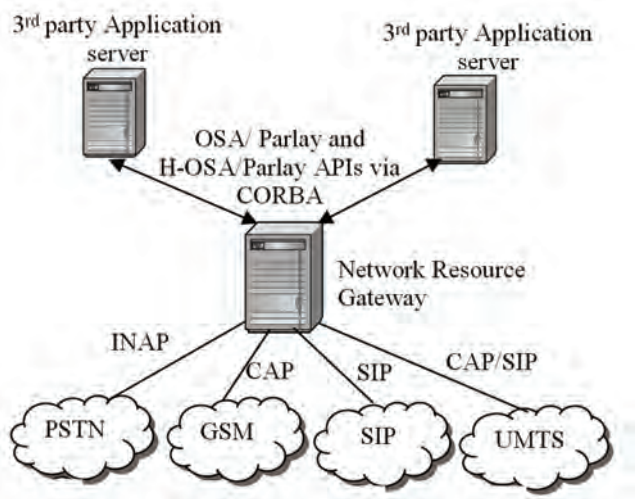

Fig. 1 Ericsson Network Resource Gateway

As it is shown in Fig. 2, by using the NRG the developers no longer need detailed knowledge of telecom network in order to use its services and with the help of SDK the need for detailed Common object request broker (CORBA) knowledge in order to use NRG is removed as well.

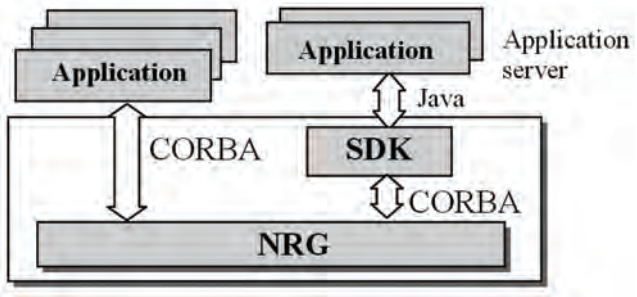

Fig. 2 An overview of NRG SDK

The NRG simulator mimics the behavior of an actual NRG node. It simulates not only the node but also the underlying

[1]Ivaylo I. Atanasov is with the Faculty of Communications and Communication Technologies, Technical University of Sofia, 1000 Sofia, Bulgaria, E-mail: iia@tu-sofia.bg

network resources (such as telecommunication network). It uses a graphical interface that allows a user to define the behavior of network resources.

The NRG SDK provides API for several services including framework, multi-party call control, user interaction, user location user status and messaging. Before an application can get access to a service, the application needs to authenticate itself towards the NRG using the framework API.

Most applications follow a similar design involving the following classes:

- Main – initializes, starts, stops and terminates the application
- Configuration – provides application with configuration data
- Graphical user interface (GUI) – provides a graphical user interface
- Feature – implements application logic
- YY_processors – use service managers to send requests to the NRG and to receive callback responses.

A feature (application) can have multiple processors.

In the next section an example of application that uses SDK APIs is considered.

III. Case Study

Let us consider a "Local entertainments" application that provides information about places of entertainment in a city. When a user dials the number of "Local entertainments" service, the application locates her position and asks the user about requested information for example, restaurants, discos or pubs. After the user enters her choice, the message is played containing information about the places of entertainments in the vicinity. The information about local places of entertainment is retrieved from a database.

The application uses the following classes available in NRG SDK:

- MPCCProcessor – a multi-party call control processor that uses service manager IpMPCCManager
- LocationProcessor – a processor used to determine the user position
- UIProcessor – a processor for user interaction that uses service manager IpUIManager
- IpMPCCManager – an interface for receiving results and notifications from IpMPCC interface
- IpUIManager – an interface for receiving results and notifications from IpUICall interface
- IpMPCC interface offering methods for multi-party call handling
- IpUserLocation interface offering methods for user location
- IpUICall interface offering methods for user interaction.

Further for the aim of the application two more classes are developed:

- DBProcessor – a database processor
- IpDataBase interface offering methods for database access.

The sequence of actions performed during application execution is shown in Fig. 3.

1 After the user dials the service number, IpMPCCManager notifies MPCCProcessor about the event.
2 The event is forwarded to the application.
3 The application requests user location.
4 The LocationProcessor sends a request for positioning to the user location service.
5 The user location service provides requested information.
6 The information about user location is forwarded to the application.
7 The application requests from the database identifications of the voice messages containing information about places of entertainment. The user coordinates are sent as parameter.
8 The DBProcessor calls DB_retrieve method to extract the requested data.
9 The result of database query is forwarded to the application. The result contains message IDs for places of entertainment.
10 The application starts dialogue with the user.
11 User interaction session is created.
12 The application requests from the UIProcessor to prompt the user about the requested entertainment type by entering digit.
13 UIProcessor requests from the user interaction service to prompt and collect information.
14 The user's choice is sent to the application.
15 The application sends to the UIProcessor the identification of the message to be sent.
16 UIProcessor requests from the user interaction service to play the message.
17 When the message ends this is reported to the UIProcessor.
18 The application requests to release user interaction session.
19 The UIProcessor frees the user interaction service.
20 The application deallocates the call-related resources.
21 The multi-party call control service is released.

IV. SLPL Description of Application Logic

The SLPL application logic description consists of definition and executive parts. In the definition part data types and methods are defined, and variables are declared.

The SLPL supports all data types defined in OSA specifications. Fig. 4 shows the definition of data type that represents the result returned by the database which contains the ID of the menu to be played and message IDs of the entertainment places in the vicinity.

Fig.5 shows variables used in the application script.

The application methods are also defined. When the user dials the "Local entertainments" service number the application is notified. The method "handleCall" handles incoming calls to the service, the method "locationReceived" is used to receive the geographical coordinates, and the method "locationTranslated" is used to get the database response in form of messages IDs corresponding to the geographical coordinates.

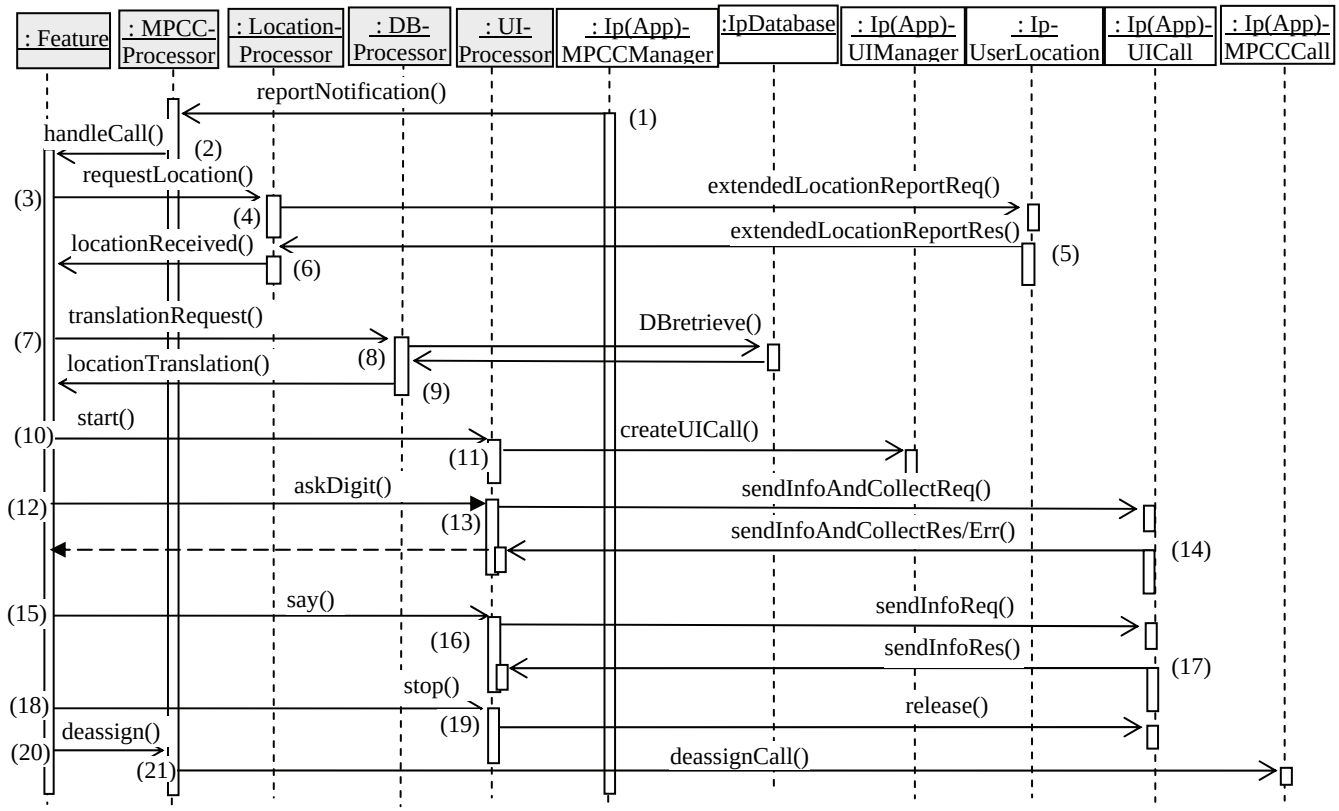

Fig. 3 Sequence diagram for "Local entertainments" applicationVariables of the types defined are declared.

```
<types>
    <structure name="PlaceDescription">
      <element name="menu" type="integer"/>
      <element name="Restaurants" type="integer"/>
      <element name="Discos" type="integer"/>
      <element name="Pubs" type="integer"/>
    </structure>
</types>
```

Fig. 4 An example of SLPL definition of structure type

```
<variables>
      <id name="theLongitude" type="float"/>
      <id name="theLatitude" type="float"/>
      <id name="thePlacesID" type="integer"/>
      <id name="theUICall" type="TpUICallIdentifier"/>
      <id name="thePlaces" type="PlaceDescription"/>
      <id name="digit" type="integer"/>
      <id name="menuID" type="integer"/>
</variables>
```

Fig. 5 An example of SLPL variable declaration

Fig.6 shows the definition of the method "handleCall" and the method "locationTranslated".

The executive part of the service logic script is built of statements. To request a service from the network-side of interface, the application logic has to invoke its methods. On method invocation its interface is specified and actual value of its argument is given. To allow synchronous communications the application has to wait for the result of network services. By invocation of application's methods, the network-side interfaces return the results of the requested service. "Case-statement" is used for multiple choices.

```
<method name="handleCall">
  <arguments>
    <argument name="aCall"
      type="TpMultiPartyCallIdentifier"/>
    <argument name="aLeg" type="TpCallLegIdentifier"/>
    <argument name="anOriginatingAddress"
      type="TpAddress"/>
  </arguments>
  <returns/>
  <body>
    <set refid="theCall" valref="aCall"/>
    <set refid="theLeg" valref="aLeg"/>
    <set refid="theOrigAddress"
      valref="anOriginatingAddress"/>
  </body>
</method>
<method name="locationTranslated">
  <arguments>
    <argument name="PlaceIDs" type="PlaceDescription"/>
  </arguments>
  <returns/>
  <body>
    <set refid="thePlaces" valref="PlaceIDs"/>
  </body>
</method>
```

Fig. 6 An example of SLPL method definition

First the application waits for call notifications. On call notification, the method "locationRequest" is invoked to receive user location. The application waits for user location data. On receiving user location, the method "translation-Request" is invoked to query database about message IDs for entertainments in the vicinity. The application waits for database response. To start interaction with the user the

method "start" is invoked. The method "askDigit" is invoked to prompt the user about her choice. The method "say" is invoked to play the message.

Fig. 7 shows the skeleton of the executive part of the script.

```
<execute>
  <wait/>
  <!—invoke method LocationProcessor.locationRequest -->
  <wait/><! – wait for user location data -->
  <!—invoke method "DBProcessor.translationRequest" -->
  <wait/> <! -- the DB result is received in variable thePlaces -->
  <set refid="theUICall">
    <invoke>
        <method name="itsUIProcessor.start">
          <arguments>
            <argument refid="theCall"/>
          </arguments>
        </method>
    </invoke>
  </set>
  <set refid="menuID" valref="thePlaces">
    <value><element name=”menu”/></value>
  </set>
  <set refid="digit">
    <invoke>
      <method name="itsUIProcessor.askDigit">
        <arguments>
          <argument name="aMsgID" refid="menuID"/>
        </arguments>
      </method>
    </invoke>
  </set>
  <case refid="digit">
    <on val="1">
      <set refid=”thePlaceID" valref="thePlaces">
        <value><element name="Restaurants"/></value>
      </set>
    </on>
    <on val="2">
      <set refid="thePlaceID" valref="thePlaces">
        <value><element name="Discos"/></value>
      </set>
    </on>
    <on val="3">
      <set refid="thePlaceID" valref="thePlaces">
        <value><element name="Pubs"/></value>
      </set>
    </on>
  </case>
  <!-- invoke method itsUIProcessor.say -->
  <!-- invoke method itsMPCCProcessor.deassign -->
</execute>
```

Fig. 7 Skeleton of executive part of SLPL script

V. Simulation of Application Logic

The SLPL interpreter is registered in the Framework of the NRG. When simulating the application behavior, the interpreter is in the role of an object of class Feature as depicted in Fig.3. The SLPL interpreter is supplied with the SLPL description of "Local entertainments" application.

When the "Local entertainments" application is started, the SLPL interpreter verifies the syntactical correctness of the description, generates an abstract syntax tree and makes the respective mappings of the abstract syntax tree onto the semantics i.e. calls corresponding Java methods. The calls of the interpreter are of methods exposed by the Ericsson NRG simulator.

In runtime on request of the user of the "Local entertainments" application, the current location is obtained and displayed in a map. When dialing the service number, an announcement is played asking for entering digit. After the user enters a digit, a message with requested information is played.

Fig. 8 shows a screenshot of service simulation

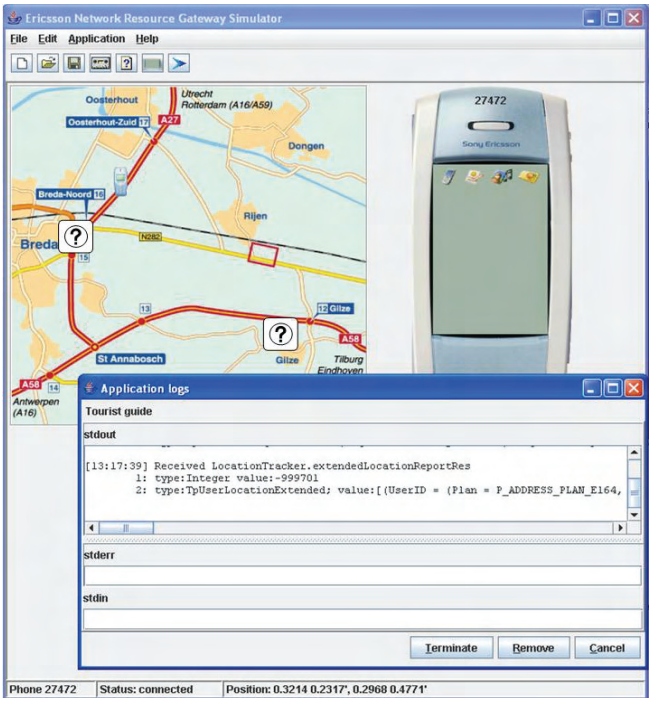

Fig. 8 A screenshot of "Local entertainments" application simulation

VI. Conclusion

In this paper an example of Parlay/OSA application is considered. The application logic is described using a new markup language SLPL. The SLPL functional applicability is verified by simulation.

References

[1] Bakker, J. Tweedie, D. and M. Umnehopa, “Evolving service Creation; New developments in Network Intelligence”, http://www.argreenhouse.com/papers/jlbakker/Bakker-telenor.pdf

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

[3] Atanasov I., E. Pencheva, “A Mark-up Approach to Add Value”, IJIT Enformatika, vol.3, Number 4, pp 267-276

[4] Ericsson Network Resource Gateway SDK (version R5A02) http://www.ericsson.com/mobilityworld/sub/open/technologies/parlay/tools/parlay_sdk