

A MARK-UP APPROACH TO SERVICE CREATION

Ivaylo Atanasov, e-mail: iia@tu-sofia.bg, phone: +359 2 965 20 50
 Evelina Pencheva, e-mail: enp@tu-sofia.bg, phone: +359 2 965 22 54
 Technical University of Sofia, 8, "Kliment Ohridsky" blvd, 1000 Sofia, Bulgaria

Abstract – The paper presents a new mark-up approach to service creation in Next Generation Networks. The approach allows access to network functions exposed by open application programming interfaces. Based on ontology analysis of the application domain, language constructions are synthesized and formally defined. Language supporting tools are developed. The approach functionality is tested by simulation.

Keywords – NGN service creation, Parlay/OSA interfaces, mark-up languages.

I. INTRODUCTION

The services and applications in Next Generation Networks (NGN) are expected to generate new revenue stream combining voice, video, multimedia, and data communications, possibly involving mobile terminals and multiple parties over multiple types of transport. Provisioning of a common platform for service and application development opens the telecommunication networks of the future up to a wealth of developers and freedom.

An important ingredient of open service platform 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]. This will greatly reduce the cost of development and it is hoped nurture innovation.

The APIs provide application developers with programmability of resources such as telecommunication protocol stacks and databases, 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 and protocol complexity.

Initiatives like Parlay/OSA (Open service access) [2] and JAIN (Java applications for intelligent networks) [3] overcome the service portability barrier by providing a common API framework delivering convergence though the implementation of multiple protocol stacks. The JAIN framework allows implementation of softswitch functionality on any machine that has Java runtime engine installed. The third-generation (3G) mobile world has defined OSA framework which allows third party to be plugged into

mobile networks. Although intended for UMTS networks, the principles of OSA are applicable in the NGN domain as well. Both JAIN and the OSA initiatives have been influenced by Parlay specifications.

The aim of Parlay/OSA is to provide a network independent application development environment by specification of APIs (Fig. 1). The Parlay/OSA APIs are programming language independent and cover the following functionalities: call control, data session control, mobility, user interaction, terminal capabilities, connectivity management, messaging and charging. The applications gain access to network functions through the framework API that covers aspects of security and management.

While those APIs can be implemented by programming languages, such as C++ and Java, eXtensible Markup Language (XML)-based languages possess many attractive features making them applicable to NGN service creation [4, 5]. XML and its derivatives found their way into almost every facet of telecommunications from service provisioning through billing records and network management systems and even scripting languages for automated voice services. XML-based 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.

Several XML-based languages have invaded the telecommunications network and some of them are intended for service creation: Call processing language (CPL) [6], Voice extensible markup language (VoiceXML) [7], Call control extensible markup language (CCXML) [8], and Service creation markup language (SCML) [9]. These languages can be used to create applications combining network functions mainly for call control and user interaction, but *none of them supports the full set of network capabilities exposed by Parlay/OSA APIs.*

In this paper we present a new XML-derived approach to NGN service creation. The approach is based on Parlay/OSA APIs and thus supports the whole palette of network functions. To illustrate the approach applicability we have defined a *new markup language* called *Service Logic Processing Language* (SLPL) and developed language supporting software tools.

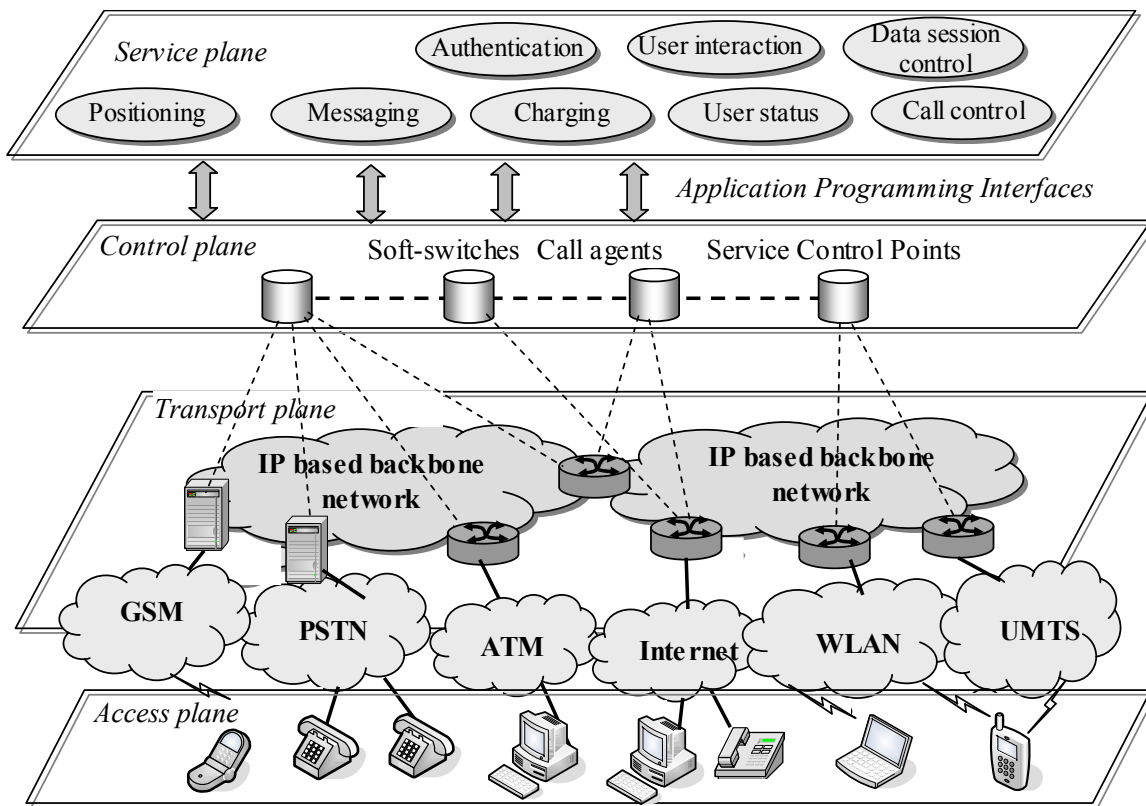


Fig. 1 NGN service deployment

These software tools include the following: a language interpreter that makes lexical and syntactical analysis of service logic description and interprets resulting abstract syntax tree; and a translator generator that processes formally described language grammar rules and generates source code that "understands" these rules. To test the approach functionality we have used a simulation environment of Ericsson's Parlay/OSA gateway called Network Resource Gateway (NRG) simulator.

The NRG simulator [11] is a utility for those wanting to develop a Parlay/OSA application without having access to a real NRG or a live telecom network. We have designed Parlay/OSA enabled applications described in SLPL. Application logic scripts are interpreted by the SLPL interpreter and their functionality is simulated by the use of the NRG simulator.

First in the paper we point out the principles of language synthesis, describe the overall structure of the SLPL application logic script, and present some of SLPL language constructions. Then we evaluate the approach according defined criteria. Some aspects of SLPL supporting software tools are discussed. Last we give an example of Parlay/OSA application which functionality is tested by the NRG simulator.

II. LANGUAGE SYNTESIS

The language used to describe NGN services has to reflect the objects and their relationships in the application domain under given constraints. Ontology analysis is used to extract the language requirements and to reason about the language constructions in the domain of service creation. The NGN applications described by the language have to access network functions without specific telecommunication knowledge using standard APIs.

Parlay/OSA interfaces are defined as a set of object types (classes). An interface defines the methods that can be called on the objects and their parameters. Consequently, the language has to provide constructions for data type and method definition. To access network function the application has just to invoke the object methods with actual parameters, so language constructions for method invocation and synchronization are needed. Exceptions can be thrown by any method, so the language has to provide means for exception capturing and processing. Any result returned by the method has to be processed in the context of application logic and the language has to offer flow control means and simple arithmetic operations. To allow time-dependent processing the language needs time-related data types and methods. In NGN data plays an essential role in almost all services.

Many services become increasingly data-centric and customized, so applications are to access databases to retrieve data they require.

Based on the domain analysis, SLPL language constructions are derived and include the following:

- Description of data types, supported by OSA interfaces and definition of variables of supported data types (statements for data types and variable definitions)
- Description of Parlay/OSA interface and application methods with parameters, type of result returned and exceptions that might be thrown (statements for interface and method definitions)
- Invocation of OSA interface methods to access network functions (invoke-statement)
- Capturing exceptions to define exception processing (try-catch-statement)
- Method synchronization to provide synchronous communications (wait-statement)
- Flow control to provide decision making, multiple choice, action reiteration etc. (if-statement, while-statement, case-statement)
- Elementary arithmetic operations to allow applying of simple algorithms (addition, subtraction, multiplication, division)
- Tools for time measuring and supervision to allow time-based decision making (timers, operation to extract time and date)
- Database access to allow retrieve, update, insert and delete data in external databases (methods for data retrieve, update and conversion).

In the next section we present some of these language constructions.

III. SERVICE LOGIC SCRIPT STRUCTURE

The structure of SLPL service script is modular and encompasses definition part and executive part. The definition part involves type definitions, variable definitions and definition of the methods supported by the application-side interfaces. The executive part is built mainly of invocations of methods supported by the server-side interfaces and processing of results returned.

SLPL supports all data types defined in Parlay/OSA interface specifications. Data types are simple (such as integer, float, string, reference and so on) and complex (enumerated type, structure type, sequence of data elements, tagged choice of data elements). Fig.2 shows an example of type definition of a tagged choice

Variables of the supported types can be defined and initialized.

```
<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.2 SLPL description of a tagged choice data type

The methods' definition part encompasses definitions of all application's methods. Usually these are methods used by network resources to report to the application results of service provided and local methods. Each method has a type (the result type), a list of arguments, and a list of exceptions that it can raise. The method body is used to get the results returned by network resources and to store them in local variables. Fig.3 shows an example of method definition in SLPL.

```
<method name="locationReportReq">
  <arguments>
    <argument name="appLocation" type="
      IpAppUserLocationRef"/>
    <argument name="users" type="TpAddressSet"/>
  </arguments>
  <returns>
    <return type="TpAssignmentID"/>
  </returns>
  <raises>
    <exceptions>
      <exception type="TpCommonExceptions"/>
      <exception name="
        P_APPLICATION_NOT_ACTIVATED"/>
    </exceptions>
  </raises>
</method>
```

Fig.3 SLPL method definition

The executive part of the service logic script is built of statements. There are different types of statements such as: statements for flow control, assignment statements, invocations of methods supported by the network-side interfaces, statements for synchronization and others. Before a method of network-side interface is to be invoked, values for the actual method arguments have to be assigned using assignment-statement.

To request a service from network-side interface, the application logic has to invoke its methods. When

a method is invoked, its interface is specified and actual values of its argument are given. In order to be complete, there are also constructions dealing with exceptional situations when invoked method reports for an error. Fig.4 shows an example of method invocation with capturing of exceptions.

```
<try>
  <invoke>
    <interface name="IpUserLocation">
      <method name = "locationReportReq">
        <arguments>
          <argument name = "application" valref =
            "theApplication"/>
          <argument name = "users" valref =
            "theUsers"/>
        </arguments>
      </method>
    </interface>
  </invoke>
  <catch>
    <exception name
      ="P_APPLICATION_NOT_ACTIVATED">
      <exit/>
    </exception>
  </catch>
</try>
```

Fig.4 An example of method invocation with exception capturing in SLPL

When the application has to wait for the result of method invocation (synchronous communications) the wait-statement is used. By invocation of application's methods, the network-side interfaces return the results of the requested service.

Flow control statements such as if-then-else-statement, case-statement and while-statement are used to process results returned. If-then-else-statement is used to check a logical condition and with case-statement a 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. Fig.5 shows an example of repeating actions in SLPL.

SLPL supports also simple arithmetic operations such as addition, subtraction, multiplication and division.

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. Two parameterless operations are provided to acquire current data and current time. When applied, the 'CurrentDate' will yield a value of the type 'Date',

which reflects the current date when it was applied. In like manner when applied, the 'CurrentTime' derives the current time in a value of the type 'Time'. The predefined types 'Date' and 'Time' are based on the type float. In many applications decisions are made on the days of week, so the enumerated type 'Day' is defined and is used to denote the days of week.

```
<set refid="Tries" value="3"/>
<while test = "Tries GT 0">
  <decrease refid="Tries" by="1"/>
  <!--invoke sendInfoAndCollectReq method of
    IpUICall interface to receive user PIN
    code in PININ-->

  <if>
    <condition test ="PIN EQ PININ"/>
    <then>
      <set refid="PIN" value="true"/>
      <goto label="CorectPIN"/>
    </then>
  </if>
</while>
```

Fig.5 An example of repeating actions in SLPL

Delays and supervisions are obtained by the use of 'Timer', which also is a predefined type. All timers used by service logic have to be declared in the same way as other data objects. The two operations that can be applied to these objects are 'set' and 'reset'. When a timer is set, it will be provided with a timeout value given as argument of the set method. The timeout value will be of type 'Time' if the expiry of the timer has the effect secondly, minutely or hourly. The timeout value will be of type Date if the expiry of the timer has the effect daily, weekly, monthly or yearly. Whenever a service logic which has timers is in a waiting state, all timers of that service logic are continuously examined and their values are compared with the value of 'CurrentDate' or 'CurrentTime'. When the current time becomes larger than or equal to the time value kept by a timer, the timer will expire.

In NGN applications and data they require will be distributed among application servers. The application may be executed on an external server and data related to the application may be stored on remote host. In order to access data, the application logic must know most likely the IP address or URL of the remote host to contact and then to formulate a request to the database server. The application logic needs knowledge or at least view of database structure or how the database servers expect to be queried for information.

Databases are full of interlinked tables of the variety information the service logic is interested in. For example, a service offering tourist guide

information has data organized in a table of historical places linked to a table of historical areas linked to a table of cities and so on. The service logic may need information about historical places in an area of a city. The information required to construct that view might be stored across several tables in the database.

Common operations on database require expression of commands, such as retrieve from, update into, insert into or delete from database. In SLPL special methods for database access are defined. The database query in a form of SQL (structured query language) statement is passed as method argument. In case of problems with the database, exceptions can be thrown, caught and processed.

Markup languages are usually descriptive ones and possess restricted power for expressing actions. The SLPL flow control language constructions and arithmetic operations draw the language near to expressiveness of programming languages.

IV. ASSESSMENT OF SLPL USING EVALUATION CRITERIA

Evaluation criteria for classification of service creation technologies are defined in [5]. In brief these criteria include the following: supported network capabilities, mapping towards reference architecture, interface abstraction, kind of interface and description language, suitability for 3rd party development and usability.

The suggested mark-up approach to service creation is based on Parlay/OSA interfaces and all network capabilities exposed by these APIs are accessible. SLPL provides application developers with functional abstraction of underlying network and allows adding value to call control, data session control, mobility, user status, charging, user interaction, messaging and others.

The second criterion defines the place of the SLPL in NGN service architecture. The SLPL interpreter is placed either at the application side of the programmability architecture (Fig.6) or at the Parlay/OSA gateway side. This means that both the operator and the 3rd party can support programmability through SLPL scripting.

The third criterion evaluates SLPL interfaces regarding interface abstraction as well as the kind of interface description language. SLPL provides high level abstraction hiding technical details of network technology from application developers. The kind of interface describing communication method by which SLPL exposes network capability is XML-based. SLPL is scripting language and the SLPL interpreter parses the script at runtime.

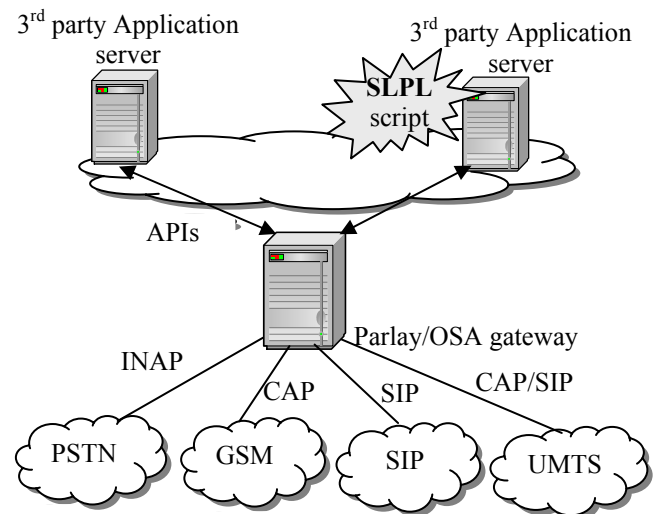


Fig. 6 SLPL script in NGN scenario

As the Parlay/OSA technology opens network interfaces for 3rd party so the SLPL can be used for a large application developer community.

SLPL might be used for service creation in many different scenarios: using scripts created by end users or using scripts created by service providers. SLPL scripts are highly customized allowing service or end user specific data to be stored in an external database.

One of the factors that determine language usability depends on availability of supporting tools. For SLPL are developed the following tools:

- Java based Augment Backus Naur Form (ABNF) translator for syntactical grammar verification and generation of basic interpreter classes
- Java-based SLPL interpreter
- Translator of Interface Description Language (IDL) descriptions into SLPL ones.

These tools are presented in the next section.

Another factor that influences on usability of the suggested service creation approach and accompanying language is measured in terms of the knowledge/experience needed. The approach is based on Parlay/OSA APIs and minimizes necessity of preliminary knowledge for network protocol and technologies.

V. SLPL SUPPORTING SOFTWARE TOOLS

Language usability depends on availability of language supporting tools – concerning scripting languages this is the interpreter. The interpretation of the logic is separated into two processing phases – front one and back one. Fig.7 shows the interpretational approach of the SLPL service execution.

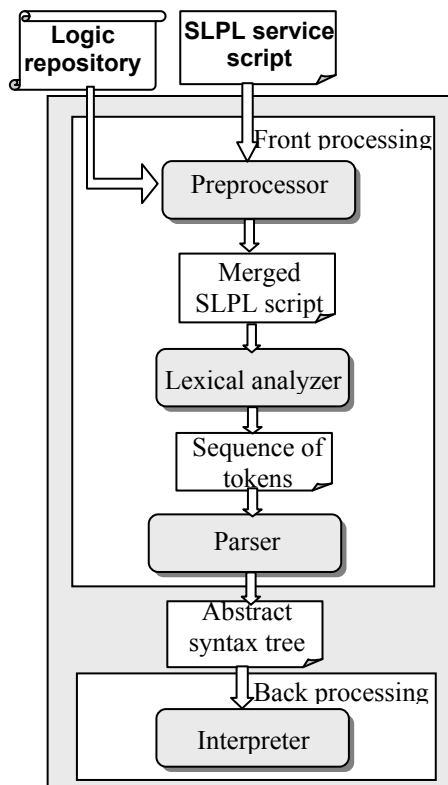


Fig.7 Script interpretation

The front processing phase is in charge of loading the service script i.e. making an instance of the logic script.

OSA interfaces are specified in Interface Description Language (IDL) which is programming language independent. A huge amount of data types on which methods operate are included in IDL specification. To reduce efforts needed for data types and method definition in SLPL an include-statement is provided. This construction allows including SLPL descriptions of methods and data types in the definition part of the service logic script. These ready-to-use parts of script are located in the script repository, accessed in shared library manner. After including definitions in the original instance, the SLPL preprocessor produces a merged, extended instance.

The main task of the SLPL lexical analyzer is to recognize the lexical units of the language like identifiers, terminal symbols, literals and so on. The extended instance of service script is lexically converted into a sequence of tokens and then the sequence is passed as input to the parser.

The SLPL parser performs syntax analysis of the input sequence of tokens in order to determine its grammatical structure with respect to SLPL formal grammar. The SLPL parser is to decide whether the sequence is acceptable in the terms of the syntactic rules of the language. If it is to be rejected, then error log is open which is omitted from the figure for sake of simplicity. Parsing transforms input sequence of

tokens into a data structure (a tree), which is suitable for later processing and which captures the implied hierarchy of the input.

For example, the language construction for synchronization wait-statement used to wait for result of network provided service is formally defined with the grammar rules shown in Fig.8.

1. wait_def	= LB"wait" [timeouted](slash GB/ GB event_list); wait for an even or timeout
2. timeouted	= "timeouted" is DQUOTE integer-val DQUOTE
3. event_list	= 1*event_list_item LB slash "wait" GB
4. event_list_item	= LB"event" a_name slash GB
5. a_name	= "name" is DQUOTE simple_name DQUOTE
6. integer_literal	= 1*DIGIT
7. simple_name	= ALPHA*(ALPHA/ DIGIT/ underscore)
8. slash	= %d47; '/'
9. is	= %d61; '='
10. underscore	= %d95; '_'
11. LB	= %d60; '<'
12. GB	= %d62; '>'
13. ALPHA	= %d65-90 / %d97-122; A-Z a-z
14. DIGIT	= %d48-57; 0-9

Fig.8 Grammar rules for definition of wait-statement

These grammar rules are built in the parser logic as a parsing table.

For the input

```
<wait timeout="20"/>
```

the SLPL lexer recognizes the type and value of the following tokens:

<	separator
wait	keyword
timeout	keyword
=	operator
"	separator
20	integer literal
"	separator
/	separator
>	separator

The sequence of tokens is passed as input to the SLPL parser and the abstract syntax tree generated is shown in Fig. 9.

During real processing for each language construction recognized correctly by the SLPL parser, an appropriate Java construction is activated.

SLPL interpreter is written in Java to achieve portability.

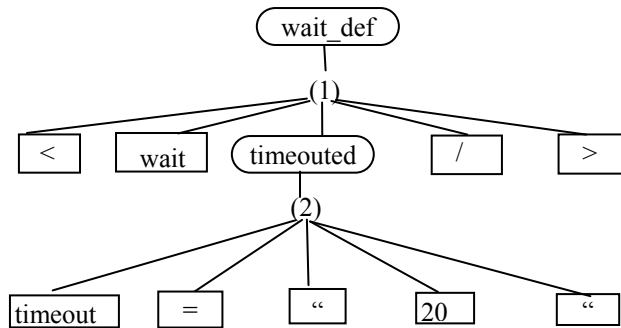


Fig.9 Abstract syntax tree built for 'wait'-statement

Another reason for Java choice as implementation language is that we use Ericsson Network Resource Gateway SDK (version R5A02) that simulates Parlay/OSA interfaces to verify the SLPL functional capabilities. The interface method calls of Parlay/OSA interface simulator are form of Java code.

The idea behind the development of SLPL is to be extensible without reprogramming of the translator producing SLPL descriptions manually. A translator-generator is developed i.e. a translator that generates translators. When supplied with language grammar rules described in ABNF, this translator-generator produces as output a Java code that 'understands' language syntax.

The translator-generator is used to generate parts of the SLPL interpreter as it is shown in Fig.10. The input sequence consists of SLPL grammar rules formally described in ABNF. Also the semantic Java rules are passed. The result is the Java source code of the SLPL interpreter.

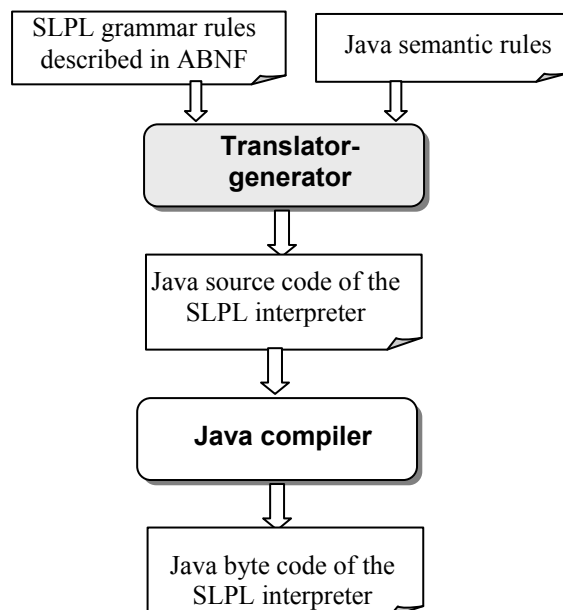


Fig.10 Process of generating Java byte code of the SLPL interpreter

The translator-generator performs lexing, parsing, semantic analysis and code generation of ABNF

grammar rules. The lexical analysis processes the input sequence of ABNF rules to produce output sequence of tokens. The translator-generator's lexer recognizes tokens defined with the basic ABNF rules [12], and the derivate ABNF rules are used to construct the parsing table. The translator-generator's parser builds an abstract syntax tree that corresponds to the grammar rules of the described language.

During code generation the syntactically correct language structures are compared with templates of semantic descriptions and converted into Java instructions.

Thus adding new rules in SLPL grammar that extend or change language expressive power, there is no need to rewrite the SLPL interpreter manually.

The translator-generator is also used for generating parts of the translator from IDL to SLPL.

Defining data types and methods in the definition part of SLPL service logic script is time consuming and tedious task if done manually. In Parlay/OSA specifications 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 Parlay/OSA specifications are described in IDL. To reduce time and effort in service script development, a translator is developed that translates IDL descriptions in SLPL descriptions.

Once generated the IDL descriptions of OSA interface data types and methods can be used as interface dependant parts in SLPL service logic scripts and they form the SLPL programming library.

The relationship between SLPL supporting tools is shown in Fig.11.

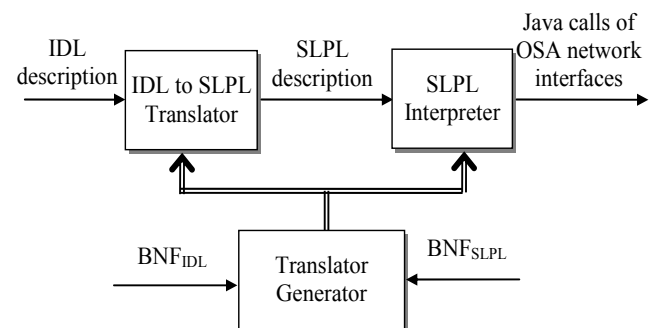


Fig. 11 SLPL supporting tools

VI. SIMULATION ENVIRONMENT

To validate the approach applicability we use Ericsson's Network Resource Gateway (NRG) Software development kit (SDK).

The NRG simulates the behavior of Service Capability Servers (SCS). The SCS provides network service to application and controls network elements

hiding network protocol complexity from application developers. The NRG simulator mimics the behavior of an actual NRG node. It simulates not only the node but also the underlying 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 offers software API libraries to Java application developers, which simplify the development of applications that use NRG capabilities.

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 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 we provide an example of application that uses SDK APIs.

VII. CASE STUDY

Let us consider a ‘Tourist guide’ application that provides tourist information about sights in a city. When a tourist dials the number of ‘Tourist guide’ service, the application locates her position and selects voice message containing information about the historical place in vicinity.

The application uses the following classes available in NRG SDK:

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

- IpUserLocation – an interface that offers methods for user location
- IpUICall – an interface that offers methods for user interaction.

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

- DBProcessor – a database processor
- IpDataBase – an interface that offers methods for database access

The configuration file of the application among the other contains a map with historical places.

The sequence of actions performed during application execution is shown in Fig.12.

- 1 When the tourist dials the number of ‘Tourist guide’ service, MPCCProcessor is notified about the event.
- 2 The event is forwarded to the application.
- 3 The application requests information about 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 identification of voice message from the database containing information about historical places. The application sends the user coordinates as parameter.
- 8 The DBProcessor calls DB_retrieve method to extract the requested data. The method is synchronous and the application waits for response.
- 9 The result of database query is forwarded to the application.
- 10 The application starts dialogue with the user.
- 11 User interaction session is created with the tourist.
- 12 The application sends to the UIProcessor the identification of the message to be sent.
- 13 UIProcessor requests from the user interaction service to play the message.
- 14 When the message ends this is reported to the UIProcessor.
- 15 The application requests to release user interaction session.
- 16 The UIProcessor frees the user interaction service.
- 17 The application must free the call related resources in the gateway.
- 18 The multi-party call control service is released.

The application logic is described in SLPL.

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.12. The SLPL interpreter is supplied with the SLPL script of 'Tourist guide' application.

When the example 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's NRG simulator.

In runtime on request of the user of the 'Tourist guide' application, the current location of the tourist is obtained and displayed in a map. When the tourist dials the service number, an announcement is played if there is a historical place in the vicinity.

Fig.13 shows a screenshot of service simulation.

VIII. CONCLUSION

A new mark-up approach to NGN service creation is suggested. The approach is based on Parlay/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 illustrate the approach are defined. The language supporting tools allow the approach usability.

OSA APIs provide medium level of abstraction and developers need some telecommunication knowledge when linking components that offer APIs, but as SLPL possesses all attractive feature of scripting languages it 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. Time-related language constructions allow time-dependent decisions in service logic description. SLPL constructions for database manipulation and query enable service logic to consult external database to retrieve service specific data.

SLPL is functionally richer than CPL and other XML-based languages for service creation. The suggested approach and accompanying language provide easy-to-use and cost effective tools for service creation in next generation networks.

REFERENCES

- [1] Bakker, J. Tweedie, D. and Umnehopa, M. [WWW document] (2006) Evolving Service Creation; New developments in network Intelligence, <http://www.argreenhouse.com/papers/jlbakker/Bakker-telenor.pdf>
- [2] Parlay Group, Parlay APIs Overview, (2004) [WWW document] http://www.parlay.org/docs/Spec3_es_20191501v010101m.pdf
- [3] Sun Microsystems, (2001) The JAIN APIs: Integrated Network APIs for the Java Platform, [WWW document] <http://java.sun.com/products/jain/WP2001.pdf> (access January 2004)
- [4] Bakker J-L, Jain, R. Next Generation Service Creation Using XML Scripting Languages, [WWW document] (2006), <http://www.argreenhouse.com/papers/jlbakker/bakker-icc2002.pdf>
- [5] Falcarin, P. Licciardi, C.A. [WWW document] (2006) Analysis of NGN service creation technologies, <http://softeng.polito.it/falcarin/docs/Annals03.pdf>
- [6] Lennox, J. Wu, X. , CPL: A Language for User Control of Internet Telephony Services, RFC 3880, (2004)
- [7] W3C, Voice eXtensible Markup Language (VoiceXML) version 1.0., W3C Note, 2000, [WWW document], <http://www.w3.org/TR/voicexml/>
- [8] W3C, Voice Browser Call Control: CCXML Version 1.0, [WWW document] 2007, <http://www.w3.org/TR/ccxml/>
- [9] Bakker, J. Jain, R. [WWW document] (2006) A service creation markup language for scripting next generation network services, [WWW document], (2005), <http://www.cs.columbia.edu/sip/drafts/draft-bakker-jain-scm1-00.txt>
- [10] Ericsson Network Resource Gateway SDK (version R5A02) [WWW document], (2005) http://www.ericsson.com/mobilityworld/sub/open/technologies/parlay/tools/parlay_sdk
- [11] Crocker, D. Overell, P. RFC 2234 - Augmented BNF for Syntax Specifications: ABNF [WWW document] (2006)

Ivaylo Atanasov received his M.S. degree in electronics from Technical University of Sofia, Bulgaria in 1992. Currently he is a PhD student in Telecommunications networks. His current position is Assistant Professor at Faculty of Communications. His main research focus is development of open service platforms for next generation networks. He is a member of IEEE society.



Evelina Pencheva received her M.S. degree in mathematics from University of Sofia, Bulgaria, and PhD degree in communications from Technical University of Sofia. Her current position is Associate Professor in Faculty of Communications. Her interests include next generation mobile applications and middleware platforms.



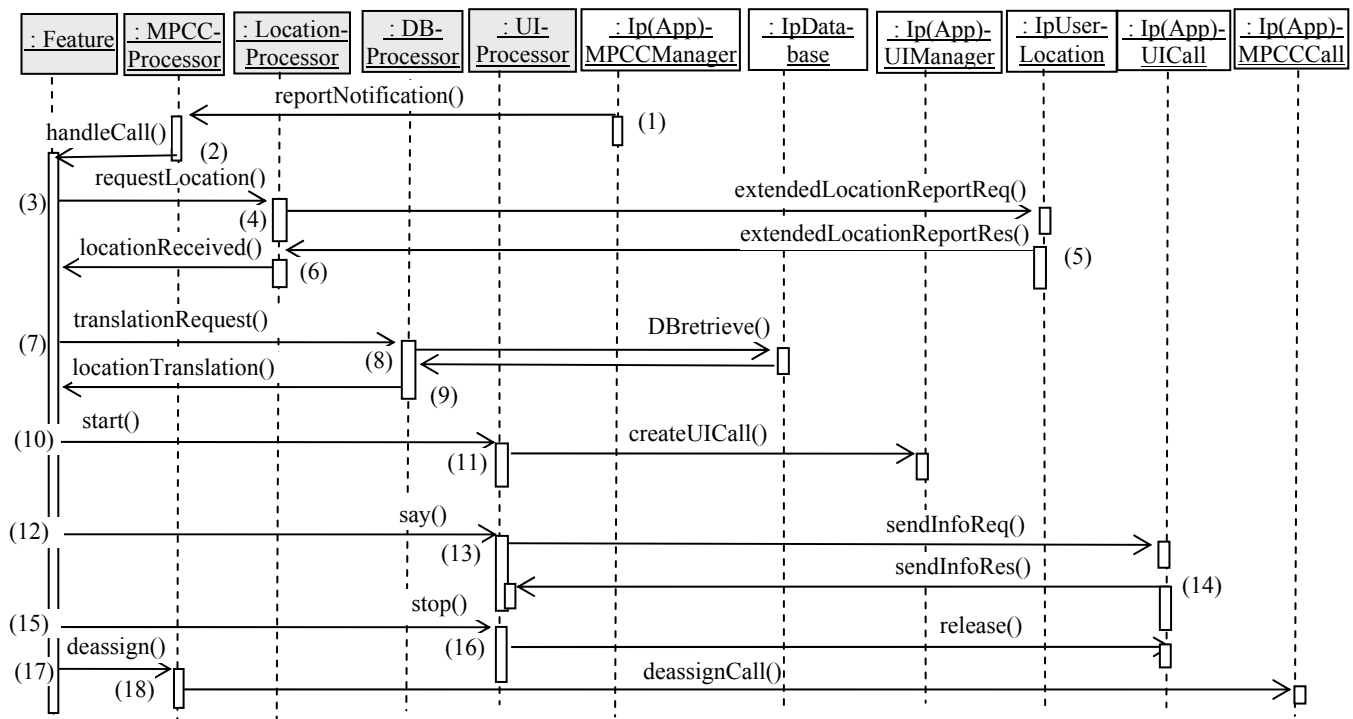


Fig.12 Sequence diagram for 'Tourist-guide' application

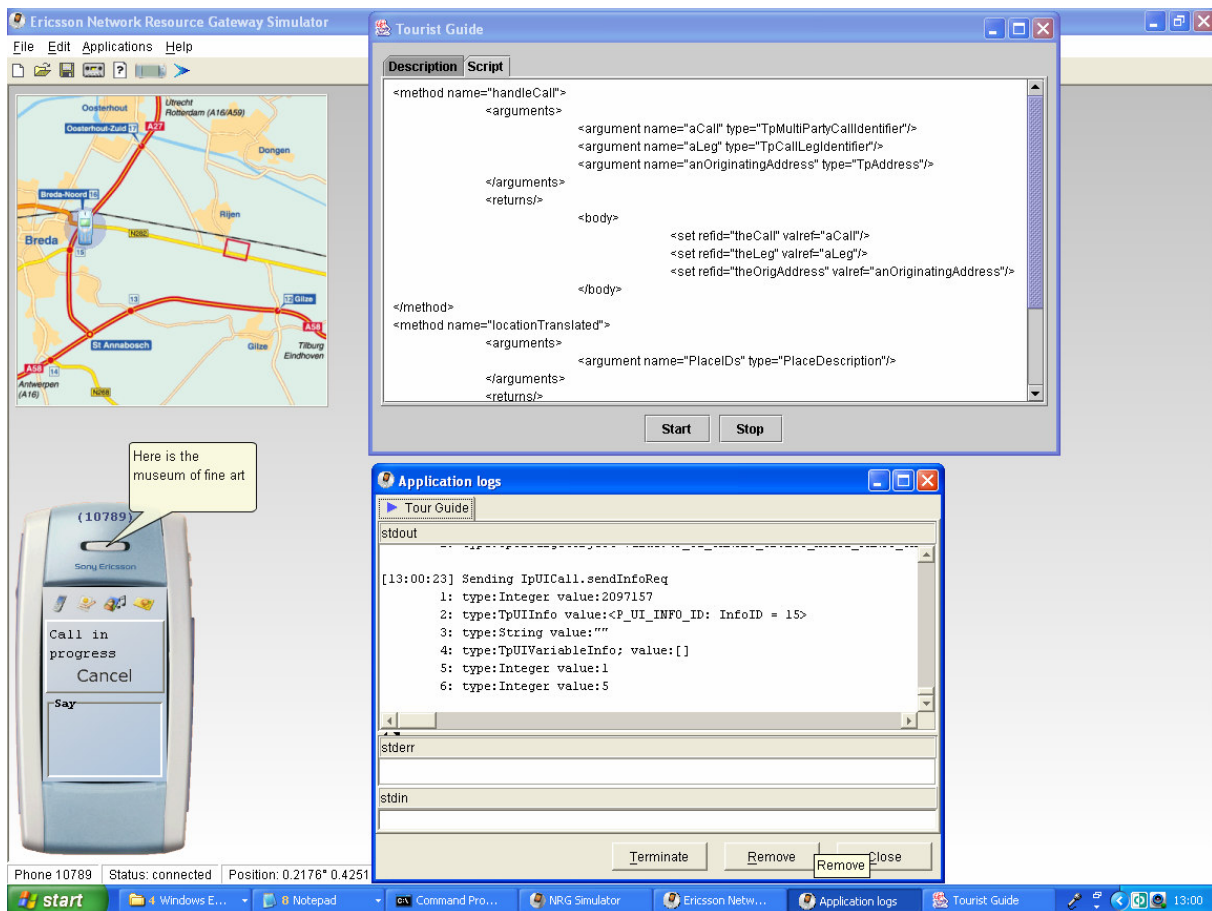


Fig.13 A screenshot of 'Tourist-guide' application simulation