A NEW SERVICE LOGIC PROCESSING LANGUAGE

Ivaylo Ivanov Atanasov, Evelina 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 language constructions of a new XML-based language for service logic processing. The language is designed to support Open Service Access (OSA) Application programming interfaces (APIs). It has more expressive power when comparing to existing XML-based languages as it supports both call-related and call-unrelated user interactions. In addition to OSA interface description, the language offers means for OSA interface method invocation and language constrictions for processing method results in the context of service logic.

Keywords: Open Service Access, XML-based language, Service logic processing

1. INTRODUCTION

In next generation networks (NGN) the network establishing connections will be separated from the network parts maintaining services. To implement this differentiation in 3G mobile networks, network components called service capabilities are introduced. With Open Service Access (OSA) service capabilities are accessible through Application Programming Interfaces (APIs), which contain management and control entities. The OSA APIs include functions for mobility, call control, data session control, user interaction, messaging and others.

One of the ways for implementing OSA is by the use of eXtensible Markup Languages (XML). XML-based scripting languages allow one to write services more rapidly and easily. While not as flexible or powerful as a programming language, XML-based scripting languages are typically easier to learn and are language and platform independent. There exist several XML-based languages that have limited expressive power restricted to call processing [1, 2].

In this paper we present a new service creation markup language called Service Logic Processing Language (SLPL) for scripting applications in next generation networks. The language provides means for OSA interface descriptions [3]. While other XML-based languages are mainly call-oriented, the SLPL supports both call-related and call-unrelated interfaces. Language constructions are proposed for method invocation [4] and processing method results in the context of service logic. The formal grammar definitions of language constructions for flow control of the logic processing are given here.

2. SLPL LANGUAGE CONSTRUCTIONS FOR FLOW CONTROL

OSA consists of 10 main interface groups. The OSA framework is a special interface that contains all support functions for OSA, most importantly security and

access functions. The other interfaces are service interfaces that allow applications to control specific network resources.

The OSA interfaces are defined as a set of object classes. A class defines the methods that can be called on the object and their parameter types. To implement a service that exploits OSA interfaces, one needs means for method invocation and processing method results.

The service logic is executed following the occurrence of the specified event. Events that can activate a service are provided by an event source. Capability servers are examples of event sources, but events may also originate elsewhere, e.g. from a user making a call or pre-configured timer.

Once activated the service logic script executes actions for calling OSA interface methods and processes the results returned. The SLPL provides language constructions for method invocation [3, 4]. To achieve flexibility in logic processing, flow control constructions have to be provided. Mandatory SLPL constructions involve if-then-else statement, case-statement, loop-statement (e.g. while) as minimum. In order to be complete there should be also support of constructions dealing with exceptional situations when invoked method of OSA interface raises an error. The results of method invocations are stored in predefined variables of respective type and if necessary compared to predefined constants or other variables.

Figure 1 shows the formal grammar definition in Backus-Naur form of case-statement and figure 2 shows the formal grammar definition for while-statement in SLPL.

1. <case statement> ::= <**CASE**><case condition><cases>
2. <case condition> ::= <**REFID**>=<quote><integer id><quote>
3. <integer id> ::= <integer constant id>|<integer variable id>|<integer expression>
4. <integer constant id> ::= <constant id>
5. <integer variable id> ::= <variable id>
6. <integer expression> ::= <expression>
7. <cases> ::= <sub cases> [<default case>]
8. <sub cases> ::= <sub case><cases>
9. <sub case> ::= <**ON**><integer constant><case statements>
10. <integer constant> ::= <integer constant literal>|<integer constant id>
11. <integer constant literal> ::= <**VAL**><is><quote><constant literal><quote>
12. <integer constant id> ::= <**VALREF**><is><quote><integer variable id><quote>
13. <statements> ::= [<statement><statements>]
14. <default switch case> ::= <**DEFAULT**><statements>
15. <quote> ::= "
16. <is> ::= =

Fig. 1 Case-statement grammar

1. <while statement> ::= <while term><statements>
2. <while term> ::= <**WHILE**><while condition>
3. <statements> ::= [<statement><statements>]
4. <while condition> ::= <logical term>
5. <logical term> ::= <logical value>|<extended logical value>
6. <extended logical term> ::= <logical term><binary logical operator><logical term>|<unary logical operator><logical term>
7. <binary logical operator> ::= <**AND**>|<**OR**>|<**XOR**>
8. <unary logical operator> ::= <**NOT**>
9. <logical value> ::= <logical constant>|< logical variable>|<compare expression>
10. <logical constant> ::= <logical reference>
11. <logical variable> ::= <logical reference>
12. <logical reference> ::= <**REFID**><is><quote><reference><quote>
13. <compare expression> ::= <term><compare operator><term>
14. <compare operator> ::= <**EQ**>|<**NEQ**>|<**LT**>|<**LE**>|<**GT**>|<**GE**>
15. <quote> ::= "
16. <is> ::= =

Fig. 2 While-statement grammar

An XML form of grammar rules for while statement is given in figure 3.

3. AN EXAMPLE OF OSA APPLICATION

Here we give an example that illustrates the SLPL flow control constructions in service logic that exploits OSA interfaces. Let us consider a mobile payment service. In this example a vending machine offers the option of paying for newspapers by mobile terminal.

First the mobile subscriber dials the service code. This event triggers the mobile payment service. The service logic has to authorize the user to verify if he can use his subscription for payments. The PIN provided by the user is verified against the PIN stored in a database. If the PIN is incorrect the user is asked again to input his identification and if the user fails three times an error message is played to him and the call is cleared. If the PIN is correct a message is played to the user asking him to choose a newspaper from a menu. Then a message is sent to the vending machine to deliver the newspaper and the user is charged with the amount due.

This example is simplified for the sake of clarity. In reality it would also be desirable to take into account the radio cell in which the order is placed, to ensure that the user is actually in the vicinity of the machine and has not dialed the special service code erroneously or maliciously.

```
<GRAMMAR>
 <RULE ID="1" NAME="while_statement">
  <RULEREF RULEID="2"/>
  <RULEREF RULEID="3"/>
 </RULE>
 <RULE ID="2" NAME="while_term">
   <KEYWORD NAME="WHILE"/>
   <RULEREF RULEID="4"/>
 </RULE>
 <RULE ID="3" NAME="statements">
  <O>
   <RULEREF RULENAME="statement"/>
   <RULEREF RULEID="3"/>
  </O>
 </RULE>
 <RULE ID="4" NAME="while_condition">
  <RULEREF RULEID="5"/>
 </RULE>
 <RULE ID="5" NAME="logical_term">
  <L>
   <RULEREF RULEID="9"/>
   <RULEREF RULEID="6"/>
  </L>
 </RULE>
 <RULE ID="6" NAME="extended_logical_
                term">
  <L>
    <P>
     <RULEREF RULEID="5"/>
     <RULEREF RULEID="7"/>
     <RULEREF RULEID="5"/>
    </P>
    <P>
     <RULEREF RULEID="8"/>
     <RULEREF RULEID="5"/>
    </P>
  </L>
 </RULE>
 <RULE ID="7" NAME="binary_logical_
                operator">
  <L>
   <KEYWORD NAME="AND"/>
   <KEYWORD NAME="OR"/>
   <KEYWORD NAME="XOR"/>
  </L>
 </RULE>
 <RULE ID="8" NAME="unary_logical_
                operator">
   <KEYWORD NAME="NOT"/>
 </RULE>
 <RULE ID="9" NAME="logical_value">
   <L>
     <RULEREF RULEID="10"/>
     <RULEREF RULEID="11"/>
     <RULEREF RULEID="13"/>
   </L>
 </RULE>
 <RULE ID="10" NAME="logical_constant">
     <RULEREF RULEID="12"/>
 </RULE>
 <RULE ID="11" NAME="logical_variable">
     <RULEREF RULEID="12"/>
 </RULE>
 <RULE ID="12" NAME="logical_reference">
   <P>
    <KEYWORD NAME="REFID"/>
    <RULEREF RULEID="16"/>
    <RULEREF RULEID="15"/>
    <RULEREF RULENAME="reference"/>
    <RULEREF RULEID="15"/>
   </P>
  </RULE>
  <RULE ID="13" NAME="compare_
                expression">
   <P>
    <RULEREF RULENAME="term"/>
    <RULEREF RULEID="14"/>
    <RULEREF RULENAME="term"/>
   </P>
  </RULE>
  <RULE ID="14" NAME="compare_operator">
   <L>
    <KEYWORD NAME="EQ"/>
    <KEYWORD NAME="NEQ"/>
    <KEYWORD NAME="LT"/>
    <KEYWORD NAME="LE"/>
    <KEYWORD NAME="GT"/>
    <KEYWORD NAME="GE"/>
   </L>
  </RULE>
  <RULE ID="15" NAME="quote">"
  </RULE>
  <RULE ID="16" NAME="is">=
  </RULE>
 </GRAMMAR>
```

Fig. 3 XML descriptions of while-statement grammar rules

Figure 4 shows the block diagram of service logic.

The first OSA interface that is used by the service logic is User interaction (IpUICall interface) which asks the user to input his PIN code. To process the PIN provided by the user, the sendInfoAndCollectReq method invocation is wrapped in a *while-statement* where the user input is verified up to tree times in case of incorrect PIN (steps 1 and 2)

If the PIN is corrects the service logic invokes the sendInfoAnd-CollectReq method again to play a menu asking the user to input his choice. Now the result returned by the IpUICall::sendInfoAndCollectRes method is processed with *case-statement* (step 4).

Having the user's choice the service logic uses Messaging OSA interface to tell the vending machine to deliver the newspaper. Then Charging OSA interface is used to charge the subscribers account.

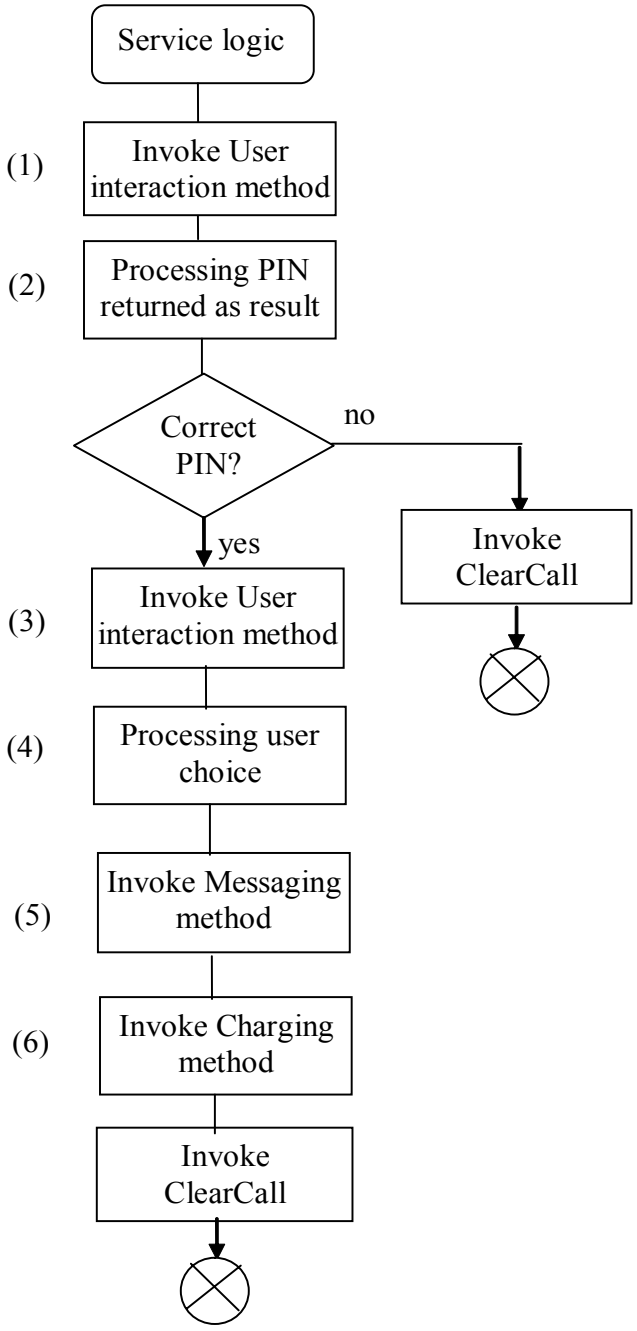

Fig. 4 Block diagram of a mobile payment service logic

Fragments of the service logic using while-statement for PIN checking are shown in figure 5.

4. CONCLUSION

A new XML-based language, SLPL, is proposed that provides a framework for NGN service creation. The SLPL supports OSA service interfaces and provides means for flow control. The language can be used as a basis for defining object parameters, as well as for the glue between OSA APIs and the application logic, which itself consists of these variously assembled objects. On the basis of this type of

framework, it is possible to conceive of whole generalized applications (a mobile payment program, for example) being defined as a set of standard, portable objects, which could then be made available as resources to a more specific application that might need to employ them.

```
<LOGIC>
    <DEFINE>
        <VARIABLES>
            <ID NAME="VID_PIN" TYPE="STRING" VAL="" DEFAULT=""/>
            <ID NAME="VID_PININ" TYPE="STRING" VAL="" DEFAULT=""/>
            <ID NAME="VID_TRIES" TYPE="INTEGER" VAL="0" DEFAULT="0"/>
            <ID NAME="VID_PINOK" TYPE="BOOLEAN" VAL="FALSE" DEFAULT="FALSE"/>
            <ID NAME="VID_CHOICE" TYPE="INTEGER" VAL="0" DEFAULT="0"/>
        </VARIABLES>
    </DEFINE>
    <EXECUTE>
        <!-- READS AUTHORIZATION PIN INTO 'VID_PIN' -->
        <WHILE TEST="VID_TRIES LT 0">
            <INCREASE REFID="VID_TRIES" BY="1"/>
            <!-- INVOKE UI 'IpUICall::sendInfoAndCollectReq' TE GET PIN INTO 'VID_PININ' -->
            <IF> <CONDITION TEST="VID_PIN EQ VID_PININ"/>
                <THEN> <SET REFID="VID_PINOK" VAL="TRUE"/> <BREAK/>
                </THEN>
            </IF>
        </WHILE>
        <IF> <CONDITION TEST="VID_PINOK NEQ TRUE"/>
            <THEN> <!-- AUTHORIZATION FAILED. STOP --> </THEN>
        </IF>
        <!-- INVOKE 'IpUICall::sendInfoAndCollectReq' TO GET MENU CHOICE -->
        <CASE REFID="VID_CHOICE">
            <ON VAL="1"> <!—SET CHOICE TO NEWSPAPER #1 --> </ON>
            <ON VAL="2"> <!—SET CHOICE TO NEWSPAPER #2 --> </ON>
            <ON VAL="3"> <!—SET CHOICE TO MAGAZINE #1 --> </ON>
            <DEFAULT> <!—SET CHOICE TO UNKNOWN --> </DEFAULT>
        </CASE>
        <!--  CONTINUE WITH DELIVERY AND CHARGE -->
    </EXECUTE>
</LOGIC>
```

Fig. 5 Fragments of service logic

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.