Approach to Formal Verification of Messaging Service Capability Server in Mobile Networks

Ivaylo Atanasov[1]

Abstract – **In this paper it is investigated how open access to messaging function in mobile networks may be deployed. The focus is on Open Service Access (OSA) interfaces for user interaction and Customized Application for Mobile network Enhanced Logic (CAMEL) applied to Short Message Service. Service Capability Server (SCS) makes translation between OSA interface methods and CAMEL Application Part (CAP) protocol. The formalism of labelled transition systems and the behavioural equivalence concept are used to verify the SCS functional behaviour.**

Keywords – **CAMEL, Open Service Access, Labelled Transition Systems, Bisimulation.**

I. Introduction

Open Service Architecture (OSA) allows third party access to communication functions in a network neutral way. Using OSA Application Programming Interfaces (API), application developers can create attractive applications without specific knowledge about underlying network technology and control protocols. Interoperability between OSA applications and specific network functions requires special type of application server called OSA Service Capability Server (SCS). The OSA SCS is responsible for translation of OSA interface method invocations into control protocol messages and vice versa.

The research focus is on OSAinterfaces for user interaction and Customized Application for Mobile network Enhanced Logic (CAMEL) applied to Short Message Service (SMS). The OSA User Interaction (UI) service provides API for call-related and call-unrelated user interactions [1]. The UI supports sending information or sending and collecting information. The mappings of OSA UI API onto CAMEL Application Part (CAP) protocol in the context ofSMS is defined in [2]. Some implementation aspects of CAMEL messaging service and OSA messaging service are discussed in [3,4] but no interworking issues are considered. In order to make interface to protocol translation, the OSA SCS needs to maintain two mutually synchronized state machines representing the application view on UI and protocol states. In the paper, we suggest a formal approach to verification of OSA SCS using the formalism of Labelled Transition Systems and the concept of bisimulation. The approach may be used for automatic generation of test cases during the OSA SCS functional verification [6].

The paper is organized as follows. In Section II, we discuss aspects of OSA deployment in a mobile network with CAMEL architecture.The formalism for Labelled Transition Systems is briefly introduced in Section III. A formal description of OSA SCS behavior is given in Section IV. Section V presentsformal descriptions of CAMEL state models for SMS events. Finally, the behavioral equivalence of state machines of OSA UI model and CAMEL SMS modelsis proved in Section VI.

[1]The author is with the Faculty of Telecommunications, Technical University of Sofia, Kliment Ohridski 8, 1000 Sofia, Bulgaria, E-mails: iia@tu-sofia.bg.

II. Functional Architecture for Open Access to Messaging Functions

A functional architecture for deployment of OSA UI interfaces in CAMEL network is presented in Fig.1. Toward the network, the OSA SCS performs functions of CAMEL gsmSCF (Service Control Function) which provides CAMEL service logic. The network node - Mobile services Switching Center (MSC) or Serving GPRS Support Node (SGSN), provides functions of gsmSSF (Service Switching Function) which is responsible for switching between SMS processing and service logic in gsmSCF. The SMS-Center (SMSC) is a node where short messages are stored before delivering.

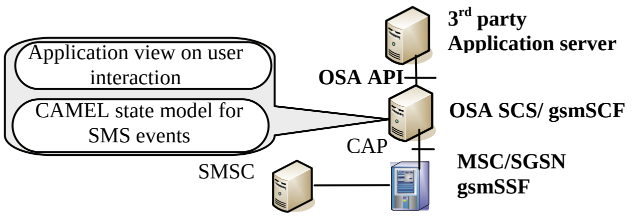
3rd party Application server
Application view on user interaction
CAMEL state model for SMS events
OSA API
OSA SCS/ gsmSCF
CAP
MSC/SGSN gsmSSF
SMSC

Fig. 1. OSA UI interface deployment in CAMEL network

In [1], it is defined a model that represents the application view of user interactions. In [5], two CAMEL models for SMS events are defined, one for mobile originating short messages and another for mobile terminating short messages. The behavior of OSA SCS regarding user interactions for short messaging needs to correspond to the specified models of both OSA application and CAMEL service logic. The formal specification of the models allows proving the behavioral equivalence, and hence the interoperability of OSA user interaction control and CAMEL service control.

III. Labelled Transition Systems and Behavioural Equivalence

To prove formally behavioral equivalence between state machines, the notion of *Labelled Transition Systems* is used [6].

Definition 1: A *Labelled Transition System* (LTS) is a quadruple (S, Act, $\rightarrow$, s_0), where S is countable set of states, *Act*is a countable set of elementary actions, $\rightarrow \subseteq S \times Act \times S$ is a set of transitions, and $s_0 \in S$ is the set of initial states.

We will use the following notations:

- $s \xrightarrow{a} s'$ stands for the transition (s, a, s');
- $s \xrightarrow{a}$ means that $\exists$ s': $s \xrightarrow{a}$ s';
- $s \overset{\mu}{\Rightarrow} s_n$, where $\mu = a_1, a_2, ..., a_n$: $\exists s_1, s_2, \ldots, s_n$, such that $s \xrightarrow{a_1} s_1 \ldots \xrightarrow{a_n} s_n$;
- $s \overset{\mu}{\Rightarrow}$ means that $\exists s'$, such as $s \overset{\mu}{\Rightarrow} s'$;
- $\overset{\hat{\mu}}{\Rightarrow}$ means $\Rightarrow$ if $\mu \equiv \tau$ or $\overset{\mu}{\Rightarrow}$otherwise,

where τ is one or more internal actions. More detailed notation description can be found in [6].

The concept of *bisimulation* [7] is used to prove that two LTSs expose equivalent behavior. The strong bisimulation possesses strong conditions for equivalence which are not always required. For example, there may be internal activities that are not observable. The weak bisimulation ignores the internal transitions.

Definition 2: [7] Two labelled transition systems $T = (S, Act, \rightarrow, s_0)$ and $T' = (S', Act, \rightarrow', s_0')$ are *weakly bisimilar* if there is a binary relation $U \subseteq S \times S'$ such that if $s_1 U t_1$: $s_1 \subseteq S$ and $t_1 \subseteq S'$ then $\forall a \in Act$:

- $s_1 \overset{a}{\Rightarrow} s_2$ implies $\exists$ t_2: $t_1 \overset{\hat{a}}{\Rightarrow}' t_2$ and $s_2 U t_2$;
- $t_1 \overset{a}{\Rightarrow}' t_2$ implies $\exists$ s_2: $s_1 \overset{\hat{a}}{\Rightarrow} s_2$ and $s_2 U t_2$.

IV. Formal Description of OSA User Interaction Model

The application view on UI object is defined in [1]. The behavior of the UI object is described by finite state machine. In Null state, the UI object does not exist. The UI object is created when the createUI()method is invoked or a network event is reported by reportEventNotification()method. In Active state, the UI object is available for requet messages which have to be sent to the network. Both sendInfoAndCollectReq()and sendInfoReq()methods have a parameter indicating whether it is a final request and the UI object has to be released after the information has been presented to the user. In Active state, when a fault is detected on the user interaction, an error is reported on all outstanding requests. A transition to Release Pendingstate is made when the application has indicated that after a certain message no further messages need to be sent to the end-user. There might be, however, still a number of messages that are not yet completed. After the last message is sent or when the last user interaction has been obtained, the UI object is destroyed. In Finished state, the user interaction has ended. The application can only release the UI object. A simplified state transition diagram for UI object is shown in Fig.2.

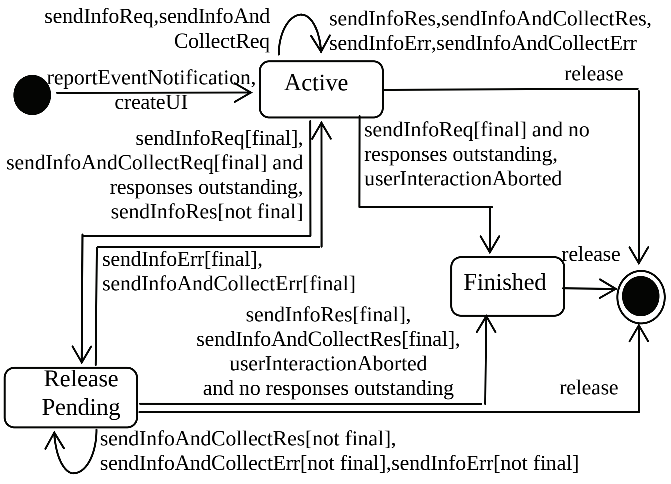
sendInfoReq,sendInfoAnd CollectReq
sendInfoRes,sendInfoAndCollectRes, sendInfoErr,sendInfoAndCollectErr
reportEventNotification, createUI
Active
release
sendInfoReq[final], sendInfoAndCollectReq[final] and responses outstanding, sendInfoRes[not final]
sendInfoReq[final] and no responses outstanding, userInteractionAborted
sendInfoErr[final], sendInfoAndCollectErr[final]
Finished
release
sendInfoRes[final], sendInfoAndCollectRes[final], userInteractionAborted and no responses outstanding
Release Pending
release
sendInfoAndCollectRes[not final], sendInfoAndCollectErr[not final],sendInfoErr[not final]

Fig. 2. OSA application view on the UI object

By $T_{AppUI} = (S_{AppUI}, Act_{AppUIH}, \rightarrow_{AppUI}, s_0)$ we denote a LTS representing the OSA application view on UI object where:

- S_{AppUI} = { Null, Active, ReleasePending, Finished };
- Act_{AppUI} = { createUI, reportEventNotification, sendInfoReq, sendInfoAndCollectReq, sendInfoRes, sendInfoErr, sendInfoAndCollectRes, sendInfoAndCollectRes, sendInfoAndCollectErr, sendInfoAndCollectErr, userInteractionAborted, release };
- $\rightarrow_{AppUI}$ = { Null createUI Active,
Null reportEventNotification Active,
Active sendInfoReq Active,
Active sendInfoRes Active,
Active sendInfoAndCollectReq Active,
Active sendInfoAndCollectRes Active,
Active sendInfoErr Active,
Active sendInfoAndCollectErr Active,
Active release Null,
Active sendInfoReq ReleasePending,
Active sendInfoRes ReleasePending ,
ReleasePending sendInfoErr Active,
ReleasePending sendInfoErr ReleasePending,
ReleasePending sendInfoRes Finished,
ReleasePending userInteractionAborted Finished,
ReleasePending release Null,
Finished release Null,
Active sendInfoAndCollectReq ReleasePending,
ReleasePending sendInfoAndCollectErr Active,
ReleasePending sendInfoAndCollectRes ReleasePending,
ReleasePending sendInfoAndCollectErr ReleasePending,
ReleasePending sendInfoAndCollectRes Finished,
Active sendInfoReq Finished,
Active userInteractionAborted Finished };
- $s_{0'}$ = { Null }.

V. Formal Description of CAMEL State Models for SMS Events

CAMEL defines state models for SMS events which provide the possibility of triggering services as a result of messaging events [5]. Service logic may brake into sending a short message. CAMEL doesn't inspect the content of any message and it doesn't trigger services on that basis; the only events CAMEL triggers a service are the ones regardingsignaling conditions. CAMEL can recognize the origin and destination addresses of the message and can use this as criteria to start a service.

The Mobile Originating(MO) SMS state model is used to describe the actions in MSC and SGSN during Mobile Originating SMS and it is shown in Fig.3. The model is started when the gsmSSF sends to the gsmSCF and InitialDPSMS message.

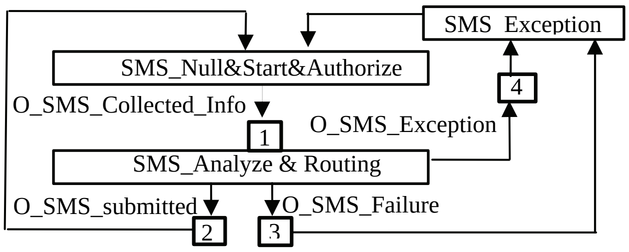

Fig. 3.CAMEL state model for MO SMS events

Entry events for SMSNull&Start&Authorize state are about previous MO SMS transfer to the SMSC completed or exception event. The detection point SMS_Collected_Info indicates that the subscription information is analysed and a MO short message is received. The CAMEL control flow between gsmSCF and gsmSSF corresponding to this detection point is shown in Fig.4.

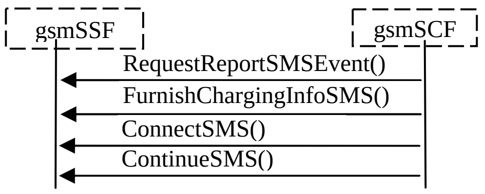

Fig. 4.CAP message flow on sending a short message

In SMSAnalyze&Routing state, information being analysed and/or translated to determine routing address of the SMSC and the short message is sent to the SMSC. The O_SMS_Submitteddetection point indicates that the short message is successfully submitted to the SMSC and it is reported by gsmSSF to gsmSCF sending anEventReportSMS message. The O_SMS_Failure detection point is armed when a failure has occurred in the SMS or command submission. The failure may have occurred internally in the MSC or SGSN or may have occurred externally, e.g. in the SMSC. Inthis case, the gsmSSF reports an error to the gsmSCF sending EventReportSMS message. An exception situation occurs when the gsmSSF reports DialogueAbort or DialogueError to the gsmSCF.

We decompose the SMSNull&Start&Authorize state into two states: SMSNull$_O$ and Start&Authorize$_O$ to distinguish between different short messages in user interactions. Using the notations of LTS, we describe formally the CAMEL state model for MOSMS events by T_{OSMS} = (S_{OSMS}, Act_{OSMS}, $\rightarrow_{OSMS}$, s_0') where

- S_{OSMS}= {SMSNull$_O$, Start&Authorize$_O$, SMSAnalyze&Routing };
- Act_{OSMS} = { InitialDPSMS$_O$, RequestReportSMSEvent, ConnectSMS, FurnishChargingInfoSMS, ContinueSMS, EventReportSMS, Release, EventReportSMS,DialogueAbort, DialogueError};
- $\rightarrow_{OSMS}$ = { SMSNull$_O$ InitialDPSMS$_O$Start&Authorize$_O$, Start&Authorize$_O$RequestReportSMSEvent SMSAnalyze&Routing, SMSAnalyze&Routing FurnishChargingInfoSMSSMSAnalyze&Routing, SMSAnalyze&Routing ConnectSMSSMSAnalyze&Routing, SMSAnalyze&Routing ContinueSMSSMSAnalyze&Routing, SMSAnalyze&Routing EventReportSMSSMSNull$_O$, SMSAnalyze&Routing EventReportSMSStart&Authorize$_O$, SMSAnalyze&Routing EventReportSMSSMSNull$_O$, SMSAnalyze&Routing ReleaseSMSNull$_O$, SMSAnalyze&Routing DialogueAbortSMSNull$_O$, SMSAnalyze&Routing DialogueErrorSMSNull$_O$};
- s_0' = {SMSNull$_O$}.

The Mobile Terminating(MT) SMS state model is used to describe the actions in MSC and SGSN during Mobile Terminating SMS, and it is shown in Fig.5.

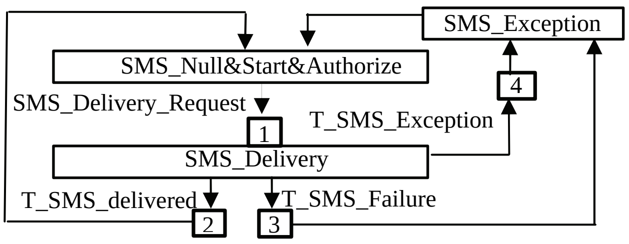

Fig. 5.CAMEL state model for MT SMS events

The SMSNull&Start&Authorize$_T$ state is entered when a short message is received in MSC from SMS-gateway MSC or previous MT SMS transfer completed or an exception event occurs. The SMS_Delivery_Request detection point indicates that the a mobile terminating SMS is received and it corresponds to the EventReportSMS message.The T_SMS_Delivereddetection point indicates that the short message has been successfully delivered which corresponds to theEventReportSMS message. The T_SMS_Failuredetection point indicates that the short message has failed which is reported byEventReportSMS message.

We decompose the SMSNull&Start&Authorize state into two states: SMSNull$_T$ and Start&Authorize$_T$ to distinguish between different short messages in user interaction.Using the notations of LTS, we describe formally the CAMEL state model for MTSMS events by T_{TSMS} = (S_{TSMS}, Act_{TSMS}, $\rightarrow_{TSMS}$, s_0”) where

- S_{TSMS} = {SMSNull$_T$, Null&Start&Authorize$_T$, SMSDelivery};
- Act_{TSMS} = { InitialDPSMS$_T$, Release, EventReportSMS, EventReportSMS, EventReportSMS, DialogueAbort, DialogueError };
- $\rightarrow_{TSMS}$ = { SMSNull$_T$ InitialDPSMS$_T$Start&Authorize$_T$, Start&Authorize$_T$EventReportSMS SMSDelivery, SMSDelivery EventReportSMSSMSNull$_T$, SMSDelivery EventReportSMSStart&Authorize$_T$, SMSDelivery EventReportSMSSMSNull$_T$, SMSDelivery ReleaseSMSNull$_T$, SMSDeliveryDialogueAbortSMSNull$_T$, SMSDeliveryDialogueErrorSMSNull$_T$};
- s_0” = { SMSNull$_T$ }.

VI. Behavioural Equivalence between State Machines in OSA and CAMEL

To prove the interoperability between user interaction model in OSA and CAMEL state machines for SMS events we have to prove that the state machine representing the OSA user interactions and the CAMEL state machines for SMS events expose equivalent behavior. The behavioral equivalence is proved using the concept of weak bisimilarity.

Proposition 1: The labelled transition systems T_{AppUI}, T_{OSMS} and T_{TSMS} are weakly bisimilar.

Proof 1: To prove the bisimulation relation between labelled transition systems, it has to be proved that there is a bisimulation relation between their states. With U it is denoted a relation between the states of T_{AppUI}, $T_{OSMS}$and T_{TSMS} where U={(Null, SMSNull$_O$, SMSNull$_T$),(Active, Start&Authorize$_O$, Start&Authorize$_T$)}. Table 1 presents the bisimulation relation between the states of of T_{AppUI}, $T_{OSMS}$and T_{TSMS} which satisfies Definition 2.In [2], a mapping between the OSA User Interaction interface methods and CAP messages in the context of SMS is defined. We use this mapping to showaction's similarity. Based on the bisimulation relation between the states of T_{AppUI}, $T_{OSMS}$and T_{TSMS} it can be stated that the state machines expose equivalent behavior.

As an example,an application that uses OSA UI interfaces creates UI object and requests a message to be sent to the user, which starts the CAMEL state model for MO SMS events. Whent the SMS is submitted in the CAMEL network, the application is informed about the result of requested operation.

VII. Conclusion

In this paper an approach to formal description of OSA user interaction and CAMEL models for SMS events is suggested. The concept of bisimulation is used to prove the behavioral equivalence.

The approach is useful in testing the conformance of a black-box implementation of OSA SCS with respect to a specification, in the context of reactive systems.

Acknowledgement

The research is funded by the Project DO-02-135/2008, funded Bulgarian Ministry of Education, Youth and Science.

References

[1] 3GPP TS 29.198-5 "Open Service Access (OSA); Application Programming Interface (API); Part 5: User Interaction Service Capability Feature (SCF)", Release 9, v9.0.0, 2009.

[2] 3GPP TR 29.998-05-5 "Open Service Access; Application Programming Interface (API) Mapping for OSA: Part 5: User Interaction Service Mapping; Subpart 4: API to SMS Mapping", Release 9, v9.0.0,2009.

[3] T. Magedanz, M. Sher, "IT-based Open Service Delivery Platforms for Mobile Networks From CAMEL to the IP Multimedia System", in *Handbook of Mobile Middleware*, Auerbach Publishers, 2006.

[4] M. Wegdam, D. Plas, and M. Unmehopa, Validation of the Open Service Access API for UMTS Application Provisioning Proc. of PROMS 2001, LNCS 2213, pp. 210-221, 2001

[5] 3GPP TS 23.078, "Customized Applications for Mobile network Enhanced Logic (CAMEL) Phase 4; Stage 2", v10.0.0, 2010.

[6] T. J´eron, Symbolic "Model-based Test Selection",*Electronic Notes in Theoretical Computer Science*, 240, Elsevier, 2009, pp.167–184.

[7] X. Chena, R. Nicola, "Algebraic characterizations of trace and decorated trace equivalences over tree-like structures", *Theoretical Computer Science*, 2001, pp. 337–361.

.

Table 1.Bisimulation Relation between OSA User Interaction states and states of CAMEL models for SMS events

Transitions in T$_{AppUI}$	Transitions in T$_{OSMS}$	Transitions in T$_{TSMS}$
Null createUI Active Null reportEventNotification Active	SMSNull$_O$ InitialDPSMS$_O$ Start&Authorize$_O$	SMSNull$_T$ InitialDPSMS$_T$ Start&Authorize$_T$
Active sendInfoReq Active, Active sendInfoRes Active, Active sendInfoAndCollectReq Active, Active sendInfoAndCollectRes Active, Active sendInfoReq ReleasePending, Active sendInfoRes ReleasePending ReleasePending sendInfoErr ReleasePending, Active sendInfoAndCollectReq ReleasePending, ReleasePending sendInfoAndCollectRes ReleasePending, ReleasePending sendInfoAndCollectErr Active	Start&Authorize$_O$ RequestReportSMSEvent SMSAnalyze&Routing, SMSAnalyze&Routing FurnishChargingInfoSMSSMSAnalyze&Routing, SMSAnalyze&Routing ConnectSMSSMSAnalyze&Routing, SMSAnalyze&Routing ContinueSMSSMSAnalyze&Routing, SMSAnalyze&Routing EventReportSMSSMStart&Authorize$_O$	Start&Authorize$_T$ EventReportSMS SMSDelivery, SMSDelivery EventReportSMSStart&Authorize$_T$
Active release Null, Active sendInfoReq Finished, ReleasePending sendInfoRes Finished, Finished release Null	Start&Authorize$_O$ RequestReportSMSEvent SMSAnalyze&Routing, SMSAnalyze&Routing FurnishChargingInfoSMSSMSAnalyze&Routing, SMSAnalyze&Routing ConnectSMSSMSAnalyze&Routing, SMSAnalyze&Routing ContinueSMSSMSAnalyze&Routing, SMSAnalyze&Routing EventReportSMSSMSNull$_O$	Start&Authorize$_T$ EventReportSMS SMSDelivery, SMSDelivery EventReportSMSSMSNull$_T$
Active sendInfoErr Active, Active sendInfoAndCollectErr Active, ReleasePending sendInfoAndCollectErr ReleasePending, ReleasePending sendInfoAndCollectRes Finished, ReleasePending userInteractionAborted Finished, ReleasePending release Null, Active release Null, ReleasePending sendInfoErr Active, ReleasePending userInteractionAborted Finished, ReleasePending release Null, Active userInteractionAborted Finished, Finished release Null	Start&Authorize$_O$ RequestReportSMSEvent SMSAnalyze&Routing, SMSAnalyze&Routing FurnishChargingInfoSMSSMSAnalyze&Routing, SMSAnalyze&Routing ConnectSMSSMSAnalyze&Routing, SMSAnalyze&Routing ContinueSMSSMSAnalyze&Routing, SMSAnalyze&Routing EventReportSMSSMSNull$_O$, SMSAnalyze&Routing ReleaseSMSNull$_O$, SMSAnalyze&Routing DialogueAbortSMSNull$_O$, SMSAnalyze&Routing DialogueErrorSMSNull$_O$	Start&Authorize$_T$ EventReportSMS SMSDelivery, SMSDelivery EventReportSMSSMSNull$_T$, SMSDelivery ReleaseSMSNull$_T$, SMSDelivery DialogueAbortSMSNull$_T$, SMSDelivery DialogueErrorSMSNull$_T$