Verifying Multiparty Call in ATM UNI Signalling Protocol

GORDAN JEZIC and IGNAC LOVREK
Department of Telecommunications
University of Zagreb
Faculty of Electrical Engineering and Computing, Unska 3, HR-10000 Zagreb
CROATIA

GIRISH BHAT
Make Systems
4000, Regency Parkway, Suite 150
Cary, NC 27511-8502
USA

Abstract: - This paper presents formal specification and verification of the multiparty call in ATM UNI signalling protocol. The protocol specification is written in the CCS process algebra (Calculus of Communicating Systems). Verification is done by using the NCSU-Concurrency Workbench. Due to the complexity of the protocol, the model is decomposed into three components. The protocol is verified by the model-checking feature of the Workbench. Each of its components is checked for safety and liveness properties using temporal CTL (Computation Tree Logic) and modal mu-calculus logic. CSCC'99 Proc.pp.1271-1276

Key-Words: - formal specification, model checking, verification, ATM signalling protocol, process algebra

1 Introduction

A typical protocol development process involves two phases: the specification and design phase in which a high level design of the protocol is specified using a partially formal protocol description language and an implementation phase in which the high level description of the protocol is translated to an implementation which is tested extensively for bugs. The bugs in the final implementation could appear either because of implementation errors or design errors. Implementation errors can be found easily. Design errors, however, are not only subtle and hard to trace, but also quite exspensive to correct as they require going back to the design phase. Therefore, it is necessary to apply techniques that detect errors early in the design phase. One solution to this problem is to use rigorous formal methods in the design process.

This paper presents formal specification and verification of the multiparty call in ATM UNI signalling protocol. The protocol specification is written in the CCS process algebra (Calculus of Communicating Systems). Verification is done by the NCSU-Concurrency Workbench [4, 5].

The model of the protocol includes the specification of the procedures for point-to-multipoint connections and model includes three users: one calling and two called users. Two users enjoy the same rights: both can initiate the basic call (point-to-point) and have of a third party join the call.

Due to the complexity of the protocol, the model is decomposed [8-10] into three components. The first one covers all procedures for point-to-point call. The second includes the procedures for a point-to-multipoint call without clearing point-to-point connection. In that case, after setting up a point-to-point connection, only one additional user can be added to the connection. These two components represent two possible “flows” of the protocol, because after establishing a point-to-point connection, it is possible either to release a point-to-point call or to connect a new party to it. The third component consists of the procedures for a point-to-point and a point-to-multipoint connection with a restriction to one passive user. A passive user can only be a called user. The third component includes one calling, one called and one added user. Thus, the relations between the first two components are checked: the procedures of releasing a point-to-point call with the procedures for connecting and disconnecting an added user.

The protocol is verified by using the model-checking feature of the Workbench. Each component of the protocol is checked for safety and liveness properties using the temporal CTL (Computation Tree Logic) and the modal mu-calculus logic.

The paper is organised in the following way. Informal description of the model, model architecture and message flows for successful establishing and clearing of a multiparty call are presented in Section 2. Formalization of the protocol, components of the model as well as model assumptios are desribed in Section 3. Decomposition of the protocol and verification results are reported in Section 4. Conclusions are given in Section 5.

2 Model Description

The procedures for establishing and clearing of a multiparty call are defined by the User-Network Interface (UNI 3.1) signalling protocol [1]. It is assumed that there are three users or ATM end-stations, each connected to the ATM network through a user network interface (UNI). Two users participate in a point-to-point call and a third one can join the connection at the request of the calling (root) user (Figure 1). The UNI has two components: one defines the control procedures for the user side (USI) and the other defines the control procedures for the network side (NSI). The USI and the NSI are assumed to be connected by means of a reliable network and that the underlying ATM network is also reliable and error-free.

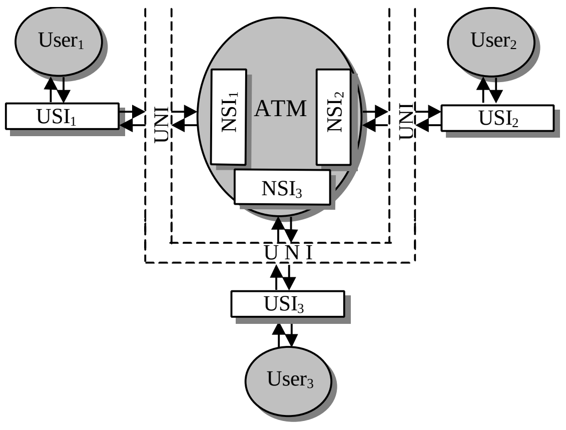
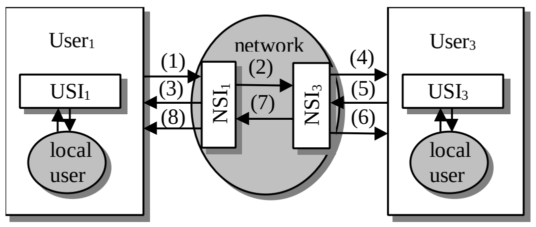


Fig. 1: Architecture of the model

A multiparty call is set up by first establishing a point-to-point connection between a calling and a called user. After this set up is complete, an additional user (party) can be added to the connection by an ADD_PARTY request from the calling user. A party may be connected to or disconnected from a multiparty call at any time after the connection is established. A party can be added to an existing point–to-point connection only via the calling user who issues an ADD_PARTY message, or be dropped from a connection at the request of either the calling user or himself (but not by the other user).

A typical execution sequence for a successful addition of a third party to an already existing (point-to-point) call is as follows (Figure 2).



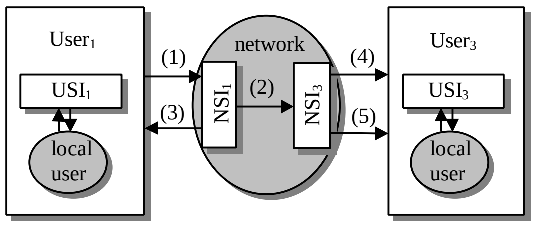
ADD PARTY	(1) (2)
CALL PROCEEDING	(3)
SETUP	(4)
CONNECT	(5) (7)
CONNECT ACKNOWLEDGE	(6)
ADD PARTY ACKNOWLEDGE	(8)

Fig. 2: Message flow for successful addition of a party

The term *User* refers to the combination of a local user and the USI. A calling user initiates the addition of a new user (party) and sends an ADD_PARTY request to the NSI_1. Index 1 refers to a calling side, index 2 to a called side and index 3 to a added (party) side. The NSI_1 forwards the request across the network and sends a CALL_ PROCEEDING message back to the calling user. When the NSI_3 (party side) receives the request, it forwards the request to the party as a SETUP message. If the party responds with a CONNECT message, the NSI3 responds with a CONNECT_ ACKNOWLEDGE message and sends a CONNECT message to the NSI_1. The NSI_1 then sends an ADD_PARTY_ACKNOWLEDGE message to the calling user. When the calling user receives this message, the multiparty call is established.

An execution sequence for dropping of a party from an existing multiparty call by the calling user is as follows (Figure 3).

The calling user indicates dropping of a party out by sending a DROP_PARTY message, which is forwarded by the NSI_1 to the NSI_3. The NSI_3 sends a RELEASE message to the party. The party responds with a RELEASE_COMPLETE message and when the calling user receives the DROP_PARTY_ ACKNOWLEDGE message, the clearing of the party is done.



DROP PARTY	(1) (2)
DROP PARTY ACKNOWLEDGE	(3)
RELEASE	(4)
RELEASE COMPLETE	(5)

Fig. 3: Message flow for dropping of a party out

3 Formal Description

Formal model of establishing and clearing of a multiparty call is written in the process algebra CCS (Calculus of Communicating Systems) [2, 3]. The terms in it are described by the following grammer (1).

$$P :: = nil \mid \alpha.P \mid (P + P) \mid (P \mid P) \mid P[f] \mid P \setminus L \mid proc\ C = P \quad (1)$$

"." represents the prefixing opeator; "+" is the summation of choice operator; "|" is the parallel composition operator; "f" is a relabelling function; "\" is the restriction operator, and "proc = " is used for defining a processes.

The model consists of 13 processes or components which are executed simultaneously; therefore they are connected by a parallel compostion operator. The model is added to the model for establishing and clearing a point-to-point call [7]. Given that only a calling user can add a party to the call, the model of a multiparty call must distinguish a calling and a called user. In this case, the model consists of three users. Two users have the same rights and both of them can initiate the point-to-point call and then can realise the multiparty call. A third user is a party user and can be only an added user. Thus, after establishing a point-to-point call, a calling user can either clear the call or initiate the procedures for establishing the multiparty call. If a calling user clears a point-to-point call, the model returns to the starting state. In the other hand, after having added a party to the call, either a calling or an added user can disconnect a party.

To establish a multiparty call, the model must have the following components:

1. the models of the local party, local calling and local called user,
2. the models of the USI on the party side, calling side and called side,
3. the models of the NSI on the party side, calling side and called side,
4. the models of UNI channels (between USI and NSI) on the party side, calling side and called side, and the network channels.

The components (models) of the calling and called side are equal these users having the same rights.

The model includes the following simplifying assumptions:

1. The channels are formalized as one-place buffers and all of them are reliable and error-free,
2. The contents (information elements) of a message are ignored,
3. The model includes point-to-point and point-to-multipoint calls, multipoint-to-multipoint calls are not considered and "calls" and "connections" are equivalent,
4. The model does not include timeouts and retransmissions.

4 Protocol Verification

This section presents some verification results. Verification is based on two important properties of the protocol: freedom from the deadlock and liveness. The first property is checked for the safety of the protocol. Checking of the second one proves that if nothing bad happens, a connection will be eventually established. The properties are verified using the model-checking feature in the Concurrency Workbench.

4.1 Model Decomposition

Due to complexity of the protocol, the model is decomposed into the following three components:

1. procedures for establishing and clearing of a point-to-point call,
2. procedures for establishing of a point-to-point call with the procedures for adding and dropping of a party,
3. procedures for establishing and clearing of a multiparty call with a restriction to one passive user in a point-to-point call.

The reason for protocol decomposition is a lack of memory and time wastage, given that over 40000 states and transitions are checked.

Decomposition of the model will be presented by messages which are important in particular components (key messages). The model for establishing and clearing of a multiparty call with the corresponding messages is shown in Figure 4.

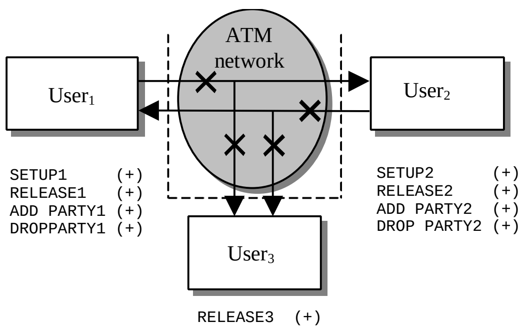


Fig. 4: Key messages for a multiparty call

Two users establish a point-to-point call (SETUP$_1$ and SETUP$_2$) after which a calling user may clear the call (RELEASE$_1$ or RELEASE$_2$) or initiate the procedures for adding a party (ADD_PARTY$_1$ or ADD_PARTY$_2$). Having added a party to the connection, a calling user or a party can initiate the procedures for dropping the party out (DROP_ PARTY$_1$ or DROP_PARTY$_2$ and RELEASE$_3$).

The first component covers all procedures for establishing and clearing of a point-to-point call (Figure 5). This component is chosen naturally, because all actions for a multiparty call are connected on point-to-point call.

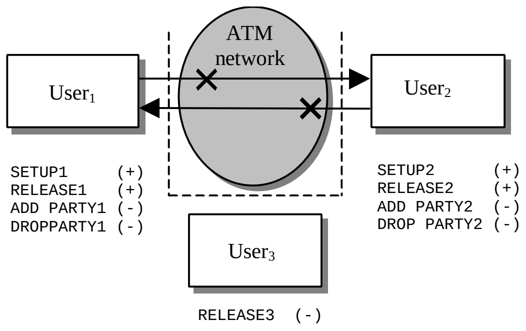


Fig.5: Key messages of the first component

There are two users that can establish (SETUP) and clear (RELEASE) point-to-point call. A party is not included.

The second component includes the procedures for establishing and clearing of a multiparty call without clearing a point-to-point connection (Figure 6). In that case, after setting up a point-to-point connection, only a party user can be added to the connection. These two components represent two possible “flows” of the protocol, because after establishing a point-to-point connection, it is possible either to release a point-to-point call or to add a party to the connection.

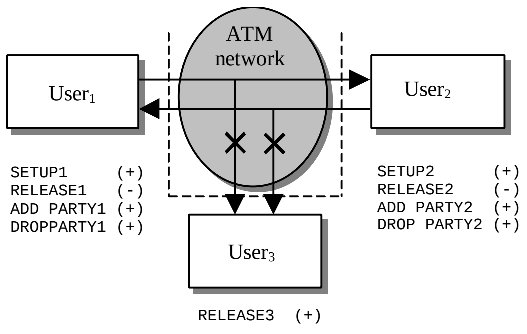


Fig.6: Key messages of the second component

There are two users that can establish (SETUP) a point-to-point call and after that a calling user can add (ADD_PARTY) a third user (party), but can not clear a point-to-point call (RELEASE). After adding the party, the procedures of dropping the party can initiate the party out (RELEASE) or the calling user (DROP_PARTY).

Third component consists of the procedures for a point-to-point and a point-to-multipoint connection with the restriction to one passive user. In Figure 7 *User*$_2$ is passive user and can be a called user only. This component defines the interrelations between the first two components and thus the procedures for clearing of a point-to-point call (RELEASE) with the procedures for establishing (ADD_PARTY) and dropping a party out (DROP_ PARTY) are checked.

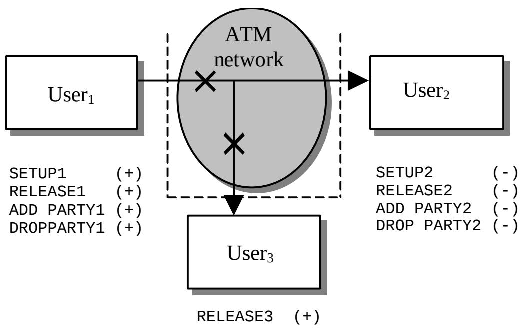


Fig.7. Key messages of the third component

There are three users: *User*$_1$ is a calling user, *User*$_2$ is a called (passive) user and *User*$_3$ is a party. After establishing of a point-to-point call (SETUP$_1$), a calling user can clear this call (RELEASE$_1$) or add a party to the connection (ADD_PARTY$_1$).

4.2 Model Checking

The properties (deadlock and liveness) are specified in a temporal logic, known as the modal mu-calculus [6]. The syntax of mu-calculus is given by the following grammer (2):

$$\Phi ::= tt \mid ff \mid X \mid \neg\Phi \mid \Phi \vee \Phi \mid \Phi \wedge \Phi \mid <\alpha>\Phi \mid [\alpha]\Phi \mid \nu X.\Phi \mid \mu X.\Phi \qquad (2)$$

The formula *tt* holds of every state, whereas the formula *ff* holds of no state. The formula $\Phi_1 \vee \Phi_2$ holds of a state if either Φ_1 or Φ_2 hold of the state; likewise $\Phi_1 \wedge \Phi_2$ holds of a state if both Φ_1 and Φ_2 hold of the state. α refers to actions. <> and [] are referred to as path modalities. The modal formula $<\alpha>\Phi$ holds of a state if the state has some α-derivative at which Φ holds, and $[\alpha]\Phi$ holds at a state if all α-derivatives of the state satisfy Φ. ν represents the greatest fixpoint operator (*max*) and μ represents the least fixpoint operator (*min*).

The mu-calculus is extended to include the CTL (Computation Tree Logic) operators [6]. The CTL formulas and corresponding mu-calculus translations that were used in the verification (3, 4):

$$\text{AG } prop \;\; \max X = prop \wedge [-\{\}]X \qquad (3)$$

$$\text{A } (prop_1 \text{ U } prop_2), \;\; \min X \;\; prop_2 \vee (prop_1 \wedge [-\{\}]X \wedge <-\{\}>tt) \qquad (4)$$

Intuitively, *A* means that the property should hold for all computations, *G* means “always” and *U* means “until”.

Each component of the protocol is checked for safety (deadlock freedom) and liveness properties. The first property says that the protocol is free of deadlocks. The following mu-calculus formula expresses this property (5):

$$can\_deadlock = \min X = [-\{\}]ff \vee <-\{\}>X \qquad (5)$$

The second property requires introducing some new visible actions in the model. These actions are used for checking of the model behaviour and the model progress as expected. The new models for the local *User*$_1$ and *User*$_2$ (these models are the same) and added *User*$_3$ are shown in Figure 8 and Figure 9. The visible actions are bolded.

Each component includes only its visible actions, characteristic for this component, which are interesting for checking. In this case, for each component some corresponding property is being checked. With respect to that, it is defined a macro (6) with parameters *a*, p_1 and p_2 which says that on all paths after an action *a* eventually a state satisfying p_2 occurs and until then every state satisfies p_1.

proc NullUser$_1$ =
'setup_req$_1$.**setup_request$_1$**.(setup_conf_USI$_1$.
NullAddPParty$_1$
+ setup_err_USI$_1$.**setup_error$_1$**.NullUser$_1$)
+ setup_ind_USI$_1$.('setup_resp$_1$.**setup_response$_1$**.
ActiveUser$_1$
+ 'setup_err1.**setup_error$_1$**.NullUser$_1$)

proc ActiveUser$_1$ =
rel_ind_USI$_1$.'rel_resp$_1$.NullUser$_1$

proc NullAddParty$_1$ =
'add_req$_1$.**add_request$_1$**.(add_conf_USI$_1$.
add_confirmation$_1$.ActiveAddParty$_1$
+ add_err_USI$_1$.**add_error$_1$**.NullAddParty$_1$)
+ rel_req$_1$.**release_request$_1$**.rel_conf_USI$_1$.
NullUser$_1$

proc ActiveAddParty$_1$=
'drop_req$_1$.**drop_request$_1$**.drop_conf_USI$_1$.
NullAddParty$_1$
+ drop_ind_USI$_1$.'drop_resp$_1$.NullAddParty$_1$

Fig. 8: Model of a local user (*User*$_1$ and *User*$_2$)

proc NullParty$_3$ =
setup_ind_USI$_3$.('setup_resp$_3$.**setup_response$_3$**.
ActiveParty$_3$
+ 'setup_err$_3$.**setup_error$_3$**.PocDodKor$_3$)

proc ActiveParty$_3$ =
rel_ind_USI$_3$.'rel_resp$_3$.NullParty$_3$
+ 'rel_req$_3$.**release_request$_3$**.rel_conf_USI$_3$.
PocDodKor$_3$

Fig. 9: Model of a party (*User*$_3$)

$$liveness\ (a, p_1, p_2) = \text{AG } ([a]\ (\text{A}(p_1 \text{ U } p_2))) \qquad (6)$$

In the first component, a liveness of the procedures for establishing (7) and clearing (8) of a point-to-point call are checked.

prop liveness_setup =
liveness (setup_request$_1$, ([setup_error$_1$]ff $\wedge$ [setup_error$_2$]ff), <setup_response$_2$>tt) (7)

prop liveness_release =
liveness (setup_response$_2$, ([setup_request$_1$]ff $\wedge$ [setup_request$_2$]ff), <release_request$_1$>tt) (8)

In the second component, liveness of the procedures for adding (9) and dropping (10) of a party and a liveness of the procedure for establishing of a point-to-point call (7) too are checked.

prop liveness_add =
liveness (add_request$_1$, ([add_error$_1$]ff /\ [setup_error$_3$]ff), <setup_response$_3$>tt) (9)

prop liveness_drop =
liveness (add__confirmation$_1$, ([add_request$_1$]ff U, <drop_request$_1$>tt \/ <release_request$_3$>tt (10)

In addition checking (7) and (8), the third component includes checking of liveness when a calling user can concomitantly add a party or release a point-to-point call (11) (interrelation between first two components).

prop liveness_add_release =
liveness (setup_response$_2$, ([add_error$_1$]ff /\ [release_request$_1$]ff), <setup_response$_3$>tt) (11)

For example, the formula (9) says that if after *add_request*$_1$ nothing bad happens (i.e. events *add_error*$_1$ and *setup_error*$_3$ do not occur), a *setup_response*$_3$ takes place.

For the last property the model had 8833 states and 30186 transitions. Verification of the properties took about 3 minutes on a SUN Ultra 1 workstation with 256 Mbytes of memory.

5 Conclusion

Formal specification and verification of the multiparty call in ATM UNI signalling protocol are reported. The specification is written in the CCS process algebra and verification is done by using model-checking feature of the Concurrency Workbench. Due to complexity of the protocol, the model is decomposed into three components and each is checked for safety and liveness properties using temporal logic CTL and modal mu-calculus logic.

In the protocol design, there was a constant interaction between the modelling and verification phases. Therefore, in reality, the border between these two phases is being lost.

Further studies would be a more detailed modelling including timeouts and retransmissions, modelling of more complex protocols with more users, possibility of negotiations and mobile agents.

References:

[1] The ATM Forum, *ATM User-Network Interface (UNI) Specification, Version 3.1*, Prentice-Hall International, 1995.

[2] Milner, R., *A Calculus of Communicating Systems*, Springer-Verlag, 1980.

[3] Milner, R., *Communication and Concurrency*, Prentice-Hall International, 1989.

[4] Cleaveland, R., Parrow, J., Steffen, B., The Concurrency Workbench: A Semantics-Based Tool for the Verification of Concurrent Systems, *ACM Transactions on Programming Languages and Systems*, January 1993., Vol. 15, No. 1, pp. 36-72.

[5] Cleaveland, R., Sims, S. T., Generic Tools for Verifying Concurrent Systems, *Proceedings of the 2nd International Workshop on Applied Formal Methods in System Design*, Zagreb, June 1997., pp. 3-8.

[6] Cleaveland, R., Sims, S., *The Concurrency Workbench of North Carolina*, *User's Manual*, September 1996., North Carolina.

[7] Bhat, G., Cleaveland, R., "Verifying the ATM UNI 3.1 Signalling Protocol, *Personal communication*, 1997.

[8] Holzmann, G. J., Protocol Design: Redefining the State of the Art, *IEEE Software*, January 1992, pp. 17-22.

[9] Lin, F. J., Liu, M. T., Protocol Validation for Large-Scale Applications, *IEEE Software*, January 1992, pp. 23-26.

[10] Hailpern, B. T., Owicki, S. S., Modular Verification of Computer Communication Protocols, *IEEE Transactions on Communications*, January 1983., Vol. 31, No. 1, pp. 56-68.