INSTITUT FÜR INFORMATIK UND PRAKTISCHE MATHEMATIK

LEHRSTUHL FÜR SOFTWARETECHNOLOGIE

Verifying Mascara Control

Natalia Sidorova
Martin Steffen

Bericht Nr. TR-ST-00-1
May 2000

CHRISTIAN-ALBRECHTS-UNIVERSITÄT ZU KIEL

Verifying Mascara Control

Natalia Sidorova[1] and Martin Steffen[2]

[1] Department of Electrical Engineering
Eindhoven University of Technology, The Netherlands
natalia@ics.ele.tue.nl
[2] Institut für angewandte Mathematik und Informatik
Christian-Albrechts-Universität
Preußerstraße 1–9, D-24105 Kiel, Deutschland
ms@informatik.uni-kiel.de

Abstract. This document reports on a series of verification experiments on the control-part of Mascara, a medium-access protocol for wireless ATM-networks.

1 Introduction

1.1 Mascara control

Let's begin surveying quickly tasks and structure of Mascara control. For a more thorough picture of the protocol, consult the specification report [4] or the official Wand documents [12, 6, 5].

As the name suggests, the *Mascara control* entity (MCL) is the part of Mascara responsible for the protocol's control and signalling tasks. It uses the services of the underlying segmentation and reassembly entity, the sliding-window entities (WDLC's), and in general the low-layer data-pump. In turn, the control layer offers its services to the ATM-layer above.

Mascara-control falls into four subentities, each divided in various sub-processes themselves. The two important and complex ones are the *dynamic control* (DC) and the *steady-state control* (SSC). Th two of minor complexity are the radio control entity (RCL) and the generic Mascara control (GMC). We will concentrate on the first two.

Mascara-control takes care of *associations*, connecting access points to mobile terminals, and *connections*, i.e., the basic data and signalling transfer channels. Both are managed by MCL either in response to requests from the upper layer or by taking initiative of its own.

The division of work between the dynamic and the steady-state control is roughly as follows: SSC monitors in various ways current associations and the quality of the radio-environment, in order to initiate in time change of associations, so-called handovers. The dynamic control's task, on the other hand, is to set-up and tear down the associations and connections, while managing the related administrative work like address-management, resource allocation, etc. Figure 1 gives an overview over the static process structure and the signal connections at a mobile terminal of the model we used in the verifications to follow.[1] The counterpart at the access point is similar.

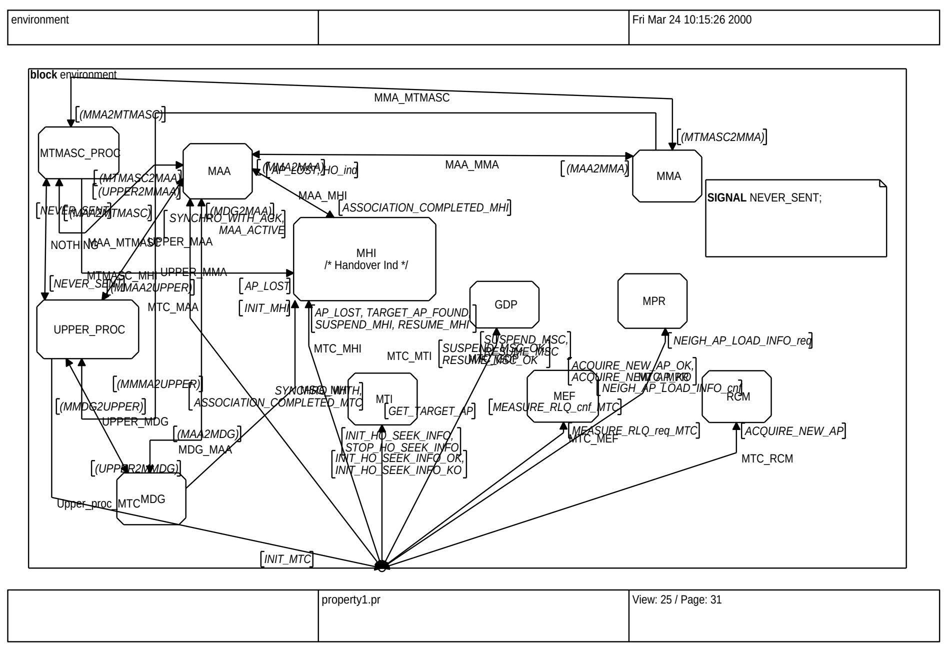

Fig. 1. Mascara-control processes and signals (MT-side)

1.2 General approach

In a series of experiments of increasing complexity, working bottom-up, we investigate the dynamic control resp. steady-state control in isolation, a combination of steady-state and dynamic control for a one-sided configuration, and a full configuration for one MT and one AP. As starting point for each of the verification experiments, we used the common, global

[1] To be precise: the picture shows part of the structure for the "MT-side-only" configuration. The process structure for the other configurations is similar and omitted here.

SDL-specification [3] with an appropriately chosen and adapted part of the control-entity. The model is closed by an environment representing the rest of Mascara and the upper layer and issuing requests to the dynamic control, where the form of the abstraction and the environment depends on the entity under investigation.

Besides the simplifications and abstractions described more concretely below, we stuck to Mascara's overall model [3] developed in the specification part of Mascara as close as possible. This especially means that we kept the process and interfaces structure of [3], but in general simplified heavily on the *data* stored and transmitted and consequently also on the parameters transmitted on the signals. In other words, a principal, general simplification we used was *data abstraction*.

To model-check various properties, we used the *Spin* model-checker [8] [9], respectively its discrete-time extension *DTSpin* [7] [2] developed in Eindhoven within the Vires project. To feed the model into *Spin*, the SDL-code is translated in two steps:

1. translating SDL to the intermediate format using *sdl2if* [13], and
2. translating the result into *Spin*'s input language Promela, using *if2pml* [10].

In case of an error found or in case of the reachability checks, we tried to produce the *shortest* trail witnessing the error or reaching the chosen control-point. In some cases, especially for the later, more complex models and properties, minimizing the trail with *Spin* was too time-consuming; in those cases we manually reduced the depth of *Spin*'s seach-space in a few steps until we found a trace of tolerable length.

1.3 Properties

The exact properties for the different configurations are given and explained in the corresponding sections. When dealing with the various set-ups, we basically followed a bottom-up approach not only proceeding from smaller entities to larger, combined ones, but also advancing from simpler to more complex properties. In all cases we routinely started with just checking rather simple *reachability* properties, basically check-pointing various crucial steps in the protocols of interest to see whether they are reachable at all. We nevertheless checked them, to make sure that the more complicated properties listed later are not trivially satisfied. The reachability checks are very easily and quickly done by just checking *assertion violations* (or, a little more complicated, checking LTL-formulas

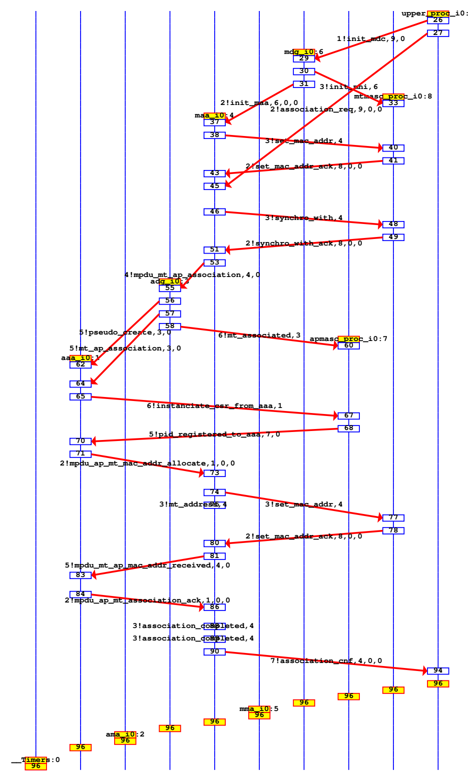

Fig. 2. Successful association

in *Spin* marking the property of interest as "undesired", while hoping for their satisfaction and the corresponding error traces). In this way we found some "obviously reachable" states being unreachable and thus a couple of unexpected errors of various kinds. Moreover, the resulting traces are rather useful illustrating characteristic desired scenarios, such as association setup, connection setup, etc.[2] giving a better understanding of the behavior of the protocol enhancing the confidence in our models (and our translations, for that matter...). We found it quite useful to compare the traces with the message sequence charts developed by the Wand-consortium [15].

[2] Cf. Figure 2 for one simple example, the complete set of such pictures is accessible via the corresponding web pages.

After a few of the reachability checks, we in general proceeded to more advanced properties, like safety or liveness properties. Depending on whether the property was found to hold or not, it is listed as error or a verified property. In the error-cases, we will discuss if the error shows up in the global model as well.

1.4 General simplifications and abstractions

Mascara-control is a sublayer of Mascara, located on top of a few lower layers. More specifically, MCL relies on

- control-segmentation and reassembly,
- wireless data-link control, and the
- MAC data pump

as lower entities of the protocol stack to perform its task. In our model, all peer-to-peer communication of the dynamic control entities has been replaced by *direct, buffered communication* between the entities, either reliable or lossy. In case of lossy channels, the scheduler as part of the data pump, whose internals are not available to us, is abstracted into fairness assumptions wrt. to loosing messages.

1.5 Further information

This part of the verification document parallels the material at the Vires verification web-pages, offering a reorganized and more distilled picture, while the net resources contain the complete verification material, including the source code of the models, version information about precise changes and repairs in the model during the verification effort, resource consumption, and, if appropriate, graphical representations of error traces encountered. Due to limitations of space (and general interest), we don't include all of this microscopic information into the report, but refer the tireless reader to

`http://www.informatik.uni-kiel.de/~seriv/Deliv/Veri`

1.6 Structure

In the following Sections, we summarize the results for three configurations and abstractions of increasing complexity: first, Section 2 and Section 3 discuss a configuration with concerning an abstract version of steady-state control, concentrating on its most complex component, the

MT target cell and the dynamic control part, afterwards a combination of dynamic and steady-state control, first for a one-sided configuration, only (Section 4) and afterwards modelling both AP's and MT's side of the protocol (Section 5). We conclude in Section 6.1 summing up the experiences and discussing the relevance of the experiments.

2 Steady-state control

We did a series of verification experiments, using tools *sdl2if*, *if2pml*, *Spin*, *DTSpin*. At first, we started with two modest goals:

- to try the combination of all the tools involved in the verification process,
- and to check the limits of the verifiers (*Spin* and *DTSpin*).

Later it turned out that we were able to achieve more than we had expected. In fact, we were not only able to go through the whole verification process on just a small example, as we intended initially, but we also managed to expose some bugs in the tools and several errors in the Steady State Control specification (i.e., its SDL model).

2.1 Experiments

We have done about 20 verification experiments whose complexity increased progressively. All of them were devoted to analysing the behaviour of MTC — the most complex component of the Steady State Control module. We went through the following phases.

First, we wanted to obtain the Promela model of MTC, by pushing the SDL specification through the pipe of translators *sdl2if* and *if2pml*. This worked quite smoothly, after we managed to get a syntactically correct SDL specification of MTC. During this phase, some minor bugs were exposed in *sdl2if* and *if2pml*, but these were easy to correct.

Second, the SDL specification of MTC had to be closed by providing an environment. We hand coded a greatly simplified environment directly in Promela. This consisted in observing that MTC communicates with its environment by several variants of simple request/confirm protocols which were coded as separate Promela processes. The verification experiments performed during the second phase were mostly inconclusive, due to the fact that *Spin* could not enumerate the state space (even with 2048MB of memory). However, one of the experiments exposed a bug in *Spin*. This was reported to Gerard Holzmann who kindly corrected it.

The state explosion problem was caused, in our opinion, by our initial environment being too nondeterministic, i.e. too abstract. But even this simple abstraction allowed to expose a deadlock (described later).

Third, we decided to build a less abstract environment by replacing our simple Promela processes with the SDL models of all the Steady State Control components which MTC relies on. Of course, we had to slightly modify the components in order to abstract out the real behaviour of other components on which the whole Steady State Control relied, in turn. After this phase we had quite a faithful Promela model for the whole Steady State Control, obtained automatically, via *sdl2if* and *if2pml*. The verification experiments performed during the third phase were more successful (described later). Also, another bug was exposed in *Spin*.

We started using *Spin* during the second and third phase while *DT-Spin* was used during the third phase only.

2.2 Results

First deadlock An experiment performed in the second phase exposed the following deadlock scenario:

During the initialization phases, Gen_MC_proc sends signals INIT_MDC to start initialising MT Dynamic Control and INIT_MSS to start initialising MT Steady State Control (both action are performed together in an atomical way). After receiving them, MDG (in MT Dynamic Control) and MSG (in Steady State Control) initialise other entities of MT Dynamic/Steady State Control.

MAA is initialized by MDG and sends SYNCHRO_WITH to MTC at the moment when MTC is not initialised yet. The only signal MTC is waiting for is INIT_MTC. So it discards this SYNCHRO_WITH, and then it gets and consumes INIT_MTC. Now MTC is waiting for SYNCHRO_WITH from MAA while MAA is waiting for SYNCHRO_WITH_ACK from MTC, resulting in a deadlock.

This error was removed by enforcing a proper initialization order between MAA and MTC. It was obtained by sending a 'synchronising' signal from MTC to MAA.

Second deadlock An experiment performed in the third phase exposed another, and this time more complex, deadlock:

The deadlock scenario actually starts when after a normal initialisation stage, MTC sends INIT_HO_SEEK_INFO to MTI. After that, MTC goes to the Associated state again and receives GET_TARGET_AP(backward_ho) from MHI. They perform the backward handover procedure at the end of which MTC is in Associated state again while MHI is in non_assoc.

Now it is supposed that MTC, MAA and MHI will go through the SYNCHRO_WITH + ASSOCIATION_COMPLETED stage, but instead of that, MTC gets the reply INIT_HO_SEEK_INFO_OK as an answer to its previous request to MTI and leaves the Associated state (MTC is inside the TIP procedure now). As a result of that, signal SYNCHRO_WITH, which is received somewhere in between, will be discarded. Consequently, SYNCHRO_WITH_ACK will not be sent back to MAA, thus MAA will be blocked and it can send ASSOCIATION_COMPLETED neither to MTC nor to MHI. Thus MHI will be blocked too and the deadlock occurs.

This error was corrected by informing MTI about the handover request.

Third deadlock Next, we considered a simplified model of the MTC environment allowing to initiate a forward handover procedure only to MTC not to MPX.

After enumerating the state space, *DTSpin* reported unreachable code:

```
line 323, state 163, "the_channel = i"
line 324, state 164, "q_rcm_0!acquire_new_ap,_pid,the_channel"
```

and

```
line 569, state 373, "ap_mac_addr = i"
line 570, state 374, "q_mhi_0!target_ap_found,_pid,ap_mac_addr"
line 571, state 375, "beacon_received = 0"
```

From this, it can be deduced that the forward handover never takes place. On the other hand the part of the code where AP_LOST was sent by MTC to MHI was reachable. Moreover this signal was eventually consumed by MHI and it started the forward handover procedure.

Now we reconstructed with Spin such a scenario (which turned out to be a deadlock) by putting assert(false) after consuming AP_LOST in MAA.

After some normal activity MHI initializes a backward handover. MTC successfully finds the new AP and tries to get back the association to its old AP for the period it will be waiting for the association with the new AP. However, the old AP has been lost, so MTC receives ACQUIRE_NEW_AP_KO being in the return_to_old_ap state. According to the code, MTC sends AP_LOST to MHI in order to initiate a forward handover, sets Beacon_received into false and goes to Non_Associated state. MHI forwards AP_LOST to MAA.

As the result we have MTC in the Non_Associated state, where it waits only for SYNCHRO_WITH from MAA; MHI in find_target_ap waiting for TARGET_AP_FOUND from MTC (or one more AP_LOST); and MAA in the state where it waits for a signal from Ccon. On the other hand, Ccon will send this signal only after receiving HO_IND from MHI. So we have a classical deadlock situation because of a circular waiting.

Analysis of the code showed that there is an error caused by mixing the situation when MT tries to get back the connection to the old AP after the TIP procedure and after the backward handover procedure. Any unsuccessful attempt to get the connection back after the TIP procedure led to deadlock too. We corrected the code to avoid deadlocks.

Functional property Finally, we decided to check some functional property of MTC. As was already mentioned, MTC communicates with its environment using several simple request/confirm protocols where the request should alternate with the confirmation. (In fact, there are several variants of this protocol, depending on whether the confirmation is a double success/failure signal, a single success signal, or no signal at all, in which case it must be guarded by a timer.)

In particular, MTC uses this alternating pattern when it communicates with GDP, in order to temporarily suspend MSC and later resume it. Since the suspend and resume messages are sent in separate parts of the SDL specification of MTC, it is not obvious, just by inspecting the SDL code, whether the suspend and resume messages alternate. A simple experiment indeed confirmed that this is always the case.

We first tried to encode this simple safety property as a so-called trace observer

```
trace {
  do
  :: gdp!suspend_msc -> gdp!resume_msc
  od
}
```

but this experiment exposed a bug in *Spin*. So the property was instead encoded as a direct observer, via `assert` statements embedded directly in the Promela code (just before sending the relevant messages), and *Spin* could enumerate the whole state space checking that the property is not violated.

In this experiment, *Spin* reported the following statistics:

```
State-vector 416 byte, depth reached 3450, errors: 0
   55959 states, stored
23.727MB memory usage for states
25.582MB total actual memory usage
```

which should be read in the following way:

`State-vector 416 byte` is the memory needed to represent one state.

`55959 states, stored` is the number of different states found in the model (all the states must be kept during the state space enumeration, so 23.727MB memory was needed for states).

`depth reached 3450` is the longest acyclic path through the model (since *Spin* must keep a state stack of at least this depth, about 1.7MB was needed in addition to the state memory).

As a conclusion, it is quite likely that with our 2048MB memory we will be able to handle a model about 75 times bigger!

False errors Since we worked with an abstraction of the Steady State Control, the errors reported by *Spin* (i.e., the erroneous traces) had to be always interpreted in the context of the whole SDL specification for MASCARA, to single out so-called false negatives.

For example, one of the verification experiments found a livelock caused by RCM: MAA sends SYNCHRO_WITH and this triggers MTC to send ACQUIRE_NEW_AP to RCM, to which RCM in turn replies with ACQUIRE_NEW_AP_KO, so MTC answers the initial SYNCHRO_WITH with SYNCHRO_WITH_ACK(fail), and this can be repeated again.

After analysing the SDL code, it turned out that this potential livelock is allowed by the MASCARA protocol.

Currently, the often tedious and time consuming task of interpreting an erroneous trace cannot be automated since it would call for a separate tool able to convert the *Spin* error report to an input of the simulation tool for SDL.

As standard *Spin* does not support modeling time, in the experiments described in the previous section we had to use an abstracted version of timers and timer operations, based on non-deterministic timer expirations. This implied that in our system we could consider time only in a qualitative way.

In order to deal with the real-time aspects in a quantitative manner, in the third phase of our experiments we used the discrete-time extension of *Spin*, *DTSpin*, developed in the framework of the VIRES project. We repeated with *DTSpin* all our experiments using the values of the timing parameters as they were defined in the SDL specification. The outcome was essentially the same as with the standard *Spin*, i.e. we detected the same deadlocks and we were able to verify the same aforementioned functional property. Moreover, it turns out that the introduction of time did not significantly increase the state space of the verified model (less than one order of magnitude).

3 Dynamic control

The dynamic control forms one large part of MCL, its basic task is to set up and tear down *associations* and *connections*. It has an upper interface to the control layer on top of Mascara. The peer-to-peer signals are sent to and received from the control segmentation and reassembly for the actual transmission. It exchanges control signals with many other entities of the Mascara-protocol.

3.1 Simplifications and abstractions

To tackle properties of the dynamic control independently of the rest of the protocol, the relevant entities had to be cut-out of the global protocol. The dynamic control relies on

- control-segmentation and reassembly,
- wireless data-link control, and the
- MAC data pump

as lower entities of the protocol stack to perform its task. In our model, all peer-to-peer communication of the dynamic control entities has been replaced by *direct, reliable communication* between the entities. From the entities of the dynamic control, only the

1. generic dynamic agents, the
2. association agents, and the
3. MVC agents

are included. This means especially, the handover indicator, nominally part of the dynamic control, has been left out in this series, since its behavior is more closely related to the task of the steady-state control and is treated together with that part. So the rest of Mascara control including handover indicator has been abstracted away and replaced by a more or less chaotic environment. Creation and termination of processes, an important part of the behavior of the dynamic control, has been replaced by statically starting a fixed number of instances and simulating creation and termination by appropriate signaling. This was motivated by the lack of process creation in the IF-language.

Figure 3 gives a global overview about the configuration, where the lower-layers have just been replaced by a direct connection between the two sides and where the rest of Mascara, especially the steady-state control, has been replaced by abstract versions (`AP_MASC` and `MT_MASC`). The model has been closed by adding a separate environment process (entity `UPPER` of the Figure).

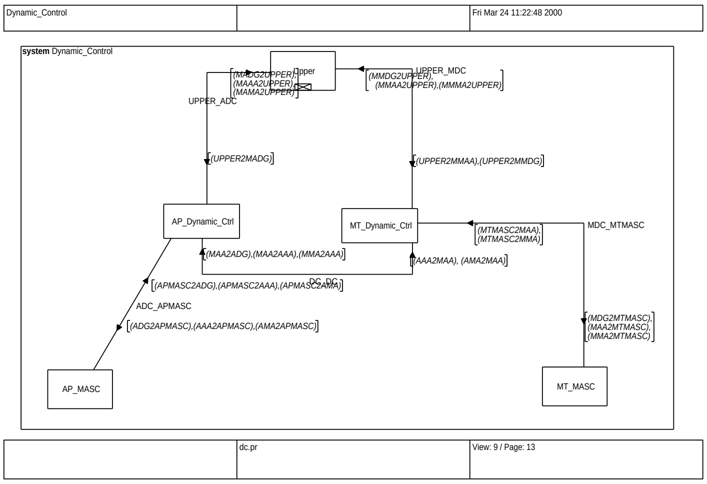

Fig. 3. Dynamic control global model

Model As sketched in Figure 3, the SDL-model of the dynamic control, fed into the translators from SDL to Promela, consists of the following parts:

- common upper layer
- AP side
  - dynamic generic agent
  - association agent
  - MVC agent
- MT side

    - dynamic generic agent
    - association agent
    - MVC agent
  - underlying medium, assuring reliable, peer-to-peer communication using direct SDL-channels.

The static structure of the processes ADG, AAA, and AMA is shown in Figure 4, the counter-part at the mobile terminal is (almost) symmetric. As usual, the complete code can be obtained from the web resources bundling the verification results.

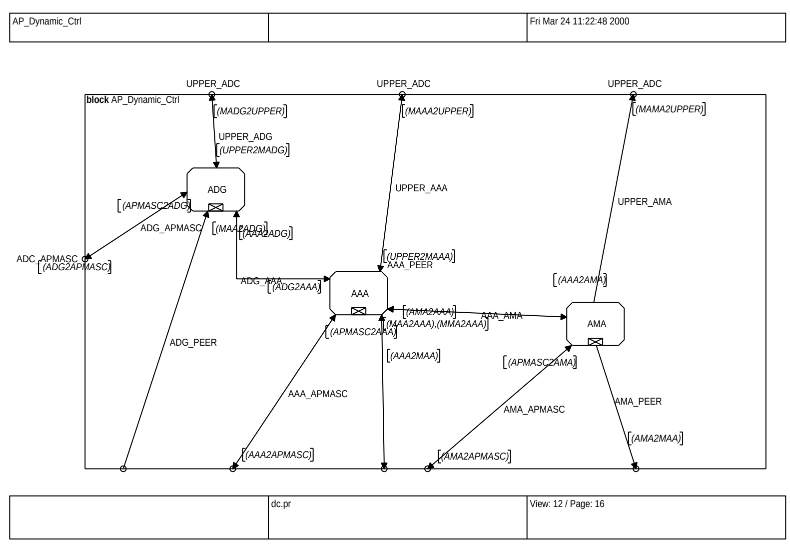

Fig. 4. Dynamic control processes, AP-side

For a beginning, we choose to start with the additional simplifying assumption, that there is only *one MT* and *one AP*. This allowed a large reduction in the state space, since in this way, the different *addresses* to be stored and transmitted can be completely abstracted away. Note that with one AP only, a proper handover cannot be directly modeled and verified, but note that *handover* is not a service of the dynamic control, but consists of an appropriate sequence of deassociation and association step, the signaling each of which *can* be modeled using this simplified setup. What *cannot* be modeled is the correct calculation of the addresses, but this is the task of the data procedures and independent of the signaling, on which we chose to concentrate here.

3.2 Results

Table 1 collects the corresponding verification results. The verification efforts concentrated on the proper functioning of the association-related functionality of dynamic control, i.e., neglecting the connection-oriented features for a start. The latter are modeled in the abstraction, but neither mentioned in the properties and nor stimulated by the environment. Hence, they do not contribute seriously to the state-space.

As described in Section 1, we started with simple reachability properties.

Reachability checks	
1. AAA associated	partial association
2. MAA associated	a successfully completed association
3. terminating an association	after AAA has been notified of a new association, something goes wrong at the MAA side
Errors	
1. race condition (1)	association request too early
2. race condition (2)	association request still too early
3. unspecified reception	`MT_Deassociation_cnf` has 2 possible receivers
4. unspecified reception	In state `aaa.wait_addr_received`, signal `Association_ind` has two possible receivers
Verified properties	
1. association/deassociation	Correct handling of association failure at the MT side. If for some reason, the MT decides a power-on handover should not be continued, then also the corresponding AAA will terminate, if it already had taken notice of association being built up.
2. no unsolicited MT-association	If the MT is associated, reported by `Association_cnf` to the upper layer, the AP must reported the successful association of the new MT before (by `Association_ind`)

Table 1. Verification results for dynamic control (association handling)

Reachability Concentrating on properties of the association handling, we check-pointed the crucial stages in the corresponding signal-exchanges. The table contains the stages, where first, the AP-side has all the necessary information, where secondly the MT-side has it, too, thus completing an association,[3] and thirdly a previously established association

[3] Cf. Figure 2 on page 4 for an example, further scenarios are available electronically at Vires' web resources.

is revoked by some signal at the AP or at the MT side. Basically, the first two scenarios are intended to capture the normal course of behavior of the association-related services, and thus allowed to compare the the resulting scenarios with the ones provided by industry [15].

Errors *Spin*, in general, found the counterexamples rather quickly. In the examples below, we show the ones with the shortest error trail.

The properties are simple untimed properties. To avoid an infinite state space, we cannot use an completely chaotic environment —this would fill SDL's input queues beyond any bound; instead we have the environment generate only a finite amount —typically a small number— of signals. To obtain the results, we manually repaired the errors detected earlier in the SDL-code and also partially in the Promela-code.

The errors encountered fall into two categories: *race conditions* and *unspecified receptions*. In the first case, signals are received in an unexpected order, such that one of the signals is just ignored, leading to a deadlock. Figure 5 shows one simple instance of this kind: during initialization at the MT-side, dynamic control is initialized and immediately afterwards, an association request is issued.

In the sealed-off and abstracted model of dynamic control used here, the signals are sent by the *upper* process used to model the environment, including the rest of Mascara's control. In the global Mascara model [3], the routing of the signals is slightly more complex than shown here: the signal `Association_req` is sent from the upper layer to MAA as shown, but `Init_MDC` actually is sent from Mascara generic control to dynamic generic control in reaction to a request to open the MAC-layer. This opening request comes from the upper layer and `Init_MDC` is issued to MDC after the radio-control manager is running, which is not modeled here. Nevertheless, as there is no confirmation to the upper protocol layer that the MAA is indeed running, there's nothing in the global model, as well, to prevent the upper-layer from sending the association request, while the MAA is not ready to receive it.

The verification gave various variants of the scenario of Figure 5. We expect that more instances of this error can be found in the dynamic control, for example for connection setup or for the interactions with the rest of Mascara, which we abstracted away in the current model. This kind of racing condition leading to unexpected reception of signals, especially in the initialization phases, has also been reported earlier in the effort to verify the *steady-state control* entity (cf. Section 2).

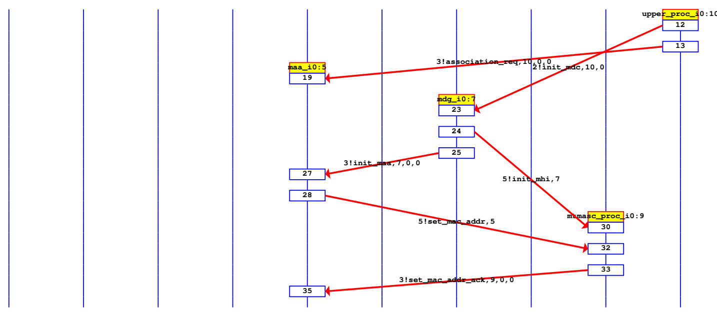

Fig. 5. Association request too early

The second type of error is caused using signals where *more than one receiver* is possible.

One such error is shown in Figure 6. It invalidates the *liveness-property*, expressing that deassociation is a confirmed service of Mascara control to the upper layer:

> If the upper layer at the MT-side requests a deassociation, eventually the deassociation is confirmed.

The trace shows that this is not the case. The error occurs, when a half-established association, i.e., when after sending the first handshake, MAA has reached state *associating_wo_addr*. At this point, the association is already being revoked by the upper layer, sending `MT_Deassociate` to the dynamic generic control. MAA should then, after a few steps, confirm the deassociation, but since the corresponding signal `MT_Deassociation_cnf` has two possible receivers, the signal is not translated, and in the trace shown the signal is missing.

Another bug of this kind invalidates the *safety property,* which could be dubbed *"no unsolicited MT-association":*

> If the MT is associated, the upper layer of the AP must have been informed by the new MT before.

Figure 7 shows the violation.[4] After receiving the 3rd association handshake of AAA, the agent should signal `Association_ind` to the upper layer and complete the set-up by giving the 4th handshake back to MT. In state *wait_addr_received* of AAA, `Association_ind` has two possible

[4] The shown part of the trace starts shortly before the second association handshake only.

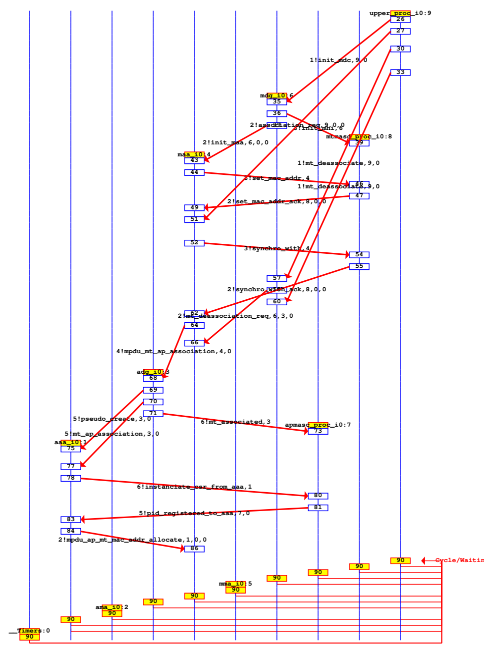

Fig. 6. Deassociation confirmation not sent at MT

receivers and therefore is not being translated. Hence the signal is missing in the scenario and model-checking reports an error.

Since the signals and signal-lists of the abstracted version are just the ones from the global model (except simplified parameters), those errors appear in the large model as well. Indeed, considering the amount of overall signals, we expect more such errors lurking in the whole protocol.

While in principle, OBJECTGEODE is smart enough to catch mis-specifications like "ambiguous receiver" and often does (for instance also for signals which have no possible receivers), sometimes, as in these two cases, they slip through the checks that the OBJECTGEODE offers. The fact that *sdl2if* refuses to translate signals with ambiguous received shows that the detection is possible at compile time and using the —undocumented— *sdl2if*-option -S one can even get a log-output ...). Which errors go unnoticed and under which conditions is not transparent to us.

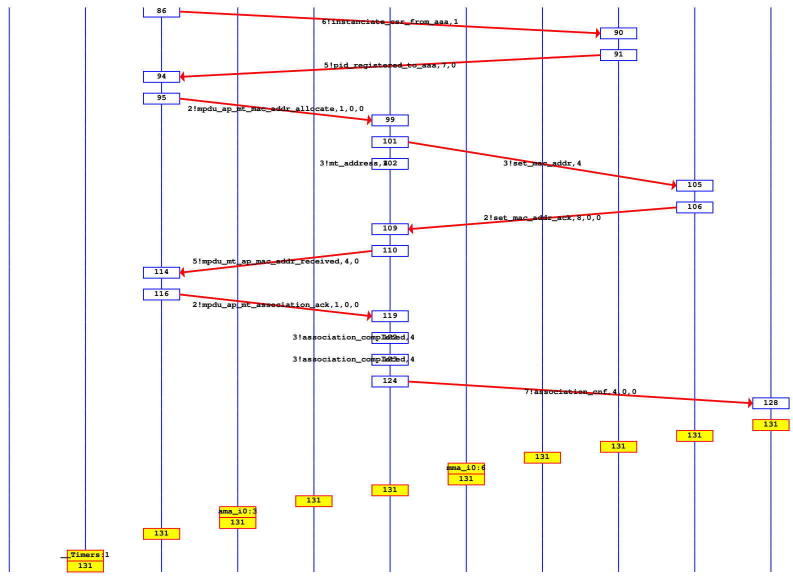

Fig. 7. Association indication not sent at AP

Verified properties While the reachability analysis of above mainly checks various standard association-related scenarios of dynamic control, which could be compared with [15], the interesting and error-prone behaviors in this part deal with the *interference* of these services. Indeed, much of the complexity of the control structure of dynamic control is due to the fact, that requests concerning services offered to the above layer must be dealt-with gracefully at any point in time, leading to consistent views about the status of the association in the entities involved. The properties (and the environment) we chose are of that kind, namely that the *interference of association set-up and deassociation* is handled properly.

The first property could be called a *"correct handling of initial association failure"* at the MT side. More specifically, it tests, whether not the AAA will falsely consider an MT to be associated, if during a power-on handover, i.e., during the very first association signal exchanges, a failure occurs. Setting φ_1 as "reception at AAA of the first association handshake", φ_2 as "failure reported at MAA", and φ_3 termination of AAA, the verified property LTL-property reads:

$$\varphi = \Box(\varphi_1 \rightarrow \Box(\varphi_2 \rightarrow \Diamond\, \varphi_3)).$$

Proposition φ_1 denotes the point in time, the association agent at the AP-side is informed for the first time about the intention of the MT to set-up a new association; before that point, the corresponding association agent does not exist. The error-reporting proposition φ_2 is set to true at MAA, if for any kind of reason, the association set-up is disrupted or, once established, terminated again. A number of failures and signals can trigger this event at MAA: 1) reception of `AP_Lost` when the radio connection breaks down, 2) revoking of the association by the upper layer with the signal `MT_Deassociation_req`, 3) failure at MTC (and lower layers) to synchronize or re-synchronize with the AP, for instance after an incommunicado phase[5], 4) the maximal number of fruitless MAA-attempts to associate with the AP is exceeded, 5) the AP turns down the association request (either using the second or the fourth handshake), or finally 6) the fourth handshake from the AAA arrives too late. Proposition φ_2 covers all the six mentioned situations. It should be noted that not all of the six situations are to be expected in a properly working protocol. Inspecting the code of [3], one would for instance conclude with a certain confidence that no incommunicado phase will actually occur during the initial association phase. Nevertheless, we simply check-point with φ_2 all possible situations. Moreover, since we are deal with an abstract version of the protocol, the behavior of the rest, especially the steady-state control, is more non-deterministic. Note also that, despite the "eventuality" in φ, the property does not depend on the assumption, the lower layers provide a non-lossy communication buffer, since, after a failure, the $\Diamond\,\varphi_3$-part is guaranteed by time-outs.

The second property is the safety property *"no unsolicited MT-association"* mentioned above: if the mobile terminal is associated, reported by `Association_cnf` to the upper layer, the AP must reported the successful association of the new MT before (by `Association_ind`).[6]

3.3 Possible extensions

The experiments reported above cover only part of what could go wrong at the dynamic control. Routinely, one could continue in the same style, using the same model, checking the same properties also for the two other types of handovers —forward and backward— since all three types of handovers are handled almost uniformly in MAA.

[5] cf. the Mascara-specification [1] for details.

[6] It's the same property as mentioned in the errors-section and shown in Figure 7. The current experiment was done after repairing the cause of the error.

The experiments covered the behavior on the level of associations. The behavior on the level of *connections* is similar, the peer-to-peer partners are then MVC-agents and not association agents. Besides that, the services offered to the upper layer are comparable (opening and closing, and additionally switching) such that corresponding properties can be checked much in the same way. The abstract model contains the corresponding entities and code already, only the environment needed adapting. Besides that it would be worthwhile to check whether there is unhealthy interference between the two layers of services, for instance, what happens with a partially established connection if the whole association is terminated etc.

4 Steady-state and dynamic control, MT-side

Following a bottom-up approach, at the next stage, we combine the models for dynamic and steady-state control. Starting from the same global model [3], both abstract models had been worked out in parallel and largely independently in Eindhoven and Kiel, and moreover investigated and cleaned-up separately (cf. Section 2 and Section 3). Targeting at properties for a complete dynamic control configuration, the experiments reported in this section were mainly intended as sanitary checks for consistency after combining back the two models. Since MTs target cell MTC, which is not present at the access point, is the most complex entity of the steady-state control, we choose the combination of steady-state and dynamic control for the mobile terminal to check for consistency of the separately developed models, before turning to the two-sided model in the next section.

4.1 Simplifications and abstractions

The starting point of the model was once more Vires' global specification of Mascara. The *configuration* consists of the control entities of *one mobile terminal*. More specifically, the processes featured in the model are:

- an upper layer as environment `upper_proc`
- processes of MT's dynamic and steady-state control
  - MT-target cell
  - handover indicator
  - one instance of MVC-agent
  - association agent
  - tip agent

- further processes of less importance for the properties checked:
  - ∗ dynamic generic agent
  - ∗ radio control manager
  - ∗ measurement functions
  - ∗ generic dynamic control
  - ∗ generic steady-state control

– abstracted rest of MT-behavior (`mtmasc_proc`)
– data pump as lower layer

Besides the mentioned simplifications, we stuck to Mascara's overall as close as possible and followed the approach described in Section 1, using the combination of *sdl2if* and *if2pml* to obain Promela-code, fed into *DTSpin*.

4.2 Results

Table 2 collects the corresponding verification results.

Reachability We started with checking simple reachability properties like "it is possible that `Synchro_With` is sent and received", "it is possible for the MT-association agent to get associated".

Errors So as we were verifying the model containing both dynamic and steady state control, each of which had been previously verified as a separate model, we mainly aimed to check that putting them together will not cause problems. This series of experiments was considered as an preparatory step in the bottom-up way from the verification of the control sub-entities to the verification of the whole Mascara control for the two-sided configuration of Section 5.

An important interaction between the dynamic control and the steady state control concerns the phase of establishing the association to an AP. Therefore, the property that should hold is "It is possible for the MT to associate to an AP". This property was successfully verified for the separate closed model of the dynamic control including a simple abstraction of steady state control whose behavior was not chaotic but regular in sense that several assumptions over steady state control behavior were made.

Some of the assumptions fitted the properties that were verified for the model of the steady state control, but some others did only under the new assumptions, that would claim that in several situations control sub-entities are always synchronized.

2. Simple *liveness property*: if the signal synchro-with is sent, then eventually the corresponding acknowledgment is received. The property does not hold, because of a *deadlock* by unspec. reception (caused by a racing condition).	MTC cannot handle `Synchro_With` in states other than `associated` and `non_associated`. MAA sending this signal waits for the acknowledgment `Synchro_With_ack`, if this doesn't come, since MTC has ignored `Synchro_With`, MAA deadlocks. The error can occur in various situations.
	Deadlock by "uncooperative" environment
	deadlock by AP requesting deassociation
	AP gets lost
	too early reassociation request
race condition	After an association has been set up, it is lost again, the upper layer requests a new association, but the signal `Synchro_With` is not received, so the deadlock occurs.
unspecified reception	signal `MT_Deassociated_cnf` has 2 possible receivers. This fact remained unnoticed by the SDL-checker, and neither the *if2pml* issued a warning, that it could be sent to the wrong place. The scenario produced by *Spin* shows where it fails.
unspecified reception	In state `aaa.wait_addr_received`, the signal `Association_ind` to be sent after the 3rd association handshake has two possible receivers (and is not being translated). This invalidates the safety property: *If the MT is associated (reported by* `Association_cnf`*), the upper layer of the AP must have been informed by the new MT before (by* `Association_ind`*)*

Table 2. Verification results steady-state and dynamic control (MT only)

At the signal exchange level the interaction between the control subentities during the association phase looks like the following: MAA sends signal `synchro_with` to MTC. If MTC is in *non-associated* state, it sends a request to RCM to tune the mobile terminal into the correspondent frequency. Depending on the answer of the RCM,MTC sends `Synchro_With_ack` with a positive or negative acknowledgment to MAA.

It is essential that `non-associated` is the only state where the target cell can process the signal `synchro_with`, in all the others it simply discard the signal. It had been already verified on the model of the steady-state control that receiving `synchro_with` in this state, MTC always acknowledges it (in one time unit).

Another essential point is that association agent is designed in such a way that after sending `synchro_with` it is *locked* until it gets the acknowledgement from MTC. This means that the protocol behavior relies on the assumption that MAA sends `synchro_with` only when MTC is in the non-associated state.

So the property we chose to verify was the following: "if the signal `synchro_with` is sent, then eventually the corresponding acknowledgement is received". We found a number of situations where this property is violated.

Several of them occur due to an (natural) assumption that the upper layer (which is outside the scope of Mascara's specification) behaves completely chaotically. It turned out that in order to get Mascara working some assumptions on the behavior of the upper layer have to be taken. Therefore, a (non-chaotic) model of the upper layer was built.

Another cause of the error had been again *race conditions.*

After repairing the model, the property was verified under condition that several assumptions on the environment, basically the upper layer, the property holds.

Verified properties Finally, we verified the property that it is possible for the mobile terminal to associate to an AP, though it is also possible that the MT will never get associated to an AP (due to the poor connection quality – in case if it is permanently impossible to tune into the frequency of any AP).

4.3 Possible extensions

The extension that we intended to do at the next verification stage is the model containing the whole Mascara control. In the current model, the

AP control was presented by an abstraction based on some assumptions about its behavior (namely, that an AP reacts to all the signals coming from the MT control as it specified in the textual description of Mascara). At the next stage it was supposed to investigate whether the combined model would produce the expected behavior.

Getting the model through the tools As it is shown in the table on the time consumption (Table 5), about a half of the time in this series of experiments were spent on getting the model through the various tools. Besides combining the two separately developed abstract models for dynamic and steady-steady state control, used in the following series for a series of experiments on a two-sided configuration of Mascara-control, an important side-effect of the current series was to hammer out a number of bugs that had went unnoticed before. As we worked with a combination of ObjectGeode, *sdl2if*, *if2pml*, *Spin*, and *DTSpin*, it took some time to identify which of these tools had been the actual source of the error, and localizing which situation, which architecture, which version of tools etc., each bug or curious behavior occurred. This required close cooperation with the the tool-developers; in case tools developed within the Vires-project had been spotted as cause of the unexplainable behavior —in most cases rather small glitches— the errors were corrected the shortest possible terms. It should be noted, that when talking about unexplainable behavior of the tool set, this also concerns the more mature tools like ObjectGeode, and *Spin*/*XSpin*, and even gcc.

5 Steady-state and dynamic control

This section reports the results about the verification of a complete Mascara control configuration, complete in sense that at least one MT and one AP are involved. Thus, it generalizes and extends the verification results for a one-sided configuration from Section 4, which dealt with a one-sided configuration and Section 3, dealing with the dynamic control part, only.

Different from the results in Section 3 and 4, the properties to be verified here concentrate on the proper operation of an existing association (which is more or less the task of the steady-state control, however, the dynamic control also contributes to it). More precisely, the model is used to verify the combined functioning of the protocols for

- *I'm alive,* for
- *incommunicado,* and for the

– backward and forward *handover*.

in the context of one single MT and one AP. Thus, neglecting the pure data transfer functionality, the investigation covers much of the most complicated behavior of Mascara in the higher layers of the protocol (layers above the data-pump and the WDLC).

5.1 Simplifications and abstractions

Concentrating on Mascara control entity, the model again replaces layers below the control-layer by simplified versions. The abstract version of the data pump is inspired by the simplified model produced in Grenoble. While preserving more or less the "official" interface with the upper layer, the data-pump and the sliding-window entities simply transfer the required signals as a *lossy communication channel* between the two instances of Mascara and especially generate `framestart` signals periodically.

The layers upper the control entities are partly merged into the association agents to simplify the model such that it can be handled by the model-checker. All the behavior related to interaction with the physical reality (measuring the signal strength, measuring the error rate etc. at the lower layers) is replaced by non-deterministic behavior.

This leads to the following *configuration,* consisting of dynamic and steady-state control entities of *one mobile terminal* and one *access point.* More specifically, the processes featured in the model are:

- at the mobile terminal
  - MT-target cell
  - hand-over indicator
  - association agent (partly serving as upper layer)
  - tip agent
  - radio control manager
- at the access point
  - tip agent
  - I'm alive agent
- lower layer: generic data pump

Model The starting point of the verification was again a stand-alone, sealed-off version of the entities mentioned, developed in Eindhoven, which was cut out of the global "official" Vires-model of Mascara. Besides the mentioned simplifications, we sticked to Mascara's overall model as close as possible.

The model has been translated by the Vires-toolset into Promela and checked with *DTSpin*. To model-check various properties, we used the model-checker *DTSpin*, a discrete-time extension of the *Spin* model-checker, developed in Eindhoven within the Vires project. To feed the model into spin, the SDL-code is translated in two steps:

1. translating SDL to the intermediate format using *sdl2if*, and
2. translating the result into *Spin*'s input language Promela, using *if2pml*

5.2 Results

Table 3 collects the corresponding verification results, where both timed and untimed sets of experiments are carried out with *DTSpin*. The results concerning the verified properties and the errors found in general depend on each other, in that errors found while trying to verify the property led to a corrected version of the model.

Unlike in the preliminary series, this time we ran into more serious state-space problems.[7] Especially for the later versions of the model (after necessary corrections and modifications) to obtain a model for which we tried to verify the timed property, the size increased considerably, *Spin*quickly ran out of memory even for minimal buffer sizes, and we had to fiddle some time to obtain a small-enough model.

Reachability

Again we started checking whether a number of crucial points in the various protocols mentioned above which contribute to the combined behavior, are reachable at all. The *reachability* checks are done by simply checking *assertion violations.* Figure 8 shows one typical behavior of one of the protocols, namely completing one single cycle of the incommunicado protocol. Further scenarios concerning the signal exchanges for various handovers are available at the corresponding web resources.

[7] Some figures concerning resource consumption can be found at the corresponding web-page.

Reachability	
1. MT can go incommunicado	
2. forward hand-over possible	
3. incommunicado scenario	
4. backward hand-over at MT	return to old AP
Errors found	
1. value out of range	number of associated MT's becomes negative
2. twice "*start-of-tip*" without end-of-tip in between	caused by break of connection during incommunicado
3. twice "*end-of-tip*" without start-of-tip in between ← illegal termination	caused by unreported typo in the SDL-model
4. incommunicado becomes impossible	caused by interference of backward hand-over and incommunicado protocol
5. illegal termination	MTI isn't aware of backward hand-over
6. infinite undetected loop in backward hand-over	In case radio quality is generally bad, MTC can loop forever in scanning the environment and no one notices.
Verified properties	
1. no illegal termination	verified after repairing of model
2. toggle-array chooses correct branches in ATI (start-tip)	verified after repairing of model
3. toggle-array chooses correct branches in ATI (end-of-tip)	verified after repairing of model
Timed properties (positive and negative results)	
1. AP gives up association before MT	
2. AP gives up too early (2)	
3. *permanent* backward hand-overs possible	MT is too eager to scan environment, without time to send MT_alive
4. AP gives up the association too early	wrong combination of values for 4 timers and 4 other program constants
5. AP gives up association too early	
6. permanent backward hand-over impossible	corrects an earlier error
7. AP will always wait long enough before giving up association	cf. a previously listed error, here a different choice of values for the timers/constants verifies the property.

Table 3. Verification results

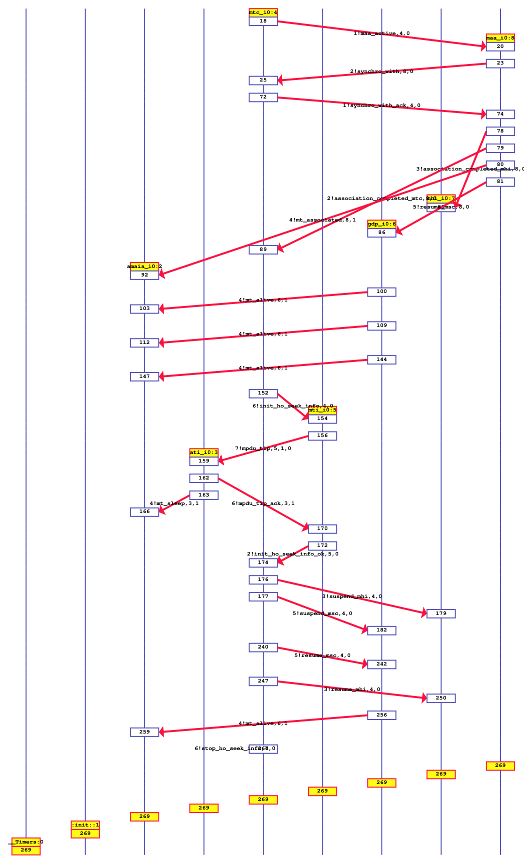

Fig. 8. Incommunicado scenario

Errors

The second block of entries of Table 3 shows the errors in overview.

The first error by an interference of *backward* and a *forward* handover, leading to erroneous behavior at the *I'm-alive-agent* at the AP-side. In the AIA it seems not to been foreseen that a deassociation request can occur “at the same time” as a forward handover. The AIA keeps track of the number of associated MTs using the variable `N_ASS`, increasing resp. decreasing this counter at each association resp. deassociation. An obvious *invariant* for this counter, that it never takes negative values, does not hold. The error trace shows the situation where the error occurs:[8] if for a given, previously established association, a maximal number of "I'm-alive"-invitations remain unanswered by the concerned mobile terminal, AIA gives up this association, it initiates a forward handover by the signal `AP_Lost` and decreases its counter. If, immediately afterwards, the upper-layer requests a deassociation, ADG relays the signal `MT_Deassociated` and the number is decreased another time *without checking* whether the association is still in operation. The relevant pieces of code are shown in Figure 9.

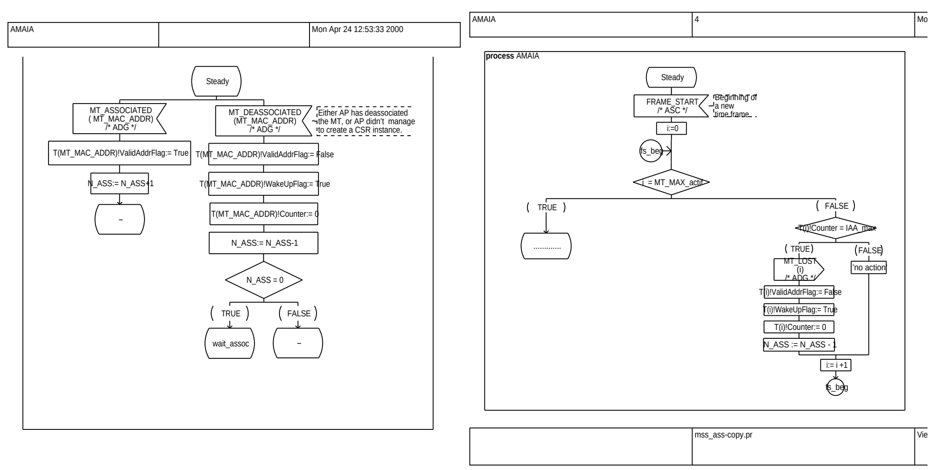

Fig. 9. Number of associated MT can get negative (AIA)

The same erroneous behavior is possible in the AIA of the global model [3], as well, if the upper layer issues the deassociation request immediately following the forward handover is initiated. Considering the behavior of the concrete Mascara protocol and the concrete environment, we see that the upper layer is being informed about the loss of association by the signal `AP_Association_Lost_ind` from ADG and it is plausible to expect that an upper layer will not request a deassociation of an MT after being told that the association has been lost. Nevertheless, due to the asynchronous nature of message passing, there's nothing in the model to prevent the upper layer to request deassociation while it has not yet received the `AP_Association_Lost_ind`-message.

[8] With 500 steps the trail is too long to be conveniently be shown here.

The error has been corrected in later versions by checking whether the MT is still associated before decrementing the counter.

The next two errors concern the proper functioning of the *incommunicado protocol.* The incommunicado protocol is a peer-to-peer protocol between a pair of Tip-agents MTI and ATI, where the ATI functions as server granting permissions to go incommunicado to the MTI-clients. As long as an association remains established, an MT once in a while goes temporarily incommunicado and turns back shortly after.[9] So each MT, driven by its timers, toggles between a silent incommunicado and the normal communication phase.

One property of the incommunicado protocol is that *never twice a start-of-tip is request without either a end-of-tip request of a MTI-timeout in between"*. The property does not hold and the error is caused by an interference of the incommunicado protocol with the *forward* and the backward handover under certain circumstances. Figure 10 shows an error trace: After and association has been established and if in the course of a *backward* handover, the MT fails to reconnect to the old AP,[10] this triggers a forward handover. In case this forward handover happens to connect to the old AP once again, the scenario shown occurs, leading to a second request for incommunicado. As a consequence, the MTI will never be allowed to go incommunicado in the new association, since the MT is already in incommunicado from the perspective of the AP.

Another error related to the incommunicado protocol concerns the reaction of the AP tip-agent to the end-of-tip request: when one of the MTI-peers has left for its incommunicado phase, it is expected to report back into duty by *one single* end-of-tip signal. Receiving *two* such signals without another start-of-tip in between is considered as an error by ATI.

In effect, receiving two end-of-tip signals leads to an *illegal termination* of ATI, and thus would result in a rather serious error, since not only the ill-behaved association would be affected, but *all* current and future associations of the AP (basically, the complete MAC-layer of the given AP would go silently out of business, and no precautions seem to have been taken, to restart the process again — or any other). The reason,

[9] For the purpose of the incommunicado-phases, the details of granting incommunicado permission, and the behavior during incommunicado, consult the specification report [4].

[10] Backward handover is the graceful form of handover, where before changing to the new AP, the connection to the old one is re-activated to allow for a proper deassociation.

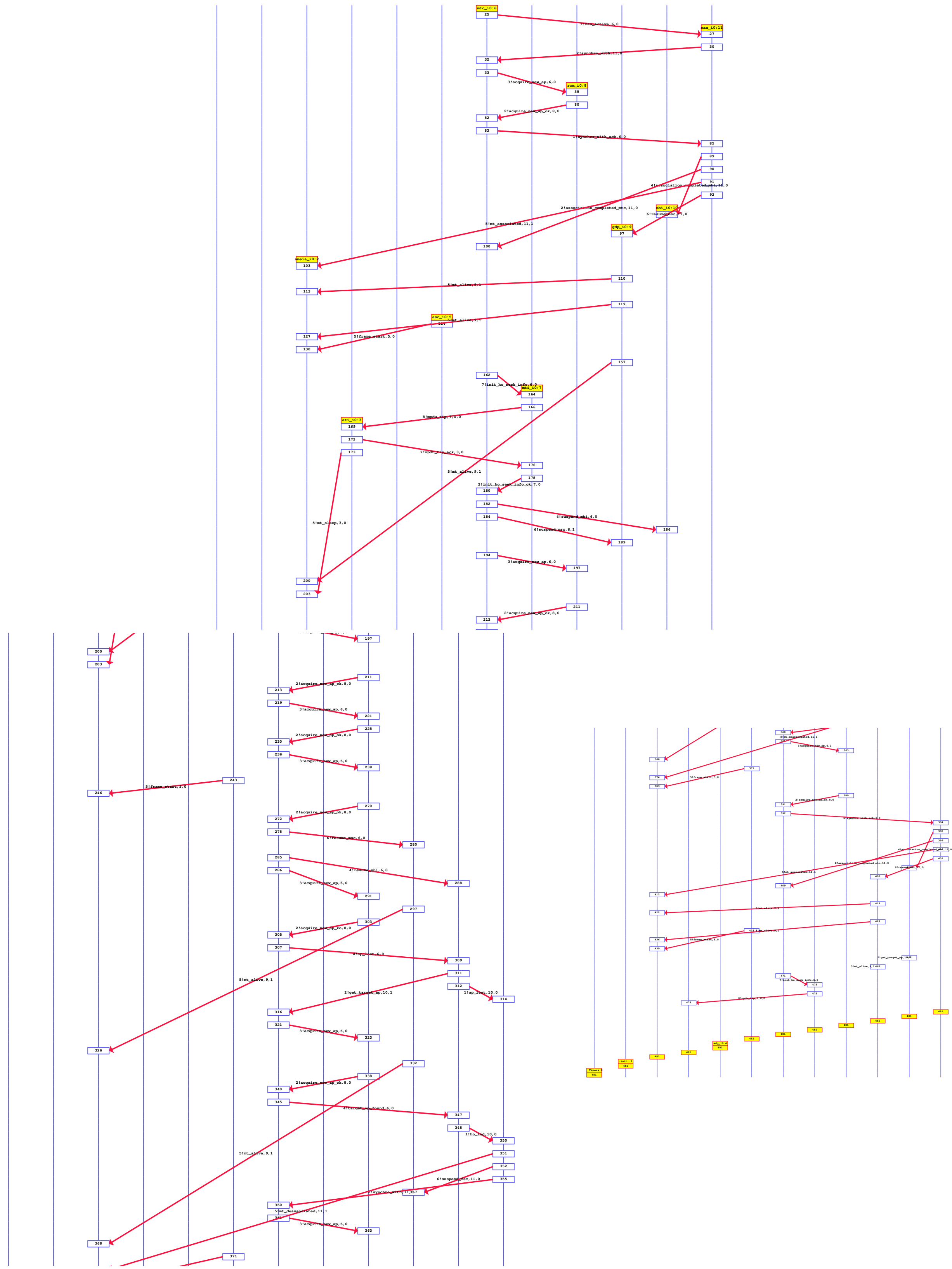

Fig. 10. Incommunicado and forward handover

why the designers expected that under no circumstances twice the end-of-tip should be received is understandable: the MTI-peer strictly toggles between incommunicado and communicating phases; even if the service requests or their (positive or negative) acknowledgments are lost, the code shows that MTI at least never signals twice the end-of-tip request. In the worst case it would just stay incommunicado.

So we checked the property, where the erroneous termination state if ATI is *reachable.* . Indeed, it is reachable, the error trace is given in Figure 11. The reason for the error is rather prosaic: it is caused by a typo in MTI, where in the signal `MPDU_Tip`, the parameter `mt_mac_addr` is used instead of the correct `my_mt_mac_address`, which make the ATI confuse the originator of the request and consequently leads to the termination of the tip-agent. ObjectGeode did not give warnings about the use of the uninitialized variable. It is not clear, whether the actual error can be traced back to [3] (let alone the industry's model), but the signals and the possible termination are the same in the global model. Besides that we consider it unwise, to simply terminate the process and in consequence, crippling the whole MAC-layer, as soon as it receives an unexpected address (just imagine a malicious MT chooses to send twice an end-of-tip or an en-of-tip with a fake address).

A related error occurs considering common behavior of the incommunicado protocol and a *backward handover.* The error trace is given in Figure 12. Just just after the incommunicado is granted to MTI, a backward handover is initiated by MHI. After this has been successfully completed by returning to the same AP again —possibly via visiting others in between— the next request for going incommunicado will be ignored and all subsequent ones as well. The error has been repaired and the mentioned property verified afterwards.

The same error —illegal termination— occurs also *after* a removing the typo, this time in connection to a *backward handover*: the MTI keeps on trying to go incommunicado, even when a handover occurs. So if after associating anew to an AP, i.e., the same AP in the scenario, ATI receives an positive start-of-tip MPDU, it gets confused and terminates.

The error has been repaired by introducing new signal `Suspend_MTI`. The signal is sent in the *"associated"*-state from MTC to MTI in response to a handover request, to reset MTI and stop it from looking for incommunicado in case it does so at that time (cf. Figure 13).

The next error concerns the behavior of the *MT target cell* (MTC). The property checked is the invariance:

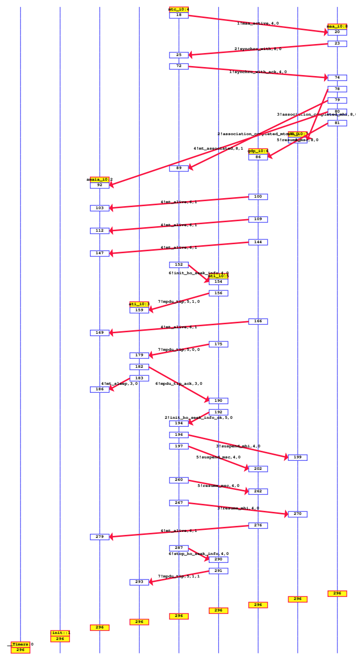

Fig. 11. Twice end-of-tip

$$\Box(\varphi_{mt-lost} \rightarrow \varphi_{ap-lost}) \qquad (1)$$

meaning intuitively that it must always be the mobile terminal that gives up an association before the access point does so. Proposition $\varphi_{mt-lost}$ describes sending the signal `MT_Lost`, whereby AP's I'm-alive-agent relinquishes the association. Similarly, $\varphi_{ap-lost}$ captures the situations, where the mobile terminal gives up the association by signaling `AP_Lost` or `HO_ind` (both from MHI).

The property is important for the correct working of Mascara-control, especially the correct management of addresses by the dynamic control entity: if the access point considers prematurely a mobile terminal to be deassociated, while in fact the MT is still trying to get back into contact, the AP might give away the addresses anew, which will cause confusion.

Whether this property holds or not depends intuitively on a number of timers and program variables. But the error discussed concerning MTC here shows up irrespective of any timer settings. A closer investigation

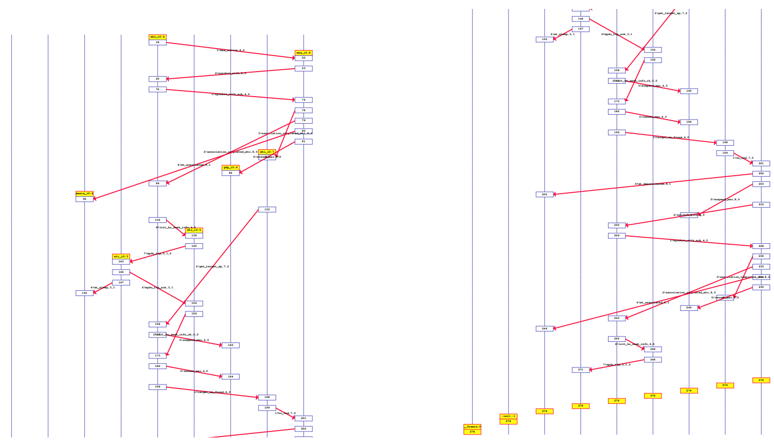

Fig. 12. Incommunicado and backward handover

of Mascara-control wrt. to this property and various timers will follow below.

Spin reports a violation of this property. The trace, given in Figure 14, can be interpreted as follows. During the initial phase of a backward handover, when MTC scans the candidate APs, it can happen that the radio quality is *generally* bad such that no appropriate candidate AP is found during the scan. MTC reacts in repeating the scan loop (without exit condition in the loop), so MTC can loop indefinitely scanning the environment and no one notices. Especially, no timer at MT breaks the loop and no alarm signal (as `AP_Lost` or `HO_ind`) is ever issued, hence $\varphi_{ap-lost}$ will never be true, AP gives up after a while, and property (1) fails to hold.

The error shows up also in the global Vires model [3]. Note that the error cannot be avoided in the abstract model by imposing fairness assumptions on the decision of the radio-control manager, here modeled by non-deterministic choice, whether the radio quality is good enough to warrant an association attempt. Neither would this be justified in the concrete model, since an endured deterioration of the radio connection cannot be ruled out. The problem is, that if a complete loss of connections should happen, at least it ought to be noticed by the responsible parts of Mascara-control, especially by MHI, and reported to the upper layers.

The error has been repaired by distinguishing forward and backward handover in MTC (cf. Figure 15).

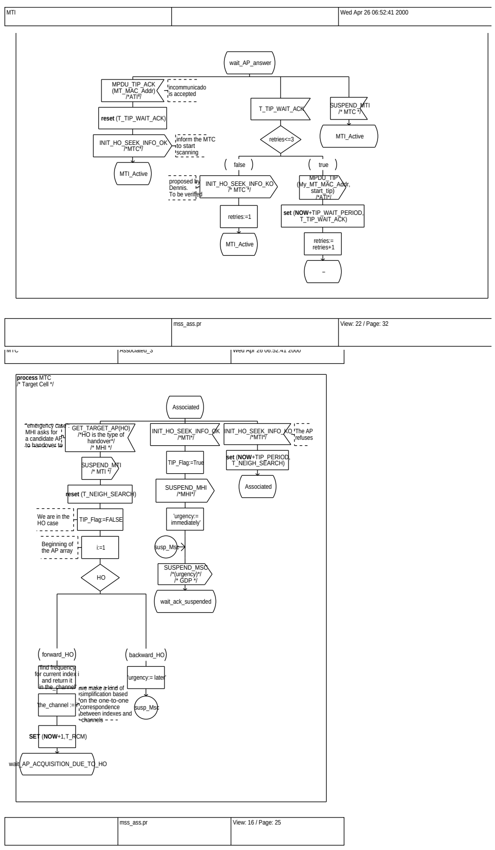

Fig. 13. Signal `MTI_Suspend`

Verified properties In the following we shortly describe the properties verified on the abstract version of Mascara-control (cf. also Table 3). The section is comparatively short, since basically it is just a re-run of the verification attempts, which failed in the previous section, after correcting the code. Besides that, we explain in some greater length the verification of the timed property (1) mentioned afore.

The first property verified proves *"absence of illegal termination* of APs Tip-agent, which means, never twice an *"end-of-tip"* without an *"start-of-tip"* in between (cf. the corresponding paragraph in the previous error-section.

The next two properties concern the behavior of a data-structure in ATI, the "toggle-array" which keeps track in which state —incommunicado or not— each associated MT currently is, and the values are updated

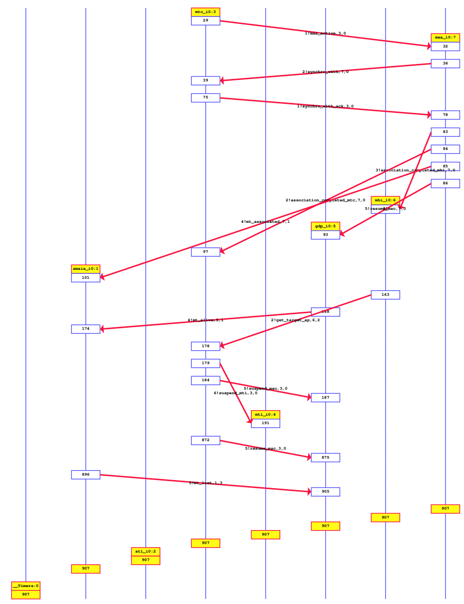

Fig. 14. Loop in backward handover

appropriately upon reception of the start-of-tip and end-of-tip server requests. Upon repairing two previously reported errors concerning the incommunicado protocol, the proper behavior of the ATI could be proved. The proof works for a non-lossy communication channel and for lossy channel under the proviso that the timer at MTI has a large enough value, such that non-lost acknowledgments reach MTI before the timeout. Since we proved that, based on the values of the toggle-array, ATI always choose the correct branch, we could remove this data structure and optimize ATI accordingly (cf. Figure 16).

Timed property The most complex property investigated in the current set-up is a *timed property* (cf. Equation 1) guaranteed by the proper behavior of mainly

- MT
  - MTC
  - MHI and data pump
  - MTI
- AP
  - ATI

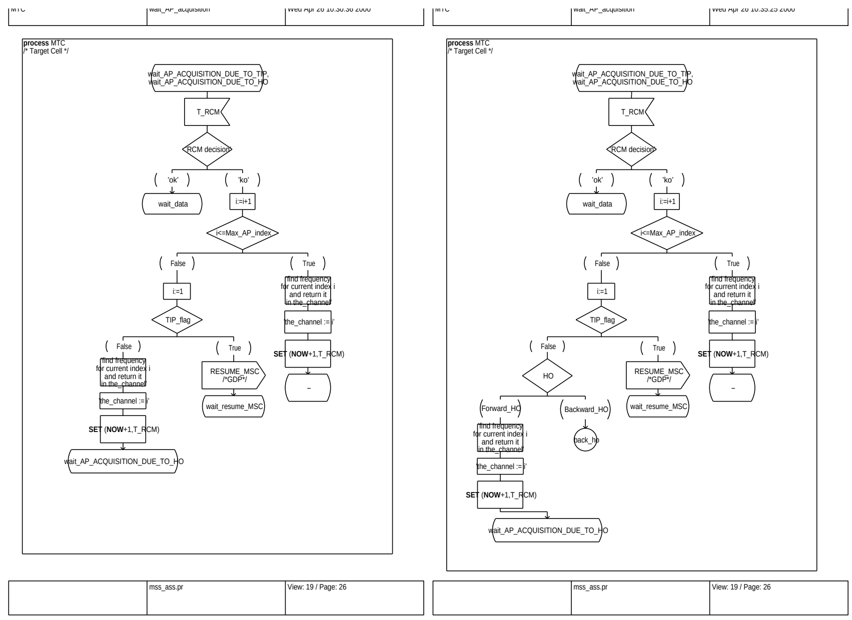

Fig. 15. Loop in MTC

- AIA

An important functionality for maintaining an established association is to determine, when the association *breaks down* (as opposed to terminating an association properly by deassociating). Driven by various timers, both AP and MT continuously check whether the current association is still functioning. (cf. the informal description [3] of the protocol for a more detailed explanation of the various entities.)

To determine that an association has gone for good, MT and AP must act independently and rely on their *local* timers, since if indeed the connection is lost, no further communication is possible in the worst case. Indeed, the functionality of Mascara's control entities investigated here is the part of the protocol which for the reasons just mentioned, relies most heavily on timers in complex ways. This was one of the reasons we chose the current setup for experiments concerning timed behavior of Mascara.

In determining the loss of connection, an important *safety requirement* is, that *"never the access point gives up the association before the mobile terminal does"*. The reason for this requirement is that in case the AP gives up the association, it is free to reuse the various addresses allocated to that association for possible new associations from a different MT. If