Verifying the Mascara Protocol
— A first step —

Dragan Bosnacki Dennis Dams Susanne Graf Leszek Holenderski
Guoping Jia Natalia Sidorova Martin Steffen Diana Tourko

May 5, 1999

Abstract

This document presents our approach to the verification of the Mascara protocol, useful abstractions and properties of the different subentities and their interplay.

1 Introduction: an approach to the verification of Mascara

For us, verification of MASCARA means verification of the SDL specification we have obtained from the WAND documents 3D1 [8], 3D2 [5] and the initial SDL skeleton that we obtained from discussions with Intracom [7]. Our specification focussed on the higher layers of MASCARA, whereas we have only a very abstract version of the Data Pump Layer, in particular the Scheduler.

It is obvious that this specification differs in many points from the implementation and we are aware of the fact, that in this context the aim of verification can only be to find a maximal amount of potential errors, and for each such error, the reality of this error must be checked in the implemented protocol and also in the abstract reference version represented by the WAND SDL code and the corresponding documents.

As we completed the SDL specifications to a large extent ourselves, we spent an important amount of time just to understand the global functioning and many details of this protocol (see [4]). This process of understanding alone allowed already to find several contradictions and potential problems in both the documents provided by WAND, and in the skeleton SDL code we started with. In addition, we have started performing experiments with automated verification, mainly using the tools `sdl2if` ([3, 2]), `if2promela`, `spin` and `dtspin`. The sliding window protocol of MASCARA as implemented in the WDLC entity has been the topic of a case study described in [1]. Furthermore, the Steady State Control entity has been subjected to a series of verifications as described in Section 3.1.

The correctness criterion of MASCARA is

> *to transmit transparently, reliably and on time (depending on the specified QoS parameters) ATM cells between an access point and an ATM Network*

Due to the complexity of this protocol, it is clearly not possible to verify this property "as such" considering the MASCARA protocol as a black box. In fact, it is impossible to generate the whole state space, even in case of a reduced configuration of a single MT and a single AP. The use of powerful abstractions — which are possible for the verification of partial properties — is essential for the feasibility of verification.

Our approach: Verification of a set of properties, necessary for the global correctness, on relatively small abstractions of MASCARA using our verification tools.

Properties to be verified: The properties we are going to verify are on one hand very general properties, such as absence of deadlock or live-lock, local consistency and response properties concerning a small number of sub-entities, as also functional properties, stating the safety of the data transmission. In the following sections, we list properties associated with the different entities which we intend to verify or for which we found already problems.

Abstractions: depending on the property to be verified, abstractions of different types will be used.

1. MASCARA has a layered structure. Properties concerning a single sub-layer (and its interfaces) allows to replace the layers below and above by their interface specifications (service requirements): these are relatively simple (non-deterministic) behaviors. In particular, this concerns properties of the control layer and the error correction layer WDLC. For the verification of safety properties of the Control Layer and the WDLC Layer, the data transmission function of the lower layers can be radically abstracted by replacing it by a direct communication through (lossy) channels with variable delay [1]. For the verification of liveness properties, it must be taken into account that during certain periods (such as handover or TIP) the transmission medium must be interrupted, for properties concerning the dialog between higher and lower layer, a more detailed description of the lower layers is necessary (see Section 2.4).

2. Limitation of the size of the considered configurations. The description of MASCARA makes intensive use of dynamic process creation: for each association between an MT and an AP, many connections may exist, and a single AP can have associations with many MTs. As the different connections within an MT and the different MTs associated with an AP are completely independent of each other (from a functional point of view), it is sufficient to consider a configuration with one or two connections in an MT and one or two MTs associated with an MT. Thus, we can always limit ourselves to consider a system with a finite control structure.

   For the verification of the correctness of the Sliding window protocol, it is sufficient to consider a single instance of a DLX and a DLR process in both MT and AP. Moreover, the data-flow in one direction can be abstracted, and only the piggybacked acknowledgments must be considered.

3. An interesting type of abstraction consists in focusing only on the control flow concerning some "interesting" subsets of signals and (finite type) variables of a subset of processes. A corresponding abstract system can be obtained by

   - considering inputs from abstracted entities as coming non-deterministically from the environment and eliminating non observable signals output to to these abstracted entities

- replacing transitions triggered by the reception of an "uninteresting" signal by an invisible "τ"-transition; some of the abstracted signal receptions may be replaced by some informal task, used as a label for the verification of a property,
- forgetting all but some of the (finite type) control parameters of "interesting" signals
- eliminating all "uninteresting" variables
- replacing the right hand side of the assignments of "interesting" variable depending on an abstracted variable by "any", meaning that nondeterministically any possible value of the assigned variable is chosen. If the abstracted variables occurring on the right hand side are also finite type or if one has some knowledge, e.g. in the form of invariants, more precise abstractions can be computed.
- all decisions depending on abstracted variables are transformed into nondeterministic decisions.

This type of abstraction leads to substantially reduced representations of the system and of the resulting state space: Typically, this kind of abstractions allows to verify, for example if,

- whenever a process waits for an acknowledgment of any message, such an acknowledgment is received after some time or some error has occurred.
- in certain states some signal can never be received
- in every process the signals are received in the order in which they are expected.

4. There are also some functionally uninteresting entities, which do only some transformation of the representation of the information to be transmitted, such as the Segmentation and Reassembly entity and the physical transmission layer. For some properties, it is sufficient to replace them by a channel, but as there is some control interaction with them, for certain properties, a non-trivial abstraction is needed. Such a non-trivial abstraction of a combined CSE/CRE and PHY layer entity, which is concrete enough to be used for simulation of the whole Mascara layer, is described in Section 2.5.

2 Abstractions and properties concerning the different subentities

2.1 Properties and Abstractions concerning the complete MASCARA

An important verification concerns the consistency and completeness of the control flow between the different components of MASCARA even within a single Mobile Terminal (respectively Access Point). In order to do this verification, we need to build a control abstraction of MT (respectively AP) considering the following sets of interesting signals and variables.

Interesting signals are control signals within MASCARA and certain signals with the outside world (upper layers, peer-to-peer control signal). Only some control parameters of these signals are considered, all other parameters and all variable which are not directly implied in the control, are abstracted.

This abstraction yields a simplified system, which has finite state space, and at the present state, we hope that it is simple enough to be verifiable with model-checkers such as CADP or Spin.

Timing aspects are not very important for the control flow, even if timers are intensively used in the specification in order to terminate waiting for a response in many situations. However, for the verification of global consistency properties within a single station, these timers can be abstracted, “timer expired” becomes a non-deterministic event. This abstraction adds many “impossible” behaviors which are undesirable from an efficiency point of view, but a violation of a safety property in this abstraction corresponds in general to a potential violation of the same property in the concrete system, at least for particular values of the timeouts). On the other hand, the satisfaction of a safety property on the abstract system *proves* its satisfaction on the concrete system.

As our SDL description is rather complete concerning all signal exchanges — but less concerning data transformations — it makes sense to build such an abstraction, on which we can verify the following properties:

- Absence of live- and deadlocks. We will probably detect problems, and some of them will just reveal some incompleteness of our SDL specification and help us to complete it.
- We can verify ordering constraints on events. For example, a property concerning the interplay of the Data Pump with all upper layer entities is

  > The Data Pump shouldn't receive any signal from upper layers before it has been initialized (by the signal INIT_MDP from Generic Mascara Control).

  Another considers the establishment of a connection: the opening of a new connection is notified to the Data Pump (in AP and MT) as well by the Generic Control Agent (in MT and the Association Agent in AP) through the signal SET_TRAFFIC_PARM as well as from the Generic WDLC through PID_REGISTERED (in order to communicate the Pid of the corresponding DLR process).

  > whenever one of the two signals is received by the Data Pump, within a some (small) amount of time (to be determined) either the other one or its nullification (RELEASE_TRAFFIC_PARM resp. PID_DEREGISTERED) should be received.

2.2 Properties and Abstractions of MASCARA Control Layer

The abstractions concerning the Control Layer are similar to those used for the whole MASCARA, but in this case we consider a system with *at least* one MT *and* one AP.

2.2.1 Steady State Control

A number of components within the Steady State Control entity perform rather trivial functions, particularly so when considering the protocol at the level of abstraction of the VIRES SDL code, in which most data operations have been omitted. These components are the Generic Control entities (ASG & MSG), the Beacon Agents (ABA & MBA), the Tip Agents (ATI & MTI), and the AP Link Status Recorder (ALS). At the MT, the Target Cell (MTC) is by far the most complicated process. At the AP, the I'm Alive Agent (AIA) is the less trivial process, as it does perform a number of data operations. Therefore, our verification efforts will first and foremost be aimed at these components. Below, we suggest a number of properties to verify for the MTC.

MTC As described in [4], the main task of the MTC is the following. When an "emergency" arises in the form of a handover request arriving from the MT's Handover Indicator (MHI), the MTC should be able to quickly select an alternative AP based on its information. To the purpose, it collects information about neighboring APs and maintains this in the form of a "prioritized" list of APs together with their channel numbers (frequencies), loads, and signal strengths.

This main task suggests the following primary correctness criterion:

> Whenever a request from MHI arrives, the MTC should react by a timely response in which it returns to MHI the best alternative AP.

This criterion is general and covers many aspects. For example, it can be made more precise by differing between Forward and Backward Handover (FW-HO, BW-HO) requests, and specify the differences between these two cases. Another issue is the "timeliness": in order to be more precise, it needs to be quantified. Yet another point is what it means to be the "best" alternative AP. And as a final example of the ways in which this criterion can be specialized, it should be noted that there are certain exceptional situations in which the MTC will not be able to show the required reaction; such exceptions should be taken into account.

Another class of correctness properties is suggested by the assumptions that other components make about MTC. There is a great deal of interaction between MTC and other processes. Apart from the communication with the MHI in order to perform its "main" service, the MTC interacts with a number of other components. Before the standard, periodical scan of neighboring APs can be started, the Tip Agent is asked to negotiate the permission to go temporarily incommunicado. In certain cases the MHI needs to be suspended and, later on, resumed again, and similarly for the Scheduler in the Data Pump. "Tuning in" to neighboring APs is achieved by sending appropriate requests to the Radio Control Manager, after which the signal strength and load are collected from the Measurement Functions (MEF, inside Radio Control) and MPDU Receiver (MPR, inside Data Pump) respectively. In order to synchronize with an AP (catch up with the Frame Header), also during the initialization, communication with the Association Agent (MAA) in the Dynamic Control takes place. Finally, the MTC may also be the one to first initiate a handover, namely when it fails to return to its old AP after scanning.

We now present two properties that we use as a starting point for the verifications of MTC. The general correctness criterion stated above can be decomposed into a response property

> Whenever a GET_TARGET_AP (with any parameter) request arrives from MHI, then eventually MTC will respond to MHI by a TARGET_AP_FOUND message.

and a pure safety property

> When MTC issues a TARGET_AP_FOUND(AP_MAC_Addr) message, then AP_MAC_Addr is the address of the best alternative AP.

The second property cannot be checked for our SDL description of the protocol, because data is not modeled. Property 1 however does not involve parameter values, making it a good candidate for our verification.

The next property is obtained by considering the assumptions that the MHI makes about MTC:

> The RESUME_MHI and SUSPEND_MHI messages (between MTC and MHI) alternate, starting with SUSPEND_MHI.

2.2.2 Dynamic control

As explained in [4], the two main sub-entities of the Mascara's dynamic control are the association agents and the MCV-agents. The "generic" agents are relatively uninteresting, at least as far as they are used for initialization and termination of the other entities.

To deal with the control entities, all lower Mascaralayers (i.e., control segmentation and reassembly, WDLC, and the whole data pump) will have to be radically simplified: Mascara control peers will exchange their signals over a lossy, buffered medium.[1]

The main challenge in dealing the dynamic control is, as the name indicates, the dynamic nature of its behavior: creating and terminating process instances on request.

Address management Dynamic control allocates and revokes addresses both on the association level and the connection level. A safety property in this respect would be, that never the same identifier is given two different stations or different connections. Such a property is first of all a local task of the managing agent and its internal data structures, but not entirely so: the agreement between the concerned peer agents on a *common* address use sometimes intricate signal exchange and handshake mechanisms for their coordination. This behavior can only be checked with at least one MT and one AP.

Other, related properties to check are, that, if the address server decides to revoke an address — because of a time-out or other problems — also the client will follow and not falsely assume that the address is still valid. Properties like this will especially be interesting to check in the setting of failed association establishment or failed connection set-up: if during a set-up the coordination fails, all steps taken on each side must be undone. So the safety criterion is, that never it happens, that one side considers an association attempt as completed, whereas the counterpart considers it as failed. In order to verify properties like, we need to come up with appropriate assumptions about the timers. Unlike in many other situations, the mentioned safety properties depend on the behavior of the timers. It seems that for an associations it suffices to assume, that the combined effect of the signals and timers of the steady state control and the dynamic control means, that the MT must be under all circumstances faster in giving up an association to an AP than vice-versa. Concerning connection set-up, the situation is comparable, but somewhat easier.

As far as the number of addresses are concerned, the obvious abstraction from the infinite or quasi-infinite reservoir of addresses, as mentioned in the introduction of this section, is to check proper behavior with only a restricted domain. Due to the layered internal structure of dynamic control, we can attempt to separate the correctness of address management for the association agents and for the MVC-agents. The separation is justified not only by the layered architecture of dynamic control, but also that association and control management are different *phases* of the behavior of dynamic control. As can be locally seen at the, any attempt to set-up a connection is locally rejected unless an association has already been successfully established. In other words, when checking properties concerning association

[1] Lossy, as Mascara's control channel operates in the non-ARQ mode.

management, the portion of the code dealing with connections can safely be ignored. Of course both phases are not completely separated, and a more complicated property could be: a partially established connections is properly terminated when its successful completion is interrupted by a deassociation.

Cooperation with the steady state control Steady state and dynamic control together constitute the main and most complicated part of Mascara's control entity and both cooperate closely. In fact, one entity of the MT dynamic control, the handover indicator, so closely cooperates with the steady state control, especially the MTC, that for verification purposes, when investigating either the dynamic or the steady-state control "as a whole", we will consider the MHI together with the steady-state control.

2.3 Properties and Abstractions of WDLC Layer

As described in [4] WDLC handles transmission errors in case of ARQ traffic (error sensitive traffic), whereas, in the case of real time traffic, it almost plays the role of a buffer: its only functional role is to eliminate "old" cells (moreover this role is shared with the Scheduler entity). For this reason, we concentrate at least in a first phase, on the verification of the modified Sliding Window protocol which is used for the error control in the case of ARQ traffic. There are two types of properties to be verified,

- safety of data transfer (no messages are lost)
- not too many cells are unnecessarily retransmitted which we will call here the liveness property of WDLC.

We different abstractions for the verification of these properties.

2.3.1 Safety of WDLC

The safety of the sliding window protocol implemented in WDLC can be verified on two configurations as shown in the following figure:

In the shown configuration (Figure 1, the observed data traffic is from MT to AP, but as there is a slight difference (essentially concerning the use of reservation requests and successfully transmitted cells) between AP and MT, we need also to consider the symmetrical situation with AP and MT exchanged.

The abstracted versions of the WDLC processes DLX and DLR, just send (abstract DLR), respectively receive (abstract DLX) the signal AcksReqsReady with the parameters expected, respectively transmitted by the concrete partner process, otherwise the abstract DLX process only transmits the acknowledgment to the lower level (with appropriate timing constraints), and the abstract DLR process forwards it (if it has not been lost) to the concrete process DLX.

The abstract transmission medium, accepts at any moment the signal TRAFFIC_UPDATED coming from the DLX processes and updates the list of cells waiting for transmission in one direction and the list of acknowledges waiting in the other direction. During each "Time frame" it transmits some of the messages and acknowledgments which are arbitrarily corrupted. On reception of the signal SUSPEND_MSC, data transmission is interrupted until the reception of signal RESUME_MSC.

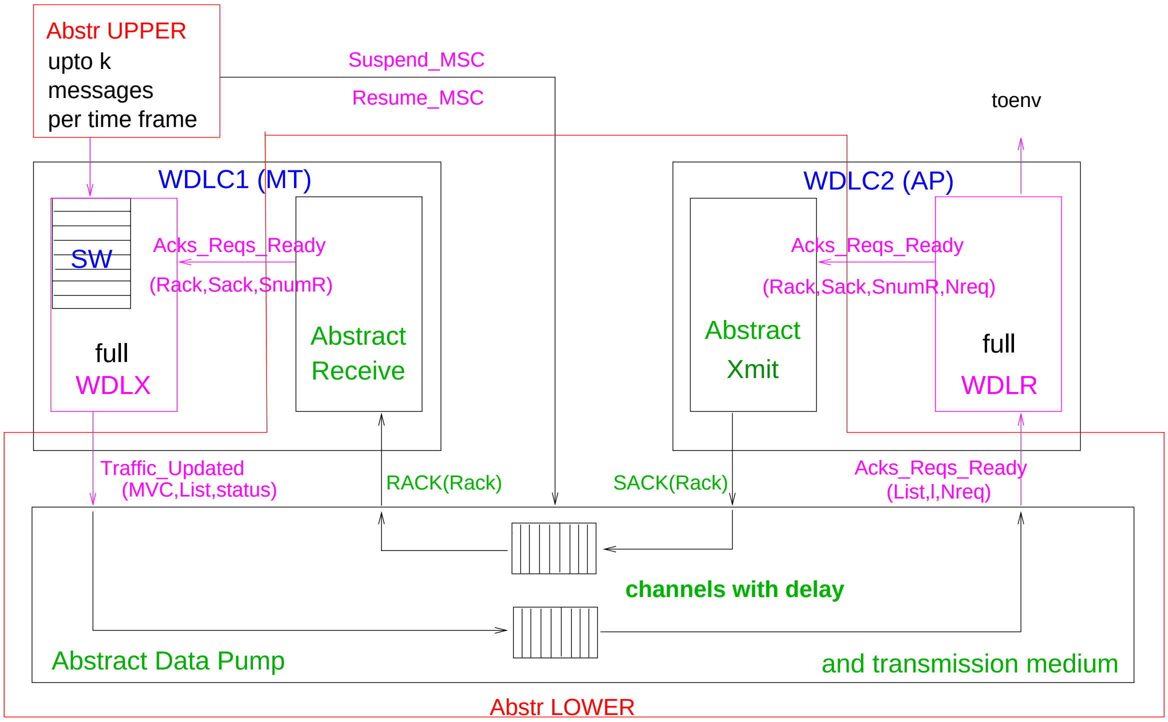

Figure 1: Abstract System for WDLC verification

The abstract upper layer sends never more message transmission requests than can be absorbed by the Sliding Window (this implements the hypothesis that the negotiated bandwidth can be guaranteed).

One possible approach consists in defining an initial version of the abstract system by writing four abstract processes “by hand” and then to simplify it by combining all the abstract processes into a single abstract environment process, and this can be done by tools. However, it would be rather “dangerous” to produce this very abstract environment directly by hand.

A somewhat different approach has been followed in [1]. As the verification with Spin or the CADP tools requires the generation of a complete state space, in order to do the verification directly on such an abstraction, only small values for the window size are possible. The safety property could be verified for the real window-size (which is 16) by decomposing the properties into smaller sub-properties and building much stronger abstractions.

2.3.2 Liveness of WDLC

It is very difficult to say at this moment, if the above abstraction is detailed enough to detect problems concerning the liveness property, but probably, one needs to go into some more details of the low level data transmission mechanism for this (see also the last item at the end of section 2.4.2). In this case we will use the more concrete Data Pump described in the next section together with a WDLC layer for a single virtual channel.

2.4 Abstraction and properties of the Data Pump Layer

The first SDL description of the Data Pump that we have obtained from reading the Wand documents and from discussion with Intracom [4, 7], is a relatively abstract and above all incomplete:

- It does not contain any details about the actually used scheduling algorithm, and therefore, our verification effort will consist in verifying properties which are independent of the actual scheduling algorithm.
- Concerning the TRE process, it does not contain any data transformation function for updating the Traffic Table:
  - on reception of update signals from either the Scheduler or WDLC,
  - or in order to add or delete new connections.
- Also the processes MPX, MPR and Slot Sequencer who in the "real" protocol build MPDUs, respectively extract them from an input buffer, do not contain any description concerning data transformation, only signal exchange is described.
- the Slot Sequencer plays the role of the PHYSICAL Layer which transmits and receives messages from the peer Slot Sequencer(s): It treats the TEDs received from MPX in order, and depending on its type, it either waits for an MPDU (RX types) — which is then forwarded to MPR — or sends the MPDU contained in the TED (TX type). SoF is considered as a particular MPDU.
- there is no timing information in the Slot Sequencer, which however is essential.

Another feature of the first SDL specification of the Data Pump (which is also a feature of the Wand SDL specification as far as we can judge it) is that the Data Pump is split into as many processes as there are functionalities, which share many data structures. For this reason, there exists a lot of internal signaling between these processes to make known information coming from outside to all concerned functional entities (such as the opening or closing of a connection, the reception of a new MAC_address, ...). Most of this information from outside is received through a process GDP (Generic Data Pump) whose only function is to forward signals from and to all processes.

2.4.1 Abstract description of the Data Pump

In the second specification — which is described in this section and described in [6] — we have decided to on hand simplify the description of the Data Pump, and on the other hand, to fill in enough algorithmic details in order that some properties can be verified. In this second version, the Data Pump deals only with a single connection (and therefore also a single association AP-MT), but the description is such that the extension to several associations and connections can be added without too much effort. The overall architecture and the signal exchange of the new Data Pump description can be found in the Figures 2 and 3, where we have separated the signals concerning control and data flow.

A significant difference with the preceding version is that the Data Pump is described now as a single process, and its functional sub-entities by *services*. Moreover, the Generic Data Pump can be eliminated completely and the 3 processes of the MPDU handler entity can

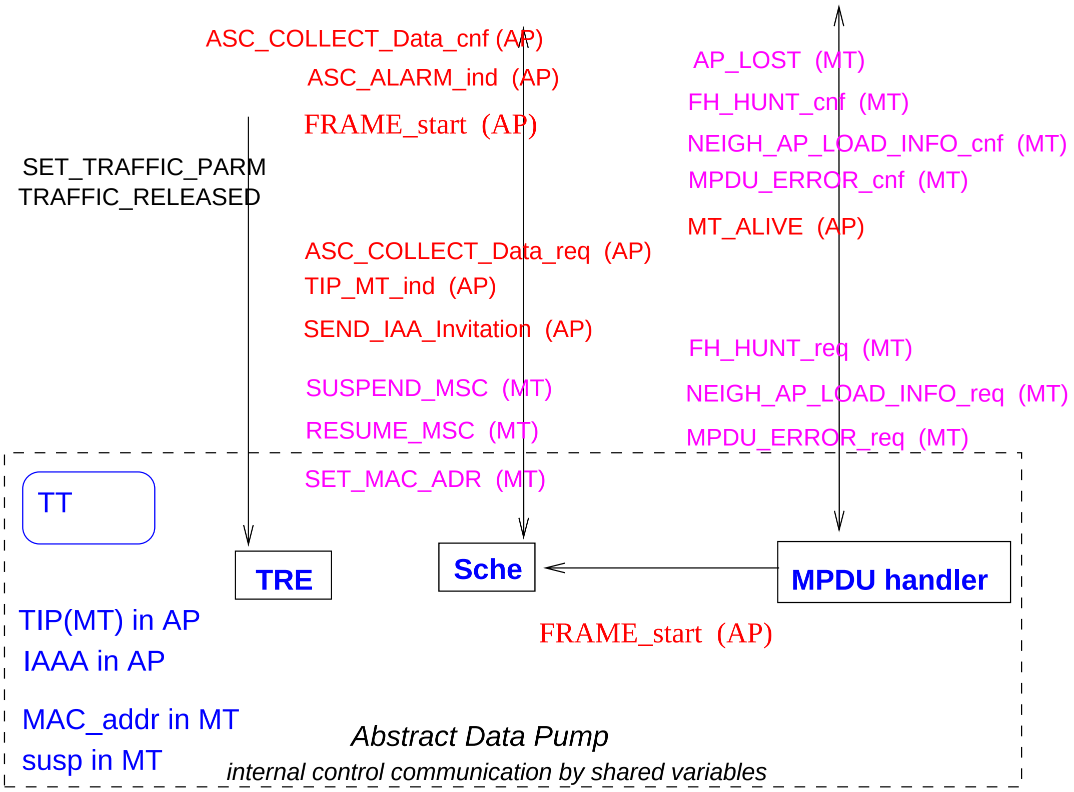

Figure 2: Control Flow in the abstract Data Pump

be grouped into a single functional entity called MPX/MPR, as MPX and MPR are active alternatively, and the Slot Sequencer is not needed as the signals exchanged between AP and MT contain MPDUs and not slots. This simplifies the description significantly, even if the abstraction level is not changed, as now variables and data structures can be shared between these sub-entities, and therefore the internal signal exchange is reduced.

The use of services would certainly allow to simplify also the original Wand SDL specification, and not only our abstract version intended for verification purposes.

The functionalities and the level of details implemented of the different sub-entities of our new version is described below:

1. In the Wand version, the Data Pump exchanges to specific time points bit-streams with the adjacent Physical layer, but in our specification MPDUs are transmitted to the Physical layer (represented simply by a channel relating an MT and an AP, or by some particular process PHY_ENV).

2. Time: in the Wand specification the time unit it is a slot, that is the time needed to transmit one ATM cell or equivalent cell. In our more abstract specification, the messages transmitted over the "transmission medium" are MPDUs. For this reason, the time unit is the time needed for the transmission of one MPDU: even if MPDUs are of variable length, this abstraction is sufficient for the kind of verification we intend to carry out. Experience will show if this time model is sufficient

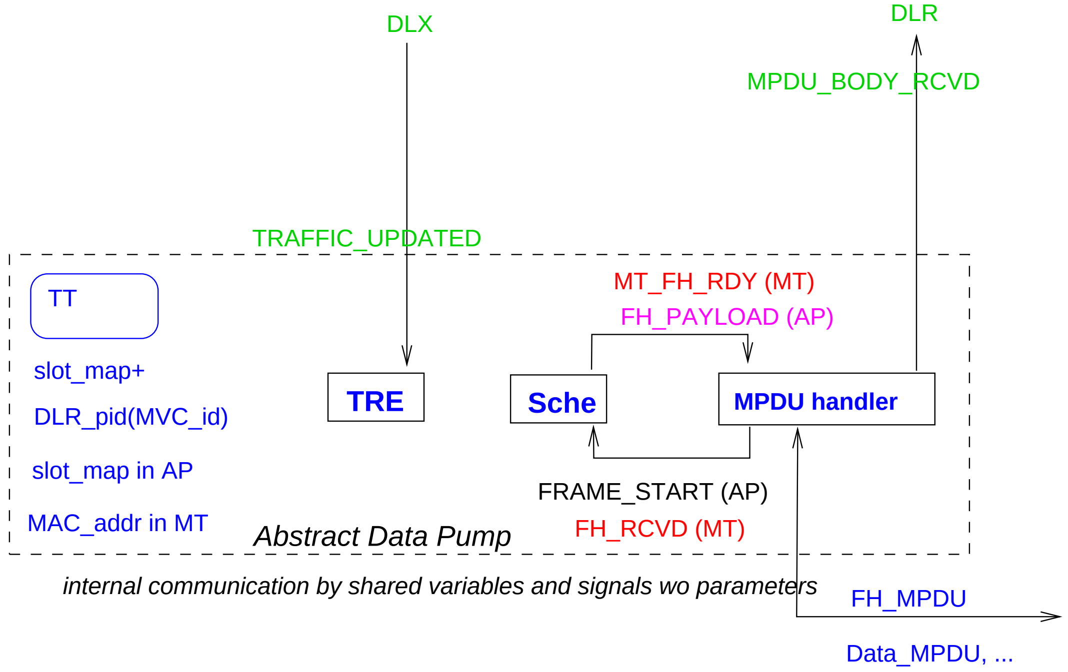

Figure 3: Data Flow in the abstract Data Pump

3. The Traffic Table TT is a global variable of the Data Pump, readable and writable by all services who need it (TRE, Scheduler and MPR). This eliminates both

   - the internal dialog between the Scheduler and the TRE process (signals GET_TABLE, SET_TABLE and UPDATE_TRE) and
   - the duplication of the information.

   The information stored in TT is almost the same as in the Wand protocol, except that there is a single entry containing the informations for up and down link and the cell pointer is replaced directly by the cell itself which we represent by an integer. Presently, we take only into account ARQ Traffic, as for non ARQ traffic relevant properties can only be verified if more about the real scheduling algorithm is known.

4. the TRE entity, which in the concrete description is the only functional entity which can change the Traffic Table, handles only the following updating requests:

   - the updating requests coming from DLX through the signal TRAFFIC_UPDATED
   - the updating request coming from the Dynamic Control SET_TRAFFIC_PARM (open a new connection) or a TRAFFIC_RELEASED (close a connection) which has the effect to add or eliminate an entry in the Traffic Table and to update the DLR_PID of the concerned connection.

   The updating of TT due to the scheduling of messages in the current time frame is done by the Scheduler, and the updating of the reservation requests piggybacked in the MPDU headers is done by MPX/MPR.

5. The functionalities of the Scheduler concern the reaction to some signals from the Control Layer (it takes over to some extent the functionality of the GDP), and the Master and the Slave Scheduling algorithms, where the Master Scheduling algorithm is only present in AP:

   - On reception of the signals INIT_MDP, CLOSE_MDP, SET_MAC_ADDR, SUSPEND_MSC, RESUME_MSC from the Control Layer, the Scheduler updates the variables MAC_addr, active, and resumed and sends the corresponding confirmations INIT_MDP_ok, CLOSE_MDP_ok, SUSPEND_MSC_ok, RESUME_MSC_ok back to the control layer.

     On reception of the the signal ASC_COLLECT_DATA_req in AP, the Scheduler sends back the signal ASC_COLLECT_DATA_conf confirm without any parameter, as we do not model the information which asked for.

     In AP, on reception of the signal SEND_IAA_INVITIATION(MT_addr), the Scheduler stores the MT_addr in the variable IAAI, and a corresponding "I am alive invitation" is inserted by MPX/MPR in the next FH_MPDU.

   - The Master Scheduling algorithm is very simple, but sufficient for verification purposes. For each open connection in the Traffic Table (there is only one for the moment) which is not suspended due to TIP, it schedules: for the down direction between 0 and the number of waiting cells, and for the up direction between 0 and RequestNumber (stored in TT) cells, and builds corresponding slot-map entries; if a transmission error occurred in the preceding time frame, either in down or up direction — that is if one of the values SuccNumRcv or SuccNumTx in the Traffic Table is 0 — no MPDU is scheduled for this direction; at the end, a slot-map entry for the contention phase is added.

   - The Slave Scheduling algorithm is simpler in AP than in MT. In AP, its only task is to fill the slot-map with the list of cells to be <u>sent</u> and to update the Traffic Table by eliminating the scheduled cells.

     In MT, the Slave Scheduler must furthermore

     – add the Reservation Requests into each up-link Slot-map entry,
     – if the slot-map contains an "I AM ALIVE INVITATION", it schedules a "I AM ALIVE RESPONSE" in the contention phase; in the WAND document it is not told explicitly how the MT Scheduler reacts to such an "I AM ALIVE INVITATION", but this is a plausible way of treating this case.

   - Most of the time the scheduler is in a neutral state where it is receptive to control signals from the upper layers.

     In AP, at the beginning of each new time frame (on reception of the internal signal FRAME_START), the Master and the Slave Scheduling algorithms are executed, the MPX/MPR service is informed via the signal FH_PAYLOAD, and the Scheduler becomes again receptive to the signals from the upper layers until the next FRAME_START arrives.

     In MT, the Slave Scheduling algorithm is executed when a new Frame Header has been received (the Scheduler is informed through the internal signal FH_RCVD when MPX/MPR receives a correct frame header MPDU), then the MPX/MPR service is informed via the signal MT_FH_PAYLOAD, and then, as in AP, the

Scheduler becomes receptive again to the signals from the upper layers until the next FH_RCVD signal arrives.

6. The 3 processes of the MPDU handler block MPX, MPR and Slot Sequencer are transformed into a single abstract MPX/MPR service. In MT, this abstract entity is either waiting to be activated, hunting for a frame header, waiting for the MT_FH_PAYLOAD signal from the Scheduler, telling that the slot-map is ready or treating one by one all elements of the slot-map. In AP, MPX/MPR either waits for a new time frame to start on reception of the signal FH_PAYLOAD from the Scheduler, on the reception of which it sends a FH_MPDU or like in MT, is treating one by one all elements of the slot-map.

   - in MT, the initial state is inactive, and a signal FH_HUNT_req from Radio Control is waited for, on which it enters the hunt mode; in hunt mode, the reception of a FH_MPDU from AP is waited for: if the received Frame Header is correct, Radio Control is informed through FH_HUNT_req(ok), the slot-map contained in the FH is stored in the global variable SlotMap, the Scheduler is informed through the signal FH_RCVD and the wait_SM state is entered; if it isn't correct, Frame header hunting continues until a correct FH_MPDU is received or the hunt timer expires; on timer expiration, the signal FH_HUNT_req(nok) is sent to Radio Control and MPX/MPR becomes inactive again.

     In the wait_SM state, the end of the Slave Scheduling algorithm is waited for, which is indicated by the signal MT_FH_payload. In this case, MPX/MPR enters the active state.
   - in AP, the initial state is inactive as in MT, and when the Data Pump becomes activated, MPX/MPR sends a FRAME_START signal to the Scheduler and enters the wait_SM state, in which the termination of both the Master and the Slave Scheduling algorithm is awaited, which is indicated by the signal FH_Payload. On the reception of this signal, the signal FH_MPDU containing the slot-map built by the Scheduler and the "I am Alive Invitation" contained in IAAI is sent to (all) the associated MT(s) and the active state is entered. Notice that in principle, the Slave Scheduling algorithm can be started after the FRAME_START signal, but for the type of verifications we can do, it makes the modeling simpler, but will not lead to different results.
   - The active state of MPX/MPR is the one in which the current slot-map is executed, and it is almost the same in AP and MT (except that AP is concerned with all MPDUs). As long as the increased slot-map built by the Slave Scheduler is not empty, the next element is taken:
     (a) if an MPDU has to be sent, this is done,
     (b) if an MPDU has to be received, either the mpdu_timeout or the reception of the corresponding MPDU is waited for, and when an expected MPDU is received, arbitrarily a value of CRC is chosen: if it is bad, the frame is skipped (=lost), otherwise the MAC_ADR and MVC_Id are extracted and the body sent through MPDU_BODY_RCVD to the corresponding process DLR process. In AP, also all the control activity of the concrete process MPR associated with the reception of an MPDU is done:
       - the reservation request is extracted from the MPDU headers and the Traffic Table is updated correspondingly

    - an MT_ALIVE signal carrying the MAC_ADR is sent to ADG

(c) if MT is not concerned by this MPDU (not the case in this version), the mpdu_timeout is waited for.

If the slot-map is empty (end of the time frame):

- in AP, a new FRAME_START signal is sent to the Scheduler and the wait_SM state is entered
- in MT, the hunt mode is entered again, but this time, no FH_HUNT_req is sent to Radio control, and if the hunt timer expires the signal AP_lost is sent to the Handover Indicator.

Notice that the rather synchronous notion of time of the ObjectGeode tool, in which time can only progress if no other activity is possible, is rather convenient at least for verifying the algorithm under the hypothesis that all the physical timing constraints (time needed to run the scheduling algorithm, build an MPDU,...) can be met.

As all activity in MT and AP is driven at some point by either reception of some signal through the Radio Link or the expiration of some timeout, the fact that MPX/MPR waits explicitly 1 unit of time after treating each element of the slot-map, forces time progress in the whole specification. This means that time is measured in MPDU transmission time which is realistic.

In order to verify that the algorithm is correct also when the signal from the upper layers are a bit late, we may translate the SDL specification into IF and declare some of the upper layer transitions as delayable, which has as a consequence that these signals arrive either timely or one instance later.

2.4.2 Properties related to the Data Pump

As the above described Data Pump specification is very abstract, it is not possible to verify many properties of the concrete Data Pump alone. There are interesting properties related to the Data Pump concerning the consistency of the different tables in which cell related information is stored, but

- in our model this has been abstracted away
- the information concerning the concrete specification we have is not sufficient in order to build a more concrete model.

The properties we intend to verify concern almost all the interplay of the Data Pump with one or several entities, except a few local consistency properties necessary to verify the correctness of the specified abstraction:

1. The properties establishing the correctness of our abstraction, include at least the following ones:
    (a) absence of deadlock and of time-lock
    (b) whenever there is request addressed to the data pump from the upper layer, there exists at least one execution path responding properly to it, (and when it is not the case, some problem has occurred). This property must be formulated for each pair "request/response" addressed to Data Pump and the potential problems which may prevent the answer from being sent instantiated.

(c) The Data Pumps of AP and MT are well synchronized, that is

- on each side, the Data Pump receives always an expected message, that is the one announced by the actual element of the slot-map
- there is always at most one MPDU underway in the Data Pump, that is each mpdu is received by the Data Pump specified as receiver in the actual element of the slot-map (or lost) before the next MPDU is sent by its sender (specified by the next element of the slot-map)

2. Important safety properties concern the interplay of the Control Layer (Association agents, TIP agent and Radio Control) with the Data Pump

   - Whenever MT and AP are associated, but the frequency of the MT is not that of the AP then, the MSC scheduler is suspended in MT, and the ASC scheduler knows it
   - Whenever MSC is suspended, it will inevitably be resumed again, either because a handover takes place or because the traffic with the old AP is taken up again; moreover, the possibility of taking up the traffic with the old AP exists
   - Whenever AP allows MT to go TIP (by sending an MPDU_TIP_ack to MT), either the TIP procedure will take place eventually or a handover procedure initiated; moreover, the possibility that the TIP procedure can take place exists.

   Following the specifications of the WAND documents 3D2 and 3D5, the second and the third properties cannot hold, as it is stated that the signals MPDU_TIP and MPDU_TIP_ACK are *not* sent through the broadcast MVC_id in the contention phase, but through the ordinary control channel. This means that:

   - the TIP procedure can never be ended by MT as the signal MPDU_TIP(end_of_tip) will never be scheduled as long as ASC considers MSC as suspended (and to resume it the signal MPDU_TIP(end_of_tip) must be received in the AP TIP agent)
   - whenever AP TIP agent notifies ASC that MT wants to go TIP (signal TIP_MT_ind) and ASC decides that TIP is possible, then it sends a positive acknowledgment to the TIP Agent (we use the signal TIP_MT_cnf as the name of the signal is not mentioned in the document) and stops scheduling messages for this MT. If it stops immediately scheduling messages for this MT, the MPDU_TIP_ACK sent from AP to MT TIP agent will never be scheduled, if it stops after some timeout, it may still schedule messages for MT when the MSC in MT is already suspended which does not compromise safety, but is an undesirable behavior.

   A solution for these problems consists in broadcasting both signals MPDU_TIP and MPDU_TIP_ACK in the contention phase (which has been done in our new SDL specification).

3. The opening of a new connection is notified to the Data Pump (in AP and MT) as well by the Generic Control Agent (in MT and the Association Agent in AP) through the signal SET_TRAFFIC_PARM as well as from the Generic WDLC through PID_REGISTERED (in order to communicate the Pid of the corresponding DLR process).

whenever one of the two signals is received by the Data Pump, within a some amount of time (to be determined) either the other one or its nullification (RELEASE_TRAFFIC_PARM resp. PID_DEREGISTERED) should be received.

However, it is not the case that no data may be received before the signal PID_REGISTERED as the opening of the connection in one direction may be a bit faster than in the other direction, and all cells which are sent too early (and therefore lost) will simply be retransmitted. The general synchronization properties between the opening of a connection in AP and MT are given in the section concerning dynamic control.

4. The properties concerning the interplay with WDLC during the data transmission are given in the section on the verification of WDLC.

5. An interesting question is to which extent the replication of the information in the Traffic Table over several processes by passing it as parameters of signals affects the efficiency of the algorithm. The most interesting example is the feedback information and the reservation requests piggybacked on cells. In the actual algorithm, this information is read in the Traffic Table at the beginning of the time-frame (during the scheduling algorithms in both AP and MT), and then transmitted to the MPDU handler who transmits the information at the moment determined by the slot-map. Especially in MT, all changes which have taken place meanwhile in the down-phase (new cells have been received or a transmission error detected) are ignored and transmitted only in the next time-frame. As a consequence, the master scheduler receives systematically somewhat outdated information. The use of a shared data structure containing the Traffic Table, would allow to delay the fixing of the information transmitted in the MPDU up to the moment when the MPDU is really constructed by MPX, that is a small amount of time before it is transmitted; this would in many cases (not in all) allow to take into account the information from the preceding down-phase and to obtain less useless retransmissions, and therefore better performances. As in our SDL specification the Traffic Table is a common data structure of all processes in the Data Pump, it is very easy to test both scenarios, and to compare them.

An implementation oriented remark is that, it is very likely that the best practical solution consists in sharing the Traffic Table not only within all entities of the Data Pump but also between WDLC and Data Pump, such that the TRAFFIC_UPDATED signal carries only new pointer values (next_to_be_sent, last_to_be_sent) to the Data Pump. We wonder if this replication of the same information in several modules is not just an effect of the constraints of SDL, more than a really intended feature.

2.5 Model of an abstract Physical Layer

Also in the case where we consider a single MT and a single AP, for scenarios in which both, the Data Pump and (parts of) the control layer are present we need an explicit model of the physical environment. The reason is that in this case, it can neither be modeled by some nondeterministic environment — only messages transmitted by one side can be received by the other side — nor by a simple channel between the two Data Pumps — the channel exists only when AP and MT have the same frequency and the MT scheduler is not suspended, where the frequency can be changed by the Radio Control entity.

Another interface that is not modeled in our concrete SDL description so far, are the CSE/CRE blocks which are supposed to encode peer to peer control signals into ATM cells, to transmit them transparently for the lower layers via the right WDLC instances and the Data Pump and to decode them on the other side. For verification, it is useless to implement the concrete protocol, but it is not realistic either to replace the whole protocol by a peer to peer channel, as again this channel does not exist if either AP and MT have not the same frequency or the MT scheduler is suspended.

For this reason, we define a particular abstract physical layer with the interface described in Figure 4 below which plays the role of both the physical layer and the peer CSE/CRE entities [6].

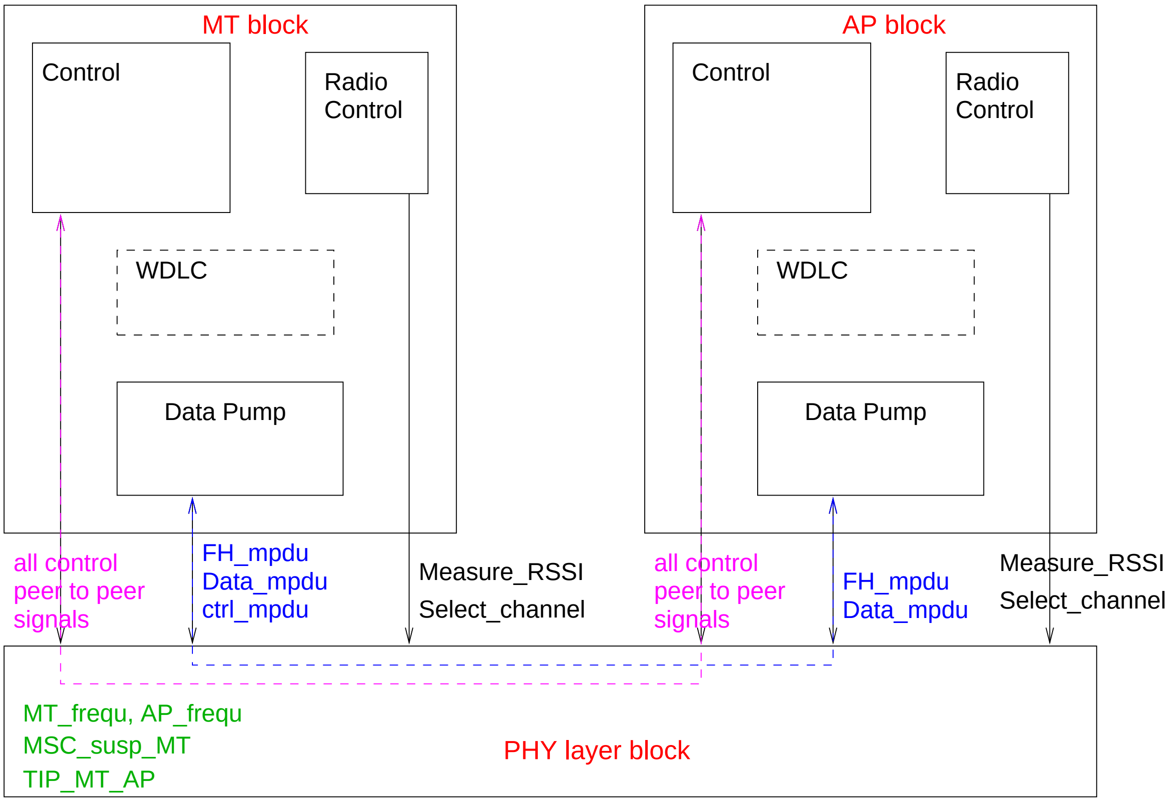

Figure 4: Signal exchange with the abstract physical layer

The functionalities of this physical layer is the following:

1. The change of the frequency by the Radio Control can take place in any state. It has the effect to change the frequency stored in MT_freq, respectively in AP_freq.

2. in the idle state, it is ready to receive an MPDU (FH_mpdu, Data_mpdu or Ctrl_mpdu) from AP (resp. MT):

   - a Data_mpdu is forwarded from MT to AP if AP and MT work on the same frequency (MT_freq=AP_freq)

- a FH_mpdu or a Data_mpdu is forwarded from AP to MT if they work on the same frequency *and* if the MSC Scheduler is not suspended (the Physical layer knows about this using the import/export constructs of SDL)
- if a Ctrl_mpdu is received from MT (in the time frame this corresponds to the contention phase), a unit_timer is set and the process goes into a state send_ctrl_signals.

3. in the state send_ctrl_signals (that is during the contention phase), PHY plays the role of the CSE/CRE entities and transmits the peer to peer control signals (in both directions) waiting in the input queue (all other signals are saved in this state and these signals are saved in all other states; in Promela or in IF, this is best modeled by using 2 distinct queues for these peer to peer control signals and for all other signals): broadcast signals are transmitted under the condition that they work on the same frequency and MSC is not suspended (MSC_susp_MT does not hold), all other control signals are only transmitted if moreover ASC has not suspended the MT (MSC_susp_AP does not hold) so that is is possible that these signals have been scheduled in the slot-map for the corresponding MVC_id). Concerning the number of control signals to be transmitted, actually we have a very eager algorithm which transmits in each time frame all pending control signals (broadcast signals can be lost, others not) by waiting the unit_timer to expire before leaving the state. Probably, in order to detect more potential problems, a more lazy algorithm must be considered (by choosing an arbitrary moment to go back to the idle state).

This physical layer represents an abstraction of the real behavior which moreover transmits control signals only if the lower layers agree, under the condition that the following property holds:

> it is always the case that when "MT and AP are associated" and "MT and AP have not the same frequency" then "MSC scheduler is suspended"

If this property (which is a correctness criterion of the protocol) does not hold, this physical layer has some impossible behaviors.

3 Verification scenarios: results and future work

3.1 Eindhoven verification experiments

We did a series of verification experiments, using all the tools to which the Eindhoven group had committed before (`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).

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 `dtspin` was used during the third phase only.

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.

We still do not know how to correct this error in the code in a safe way.

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.

Future work

After the encouraging initial results, we intend to do more ambitious experiments in the future. There are several possibilities for this.

One straightforward direction is to verify the access point (AP) side of the Steady State Control — so far it was done for the mobile terminal (MT) side only. The system could be built around the I_Am_Alive agent, the most complex entity on the AP side. The AP Steady State Control has roughly the same complexity as the MT Steady State Control, so it is to be expected that it can also be successfully checked.

A more challenging task will be to try a system that consists of several mobile terminals and access points. In that way we can check the "mobile" aspects of the protocol. Although it seems that for the time being there is a promising margin for the memory resources, the main problem will certainly be how to alleviate the state space explosion. At the moment it is hard to predict the possible number of MTs or APs that can be handled successfully. Definitely, to handle arbitrary configurations will call for parameterisation and symmetry techniques, and (automatic) abstractions.

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.

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). That is why we believe that in the future also the real-time properties of the system can be treated successfully.

Acknowledgements

We gratefully acknowledge VERILOG for giving us access to their *Object*GEODE SDL environment [9].

References

[1] K. Baukus, Y. Lakhnech, K. Stahl, and M. Steffen. Divide, abstract and model-check. Technical report, 1999.

[2] M. Bozga, J.Cl. Fernandez, L. Ghirvu, S. Graf, J.P. Krimm, L. Mounier, and J. Sifakis. IF: An intermediate representation for SDL and its applications. In *SDL forum 99, Montreal*, June 1999.

[3] M. Bozga, S. Graf, L. Mounier, and J. Sifakis. The intermediate representation IF. Technical report, Verimag, October 1998.

[4] Dennis Dams, Susanne Graf, Guoping Jia, Martin Steffen, and Diana Tourko. Understanding the Mascara protocol: An informal specification. Technical report, April 1999.

[5] Ioannis Dravapoulos, Nikos Pronios, Spyros Denazis, Nikosd Passas, Dimitris Skyrianoglou, Geert Awater, Neda Nikaein, Alain Enout, Stephane Decrauzat, Thomas Kaltenschnee, Thorsten Schumann, Jürg Meierhofer, and Stefan Thömel. *The Magic WAND, Deliverable 3D2, Wireless ATM MAC*, September 1997.

[6] Guoping Jia and Susanne Graf. SDL-code of an abstract Data Pump, physical layer and CSR coding/decoding entity. Circulated within the Vires-project, 1999. Version from Mai 1.

[7] Florence Pagani. A tentative specification of the Mascara protocol. Circulated within the Vires-project, 1998. Version of October 26.

[8] Nikos Pronios, Ioannis Dravapoulos, Ioannis Pavlidis, Ioannis Marias, Nikos Passas, Geert Awater, Martin Janssen, Jürg Meierhofer, Beat Keusch, Adrew Lunn, and Frederic Bauchot. *The Magic WAND, Deliverable 3D1, Wireless ATM MAC Overall Description*, December 1996.

[9] Verilog. Object*GEODE SDL Simulator - Reference Manual.* http://www.verilogusa.com/solution/pages/ogeode.htm, 1996.