Verifying the MASCARA protocol in SPIN*

An experience report

Dragan Bošnački[1], Dennis Dams[2],
Leszek Holenderski[1], Natalia Sidorova[2]

[1] Dept. of Computing Sci., Eindhoven University of Technology,
PO Box 513, 5600 MB Eindhoven, The Netherlands
{dragan, lhol}@win.tue.nl

[2] Dept. of Electrical Eng., Eindhoven University of Technology,
PO Box 513, 5600 MB Eindhoven, The Netherlands
{dennis, natalia}@ics.ele.tue.nl

Abstract

We present a case study in using SPIN to partly analyse a complex industrial protocol called MASCARA. The purpose of MASCARA is to extend the ATM networking protocol with wireless communication over radio links. Our analysis exposed several non-trivial errors in the SDL model of MASCARA. In order to perform our verification experiments we had to use a version of SPIN extended with a front-end which allows to translate SDL to Promela. Since MASCARA is a real-time protocol we also had to extend SPIN with a back-end which allows to analyse timing properties.

1 Introduction

The paper presents an experience report on the application of SPIN in the context of the Esprit Long-Term Research project VIRES (Verifying Industrial Reactive Systems) [12]. The goal of VIRES is to enhance existing tools for automated verification of reactive systems, and to apply the enhanced tools to the verification of a complex industrial protocol.

The protocol is an extension of the ATM (Asynchronous Transfer Mode) networking protocol to wireless communication over radio links. Such a wireless ATM protocol offers to mobile users the advantages of the standard wire-based ATM protocol: high reliability, capacity-on-demand, and service-independent transport.

A wireless ATM network comprises of a fixed wire-based ATM network which links a number of geographically distributed access points. These access points transparently extend ATM services to mobile users via radio links operating in the 5.2 GHz frequency band and achieving data rates of 23.5 MBits/s. The protocol that mediates between mobile stations and the access points is called MASCARA. It was developed by the WAND (Wireless ATM Network Demonstrator) consortium [13].

*This research has been supported by the LTR project VIRES (Verifying Industrial Reactive Systems) of the European Community.

WAND has provided the VIRES consortium with the specification of MASCARA. Since the specification is mostly informal, the protocol description is being further formalised within VIRES, and some correctness properties are being investigated. For a formal description of MASCARA, the SDL specification language has been chosen in VIRES.

One of the tools used in VIRES is SPIN, and its application to the verification of MASCARA is the central issue of this paper. In order to perform verification experiments we had to extend SPIN in two ways.

First, the verification of MASCARA in SPIN requires a translation from SDL to Promela. In order to minimise the probability of introducing errors, this translation had to be performed automatically, by a tool developed within VIRES.

Second, MASCARA is a real-time protocol and its SDL specification uses timers to specify quantitative timing constraints between events. Since Promela can not directly deal with timing constraints, it has been extended with discrete time, and the underlying SPIN verification engine has been adapted accordingly.

The front-end and back-end tools are being developed within VIRES and have successfully been used in our verification experiments in which we formally analysed a central component of MASCARA. This analysis exposed several errors in the SDL model of MASCARA. Among the errors that were found are a number of deadlocks and a livelock. Two functional properties that state a proper alternation of some request and confirm messages were analysed. One was proved and the other was disproved.

The paper is organized as follows. In section 2 we give an overview of the MASCARA protocol. Section 3 is devoted to the description of the additional tools, beside `spin`, we needed in our verification experiments. The verification experiments are described in section 4. Finally, we conclude with section 5.

2 Functional description of the MASCARA protocol

2.1 General description of MASCARA

MASCARA (Mobile Access Scheme based on Contention and Reservation for ATM) is a protocol designed for wireless ATM (Asynchronous Transfer Mode) communication in local area networks. It has been developed within the WAND (Wireless ATM Network Demonstrator) project [13]. The task of the MASCARA protocol is to provide wire-like services to ATM connections while the transmission medium that lies underneath is a wireless radio link. In addition, the protocol has to control the access to this shared medium. This transparency with respect to ATM connections is a challenge, as standard ATM has not been designed to work in a wireless environment, but rather assuming a quasi error-free transmission medium that provides full duplex and point-to-point connections. In contrast with that, the radio medium is characterised by a high bit error rate (BER), the transmission mode is typically broadcast (or at least multicast) and the scarcity of available radio bandwidth calls for a time division duplex (i.e. half duplex) mode. All this leads to the necessity of extra functionality allowing to cope with these dissimilar environmental properties.

The purpose of a wireless ATM network is to provide native ATM service to its users. Therefore the network architecture can be presented as an extension to a fixed-wire ATM network (see Figure 1). The access network is based on a centralised approach, where several Mobile Terminals (MTs) communicate with the ATM core network through an Access Point

(AP). The communication between MT and AP goes via the wireless radio link. The protocol stack, corresponding to this network architecture, is displayed in Figure 2.

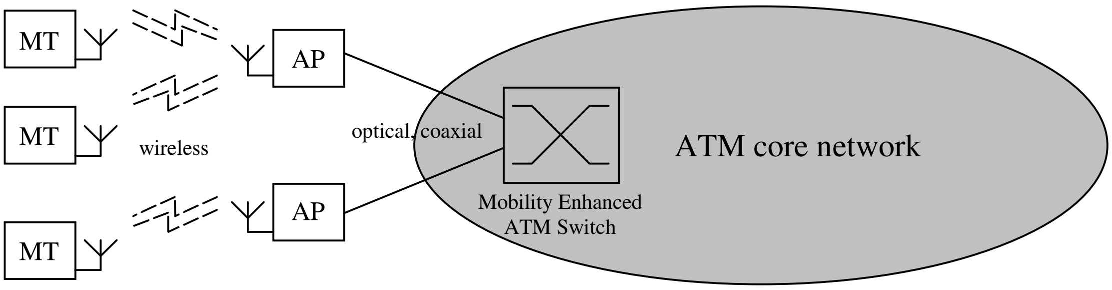

Figure 1: Architecture of a wireless ATM network

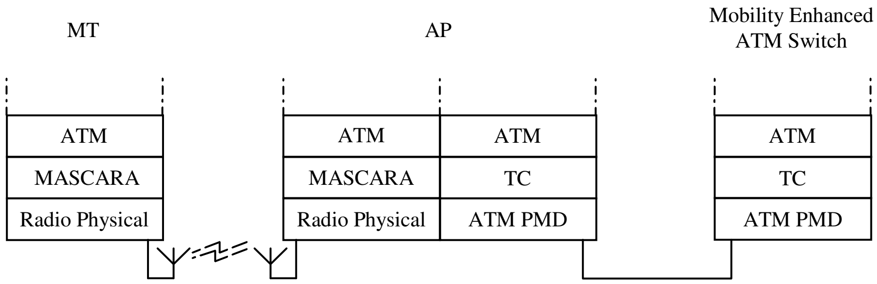

Figure 2: Architecture of WAND system

In terms of MASCARA the communication channel between AP and MT is an *association*. It is possible to have several *connections* within one association.

The protocol consists of the following high level functional entities (see figure 3):

- Layer Control Protocol and Message Encapsulation Unit (together called Inter Control Communication) — management of the information exchange between the MASCARA layer and the entities located within the upper layers.
- MASCARA Control (MCL) — the internal control functions, peer-to-peer (AP-MT) signalling and control communication of the MASCARA protocol. Inter alia, it is involved in procedures related to association management and handover.
- Wireless Data Link Control (WDLC) — recovering the wireless radio link quality by using error control at ATM cell level; flow control by means of a sliding window protocol; extracting the data cells arriving through the Radio Physical layer.
- Control Segmentation and Reassembly (CSR) — in one direction, segmentation of peer-to-peer control messages that it receives from MCL and forwarding them to the WDLC Transmitter for further, transparent transmission; in the opposite direction, reassembling ATM cells that it receives through the WDLC Receiver into control messages and delivering them to MCL.

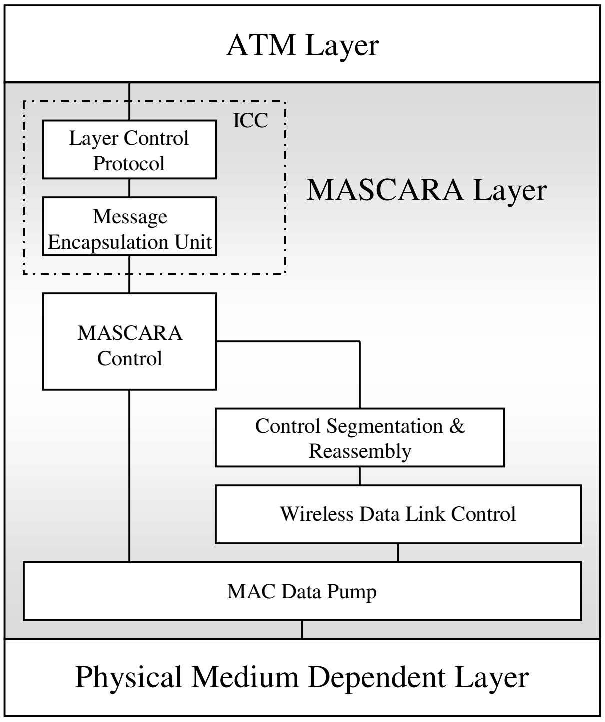

Figure 3: Functional entities of MASCARA

- MAC Data Pump (MDP) — control of the interface with the Radio Physical Layer; scheduling which allocates ATM cells into time slots according to different ATM traffic contracts; dealing with incoming cells, error control and cyclic redundancy check calculations (the latter as a service to the WDLC).

A crucial feature of MASCARA is the support of *mobility*. An MT located inside the area cell of an AP is capable of communicating with it. Whenever MT moves outside the area cell of its current AP, it has to perform a so-called *handover* (HO) to an AP whose area cell MT has moved into. A handover must be managed transparently with respect to the ATM layer, maintaining the agreed quality of service (QoS) for the current connections. So the protocol has to detect the need for a handover, select a candidate AP to switch to and redirect the traffic with minimal interruption.

It is the MASCARA Control entity that is responsible for the handling of mobility issues. That is why our verification efforts were put on the MASCARA Control, particularly on the parts of MT Control managing the handover procedure.

2.2 MASCARA Control (MCL)

The purpose of MCL is to ensure that an MT remains associated to the ATM network through some AP with good quality and minimal interruptions for as long as the ATM connection is valid. MCL carries out the periodical monitoring of the current radio link quality, gathering the information about radio link qualities of its neighbouring APs to handover to in the case of deterioration of the current association link quality, and switching from one AP to another in the handover procedure.

One distinguishes 2 types of handovers in MASCARA: *backward handover* and *forward handover*. The forward handover procedure starts when the connection to the current AP is lost and MT urgently needs to find another AP. Backward handover takes place when MCL notices the deterioration of the quality of the current association. Then it looks for the 'best' alternative AP to switch to. After the alternative AP has been found, the MT tunes to its old AP. It keeps the association to the old AP until it gets the association to the new one. So the MT is able to perform its normal activity in the period the upper layer accomplishes its part of the procedure of associating to the new AP.

We give a brief description of the MCL sub-entities (see figure 4).

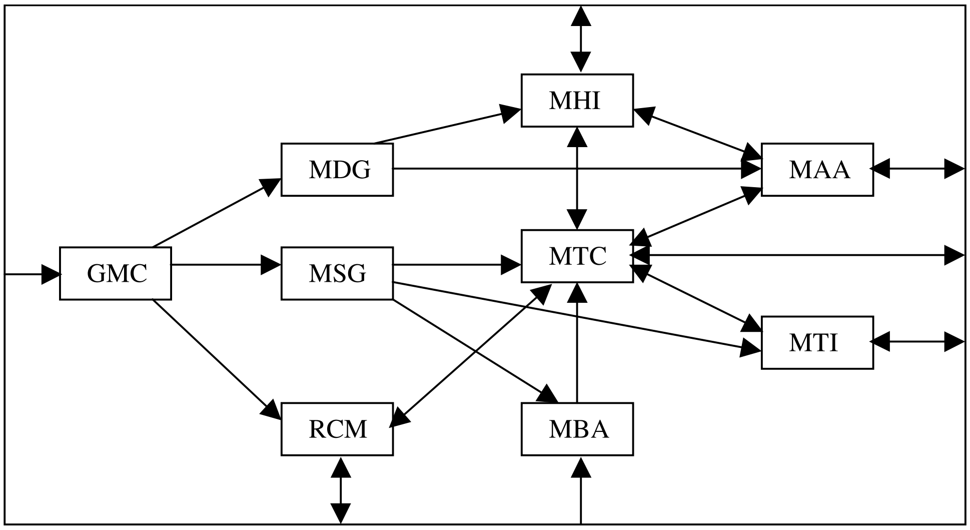

Figure 4: Structure of MCL

Initialisation and termination procedures have a tree-like structure. *Generic MASCARA Control* (GMC) brings into operation and terminates *MT Dynamic Generic Agent* (MDG), *MT Steady State Generic Control* (MSG) and *Radio Control* (RCM). MDG and MSG are responsible for starting and terminating their lower sub-entities (MAA and MHI for MDG; MBA, MTI and MTC for MSG).

Radio Control (RCM) supplies services of communication with the Radio Physical layer. Upon request it instructs the radio modem to tune into a specific radio frequency and reports back whether it succeeded or not.

MT Association Agent (MAA) communicates with the upper layers to make an association to an AP and performs the address mapping.

MT Beacon Agent (MBA) is responsible for extracting and distributing information about neighbouring APs using beacon control cells. This information (frequencies, addresses) is used by the MT during the periodic check of the radio link quality of the reachable APs and in the handover case.

MT Handover Indicator (MHI) is responsible for monitoring the quality of the current association and manages a part of a handover procedure. This entity is crucial in handover scenarios. After initialisation, MHI waits for the MT to become associated. When that has happened, it sets its timer to expire when the first periodic check of the quality of the current association should be executed. This check is performed during normal MT functioning. Every expiration of the timer, MHI collects detailed information about the association quality via some entities located outside of MASCARA control and uses this information to decide whether a backward handover is needed. If not, it sets the timer again. If handover is needed, it informs MT Target Cell (MTC) about it.

When it gets a signal informing that the AP is lost, i.e. a forward handover should take place, it resets its timer and informs other entities about starting the handover. When MHI gets the signal informing that the handover has succeeded and containing the address of the AP to switch to, it forwards this information to the upper layer, which resends it later to MAA (see above) whose job will be to complete the association.

MT TIP Agent's (MTI) task is to negotiate permission for the MT to go temporarily incommunicado (enter TI-mode). This means that the MT will be temporarily out of reach for the current AP, contacting other APs to get information about its radio environment. When MTI receives a request from MTC to go temporarily incommunicado, it subsequently asks the AP side for such permission. Then two things may happen: the acknowledgement arrives or the timer restricting the period of waiting for the reply expires.

If the acknowledgement arrives, the timer is reset and MTC is informed that the MT is allowed to enter TI-mode. Afterwards MTC informs MTI that the TI procedure is completed and MT has returned to its old AP. Then MTI informs the AP side that MT is back.

If the timer expires but the maximum number of possible retries has not been exceeded yet, MTI tries again to get TI permission to go temporarily incommunicado. Otherwise MTI sends a negative acknowledgement to MTC.

MT Target Cell (MTC) is a key entity for the verification of the handover scenarios. It has multiple tasks. First, it assists MAA in the association procedure. Second, it acts as a slave to MHI assisting in the handover procedure (HO-mode). And third, it autonomously performs a periodical check of the radio link quality for neighbouring APs (TI-mode).

Being initialised, MTC first has to synchronise with an AP that means tuning into the right frequency and catching a frame header. Synchronisation requests are received from MAA and, after proper address-to-frequency mapping, forwarded to RCM. After relaying RCM's response back to MAA, MTC waits for the notification from MAA that MT has been associated. The synchronisation and association procedure takes place not only during start-up, but also in case of a handover to another AP. So the synchronisation procedure can be entered irrespective of the fact whether MT is currently associated or not.

As soon as the association procedure is completed MTC sets a timer that is intended to expire when the first TIP-procedure should happen. Scanning alternative frequencies (TI-mode) is the standard behaviour of MTC. The goal of that is to maintain an up to date, ordered list of alternative APs that can be used in a handover case.

Upon the timer expiration, MTC starts the *TI procedure* with the suspension of MHI that should be prevented from checking properties of the current association while there is no real association. MTC also suspends MT Scheduler (MSC) in MDP in order to stop the data flow until tuning back into the old AP frequency. MTC orders RCM to tune into the frequencies of all the neighbouring APs and check the quality of the connection.

When the end of the list has been reached, MTC requests MSC and MHI to resume their operations. Further it orders RCM to tune into the frequency of the old AP. If RCM has succeeded to synchronise with the old AP again, MTC informs MTI that the TI-mode has been ended. MTI in its turn informs the AP side.

RCM can give a negative answer to the request to tune back to the old AP. In other words: MT has lost its connection while it was in TI-mode. In that case MT enters a forward handover scenario. It is called forward handover because MT cannot exchange information with its old AP to allow for a graceful handover (meaning proper de-association and re-association). MTC informs MHI that it has lost the association with its AP.

Assisting in handover procedure (HO-mode) is another function of MTC. When MT is associated to some AP and currently not in TI-mode, MTC is able to consume a request from MHI asking for the best candidate AP to handover to.

After such a request MTC suspends the activity of some entities and goes through the AP-list built in TI-mode until it finds an AP whose radio link quality is over a threshold.

It is not guaranteed that MTC will succeed in its mission to find an alternative AP. It might happen that the AP-list gets exhausted. When this occurs, MTC just starts all over again from the beginning of the AP-list in hope that some 'good' AP will appear in the neighbourhood.

Have found an alternative AP, MTC resumes the previously suspended entities, informs MHI about the candidate AP it has found and waits for the actual de- and re-association to take place. In case of a backward handover, MTC sends a request to RCM to tune into the frequency channel of the old AP. If this attempt was successful, meaning that the MT is still associated to its old AP, the MT remains associated with this AP until it gets associated with the new one.

It may happen that RCM does not succeed in tuning back into the old AP frequency, meaning that the MT has lost the association with its old AP while it was in HO-mode. Then MTC informs other entities that it has lost its old AP and waits for getting the association with a new AP.

3 Tools

Assuming that the reader is familiar with Promela and `spin`, in this section we give a short description of the other languages and tools that we used in our experiments. The SDL specification was translated through the pipe of the translators `sdl2if` and `if2pml`. `sdl2if` is a translator from SDL to the intermediate Format (IF) language for representation of timed asynchronous systems, that we describe in the next subsection. The obtained IF output was further transformed by the `if2pml` translator into DT Promela – a discrete-time extension of Promela. DT Promela files served as inputs for `spin` and its discrete-time extension `dtspin`. The experimental results had to be checked against the SDL specification. The whole process is represented schematically in Figure 5.

All the tools (except `spin`, of course) described in this section are developed in the framework of the VIRES project. The language IF and the translator `sdl2if` were created at Verimag, Grenoble, while `dtspin` and `if2pml` were developed at the Eindhoven University of Technology. Part of the work on `dtspin` was done at Bell Labs in collaboration with Gerard

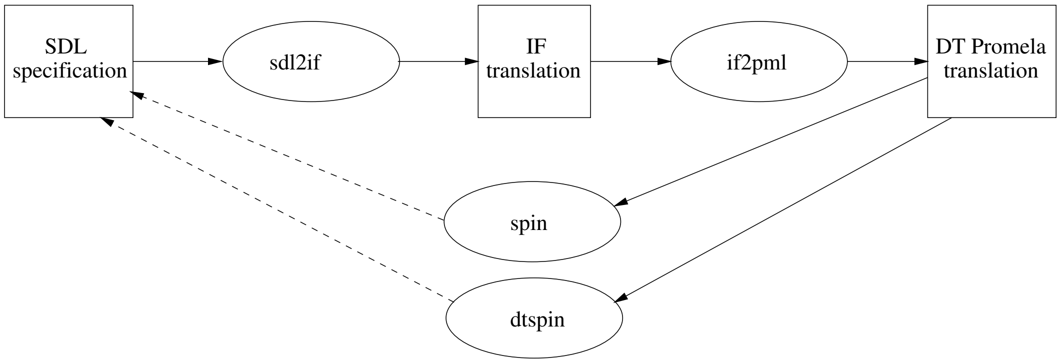

Figure 5: The tools used in the verification.

Holzmann.

3.1 IF and `sdl2if`

The language IF (Intermediate Format) [4, 5] was developed as an intermediate representation language for timed transition systems. The main motivation behind IF is to design an intermediate formalism for connecting several industrial formal description techniques (FDT), such as LOTOS and SDL, to verification tools developed in the research community. In the brief overview of IF that follows, we present mainly the features that are important for the translation from SDL.

A system in IF is represented as a set of processes that communicate by exchanging signals (messages) through a set of buffers. In the translation from SDL each process is assigned a unique buffer, which we will refer to as *the input buffer* of the process. Apart from the standard language features, like global variables and user defined data types, one can define *clocks* and *timers* in order to capture the timed behavior of the system. Clock values increase with the passage of time and can be reset to zero (like in timed automata [1]), while the timer values, after being set to some value, decrease towards zero with the passage of time. In the translation from SDL only timers are used.

A process in IF consists of a set of local variables, a set of control states and a set of control transitions between the states. Several attributes are defined for each state. The most important for us are `nostable`, used to implement the atomicity of the transitions and `discard` giving the lists of signals that are only removed from the buffer without triggering any further action in the state.

In the SDL translations we deal with two types of IF transitions: *input transitions* which are triggered by a signal read from the communication buffers, and *internal transitions*, which are spontaneous and independent on the input buffer's content. The syntax of the input transitions is

`from` *state1* `input` *sig* `from` *buf* `do` *body* `go` *state2*

where *state1* and *state2* are the source and target states, respectively, *sig* is a signal, *buf* is a buffer, and *body* is an atomic sequence of actions that can be outputs to other buffers, assignments between ordinary discrete variables, and sets (resets) of timers (clocks). The meaning of the transition is that if the process is in *state1* and *sig* can be read from *buf*, then *body* is executed, and after that the process goes to *state2*. All these actions are done atomically. The internal transitions have the same structure except for the `input` part that is omitted. The formal semantics of IF is beyond the scope of this paper and can be found in [4].

`sdl2if` [5] is a prototype translator from SDL to IF, implemented using the SDL/API Interface provided by ObjectGeode [11]. Currently `sdl2if` is able to translate a broad static subset of SDL, meaning that the dynamic creation of processes and some mobility aspects are not covered.

3.2 DT Promela and dtspin

`dtspin` is an extension of `spin` with discrete time. A more detailed description of `dtspin` and its implementation can be found in [3] and `http://win.tue.nl/~dragan/DTSpin.html`.

In `dtspin` we assume a time model in which time is divided into slices indexed by natural numbers. One can see the slice indices as readings of a fictitious global digital clock, that ticks at the end of each slice. System events can happen not only at, but also between the clock ticks. The events happening in the same slice are assigned the same clock value. The elapsed time between events is measured in ticks. In that way we obtain a quantitative estimation for the elapsed time between events belonging to different slices. Within the slices however, we can only have a qualitative relation that one event happens before another, as in the time free case. In our model, time passes only if no other action in the system is possible.

In order to capture the timing features, standard Promela is extended into DT Promela. For this purpose a new data type `timer` that corresponds to discrete-time countdown timers is introduced. Furthermore, two new statements, `set` and `expire`, that operate on `timers` are added. Their syntax is `set(`*tmr*`,` *val*`)` and `expire(`*tmr*`)`, where *tmr* is a `timer` variable and *val* is of type `short` integer. With `set`, timer *tmr* is assigned value *val*, while `expire` tests whether the value of *tmr* is zero. There is also a third, implicit, operation on timers which is not available for the user as a language statement. This is the `tick` operation that decreases by 1 the value of the active timers every time slice. The statements `set` and `expire` are sufficient for modeling all timing features of the system. With them one can define new compound statements as Promela macro definitions.

The timed extension is fully compatible with the standard `spin` package and all properties that are on `spin`'s repertoire can be also validated for discrete-time models. Besides qualitative properties, a broad range of quantitative timing properties can be verified, by using boolean expressions on timer values in the assertions and LTL formulae.

It is worth noting that `dtspin` is compatible with the partial order reduction of standard `spin` [8, 10] and moreover it uses an adaptation of this algorithm for timed systems [3].

3.3 `if2pml`

From the very beginning `if2pml` was intended to be a tool that performs the second phase of the translation from SDL to Promela. This had the following two consequences.

First, we decided that the translator would support only a subset of IF. As IF is intended to be an intermediate language for a variety of high-level specification formalisms, it is quite expressive, but not all constructs are needed to represent SDL. So, we concentrated only on the features that were relevant to the translation from SDL.

```
#define timer bool

#define set(x,t)        x=true
#define reset(x)        x=false
#define expire(x)       x==true -> x=false
```

Figure 6: Macro definitions of timers and timer operations.

The second consequence was to take DT Promela as a target language. This was necessary because standard Promela does not support time while SDL does. However, for properties that do not depend on time we could use virtually the same DT Promela models also with `spin`. This has an advantage because usually `spin` consumes less resources than `dtspin`. The main problem was how to abstract timers properly, i.e., how to handle the features of DT Promela that are not supported by `spin`. We solved this by defining `timer`, `set`, `reset` and `expire` as Promela macro definitions in a special header file (given in Figure 6) that was included in each of our models.

We define `timer` to be a synonym for type `bool`. In our timer abstraction we allow timers to expire nondeterministically at any time, with the constraint that a timer can expire only if it is set (implemented in the `expire` macro). Because in DT Spin the basic timer operations are also implemented as macro definitions, we were always able to use the unchanged Promela model that was produced by `if2pml`. The only intervention that was needed when switching from `dtspin` to `spin` and vice versa, was to change the contents of the special header file `dtimesdl.h`. This file is always included by `if2pml` in the beginning of the DT Promela models.

One can say that in the overall SDL to Promela translation, `sdl2if` resolves the hierarchical aspects of SDL by "flattening" the hierarchy of blocks down to bare processes and appropriately resolving the sources, destinations and priority of the signals exchanged between the processes. On the other hand, `if2pml` performs the translation of the language core, centered around the implementation of the transitions (issues like preserving the atomicity of the transition and implementation of the discard mechanism).

As in IF, Promela models are also sets of processes that communicate via buffers. This provides an almost one-to-one translation of these concepts. Also, most of the data types can be directly translated to their Promela counterparts. An exception are the enumerated types that are translated as sequences of `#define`s. Promela's declarator `mtype`, that allows introduction of symbolic names for constants values, is not a solution. The reason is that there can be only one `mtype` declaration in the Promela source. [1] Another problematic issue is the range type. The translator accepts only the ranges starting with zero, because of the similar constraint on the array indices in Promela. As the concepts of timers in SDL and DT Promela coincide, the translation of the IF statements defined on timers (`set`, `expire` and `reset`) is a simple mapping to their DT Promela namesakes.

The translation of the structure of the processes and transitions is slightly more involved.

First, when composing the translation of an IF process, in some sense `if2pml` has to reverse the effect of `sdl2if`. This is because in SDL sources the transitions are listed immediately

[1] The new versions of Spin will allow to define several `mtype`s, which will make the translation of the enumerated type via `mtype` possible.

```
...
process proc1: buffer buf1;
state
   ...
   state1
      discard sig3 in buf1
   ...
   state2
   ...
transition
   ...
   from state1 input sig1 from buf1 do body1 to state2;
   ...
   from state1 input sig2 from buf1 do body2 to state3;
   ...
```

Figure 7: A skeleton of an IF program.

after the states they originate from, while in IF they are given as separate lists of states and transitions. In the Promela translation we have to again "reunite" the transitions with the corresponding states.

One of the most important tasks from a semantical point of view is to preserve the atomicity of transitions during the translation. We achieve this by putting the transitions into an `if` statement surrounded by an `atomic`. The discard mechanism is implemented via self-looping to the same state, after reading a signal that has to be discarded in the state. IF states are translated into Promela labels. An example of the translation scheme can be seen by comparing the IF source given in Fig. 7 and its Promela translation in Fig. 8.

There is a point related to the atomicity of IF(SDL). It concerns the fact that `sdl2if` breaks some complex SDL transitions into two or more IF transitions. (For instance, this happens if the SDL transition contains a choice.) In order to connect these IF transitions, new states (that do not exist in the SDL source) are introduced between them. In IF terminology those states are called *non-stable* states and are labeled with `nostable` (see section 3.1). In a non-stable state time cannot pass and the transitions to and from this state cannot be interleaved with any other transition, i.e. they are done atomically.

The non-stable states do not fit the translation schema for the ordinary states. The reason is that in the solution for the ordinary states we jump outside the `atomic` construct when going into a new state. (See for example the jump from `state1` to `state2` in Fig. 7.) According to the Promela semantics, the atomicity is lost in this case. To avoid this, we move the label that corresponds to a non-stable state inside the `atomic`. In that way the jump in the Promela program is from one atomic inside another atomic with preserving atomicity. [2] For example, the IF code from Fig. 9 will be translated into the Promela code given in Fig. 10. Notice that if the label `ns_state2` had been outside `atomic` the atomicity would have been lost after the jump from `state1`.

The implementation of `if2pml` is still in a prototype stage and some SDL features are not

[2] Because of that, we need to introduce a dummy `skip` statement before the label that corresponds to the non-stable state.

```
...
proctype proc1() {
...
state1:
atomic {
if
:: buf1?sig1->translated_body1; goto state2;
:: buf1?sig2->translated_body2; goto state3;
...
:: buf1?sig3->goto state1; /* discard */
fi
}
...
state2:
atomic{
...
}
...
}
...
```

Figure 8: Promela translation of the structure from Fig. 7

```
state
   state1
   ...
   ns_state2: nostable
   ...
transition
   ...
   from state1 input sig1 from buf1 to ns_state2
   ...
```

Figure 9: Example of a non-stable state in IF.

```
...
state1: atomic {
  if
  ...
  :: buf1?sig1->goto ns_state2;
  ...
  if;
}
...
atomic {
  skip;
  ns_state2:
  if
  ...
  if;
}
...
```

Figure 10: Translation in Promela of a non-stable state.

covered yet. The most notable omissions are the save mechanism, the dynamic creation of processes, and the abstract data types. The implementation of save in Promela is possible, but rather involved. Besides, it would mean an unnecessary complication of the obtained Promela outputs and burden on the state space during the verification. The dynamic process creation is not implemented because it is not supported yet by `sdl2if`. Unlike save, this feature is possible to handle in Promela via the `run` statement in a quite straightforward way. Finding a satisfactory solution for the abstract data types remains a future work for both `sdl2if` and `if2pml`.

4 Verification experiments

We did about 20 verification experiments whose complexity increased progressively. Here we describe only several of them, namely those whose results were conclusive. All the experiments were devoted to analysing the behaviour of MT Target Cell (MTC) which is the most complex component of the MASCARA Control (MCL) module.

We first obtained the Promela model of MTC, by pushing the SDL specification through the pipe of translators `sdl2if` and `if2pml`. During this phase, some minor bugs were exposed in `sdl2if` and `if2pml`, but these were easy to correct. The verification experiments were done in two phases, described below.

4.1 Phase I

Before the verification experiments could be started, the Promela model of MTC had to be closed, by providing an environment which would simulate the behaviour of those MCL components which MTC communicated with. We hand coded a greatly simplified environment directly in Promela. This consisted in observing that MTC communicates with its environment using several variants of a simple request/confirm protocol in which the request alternates with the confirmation. In fact, MTC uses several variants of this protocol: the confirmation can be just a single signal, a double success/failure signal, or it can be guarded by a timer. The respective request/confirm protocols were coded as separate Promela processes running in parallel.

The verification experiments performed during the first phase were mostly inconclusive, due to the fact that `spin` could not enumerate the state space (even with 2048MB of memory). The state explosion problem was caused by the initial environment being too abstract (too nondeterministic), due to the naive modelling of all the request/confirm protocols as independent Promela processes. In the real environment of MTC, the protocols are strongly dependent, arranged as cascades of the simple request/confirm protocols triggered by a main request to MTC. But even the oversimplified initial abstraction allowed to expose a real deadlock in MCL. The deadlock is described below.

First deadlock. Most of components of MASCARA must be initialised. The components are arranged in a tree-like hierarchy, and the initialisation phase is triggered by sending the *init* signal to the root. Each node triggered by the *init* signal resends the signal to all its children which need to be initialised. However, it has turned out that a possible race condition in such a cascade of *init* signals had been overlooked. `spin` found a trace in which an already initialised component was able to request a service from another, yet uninitialised component. Such a request was simply discarded and thus a confirmation was never sent back, leading to a deadlock in MCL.

4.2 Phase II

We decided to build a less abstract environment, by replacing our simple Promela processes with the SDL models of all the MCL components on which MTC relied directly. Of course, we had to slightly modify the components in order to abstract out the real behaviour of other components on which the whole MCL relied, in turn. After this phase we had quite a faithful Promela model for the whole MCL. The model was obtained automatically, with `sdl2if` and `if2pml`. The verification experiments performed during the second phase were more successful. We exposed further problems with structural properties of MCL (deadlocks and livelocks), verified some functional property and disproved another one. These are described below.

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

MTC asks for permission to start the TI procedure. However, it may happen that before getting the reply for this request the deterioration of the current radio link quality is diagnosed by MHI, and thus MTC gets an order to start a backward handover. After finding the alternative AP, the synchronization with this new AP is supposed to be started. Instead, MTC gets permission for the TI procedure as an 'latish' answer to its previous request, so it starts this procedure.

Since this is a completely unexpected behaviour, the synchronization request that comes from MAA to MTC (while MTC is in the TI procedure) is discarded. Consequently, its confirmation is not sent by MTC to MAA and this results in MAA being blocked while waiting for the confirmation. MCL reaches a deadlock situation.

Third and fourth deadlock. In another experiment we enumerated the state space and `spin` reported that there is some unreachable code.

Surprisingly its analysis showed that the forward handover never takes place. However, the part of the code where the order to start the forward handover procedure is sent, was reached. Moreover, this signal was eventually consumed by the Handover Indicator and it started the forward handover procedure.

We reconstructed with Spin such a scenario and it turned out to be a deadlock: we had MTC waiting for the signal from MAA, that was waiting for the signal from the upper layer, that in turn could not send it before receiving the signal from MHI, but this one was waiting for the signal from MTC. So we had a classical deadlock situation because of a circular waiting.

To make a long story short, an error happened during the unsuccessful attempt to get back the connection to the old AP after the backward handover procedure.

In another experiment, `spin` exposed one more deadlock which occurs after loosing the connection to the old AP during the TI procedure.

Livelocks. One of the verification experiments performed in the first phase found a livelock, but after analysing the SDL code, it turned out that this potential livelock was not allowed by the MASCARA protocol. It was introduced by our oversimplified abstraction of the environment. This was corrected in the less abstract environment considered in the second phase.

On the other hand, another verification experiment performed in the second phase found a livelock which turned out to be in fact allowed by the MASCARA protocol.

4.3 Functional properties

After discovering several problems with structural properties of MCL (deadlocks and livelocks) we attempted to analyse some functional properties. Two experiments concluded in interesting results.

A proved property. As already mentioned, MTC communicates with its environment using several simple request/confirm protocols in which the request should alternate with the confirmation.

In particular, MTC uses this alternating pattern when it communicates with MAC Data Pump, in order to temporarily suspend MSC (the scheduling process that is contained in MAC Data Pump) 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` which stopped with a strange error message `queue 2 deleted`. So the property was encoded instead as a direct observer (using the `assert` statements) embedded in the Promela code of MCL. `spin` could enumerate the whole state space checking that the property is not violated.

A disproved property. The main task of MTC is to periodically update a list that contains information about the quality of radio links with neighbouring APs. A special protocol for updating the list is called the TI procedure. As it was already mentioned there is no connection to the associated AP during the TI procedure. So the correct decision on the necessity of a backward handover can not be made. It is supposed that no backward handover request should be sent during the TI procedure.

The property can be encoded as a simple observer for the following safety property: a handover request never comes in between the two messages which mark the beginning and the end of the TI procedure. The verification experiment which was supposed to confirm this property showed instead a trace which violated the property.

Unfortunately, the erroneous trace could not be simulated by the SDL model of MCL, so it seemed that the violation of the property was a false error caused by abstracting from time. However, when we repeated exactly the same experiment with `dtspin` we got a different erroneous trace which could then be simulated by the SDL model. Thus, `dtspin` allowed us to find yet another error in MASCARA protocol.

The explanation of the different results obtained with `spin` and `dtspin` is simple: the TI procedure is triggered by a timer so the behaviour of the TI protocol could indeed depend on proper timing. As explained in section 3.3, timers can expire in `spin` at any time, so `spin` can produce a wrongly timed trace which is then not accepted by an SDL simulator which allows only properly timed traces.

4.4 Some statistics

We have to admit that when we planned the first series of verification experiments, which we present in this paper, we expected to quickly reach the limits of `spin` and `dtspin`. We were proved wrong.

In the experiment mentioned in section 4.3, `spin` reported the following statistics:

```
State-vector 416 byte, depth reached 3450, errors: 0
  55959  states, stored
  23.727 memory usage for states (in MB)
  25.582 total memory usage (in MB)
  14.2   total time (in seconds)
```

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).

On the other hand, `dtspin` reported the following, and only slightly worse, statistics:

```
State-vector 428 byte, depth reached 3988, errors: 0
  56818  states, stored
  24.773 memory usage for states (in MB)
  27.361 total memory usage (in MB)
  14.8   total time (in seconds)
```

Since this experiment consumed the most resources, among all the experiments performed in the second phase, it is quite likely that with our 2048MB memory we will be able to handle a model about 75 times bigger.

5 Conclusions and future work

SPIN has turned out to be a useful tool in formal analysis of a complex industrial protocol, when supported by the presented extensions. It performed well in both exposing errors, during an early stage of the protocol's design, and in increasing a designer's confidence in the correctness of the design.

After the encouraging initial results, we plan to attempt a new series of experiments. There are several possible extensions.

One straightforward direction is to verify the access point (AP) side of the MASCARA Control — so far it was done for the mobile terminal (MT) side only — which has roughly the same complexity as the MT side, so it is 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 automatic abstraction techniques, including parameterisation and symmetry.

Since it turned out that the introduction of time did not significantly increase the state space of the verified model, we believe that at least some of the real-time properties of the system can be treated successfully.

Acknowledgments

We gratefully acknowledge VERILOG for giving us free access to their *Object*GEODE SDL tool. We would also like to thank Marius Bozga, Lucian Ghirvu, Susanne Graf and Gerard Holzmann for fruitful discussions during the implementation of the `if2pml` tool. We thank Frank van Wijk for sharing his pictures with us.

References

[1] R. Alur, D.L. Dill, *A Theory of Timed Automata*, Theoretical Computer Science, 126, pp.183-235, 1994.

[2] S. Bornot, J.Sifakis, S. Tripakis, *Modeling Urgency in Timed Systems*, International Symposium: Compositionality - The Significant Difference, Malente, Germany, 1998, to appear in LNCS.

[3] D. Bošnački, D. Dams, *Integrating Real Time into Spin: A Prototype Implementation*, S. Budkowski, A. Cavalli, E. Najm, editors, Formal Description Techniques and Protocol Specification, Testing and Verification (FORTE/PSTV'98), Kluwer, 1998.

[4] M. Bozga, S. Graf, L. Mounier, J. Sifakis, *The Intermediate Representation IF*, Technical report, Verimag, 1998.

[5] M. Bozga, J-C. Fernandez, L. Ghirvu, S. Graf, J.P. Karimm, L. Mounier, J. Sifakis, *If: An Intermediate Representation for SDL and its Applications*, In Proc. of SDL-FORUM'99, Montreal, Canada, 1999.

[6] H. Bowman, G. Faconti, J-P. Katoen, D. Latella, M. Massink, *Automatic Verification of a Lip Synchronization Algorithm Using UPPAAL*, 3rd Inl. Workshop on Formal Methods for Industrial Critical Systems, Amsterdam, CWI, 1998.

[7] I. Dravapoulos, N. Pronios, S. Denazis *et al.*, *The Magic WAND, Deliverable 3D2, Wireless ATM MAC*, September 1997.

[8] G. Holzmann, D. Peled, *An Improvement of Formal Verification*, PSTV 1994 Conference, Bern, Switzerland, 1994.

[9] A. Olsen *et al.*, *System Engineering Using SDL-92*, Elsevier Science, North-Holland, 1997.

[10] D. Peled, *Combining Partial Order Reductions with On-the-Fly Model Checking*, Computer Aided Verification CAV 94, LCNS 818, pp. 377-390, 1994.

[11] Verilog, *ObjectGEODE SDL Simulator - Reference Manual*, `http://www.verilogusas.com/solution/pages/geode.htm`, 1996.

[12] VIRES, *Verifying Industrially Relevant Systems*, Esprit Long Term Research Project #23498, `http://www.win.tue.nl/cs/fm/vires`, 1996.

[13] WAND consortium, *Magic WAND - Wireless ATM Network Demonstrator*, `http://www.tik.ee.ethz.ch/ wand`, 1996.