Verification Experiments on the MASCARA Protocol*

Guoping Jia and Susanne Graf

VERIMAG** – Centre Equation – 2, avenue de Vignate – F-38610 Gières – France
Guoping.Jia@imag.fr, Susanne.Graf@imag.fr

Abstract. In this paper, we describe an experiment in verifying a real industrial protocol for wireless ATM, called MASCARA. Several tools have been used: SDL has been chosen as the specification language and the commercial tool *Object*GEODE (TELELOGIC) has been used for creating, maintaining and modifying SDL descriptions. The IF tool-set has been used for minimization and comparison of system models and verification of expected properties. All specification and verification tools are connected via the IF language, which has been defined as an intermediate representation for timed asynchronous systems as well as an open validation environment. Due to the complexity of the protocol, static analysis techniques, such as live variable analysis and program slicing, were the key to the success of this verification experiment. The results obtained give some hints concerning a methodology for the formal verification of real systems.

1 Introduction

Model checking [CE81,QS82] is by now a well established method for verifying properties of reactive systems. The main reason for its success is the fact that it works fully automatically and it is able to reveal subtle defects in the design of complex concurrent systems. Different academic tools have been developed for supporting these techniques. Not only hardware but also telecommunication industries are beginning to incorporate them as a component of their development process. For example, the commercial SDL design tools *Object*GEODE [Ver96] and TAU [TA99] provide some verification facilities going beyond interactive or random simulation. A major challenge in model checking is dealing with the well-known *state explosion* problem. This limits its large scale use in practice. In order to avoid this problem, many solutions have been investigated. Examples of such approaches are on-the-fly model-checking [JJ89a,Hol90], symbolic model-checking [BCM$^+$90,McM93], partial order reduction [God96,GKPP94], symmetry reduction [ES94], abstraction [CGL94,LGS$^+$95], compositional minimization [GS90,KM00] and more recently static analysis reduction [BFG00a] to name a few of many. Applying model checking to software system verification, in particular to large industrial systems, needs to combine these techniques.

In this paper, we present a detailed experiment report on the verification of an industrial protocol, called MASCARA (Mobile Access Scheme based on Contention And Reservation for ATM) [DPea98]. The protocol is a specific medium access control (MAC) protocol, which has been designed for wireless ATM communication and has been developed within

* This work was supported by the VIRES project (Verifying Industrial REactive Systems, Esprit Long Term Research Project No.23498)

** VERIMAG is a joint laboratory of CNRS, Université Joseph Fourier and Institut Nationale Polytechnique de Grenoble

the WAND (Wireless ATM Network Demonstrator) consortium [WAN96]. SDL [IT94] has been chosen as the specification language by the designers and we have used the commercial tool *Object*GEODE for maintaining the SDL description of the protocol. The IF tool-set [BFG+99b,BFG+99a] has been used for analysis of the protocol. All specification and verification tools of this tool-set have been connected via the IF language, which is an intermediate representation for timed asynchronous systems in an open validation environment . In order to deal with the complexity of the protocol, all the available reduction techniques of the tool have been used in combination to reduce the state graphs of the protocol. The results obtained in the experiment give some hints on a methodology for the formal verification of large systems.

The paper is organized as follows. Section 2 gives an overview on the MASCARA protocol and a brief description of the IF language and its validation environment. Section 3 is the heart of the paper, where we describe in detail the verification of the protocol. The results are compared and discussed. We conclude with a brief summary and some directions of future work.

2 The Context

2.1 The IF Language and Validation Environment

The IF language [BFG+99a,BFG+99b] has been defined as an intermediate representation for timed asynchronous systems. In IF, a system is described by a set of parallel processes communicating either asynchronously through a set of buffers, or by rendez-vous via a set of gates. Buffers can have various queuing policies (fifo, bag, etc.), can be bounded or unbounded, reliable or lossy, and delayable or with zero delay. Processes are timed automata with urgencies [BST97], extended with discrete (data) variables. Process transitions are triggered by inputs and/or guards and perform variable assignments, and clock settings and signal outputs. An urgency out of *eager, delayable, lazy* is associated with each transition, defining its priority with respect to time progress. This makes IF very flexible and appropriate on one hand as underlying semantics for high level specification formalisms such as SDL or ESTELLE used in commercial design tools and on the other hand as an intermediate representation for tool interconnections as it is powerful enough to express the concepts of the languages used in the main verification tools of the domain, such as LOTOS [BB88] and PROMELA [Hol91].

The IF validation environment provides a complete validation tool chain allowing to transform high level SDL descriptions through the intermediate representation into the input formats of several verification tools (see Figure 1) [BFG+00b]:

- **The specification level tools.** IF does not itself provide facilities to edit SDL specifications. For this it relies on the commercial tool-set *Object*GEODE developed by TELELOGIC and supporting SDL, MSC and TTCN. It includes graphical editors and compilers for each of these formalisms and provides beside step-by-step and random-walk a model-checking facility using never claims, similar as in Spin, to help the user to debug an SDL specification. In the verification experiment, *Object*GEODE has mainly been used for modifying the SDL description of the MASCARA protocol and for visualizing MSCs generated from diagnostic sequences generated by other tools.

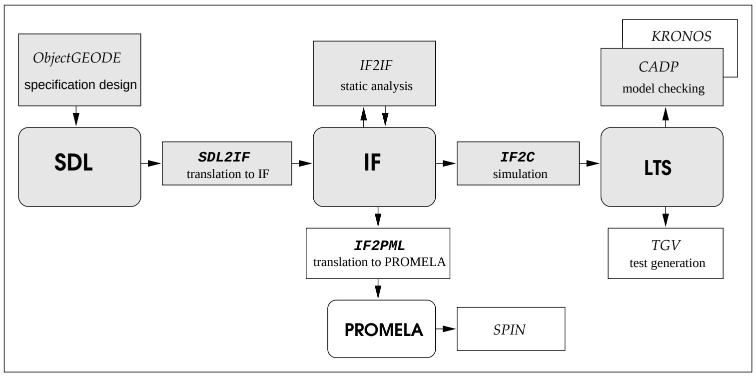

Fig. 1. The IF validation environment

- **The intermediate level tools.** Based on an API provided by *Object*GEODE, a translator, SDL2IF, generates operationally equivalent IF specification from SDL [BFG+99b]. At the IF level, a tool called IF2IF implements various static analysis techniques such as *dead variable analysis*(LIVE), *slicing*(SLICE), *constant propagation* and *clock reduction*. IF2IF transforms with a small cost a given IF description into a semantically equivalent one (with respect to a set of properties) with a reduced model, where a typical factor of reduction observed in many examples is between 1 and 3 orders of magnitude. Any back-end tool connected to IF can profit from these reductions. For example, the SPIN tool [Hol91] which has been connected via a translator IF2PML [BDHS00].
- **The verification tools.** CADP [FGK+96] has been designed as a tool-set for the verification of LOTOS specifications. Its model generation and analysis tools, such as ALDEBARAN and EVALUATOR, have been adapted for IF-specifications and can be used for generation, minimization, comparison of state graphs and verification of properties specified as alternation-free μ-calculus formulas either on-the-fly or on an already generated model. Diagnostic sequences are computed and can be translated into MSCs to be displayed in a user friendly manner. Other tools, such as KRONOS [Yov97] for the verification of real-time properties and TGV [FJJV97] for automatic test generation, can work directly on IF specifications but have not been used in our experiment.

2.2 The MASCARA Protocol

The MASCARA (Mobile Access Scheme based on Contention And Reservation for ATM) protocol [DPea98] is a special medium access control (MAC) protocol designed for wireless ATM (Asynchronous Transfer Mode) communication and developed by the WAND (Wireless ATM Network Demonstrator) consortium [WAN96]. A wireless ATM network extends transparently services to *mobile terminals* (MTs) via a number of geographically distributed *access points* (APs). The task of the MASCARA protocol is to mediate between APs and MTs via a wireless link. The protocol has a layered structure, where we consider only the highest layer, the MASCARA control layer (MCL).

The Purpose of the MASCARA Control Layer (MCL) is to ensure that mobile terminals (MTs) can initially establish and then maintain an association with an access point (AP) with good quality and minimal interruptions as long as the ATM connection is valid. It carries out a periodical monitoring of the current radio link quality (gathering the information about radio link qualities of its neighboring APs to hand-over to in the case of deterioration of the current association link quality) and switching from one AP to another in the hand-over procedure. Since several MTs communicate through a single AP with the ATM core network, MCL is different on the AP and the MT side.

MCL consists of four parts: *dynamic control* (DC), *steady state control* (SS), *radio control* (RCL) and *generic* MASCARA *control* (GMC). We describe in detail the dynamic control part, which we have concentrated on in this verification experiment.

Dynamic Control (DC) The main task of the dynamic control (DC) is to set up and release *associations* and virtual *connections*. It consists of the following entities: *dynamic generic agent*, *association agent* and MAC *virtual channel* (MVC) *agent*.

The dynamic generic agent is the top-level entity of the dynamic control and its task is *association management*. It dispatches upper layer requests concerning existing associations and connections to the right association agent, manages MAC-layer addresses for the associations, and informs the upper layer about lost associations.

The association agent of an MT and its peer entity in the (to be) associated AP are responsible for managing and supervising a single *association*. Each association can carry a variable number of connections via virtual channels. The task of the association agent peers is to create the virtual channel agents, map the addresses between the ATM-layer and the MAC-layer connection identifiers and forward requests. Since each MT is at any time associated with at most one AP, there exists one association agent per MT. While, whereas each AP has one association agent for every associated MT.

An MVC agent of an MT and its peer entity in the AP manage a single *connection* on a virtual channel. Beside address mapping from the ATM-layer to the MAC-layer connection identifiers, the MVC agents are in charge of *resource allocation* for the connection they manage over the wireless channels.

3 Verification of the MASCARA Protocol

The overall description of the MASCARA protocol which we got after completion of the model obtained by the designers is *300* pages of SDL textual code. This makes it impossible to envisage to push the "**verify**" button on the protocol as a whole. We need some methods to cut the problem into pieces and still get meaningful results.

This approach is made easier by the fact that MASCARA is a layered protocol. We concentrate the verification experiment on the MASCARA control layer (MCL), for which the SDL description could be made reasonably complete. Here we report on the verification of the dynamic control (DC). Another verification experiment has been carried out on steady state control (SS) [BDHS00]. In this section, we first present the experiment system and the assumptions and simplifications we made. Then, we list some of the correctness properties to be verified and describe in detail the approaches to perform the verification. Finally, we present the verification results and discuss some problems encountered.

3.1 The Experimental System

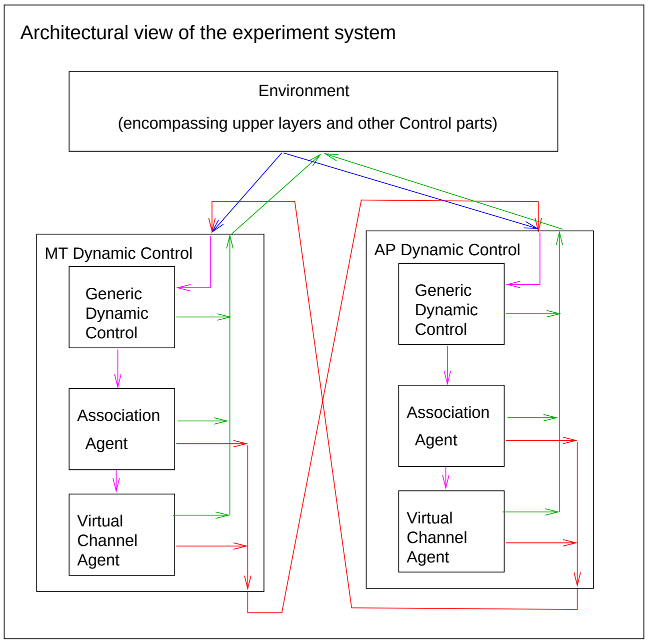

Fig. 2. Architectural view of the experimental system

Architecture. Dynamic control (DC) has an interface to the MASCARA management layer (called upper layer here), and exchanges control signals with lower layer entities of the MASCARA protocol. For verification, we abstract all lower MASCARA layers to a pair of buffered communication channels. These channels can be supposed as reliable as signal losses of the physical medium are treated in the lower protocol layers. The architecture of the SDL model used for verification can be seen in Figure 2 and consists of the following parts:

- AP Dynamic Control has itself a hierarchical structure: all signals from outside are received by the *Generic Dynamic Control* process, and either treated directly or forwarded to the relevant *Association Agent*, which on turn, either treats the signal itself or forwards it to the relevant *Channel Agent*.
- MT Dynamic Control has the same architecture as AP Dynamic Control, but the implemented protocols are of an asymmetric nature.
- An "*Environment*" process which consists of abstract versions of the upper layer and of the other MASCARA Control entities, in particular steady state control.

We assume that only one MT can be associated and only one connection can be opened, i.e., only one pair of association agents (AAA/MAA) and one pair of MVC agents (AMA/MMA) are considered, which is sufficient for the correctness properties we have verified.

Environment. Verification should be carried out under a general environment with realistic restrictions. As we have not obtained information on the MASCARA upper layer, we considered initially an open system with an unconstrained upper layer, which would allow us to obtain the "*most general*" verification results. But, contrary to LOTOS, SDL communication is via unbounded channels. This leads to infinitely growing channel contents and thus an infinite state model in case that the environment sends requests too often, which is typically the case in "reactive" systems always ready to to treat requests from the environment.

The approach we have chosen to provide a more restricted, but still realistic environment consists in restricting the number of requests it can make per time unit. We assume that within one time unit, no more than "N" requests can be sent by the environment. Considering system transitions as *eager* – that means that time can only progress when no system transition is enabled – this provides an environment suitable for our purpose. The system has never to deal with more than "N" requests simultaneously which leads, in the MASCARA protocol, to bounded channel contents. The success of the method depends on the use of a realistic bound. We use $N = 4$ which is probably realistic, and goes certainly further than most comparable verification experiments.

The role of time. This protocol makes use timeouts, essentially to avoid indefinite waiting for a response. This type of systems can usually be verified as an untimed system, where the occurrence of a timeout is treated as a non-deterministic event. As we use time as a convenient means to slow down the environment, we cannot adopt exactly this solution. We consider

- the transmission delay through the channels between AP and MT as 1 (the value is arbitrary) and all other transmission delays as zero and
- the maximal waiting time for response as 2
- all system transitions as urgent

This has as consequence that responses and corresponding timeouts occur in the same "time slice" and thus can occur in any order, and still time can be used to slow down the environment as it can send in each time slice only a limited number of requests.

3.2 Properties

As it is often the case, the designers of the system did not provide us with requirements that the system must satisfy. Therefore, we considered

- generic properties such as deadlocks
- and for each request from the environment (such as association, deassociation, connection opening, connection release,...) a set of "*response properties*", where i

In a first time we verified very weak properties, such as "potential response", and the more the system became debugged, the more we strengthened them. As an example, we show the strongest response property considered for "association request".

Association Establishment This property refers to association establishment between an MT and an AP which is obtained by a four-way handshake protocol initiated by MT. The signals between the system and the upper layer are:

– the request for association establishment ("*a_req*") from the upper layer to MT.
– the response to association request ("*a_cnf_(rc)*")[1] by MT to the upper layer.
– the indication of association establishment ("*a_ind*") AP to the upper layer.

the response property we considered for checking the correctness of the association establishment, is the following requirement:

> "Any *association request* received by the MT is *eventually* followed by an *association confirmation* with either a negative or a positive result. In the second case, an *association indication* has already been or will *eventually* be emitted by the AP."

Expression of Properties We have used several formalisms for the expression of properties:

1. We expressed some properties by sets of temporal logic formulas. Popular temporal logics are Computational Tree Logic (CTL)[CGP99] and Linear Temporal Logic (LTL)[MP92]. The model-checker EVALUATOR is based on the use of the alternation-free μ-calculus[Koz83] which is more expressive, but more difficult to use. However, there has been defined a set of macros going beyond the modalities of CTL or LTL, such that most properties can be expressed without problems at least by an expert. For verification (see Section 3.3), we decomposed the above correctness criterion into simpler properties (Table 2). The following formula expresses the requirement *Req1*, where "*all*", "*inev*" and "*TRUE*" are macros which should be intuitive enough.

$$all[a\_req](inev\ \ < a\_cnf\_* > TRUE)$$

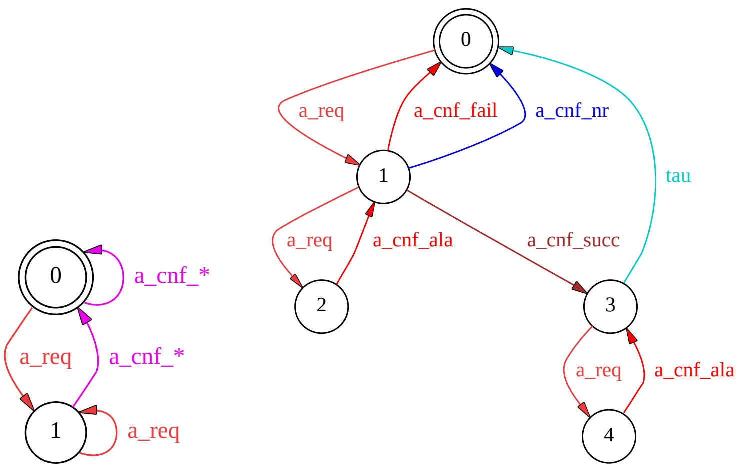

Fig. 3. *Req1* and a refinement of *Req1* expressed as LTS

[1] A return code is attached with the response signal which indicates positive (e.g. *success*) or negative (e.g. *failed*, *already associated* or *no response*) result of the corresponding association request.

2. CTL and LTL are not powerful enough to express *all* useful properties. Moreover, they are difficult to use by a non expert for the expression of complex state and event sequencing properties often required for protocols, especially if they are not covered by the set of macros. Finite automata are another formalisms for the expression of properties which is powerful yet relatively easy to use. In our tools we use labeled transition systems LTS, where the transition labels are signal inputs and outputs, possibly with their parameters (exchanged between the system and the environment or within the system itself). Figure 3 shows on the left side an LTS expressing *Req1* of Table 2 and on the right side a refinement of it. This refined version, which corresponds more to the really expected behaviour is to express as an LTS than as a temporal logic formula.
3. We have tried to apply so called "*visual verification*" which gives often very satisfactory results. It consists in "*computing*" the *exact property* of the system with respect to a set of observed signals: all *unobservable* signals are hided and the (completely constructed) state-graph is minimized with respect to an appropriate equivalence relation, such as observational[Mil80] or safety[BFG+91] equivalence. In this particular protocol, the obtained minimal state-graphs were most of the time still too complicated to be inspected visually.
4. We have used Message Sequence Charts (MSC). But we have not created them ourselves (in order to express negations of universal properties) but we have generated them from diagnostic sequences, showing the violation of some property, generated by our verification tools. In Figure 5 an example of such a "diagnostic MSC" can be found.

3.3 Verification Methodology

Given a property φ and a system $\mathcal{S}$, described as a parallel composition of a number of subsystems $\mathcal{S}_i$, the goal of model checking is to verify whether $\mathcal{S}_1 \parallel \cdots \parallel \mathcal{S}_n \models \varphi$ holds.

It is well-known that the size of the state graph grows exponentially with the number of components $\mathcal{S}_i$ (and also the number of variables). For the MASCARA protocol, even if only a single association and a single connection is considered, the state space is too large to be analyzed without application of any reduction technique.

We combined the use of all reduction techniques available in our tools and applied them in the order depicted in Figure 4. We explain the results observed using the different techniques in the following paragraphs.

Static Analysis Reduction Techniques: *static analysis techniques* is applied at the program level to reduce the number of states and transitions of the model associated with the specification and thus make model checking feasible:

1. Dead variable analysis transforms an IF specification into an equivalent one by adding systematic reset statements of "*dead*" variables. A variable is "*dead*" at some control point if its value is not used before it is assigned again. This transformation preserves all event-oriented properties of the original specification while the global state space and consequently the exploration time are reduced.
2. Program slicing automatically extracts portions of a program relevant for a specific property, called *slicing criterion*: for example, a set of variables or signals at program points of interest for the given correctness property. The reduction is obtained by eliminating

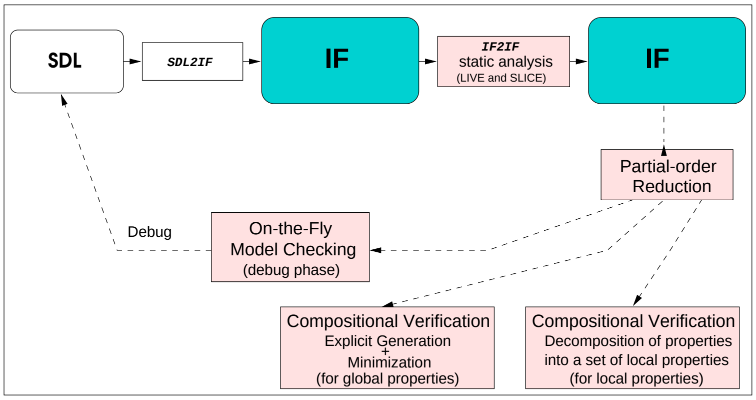

Fig. 4. Overview of the verification methodology

program parts which do not influence the slicing criterion. There is a different model for each property (or a set of properties). This transformation preserves safety properties depending only on the slicing criterion, while it results smaller IF-program.

These reductions are performed on the structural program level description of the system, before model generation and their complexity is completely unproblematic. They can, and should, be always applied, independently of the verification method or tool being used later on.

Partial Order Reduction [God96,GKPP94] is a method which consists in verifying properties of a concurrent system without exploring all interleavings of concurrent executions of independent transitions. It preserves properties on orderings of events as long as they are considered dependent. A simple version of this method has been implemented in the IF tool-set. In order to use partial order reduction jointly with compositional techniques, we need to consider all signal input from the "*environment*" as dependent, no matter the property to be verified. It is well-known that partial order reduction allows a significant reduction of the size of the state-graph, and should therefore always be applied during generation or traversal of the model.

Atomicity Reduction A well-known reduction method consists in considering as "model steps" sequences of internal steps. This is correct as long as each sequence contains at most one read or write action of a global variable and at most one modification of observable variables. SDL greatly facilitates the use of this method: transitions start reading a signal from a message buffer, and then execute a set of actions consisting in local assignments and signal outputs. All properties we consider are expressed in terms of signal inputs and outputs and in Mascara there are never two observable outputs sent within a single SDL transition; thus we can consider complete SDL transitions as atomic steps. The reduction obtained by this method is in general tremendous. As it is applied by default in the step-function of the IF tool-set, we can not show its effect.

On-the-Fly Model Checking: In "*on-the-fly*" model checking [JJ89b,FM91,Hol91] verification is performed during a possibly exhaustive traversal of the model. This method is very helpful, in particular at the first stage of verification for the debugging of the initial specification, as it exposes fast many errors and omissions, even of systems which cannot be verified completely. It should be noted, however, that only for very simple safety properties, the underlying model of the on-the-fly analysis has the same size as the system model alone. For more complex properties, on-the-fly verification explores a model which can be significantly bigger than the system model alone, and some of the advantage of not storing transitions vanishes. In the particular case of the Mascara protocol, there was no verification of a non-trivial property that we could do on-the-fly, but for which we could not generate the state graph.

Compositional Verification: We have applied two different types of *compositional verification*. The first one is based on property decomposition [Kur94], and the second one is based on compositional generation of a state graph minimized with respect to a behavioral equivalence [GS90]. For the application of both methods, we split the system into smaller functional parts, namely, AP dynamic control and MT dynamic control.

1. a) decompose a global property of a system into a set of *local* properties of the considered subsystems;
   b) verify each local property on the corresponding subsystem — using a particular environment representing an abstraction of the remaining subsystems.

   All safety properties which hold on a small configuration hold also on the complete system. This method is very convenient to use, under the condition that the global properties can be decomposed and that it is sufficient to include a very abstract view of the neglected subsystems. For example, *Req1 Req2*, *Req3a* and *Req3b* of Table 2 below, are such local properties of MT which can be verified on a state graph, generated by abstracting the AP part almost as the Chaos process (making sure however that it is not too active). see Table 1.

2. a) generate the state graph of each subsystem (AP and MT) separately, considering the same weak abstraction of the other subsystem as in the first method, and reduce it with respect to weak bisimulation using the ALDEBARAN tool;
   b) apply parallel composition on the reduced models (as communication between AP and MT is via a pair of buffers, these buffers are the global variables of the system and need to be considered as such for parallel composition [KM00])
   c) verify the global correctness properties on the generated global model.

   This method preserves all safety properties on observable events. *Req3* of Table 2 below, for example, can be evaluated on the state graph DC1 mentioned in Table 1.

The first method allows to work with smaller models than the second one as no abstraction of the global state graph need to be constructed. Unfortunately, it can sometimes be quite difficult to find an appropriate decomposition of properties and to provide a useful abstraction of the "*non-considered parts*" of the system. For example, the decomposition of *Req3* into *Req3a* and *Req3b* is only correct if the communication channels between AP and MT can be considered as reliable. Notice that the second method needs not to rely on a correct environment abstraction [GS90], but this variant is not implemented in our tool for systems communicating through asynchronous buffers.

3.4 Complexity

Table 1 gives an overview of a subset of the state graphs we have generated using different reduction techniques and allows to compare their sizes.

Execution Time With respect to execution time, the following observation can be made: execution times are roughly proportional to the size of the generated graphs, which means that the different reduction methods do not introduce any significant overhead. For static analysis reduction this result is not surprising. For partial order reduction it is the case because we use a simple static dependency relation. Table 1 shows only minimization results for relatively small graphs (AP4a and MT4a) so that minimization time is small anyway. Nevertheless, it can be seen that minimization for observational equivalence is more expensive than for safety equivalence, as the computation of the transitive closure transition relation "$\tau * a\tau *$" is required (where τ represents a non-observable and a an observable transition).

State Graph Size We can see that application of dead variable analysis (LIVE) and partial order reduction (PO) alone reduces the original state graph by 1 to 2 orders of magnitude. The combination of LIVE and PO gives more than the combined reduction of each of these techniques applied in isolation. Notice that for AP, the efficiency of PO and LIVE are about the same, whereas for MT, LIVE performs better; in other case studies we also had the situation where PO performed better, so that one can say that with a negligible cost, applying them together, most of time, one obtains good reduction (here 3 orders of magnitude)

Obviously the reduction obtained by the application of slicing depends heavily on the considered properties, and it is impossible to make general statements. In our case-study, we get similar results when slicing according to the 4 main sub-protocols (1 to 2 additional orders of magnitude); connection opening is more complicated than the others (it involves more signal exchanges than the others) and thus we get a bit less reduction.

It was impossible to generate or verify on-the-fly the state graph of the global system as a whole, thus we started to consider AP and MT in isolation (see first two parts of the Table 1). Finally, we were able to compositionally generate a reduced model of the global system using compositional generation, under the condition to use both LIVE and partial order reduction for the generation of the subsystems.

3.5 Verification Results

We did a large number of verification experiments with increasing complexity. Initially, many deadlocks were found which were mainly due to the interaction the different "*request/response*" sub-protocols. It should also be mentioned that the feature of implicit (that is silent) discarding unexpected signals in SDL made the analysis of the generated diagnostic sequences of deadlock traces more difficult. Using a different translation from SDL to IF, this problem has disappeared.

Table 1. State graphs of the dynamic control

Entities	Generation Approaches		N. of Tran.	N. of Stat.	Time (hh:mm:ss)
AP Dynamic Control (with an abstract version of MT part)	AP1: Direct generation		30 689 244	7 308 400	3:27:35
	AP2: Partial-order reduction alone		1 807 095	895 249	37:26
	AP3*a*: LIVE optimization alone		1 536 699	351 202	12:22
	AP3*b*: Minimization of AP3a (strong bisimulation)		1 189 032	265 888	5:27
	AP4*a*: LIVE + Partial-order		52 983	28 069	1:53
	AP4*b*: Minimization of AP4*a* (strong bisimulation)		38 952	20 312	18
	AP4*c*: Minimization of AP4*a* (hide/rename + observ. equivalence)		18 521	11 265	1:04
	AP4*d*: Minimization of AP4*a* (hide/rename + safety equivalence)		12 210	3 502	22
	AP5: SLICE + LIVE (w.r.t. properties)	AP5*a*: Association Establishment	12 664	4 556	9
		AP5*b*: Deassociation	10 739	3 940	5
		AP5*c*: Conn. Open	36 603	11 616	28
		AP5*d*: Conn. Release	15 958	5 636	9
	AP6: SLICE + LIVE + Partial-order (w.r.t. properties)	AP6*a*: Association Establishment	2 885	1 630	3
		AP6*b*: Deassociation	2 972	1 703	3
		AP6*c*: Conn. Open	8 099	4 583	9
		AP6*d*: Conn. Release	4 262	2 476	5
MT Dynamic Control (with an abstract version of AP part)	MT1: Direct generation		12 811 961	4 388 765	2:51:58
	MT2: Partial-order reduction alone		7 433 859	3 099 928	1:30:57
	MT3*a*: LIVE optimization alone		325 312	63 628	1:03
	MT3*b*: Minimization of MT3*a* (strong bisimulation)		246 970	47 489	1:37
	MT4*a*: LIVE + Partial-order		20 913	6 580	7
	MT4*b*: Minimization of MT4*a* (strong bisimulation)		12 241	3 927	8
	MT4*c*: Minimization of MT4*a* (hide/rename + observ. equivalence)		1 804	1 148	13
	MT4*d*: Minimization of MT4*a* (hide/rename + safety equivalence)		1 380	499	3
	MT5: SLICE + LIVE (w.r.t. properties)	MT5*a*: Association Establishment	16 854	4 018	5
		MT5*b*: Deassociation	16 522	3 967	5
		MT5*c*: Conn. Open	15 823	3 754	4
		MT5*d*: Conn. Release	16 135	3 820	4
	MT6: SLICE + LIVE + Partial-order (w.r.t. properties)	MT6*a*: Association Establishment	2 845	977	3
		MT6*b*: Deassociation	2 411	985	2
		MT6*c*: Conn. Open	2 801	969	2
		MT6*d*: Conn. Release	2 162	855	2
Dynamic Control (AP + MT)	DC1: Composition of models (AP4*c* $\times$ MT4*c*)		1 142 215	218 130	–

As we had obtained almost no information on the environment of the MASCARA layer, we considered initially the case where the request from the environment can be sent in any order. This lead to a number of error traces which we considered to be "probably because of too loose assumptions on the environment" and we added corresponding restrictions for subsequent verifications. The graphs mentioned in Table 1 have been obtained using the most restrictive environment.

Property: **Association Establishment**		
Req1.	After reception of an association request by MT, an association confirmation with either *positive* or *negative* return value will be eventually sent by MT to the upper layer.	**TRUE**
Req2.	After reception of an association request by MT, there exists an execution path where an association confirmation with *positive* return value is sent by MT to the upper layer.	**TRUE**
Req3.	Whenever the association confirmation with *positive* return value is sent by MT, an association indication will be or has already been sent by AP to the upper layer.	**FALSE**
Req3a.	Whenever AP receives the third handshake message (MPDU_MT_AP_addr_received), it will eventually send the fourth handshake message (MPDU_AP_MT_Assoc_ack) to MT, and an indication of the association (a_ind) to the upper layer.	**TRUE**
Req3b.	Whenever MT receives the fourth handshake message (MPDU_AP_MT_Assoc_ack) from AP, it will eventually send a successful confirmation (a_cnf_succ) to the upper layer.	**FALSE**

Table 2. Properties and Verification results for Association Establishment

Table 2 lists the verification results for the properties concerning association establishment. We performed the verification in an incremental manner, starting with weak properties, weaker than those mentioned in the table, for each subsystem (AP and MT), and finally ending up with the strong properties of the table which we verified either on the relevant subsystem or on the reduced version of the global system.

The local property *Req3b* (as well as the global property *Req3*) does not hold. A diagnostic was produced by the tool EVALUATOR. Figure 5 gives an MSC scenario of such a trace. Its analysis shows that this violation occurs when a deassociation request is sent before the association has been confirmed. In case of a deassociation request, a negative association confirmation is sent to the environment independently of the success of the handshake protocol with AP; and this is a correct behaviour. Thus, *Req3a* should be replaced by the following weaker requirement (which holds):

> When MT receives an MPDU_AP_MT_Assoc_ack from AP, it will eventually a successful confirmation (a_cnf_succ) to the upper layer, except if it has already received or meanwhile receives a deassociation request from the upper layer.

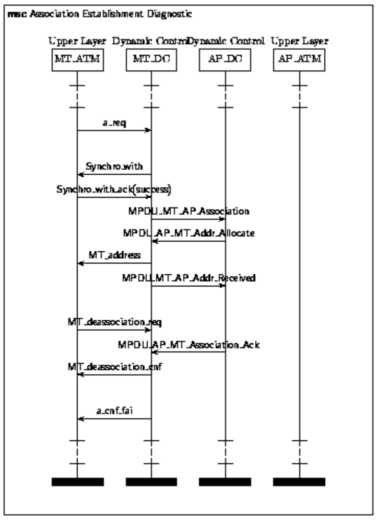

Fig. 5. MSC showing the non satisfaction of *Req3b*

4 Conclusion and Perspectives

In this paper, we have presented an experiment report on the verification of an industrial ATM protocol. The aim of this verification exercise was much less the actual verification of the protocol, than rather to experiment and improve the verification facilities of the IF tool-set and to analyze the difficulties occurring with such a big protocol and how to deal with them. We believe that we have at least partially succeeded. The main difficulties that we encountered together with some responses or some to-do list are the following ones, in the order in which they appeared:

1. How to extract with a reasonable effort a subsystem from a large SDL description (such as a single layer from a whole protocol stack)? The fact that we did not enlarge verification from the initially considered sub-system to a larger subsystem is partially due to the fact that it is such a time consuming hand-work to cut large SDL descriptions in pieces, or to recombine separately modified subsystems at a later stage. Hopefully, the integration of on one hand UML oriented features in SDL design tools, which allow to trace interface changes, and on the other hand static analysis methods allowing to "cut out" subsystems in a clean manner, will eliminate this problem.
2. How to get reasonable abstraction of the neglected parts of the system? In a protocol stack,
   - the lower layers can often easily be abstracted by communication channels with particular properties: they may be reliable or lossy, ordered or unordered, delayable or immediate.
   - The upper layers can often be considered as unconstraint or only weak order constraints are necessary, a part from the fact that the flooding of the system with

infinitely fast sequences of requests must be avoided in order to make state space exploration tractable. Fortunately, for the verification of safety properties it is always reasonable to bound the number of requests of the upper layer per time unit.

For other subsystems which are not related in such a simple way with the subsystem under study, slicing is one way to get a simplified description, but in our example this was not sufficient. General abstraction techniques as those implemented in the InVest tool [BLO98] should be connected with IF.

3. How to get requirements? A part from "deadlock-freedom", there exist only few "generic" properties which must hold for any system. Communication protocols, can often be viewed as reactive systems which must react "in an appropriate way" to a number of "requests" from the environment; moreover this is true for every sub-layer of the protocol. In an ideal case, the designer should be able to help in expressing the appropriate "response properties". In absence of help we used the following strategy, which turned out to work quite well: we started with very weak response properties and strengthened them as long as they hold. When we found a violated property, we analyzed a number of diagnostic sequences (or graphs) produced by the model-checkers in order to find out if it was likely to be
   - a real problem of the system,
   - a too loose environment
   - or a too strong property

   and we made corresponding changes.
4. How to express properties? For simple properties, Temporal Logic is very convenient, but for more complicated ones, for example taking into account a number of exceptions under which the desired response need not to occur, temporal logic is cumbersome. Labeled transition systems allow to express more complicated properties on message exchanges. MSCs express the existence of sequences with certain characteristics, and are therefore more appropriate for the representation of "bad scenarios" or "never claims". We believe that a generalization of MSCs, such as *Live Sequence Charts* [DH99], could be very useful.
5. How to analyze negative verification results? In the case where the violation of a property is due to (a set of) execution sequences, we translated these sequences into message sequence charts which we than replayed on the SDL specification using a facility of *Object*GEODE. This was convenient as long the sequences were not too long. Using an abstraction criterion makes the sequences shorter, but introduces non-determinism, which is another source of complexity, and hides sometimes away the problematic point. Our experience is probably not very encouraging in this point: we found that only once we had a good understanding of the protocol, we could detect subtle errors with a reasonable effort.
6. How far can we go with system verification? We hope that we have demonstrated that using an appropriate strategy, we can verify automatically reasonably small subsystems or components of large systems. For the verification of global properties of large systems, automatic verification using state space enumeration, combined with whatever reduction strategies, seems out of reach. To go a step further we applied two — out of the large number of compositional approaches proposed in the literature — which could be applied using the facilities of our tool-set.
   - compositional construction of a state-graph reduced with respect to some equivalence relation. Our results show that this method will probably not scale, unless the inter-

faces between the subsystems are very concise, or we can provide automatic means for getting precise enough abstractions of large parts of a system.
– compositional verification based on property decomposition. We believe that this method can scale, even if there are at least two problems which can make its application difficult:
  - the decomposition of a global system with a large number of subsystems can be very hard (we applied it to a system with only two subsystem and a very simple communication structure)
  - as for the first method an abstraction of the environment of the considered subsystem is needed, even if one can hope that less concise abstractions are enough.

Acknowledgments

We gratefully acknowledge TELELOGIC for giving us access to their *Object*GEODE SDL environment. We would also like to thank Marius Bozga, Lucian Ghirvu and Laurent Mounier for fruitful discussion and kind help during the verification experiment.

References

[BB88] T. Bolognesi and E. Brinksma. Introduction to the ISO Specification Language LOTOS. *Computer Networks and ISDN Systems*, 14(1):25–29, January 1988.

[BCM+90] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Proceedings of the International Conference on Logic in Computer Science'90*, 1990.

[BDHS00] D. Bosnacki, D. Dams, L. Holenderski, and N. Sidorova. Model Checking SDL with SPIN. In *Proceedings of TACAS'2000, Berlin, Germany*, LNCS, 2000.

[BFG+91] A. Bouajjani, J.Cl. Fernandez, S. Graf, C. Rodriguez, and J. Sifakis. Safety for Branching Time Semantics. In *18th ICALP*, number 510 in LNCS, July 1991.

[BFG+99a] M. Bozga, J.-C. Fernandez, L. Ghirvu, S. Graf, J.P. Krimm, and L. Mounier. IF: An Intermediate Representation and Validation Environment for Timed Asynchronous Systems. In *Proceedings of FM'99, Toulouse, France*, LNCS, 1999.

[BFG+99b] M. Bozga, J.-C. Fernandez, L. Ghirvu, S. Graf, J.P. Krimm, L. Mounier, and J. Sifakis. IF: An Intermediate Representation for SDL and its Applications. In *Proceedings of SDL-FORUM'99, Montreal, Canada*, June 1999.

[BFG00a] M. Bozga, J-C. Fernandez, and L. Ghirvu. Using Static Analysis to Improve Automatic Test Generation. In *Proceedings of TACAS'2000, Berlin, Germany*, LNCS, 2000.

[BFG+00b] M. Bozga, J.Cl. Fernandez, L. Ghirvu, S. Graf, J.P. Krimm, and L. Mounier. IF: A Validation Environment for Timed Asynchronous Systems. In E.A. Emerson and A.P. Sistla, editors, *Proceedings of CAV'00 (Chicago, USA)*, LNCS. Springer, July 2000.

[BLO98] S. Bensalem, Y. Lakhnech, and S. Owre. Computing Abstractions of Infinite State Systems Compositionally and Automatically. In A. Hu and M. Vardi, editors, *Proceedings of CAV'98 (Vancouver, Canada)*, volume 1427 of *LNCS*, pages 319–331, June 1998.

[BST97] S. Bornot, J. Sifakis, and S. Tripakis. Modeling Urgency in Timed Systems. In *International Symposium: Compositionality - The Significant Difference (Holstein, Germany)*, volume 1536 of *LNCS*. Springer, September 1997.

[CE81] E.M. Clarke and E.A. Emerson. Design and Synthesis of Synchronisation Skeletons using Branching Time Temporal Logic. In *Proceedings of IBM Workshop on Logics of Programs*, volume 131 of *LNCS*, 1981.

[CGL94] E.M. Clarke, O. Grumberg, and D.E. Long. Model checking and abstraction. *ACM Transactions on Programming Languages and Systems*, 16(5):1512–1542, September 1994.

[CGP99] E.M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking.* The MIT Press, 1999.

[DH99] W. Damm and D. Harel. Lscs: Breathing live into message sequence charts. In *FMOODS'99 IFIP TC6/WG6.1 Third International Conference on Formal Methods for Open Object-Based Distributed Systems*, 1999.

[DPea98] I. Dravapoulos, N. Pronios, and S. Denazis et al. *The Magic WAND, Deliverable 3D5, Wireless ATM MAC, Final Report*, August 1998.

[ES94] E. A. Emerson and A. P. Sistla. Utilizing symmetrie when model checking under fairness assumptions. submitted to POPL95, 1994.

[FGK+96] J.-C. Fernandez, H. Garavel, A. Kerbrat, R. Mateescu, L. Mounier, and M. Sighireanu. CADP: A Protocol Validation and Verification Toolbox. In *Proceedings of CAV'96, New Brunswick, USA*, volume 1102 of *LNCS*, July 1996.

[FJJV97] J.C. Fernandez, C. Jard, T. Jéron, and C. Viho. An Experiment in Automatic Generation of Test Suites for Protocols with Verification Technology. *Science of Computer Programming*, 29, 1997.

[FM91] J.-C. Fernandez and L. Mounier. "On the fly" Verification of Behavioural Equivalences and Preorders. In *Workshop on Computer-Aided Verification, Aalborg University, Denmark*, LNCS, July 1991.

[GKPP94] R. Gerth, R. Kuiper, D. Peled, and W. Penczek. A partial order approach to branching time logic model checking, April 1994.

[God96] P. Godefroid. *Partial-Order Methods for the Verification of Concurrent Systems - An Approach to the State Explosion Problem*, volume 1032 of *LNCS*. Springer, January 1996.

[GS90] S. Graf and B. Steffen. Compositional Minimisation of Finite State Processes. In *Proceedings of CAV'90, Rutgers*, number 531 in LNCS, 1990.

[Hol90] G. J. Holzmann. Algorithms for automated protocol validation. *AT&T technical Journal*, 60(1):32–44, January 1990.

[Hol91] G.J. Holzmann. *Design and Validation of Computer Protocols.* Prentice Hall Software Series, 1991.

[IT94] ITU-T. *Recommendation Z.100. Specification and Description Language (SDL).* 1994.

[JJ89a] C. Jard and T. Jeron. On-Line Model Checking for Finite Linear Temporal Logic Specifications. In *Workshop on Automatic Verification Methods for Finite State Systems, Grenoble.* LNCS 407, Springer Verlag, 1989.

[JJ89b] C. Jard and T. Jeron. On-Line Model-Checking for Finite Linear Temporal Logic Specifications. In J. Sifakis, editor, *Proceedings of the 1st Workshop on Automatic Verification Methods for Finite State Systems (Grenoble, France)*, volume 407 of *LNCS*, pages 189–196. Springer Verlag, Juin 1989.

[KM00] J.P. Krimm and L. Mounier. Compositional State Space Generation with Partial Order Reductions for Asynchronous Communicating Systems. In *Proceedings of TACAS'2000, Berlin, Germany*, LNCS, 2000.

[Koz83] D. Kozen. Results on the Propositional μ-Calculus. In *Theoretical Computer Science.* North-Holland, 1983.

[Kur94] R.P. Kurshan. *Computer-Aided Verification of Coordinating Processes: The Automata-Theoretic Approach.* Princeton University Press, Princeton, New Jersey, 1994.

[LGS+95] C. Loiseaux, S. Graf, J. Sifakis, A. Bouajjani, and S. Bensalem. Property preserving abstractions for the verification of concurrent systems. *Formal Methods in System Design*, 6, Issue 1:11–44, jan 1995. first published in CAV'92, LNCS 663.

[McM93] K.L. McMillan. *Symbolic Model Checking: an Approach to the State Explosion Problem.* Kluwer Academic Publisher, 1993.

[Mil80] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer-Verlag, 1980.

[MP92] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag, 1992.

[QS82] J.P. Queille and J. Sifakis. Specification and Verification of Concurrent Programs in CESAR. In *International Symposium on Programming*, volume 137 of *LNCS*, 1982.

[TA99] Sweden Telelogic A.B., Malmö. *Telelogic TAU SDL suite Reference Manuals*. `http://www.telelogic.se`, 1999.

[Ver96] Verilog. Object*GEODE SDL Simulator - Reference Manual*. http://www.verilogusa.com/products/geode.htm, 1996.

[WAN96] WAND. *Magic WAND - Wireless ATM Network Demonstrator*. http://www.tik.ee.ethz.ch/wand, 1996.

[Yov97] S. Yovine. KRONOS: A Verification Tool for Real-Time Systems. *Software Tools for Technology Transfer*, 1(1+2):123–133, December 1997.