Compositional Verification of a Third Generation Mobile Communication Protocol

Sari Leppänen
Nokia Research Center
Mobile Networks Laboratory
P.O.Box 407, 00045 Nokia Group, Finland
Sari.Leppanen@Nokia.Com

Matti Luukkainen
University of Helsinki
Department of Computer Science
P.O.Box 26, 00014 University of Helsinki, Finland
mluukkai@cs.helsinki.fi

Abstract

Model-checking has turned out to be an efficient and relatively easy-to-use technique in the verification of formally described programs. However, there is one major drawback in using model-checking: the state explosion, i.e., the behavior models of real-life programs tend to be extremely large. In the article it is shown how the theories of behavioral equivalences with a compositional style of behavior model generation can alleviate the state explosion in verifying the externally observable properties of SDL descriptions. The practical usability of the method is evidenced with a case study that is taken from ongoing development of third generation mobile communication systems.

Keywords

Formal verification, model-checking, Compositional methods, SDL Communication protocols, UMTS

1. Introduction

Testing and debugging concurrent and reactive programs, such as communication protocols, is an extremely tedious task, partly due to the nondeterminism caused by the computation environment. If a program is described with a formal language, such as SDL [10], the behavior of the program can be defined by means of a mathematically defined structure, such as a *behavior graph*, which describes all the possible computation sequences of the program. The more general term *state space* is often used when talking about behavior models of programs.

If the correctness requirements of a formally defined program are given in a mathematical notation, such as temporal logic [24, 4] or state automaton [15], an algorithm called *model-checker* [5] can be used to check whether the program respects its correctness requirements. The model-checker goes through every possible computation sequence of the program, thus it is said to be an *exhaustive* verification technique. Because all the possible execution sequences are covered, model-checking gives a total confidence of the programs' correctness.

Model-checking has turned out to be an efficient and easy-to-use technique in program verification. However, there is one major drawback in using exhaustive model-checking: behavior graphs of real-life programs, telecommunication protocols for example, tend to be extremely large. In literature this problem is often referred to as *state explosion*.

One successful technique in alleviating the state explosion is the 'divide-and-conquer'-like compositional method of building systems state space (see for example [17, 8, 6]),where the system state space is generated from minimized state spaces of its sub-components. The compositional method can be particularly efficient if those aspects of the system that are unnecessary for the verification are abstracted away. Usually these compositional techniques work in the context of process algebra, where a synchronous communication paradigm is used. In [13] it was shown how this technique can be used in the context of SDL, where the communication is by nature asynchronous.

In this article we apply this compositional technique to the verification of a third generation mobile communication protocol. By third generation we mean the future mobile systems in the *IMT-2000 family* (see Figure 1). The core network of the third generation can be considered as an evolution of second generation systems, based mainly on GSM. The main difference between second and third generation systems lies on the radio access technique. Mainly in Europe and Japan *3GPP* which is a standardization forum established by several standardization organizations,[1] standardizes *UMTS (Universal Mo-*

[1] The standardization organizations involved in the creation of 3GPP

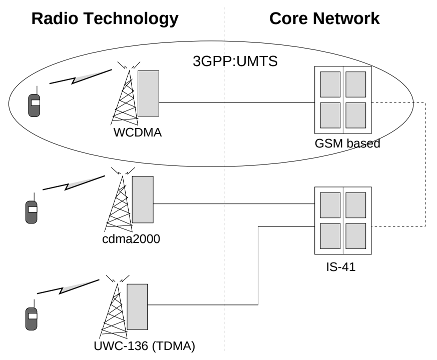

Figure 1. IMT-2000 family of systems

bile Telecommunications System) using *WCDMA (Wideband Code-Division-Multiple-Access)* radio access technique. The main objective of 3GPP is to harmonize various WCDMA specifications and achieve a common WCDMA standard. Different companies, such as manufacturers and operators, are members of 3GPP through the respective standardization organization they belong to.

The article is structured as follows: Section 2 motivates the use of labeled transition systems as a behavioral model of SDL descriptions. Section 3 shows how the behavioral equivalence theories and compositional style of state space generation could be used in alleviating the state explosion. In section 4, the compositional method is adapted to the context of SDL. In section 5 the analyzed protocol is described and section 6 shows the results of verification. Conclusions are finally drawn in section 7.

2. Externally observable behavior

The behavior of a SDL description can be defined by means of a *behavior graph* that consists of nodes and transitions between the nodes. A node describes a state of the program (i.e. values of the program variables, contents of message queues . . .) at one moment of time and transitions describe the atomic actions (i.e. assignment, sending or receiving of a message . . .) that change the state of the program. Transitions may be labeled with the corresponding action names. Intuitively, the behavior graph of a SDL description contains all the possible computation sequences of the described system. The behavior graph corresponding to a SDL description can be obtained by interpreting the description according to the formal semantics of SDL which is defined in Annex F.1 of the Z.100 standard [11].

When using formal techniques in developing reliable software, good tool support is extremely important. Commercial SDL tools, like SDT [14] and ObjectGEODE [22] offer some means for model-checking-like automatic verification. However, there is a serious drawback in the model-checking facility of both the tools. When using exhaustive model-checking to verify a program, both of the tools save the complete behavior graph of the program to the computer's memory. Since SDL descriptions usually comprise lots of data variables, generation of the complete behavior graph is extremely memory consuming. Thus, the size of the programs that can be exhaustively model-checked is limited.

State explosion is handled within the tools by using non-exhaustive model-checking methods, such as checking only some randomly selected computation paths or using the bit state hash technique of [7]. Drawback of the non-exhaustive methods is that only part of the computations are checked and thus, they do not give full confidence about the correctness of a program.

We are sometimes interested only in the external behavior of a system, in other words, the activity of the system which is visible to an external observer, such as the communication actions between the system and its environment. If we specify a communication protocol for example, it is enough that the service it offers (i.e. the external behavior visible to the user of the protocol) is consistent with the correctness requirements of the protocol. Thus, as long as the protocol behaves as expected, we more or less do not care how the behavior is achieved.

Externally visible actions of a system specified in SDL are the $input$ and $output$ statements it uses to communicate with the environment. The rest of the actions within the system and its internal state (e.g. values of different variables) are not interesting if only the external behavior is concerned. As a consequence, the external behavior of a SDL description can be captured by a simpler structure than behavior graph, namely a *labeled transition system*, or Lts in short.

A labeled transition system is a quadruple (S, Σ, Δ, s_0), *where*

- S *is a set of states,*
- Σ *is the set of observable action labels,*
- $\Delta \subseteq S \times \Sigma \cup \{\tau\} \times S$ *is the transition relation, (*τ *denotes an internal action i.e., an action invisible for the external observer), and*
- $s_0 \in S$ *is the initial state.*

The external behavior of a program is easily modeled as a labeled transition system, where the transition labels in Σ correspond to the external communication events and the internal action τ is used to denote some internal activity within a program.

were ARIB(Japan), ETSI (Europe), TTA (Korea), TTC (Japan) and T1P1 (USA)

3. The Equivalence Theory

With the Lts representation of programs, we have the possibility of comparing syntactically different programs with respect to their behavior. Thus, we can check if two programs P_1 and P_2 behave similarly, or if a program $Impl$ is a 'valid implementation' of another program $Spec$. Next we formally define what is intended by saying that two programs behave similarly, in other words the concept of behavioral equivalence is defined. There is a vast amount of different equivalences in the literature, see for example [21]. In the following we use the CFFD equivalence [19, 20] as our notion of behavioral equivalence.

The following customary definitions are used:

- $P - a \rightarrow P'$ means that there is transition a from state P to the state P', in other words $(P, a, P') \in \Delta$.
- $P - a \rightarrow$ means that there is a state P', such that $P - a \rightarrow P'$.
- $\neg(P - a \rightarrow)$ means that it is not possible to find a state P', such that $P - a \rightarrow P'$.
- $P - a_1 a_2 \ldots a_n \rightarrow Q$ iff $\exists P_0, \ldots, P_n$ such that $P_0 = P, P_n = Q$ and $\forall i \in \{1 \ldots n\} : P_{i-1} - a_i \rightarrow P_i$.
- $P = b_1 b_2 \ldots b_n \Rightarrow Q$ iff $P - \tau^* b_1 \tau^* b_2 \tau^* \ldots \tau^* b_n \tau^* \rightarrow Q$, where τ^* denotes a finite or empty sequence of internal actions.

To define the CFFD equivalence, the following concepts are also needed:

- $\sigma \in \Sigma^*$ is a *divergence trace* of a state P iff $\exists Q : P = \sigma \Rightarrow Q \wedge Q - \tau^\infty \rightarrow$, where τ^∞ denotes an infinite sequence of internal actions, and Σ^* an finite sequence of action names drawn from set Σ.
- The set of divergence traces of a state P is denoted by $div(P)$. Divergence traces of a Lts corresponds to the divergence traces of its initial state.
- A state P is *stable*, if $\neg(P - \tau \rightarrow)$. Predicate $stable(P)$ denotes stability of the state P. A Lts is stable if and only if it the initial state is stable.
- Pair (σ, A), where $\sigma \in \Sigma^*$ and $A \subseteq \Sigma$ is a *stable failure* of state P, if and only if $\exists Q : P = \sigma \Rightarrow Q \wedge stable(Q) \wedge \forall a \in A : \neg(Q - a \rightarrow)$.
- The set of stable failures of a state P is denoted by $sfail(P)$. Stable failures of a Lts corresponds to the stable failures of its initial state.

Finally we define the CFFD equivalence relation between two Lts's.

Labeled transition systems $lts = (S, \Sigma, \Delta, s_0)$ and $lts' = (S', \Sigma', \Delta', s_0')$ are CFFD-equivalent, $lts_1 =_{CFFD} lts_2$, iff

- $stable(s_o) = stable(s_o')$,
- $sfail(s_0) = sfail(s_0')$,
- $div(s_0) = div(s_0')$.

If two programs, P and P' are CFFD equivalent, the intuitive meaning is that both have the same set of possible computation sequences and, furthermore, P can deadlock after a sequence of actions if and only if P' can also deadlock after the same sequence of actions. The computation sequences that lead to a divergence (infinite sequence of internal actions) are also the same for the two processes.

In [12] it was proven that

> *CFFD is the weakest congruence relation for finite state systems, with respect to the composition operators of process algebras, which preserves the truth of nexttime-less linear temporal logic (or LTL' for short) [24].*

In practice compositionality means that we can replace a Lts with a CFFD equivalent Lts in the context of any of the process algebraic operators. It should be noted that this is not the case with all behavioral equivalences.

Because of compositionality, we can use the following method in building a minimized[2] Lts describing the behavior of a given system. Assume that the correctness criteria of a system are given in REQ which is a set of LTL' formulae. Let us consider that our system consists of several concurrent entities. Naturally all the entities are modeled as separate Lts's, $lts_1, \ldots, lts_n$. Let us assume that $||$ is a parallel operator that enforces synchronization on all the common actions of the combined Lts's, similarly as CSP parallel operator [9]. A Lts describing the behavior of the system is achieved by parallel composition of the separate components $sys = lts_1 \mid\mid \ldots \mid\mid lts_n$. After building the Lts describing systems behavior, a model-checker is used to test if the system respects the requirements given by the LTL' formulae in the set REQ.

A much more efficient and memory-saving way to build the Lts of the system is, first to minimize the separate system components with respect to CFFD-equivalence, and after that combine the minimized components ${lts_1}^{min}, \ldots, {lts_n}^{min}$ to form the Lts describing the behavior of system $sys^{min} = {lts_1}^{min} \mid\mid \ldots \mid\mid {lts_n}^{min}$. Because CFFD is a congruence relation, it is guaranteed that

[2] Sometimes finding the minimal equivalent Lts is computationally hard, and we have to use some heuristics to find condensed, but not necessarily minimal Lts's.

$sys =_{CFFD} sys^{min}$. Furthermore, because CFFD minimization preserves the truth of all LTL' formulae, the correctness of the system can be detected by model-checking the minimized system description sys^{min} against the formulae in the set REQ. More formally, the following is true for all LTL' formulae ϕ: $sys \models \phi$ iff $sys^{min} \models \phi$.

The fact that CFFD is the *weakest* LTL' preserving congruence relation guarantees that the minimized Lts is optimally small and still contains all the information needed in verification.

The compositional method of building Lts's is not restricted to the use of CFFD and linear temporal logic only. We can use other equivalence relations, as well, which are strong enough to preserve all the interesting properties and which are congruent at least with respect to the parallel operator.

4. Compositional state space generation from SDL descriptions

Externally visible actions of a system specified in SDL are the $input$ and $output$ statements it uses to communicate with the environment. The rest of the actions within the system, and its internal state (e.g. values of different variables) are not interesting if only the external behavior is concerned.

As stated in previous sections, labeled transition system is a convenient mathematical structure to describe the external behavior of a SDL system. Lts describing the behavior of a SDL system is easily obtained from the systems behavior graph in the following way.

- For every node in the behavior graph there is a state in the Lts.
- If there is a transition between two nodes in the behavior graph, there should also be a transition between the corresponding states in the Lts.
- If a transition in the behavior graph is caused by an input or an output action, the corresponding transition in the Lts should be labeled with the action name.
- Other transitions in the Lts should be labeled with τ, thus they are invisible to the external observer.
- The state in Lts corresponding to the root node of the behavior graph is the initial state.

If we would like to verify the external behavior of a system specified in SDL, we can generate the corresponding Lts and use a suitable minimization to make the model-checking less memory and time consuming. Most transitions in a Lts describing the external behavior of a system are internal ones, thus it is evident that the minimized Lts is much smaller than the original one.

The same analogy as above can be used to define the external behavior of an arbitrary component (usually a process) in a SDL specified system. Externally visible actions of a component are the $input$ and $output$ statements it uses to communicate with the rest of the system and the environment. Thus, the external behavior of an arbitrary component can also be described by means of a Lts. Note that a channel in a SDL description can also be modeled as a Lts. So, the asynchronous synchronization of SDL components is achieved by modeling the SDL channels explicitly with labeled transition systems, that in turn communicate synchronously with the real system components.

We can now use the following procedure to build compositionally a minimized Lts describing the behavior of a complete SDL specified system.

1. Produce the behavior graphs ($bg_1 \ldots bg_n$) from the separate system components.

2. Translate $bg_1 \ldots bg_n$ to corresponding labeled transition systems $lts_1 \ldots lts_n$.

3. Minimize the Lts's with respect to a suitable equivalence relation.

4. Model the channels combining separate system components as Lts's. Note that a limited channel length has to be assumed to keep the system finite stated.

5. Combine the minimized Lts's and channels by using a synchronous parallel operator, and minimize the result.

The resulting Lts actually describes more than merely the external behavior: it also contains systems internal signalling. The Lts which describes only the external behavior of the system is obtained from the Lts resulting from the above procedure by transforming the internal signalling actions to internal actions. This transformation can be done by using the *hide*-operator of the LOTOS process [3] algebra. The operation just renames the corresponding labels with τ's. After hiding, the Lts is minimized again. The resulting Lts describes the external behavior of the original SDL specified system. If CFFD equivalence is used in minimization, all the properties of the original system which are expressible with Linear temporal logic can be checked from the generated Lts.

If the internal signalling of the system or parts of it are of interest, hiding internal signals can be skipped (or done only for those signals that are not needed). Thus, the compositional verification method is not limited to external behavior only.

5. The RLC-protocol

In this chapter we look at an application of the compositional state space generation method to a link layer data transfer protocol specified for the UMTS (*Universal Mobile Telecommunications System*) system [16]. Let us first have a short general review of the motivation and basic principles of the UMTS system.

The explosive growth of users and applications in mobile telecommunications increases the requirements for the capacity in future mobile systems. The need for higher transmission speed is inevitable while using mobile telecommunication more for data transfer than speech.

Increment in capacity and in transmission speed require either more frequency band or more efficient usage of the current one. Particularly in metropolitan areas bandwidth is a scarce resource and the competition between the operators is severe. Thus new and more efficient radio access techniques are in a key position in the development of future mobile telecommunication systems.

WCDMA (Wideband Code-Division-Multiple-Access) is a radio access technique used in *UMTS (Universal Mobile Telecommunications System)*. In traditional radio access techniques, like *FDMA (Frequency-Division-Multiple-Access)* and *TDMA (Time-Division-Multiple-Access)*, the highest amount of users is determined by the amount of frequencies or timeslots in use. In CDMA technique users are separated by a user-specific code, which is used to code the transferred data. All users are in all frequencies at the same time. Thus, the system does not have any artificial limits for the amount of users and the radio network capacity is more efficiently used. There are two versions of the CDMA technique: *NCDMA (Narrowband CDMA)* and WCDMA (Wideband CDMA). In WCDMA the radio bandwidth used is wider than in the second generation systems (such as GSM) where narrowband techniques are used, which increases the capacity of the network. However, WCDMA does not achieve the performance level of existing fixed networks using wideband radio access techniques when measured in network capacity and transmission speed.

UMTS network architecture (see Figure 2) design follows the GSM principle of distributing the intelligence of the system throughout the network [1]. UMTS network consists of *UE (User Equipment)*, like a mobile phone, *UTRAN (UMTS Terrestrial Radio Access Network)* and *CN (Core Network)*. UTRAN manages the radio path control and CN controls the call establishment and release. There are two standardized and fully open interfaces in UMTS. *Uu* is specified between the UE and UTRAN. *Iu* interface connects UTRAN to CN.

UTRAN is divided into several subsystems, called *RNS (Radio Network Subsystem)*. RNS is a logical entity consisting of several Node Bs connected to one *RNC (Radio Network Controller)*. Node B maintains the radio interface and relays voice and data to and from UE. RNC is the most essential network element in UTRAN. It controls the radio access network and performs radio resource management. Complexity of the CDMA technique is perhaps most visible in the functionality of RNC, which is much more complicated compared to the corresponding network element in GSM.

The protocol stacks in UTRAN network elements cover L1, L2 and L3 according to the OSI reference model. In Figure 3 we illustrate the protocols establishing the communication between UE and RAN. *RLC (Radio Link Control)* protocol, the target of our case study, is a L2 data link protocol defined for the Uu interface terminating in UE and RNC. Together with the other L2 protocol *MAC (Medium Access Control)*, the protocol provides a reliable data transfer service via unreliable air interface. MAC mainly handles the access to the physical layer while RLC manages the actual data transfer taking care of flow control, retransmission, error correction etc. Also *PDCP (Packet Data Convergence Protocol)* is considered to be part of the link layer. The protocol optimizes transmission in the radio interface by compressing the data packets before sending.

The RLC protocol is located above the MAC protocol (see Figure 4). MAC provides a symmetrical, unreliable data transfer service over the radio interface. We model the MAC service as a channel where data can be lost. On the other hand we assume that data can not be corrupted or duplicated. The length of the channel is one and in case the channel is full it is blocks the sender.

RLC provides to the upper layers several services related to data transferring. According to the specification [2] the protocol performs RLC connection establishment and release, transmits data in transparent, unacknowledged or acknowledged mode, allows QoS (Quality of Service) setting dynamically during the data transfer and notifies the upper layer of the unrecoverable protocol errors. We are concentrating here on the verification of reliable data transfer service in acknowledged mode. We also include RLC connec-

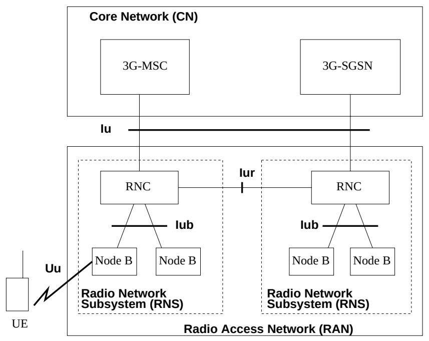
Core Network (CN)
3G-MSC
3G-SGSN
Iu
Iur
RNC
RNC
Iub
Iub
Uu
Node B
Node B
Node B
Node B
Radio Network Subsystem (RNS)
Radio Network Subsystem (RNS)
UE
Radio Access Network (RAN)

Figure 2. UMTS architecture

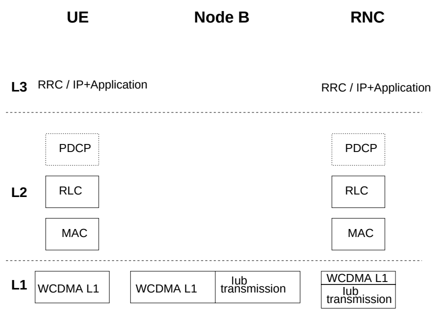

Figure 3. UMTS protocol layers

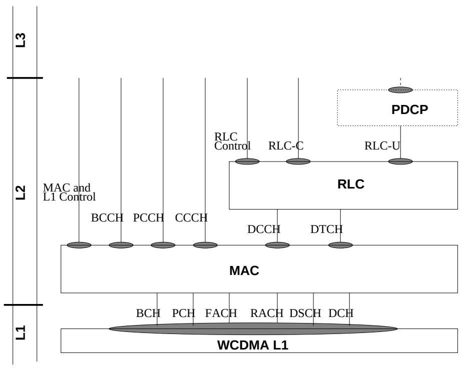

Figure 4. UMTS Link layer

tion establishment and notification of unrecoverable errors into our model to make the functionality of it sound and complete.

The RLC connection establishment procedure as a whole consists of receiving one message from the upper layer *RRC (Radio Resource Control)* protocol. The connection establishment message is received approximately at the same time at both ends of the protocol, i.e., in UE and RNC.

The acknowledged data transfer service transmits upper layer *PDUs (Protocol Data Unit)* and guarantees delivery to the peer entity. In case RLC is unable to deliver the data correctly, the user of RLC at the transmitting side is notified. The acknowledged data transfer mode has the following characteristics [2]:

- Error-free delivery: The receiving RLC entity delivers only error-free *SDUs* (*Service Data Unit*, upper layer PDUs) to the upper layer.

- Unique delivery: RLC delivers each SDU only once to the receiving upper layer by using duplicate detection function.

- In-sequence delivery: RLC provides support for in-order delivery of SDUs, i.e., RLC delivers SDUs to the receiving upper layer entity in the same order as the transmitting upper layer entity submits them to RLC.

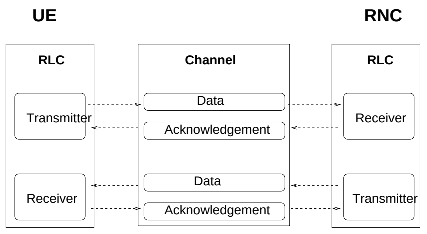

Figure 5. Architecture of RLC SDL model

- Out-of-sequence delivery: Alternatively to the in-sequence delivery, it is possible to allow that the receiving RLC entity delivers SDUs to the upper layer in a different order than delivered to it on the transmitting side.

- Ciphering: This service is not yet defined in the specification.

We verify the in-sequence delivery of the protocol. There are several alternative *ARQ (Automatic Repeat reQuest)* schemes to choose from. We employ *stop-and-wait*, perhaps the simplest one, for our verification model. Each data packet has to be acknowledged before a new one is accepted from the user, and in case RLC is unable to deliver the data according to the requirements it notifies the transmitting upper layer entity.

6. Results of verification

We constructed a SDL model describing the functional behavior of RLC protocol with respect to the features and requirements to be verified. This case study is partly related to a project within Nokia where UTRAN protocols were described with SDL. At Nokia, as almost everywhere within the telecommunication industry, an increasing trend is to use the SDL language as a protocol development language.

To make the size of the specification manageable we had to make some abstractions both for the behavior and the data in the protocol. We left out the routing processes, which are fundamental in the implementation model but mostly uninteresting and unnecessary overhead for our verification model. We integrated the decoding and encoding processes into the main RLC process, so the processes are not explicitly shown in the SDL description.

In the SDL description both of the peer RLC protocol entities are divided into transmitting and receiving units, which are executed parallel and independently of each other. The transmitting unit is communicating with the receiving unit of the peer entity via the unreliable channel below. The channel consists of two data channel and acknowledgment channel pairs for both directions. The architecture of RLC's SDL model is illustrated in Figure 5.

For the case study we used Sun Enterprise 3500 equipped with 6 CPU's 336Mhz each and 2 GB of memory. We first tried to construct the state space as a whole with SDT Validator [14] tool, but after generating 4 million states the tool ran out of memory. In other words exhaustive verification with the commercial tool was not possible.

Then we applied the compositional method of section 4 in building the behavioral model for the protocol. We proceeded as follows. In the first phase we produced the behavior graphs from each process in the SDL model of the RLC protocol with the SDT Validator, and translated them into corresponding Lts's. The data transmitting component on each side had 643 Lts states and the receiving component 89 states. We generated Lts's corresponding to the channel components with LOTOS to Lts generator of verification tool ARA [18]. Each of these Lts's has 6 states. So, in our verification model the underlying MAC service is replaced entirely with labeled transition systems that simulate the behavior of MAC service under the assumptions given in section 5.

Now we can approximate that if we would have been able to construct the whole state space with "brute-force" approach, i.e., without using the compositional method, the size of it would have been something between 4 million (the number of states generated with the SDT validator before it ran out of memory) and $1,8 \cdot 10^{25}$ states.

After minimizing the components with respect to the CFFD equivalence we first combined the transmitting component on the UE side and the channel component transmitting data from UE to RNC. The corresponding receiving site was constructed by combining the RNC receiving component and the acknowledgment channel. In each of these combinations we hid all the internal actions and minimized the result. After combining these two and minimizing the result we got a Lts with 1077 states. The only visible actions were the data transmission requests by the user and the corresponding data indications to the user on the other side. The protocol error indication message for the user was also visible in the Lts. All the activities in state space generation, namely minimization and parallel composition of Lts's and hiding action labels, were done with verification tool ARA.

The resulting Lts models the functionality of RLC data transfer in the acknowledged mode with the restrictions and assumptions concerning the channel and ARQ scheme we described above. Because the functionality of RLC data transfer is symmetric for both network elements and they execute in parallel completely independently of each other, the verification results for this model can be generalized to hold in the bidirectional model, as well.

Once the model was generated, the verification task itself was to be carried out. The first property verified was that the protocol is free from deadlocks. The use of CFFD equivalence enabled us to use a particularly simple scheme for checking deadlocks: We hid all the actions of the system and minimized the result, and checked whether the minimized Lts contains any deadlocks. Had there been a deadlock in the original Lts there would also have been a deadlock in the abstracted and minimized Lts.

With this scheme we actually found some deadlocks. In order to analyze the reason for these, we had to do further analysis without hiding all the actions. The deadlock analysis revealed that our original SDL model needed some modifications (eg. the addition of some timers). We worked over the SDL model and repeated the deadlock checking. After several iterations we generated the deadlock-free Lts described above.

As we already mentioned in the previous section, the in-sequence property of the protocol, i.e., that it delivers messages in correct order, was also verified. In the verification of this property the theory of *data independence* [23] was used. Data independence guarantees that if a transfer protocol does not use the user data its logic and does not alter its contents, it is enough to consider just a small number of different types of user data frames, even if the domain of the frames is extremely large. It is actually enough to consider just two types of user messages in verifying a safety property like this one.

We constructed a user process, which tries to send a sequence of messages of the form 0^*1^*. Thanks to the theory of data independence, the property of in-sequence delivery could then be verified by observing that the sequence which was received at the other end was also of the same form 0^*1^*. The verification was done by constructing a monitor process observing that the sequence of received messages was indeed the required one.

The above property is clearly expressible with Linear temporal logic (if executability of an action would be thought as an atomic proposition). In [12] it is also shown that hiding unnecessary details from a model does not change outcome of a LTL model checking.[3] Now we can come to the conclusion that because minimization of our model was done with respect to the CFFD-equivalence, it is guaranteed that our verification result also holds in the original, unminimized system.

Verification of the property was carried out successfully, in other words, besides being free of deadlocks, our protocol also provides the in-sequence message delivery. The ARA tool was also used in verifying both the deadlock freeness and in-sequence delivery. Since the Lts representing the behavior of the protocol, i.e. the production of compositional state space generation, was so small (only 1077 states), the model checking itself, which would have been impossible for the full model, was extremely quick.

[3] Clearly the definition of unnecessary depends on the formula that is to be verified.

7. Conclusions

In the text we pointed out the inability of current commercial SDL tools to cope with state explosion in exhaustive model-checking. We revised a previously proposed [13] method where the state explosion of verifying SDL descriptions could be alleviated by exploiting the theories of behavioral equivalences, which have been developed mainly in context of process algebra. In the method labeled transition system representations are generated from the components of SDL description. The asynchronous communication of SDL is simulated by representing channels explicitly with labeled transition systems. The key step of the framework is minimization of these Lts's with respect to a suitable equivalence relation. One such is the CFFD equivalence which preserves the truth of nexttime-less linear temporal logics (LTL'), thus the minimization with respect to CFFD equivalence does not change the outcome of LTL' model-checking.

The contribution of the paper was the application of this framework to a third generation mobile communication protocol RLC, which is currently under standardization of 3GPP. The success of the verification task clearly demonstrated that even if our tool support was in some sense inadequate (the tool ARA [18] that we used dates has been constructed in 1994), the compositional method (which works naturally with a synchronous communication paradigm) also works well with industrial-sized SDL-systems that are asynchronous by nature.

8. Acknowledgments

The authors gratefully acknowledge the financial support from Nokia Foundation. The authors also wish to thank Professor Antti Valmari for extremely helpful comments.

References

[1] 3GPP. General UMTS Architecture, 1999.

[2] 3GPP. RLC Protocol Specification, draft 1.2.1", 1999.

[3] T. Bolognesi and E. Brinksma. Introduction to the ISO Specification Language LOTOS. *Computer Networks and ISDN Systems*, 14:92–100, April 1987.

[4] E. Clarke and E. Emerson. Design and Synthetis of Synchronization Skeletons Using Branching Time Temporal Logics. In *Workshop on Logic on Programs*, number 131 in Lecture Notes in Computer Science. Springer-Verlag, 1981.

[5] E. Clarke, E. Emerson, and A. Sistla. Automatic Verification of Finite State Concurrent System Using Temporal Logic Specifications. In *ACM Transactions on Programming Languages and Systems*, volume 8, pages 244–263, 1986.

[6] E. Clarke, D. Long, and K. MacMillan. Compositional Model-Checking. In *Proceedings of the Fourth IEEE Symposium on Logic in Computer Science*, pages 353–362, 1989.

[7] G. Holtzman. *Design and Validation of Computer Protocols*. Prentice-Hall, 1991.

[8] S. Graf and B. Steffen. Compositional Minimization of Finite State Processes. In *Proceedings of 2st Workshop on Computer Aided Verification*, number 3 in Series in Discrete Mathematics and Theoretical Computer Science, pages 57–73. AMS-ACM DIMACS, 1990.

[9] C. Hoare. *Communicating Sequential Processes.* Prentice Hall, 1985.

[10] ITU-T. *Recommendation Z.100 - CCITT Specification and Description Language.* ITU, 1993.

[11] ITU-T. *Recommendation Z.100 Annex F.1 - CCITT Specification and Description Language.* ITU, 1993.

[12] R. Kaivola and A. Valmari. The Weakest Compositional Semantic Equivalence Preserving Nexttime-less Linear Temporal Logic. In *CONCUR 92, 3nd International Conference on Concurrency Theory*, number 527 in Lecture Notes in Computer Science, pages 207–221, 1992.

[13] M. Luukkainen and A. Ahtiainen. Compositional Verification of SDL descriptions. In *SAM98, 1st Workshop of the SDL Forum Society on SDL and MSC*, 1998.

[14] Telelogic Malm AB. *SDT 3.5 Users Guide, SDT 3.1 Reference Manual* . Telelogic, 1999.

[15] W. Thomas. Automata on Infinita Objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, Volume B, Formal Models and Semantics, pages 133–191. Elsevier, 1990.

[16] www.umts-forum.org, 1999.

[17] A. Valmari. Compositionality in State Space Verification Methods. In *Application and Theory of Petri Nets 1996, 17th International Conference*, number 1091 in Lecture Notes in Computer Science, pages 29–56. Springer-Verlag, 1996.

[18] A. Valmari and R. Savola. Verification of the Behaviour of Reactive Software with CFFD-semantics and ARA tools. In *Proceedings of ESA International Symposium On-Board Real Time Software*. ESTEC, Nordwijk, The Netherlands, 1995.

[19] A. Valmari and M. Tienari. An Improved Failures Equivalence for Finite-State Systems with a Reduction Algorithm. In *Protocol Specification, Testing and Verification XI*, pages 3–18. North-Holland, 1991.

[20] A. Valmari and M. Tienari. Compositional Failure-Based Semantic Models for Basic LOTOS. *Formal Aspects of Computing*, 7:440–468, 1995.

[21] R. van Glabbeek. The Linear Time Branching Time Spectrum II: The Semantics of Sequential Systems with Silent Moves. In *CONCUR 93, Fourth International Conference on Concurrency Theory*, number 715 in Lecture Notes in Computer Science, pages 66–81. Springer-Verlag, 1993.

[22] Verilog. *GEODE - Technical Presentation*. Verilog, 1994.

[23] P. Wolper. Expressing Interesting Properties of Programs in Propositional Temporal Logic. In *Proceedings of the 13th ACM Symposium on Principles of Programming Languages*, pages 184–193, 1986.

[24] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems, Specification.* Springer-Verlag, 1991.