Correct Performance of Transaction Capabilities

Thomas Arts[†] and Izak van Langevelde[‡]

[†]Ericsson
Computer Science Laboratory
Box 1505, SE-125 25 Älvsjö, Sweden
Email: thomas@cslab.ericsson.se

[‡]CWI
PO Box 94079, 1090 GB Amsterdam, The Netherlands
Email: izak@cwi.nl

Abstract

The correctness of an optimisation of the Transport Capabilities Application Part of the Signalling System No. 7 is formalised as a branching bisimulation which is relaxed to allow certain actions to be executed in any order. It is demonstrated how this correctness can be checked by a combination of an automated test of branching bisimulation and a manual test of commutation. Using this approach, two bugs in the design were found and eliminated.

1 Introduction

Ericsson's move of the implementation of the Signalling System No. 7 [14] from PLEX, a proprietary language for multi-processor architectures, to ERLANG[1], provided a splendid opportunity to optimise the design of the Transaction Capabilities Application Part (TCAP) of the protocol stack. Although the applied strategy of merging components of TCAP, making obsolete internal communications, apparently yielded a more efficient design, the question whether the optimisation was correct could not readily be answered. Both the fact that original and optimised design were specified in different languages, and the fact that the mere definition of correctness was unclear, made answering this question a true challenge.

The key to the solution was sought in lifting both the original and the optimised design to a common abstraction layer. The two designs were specified in μCRL [12], a language rooted in the Algebra of Communicating Processes (ACP [5]) extended with abstract data types [4]. For both specifications a labeled transition system was generated, making it possible to automatically check the equivalence of the original and the optimized design using a suitable equivalence relation. However, a relation truly covering the relevant notion of equivalence was not found in literature. An answer was found in an extension of branching bisimulation [8] with commutativity of certain pairs of actions.

This paper describes the formalisation of the original and the optimised design and their equivalence, and the use of software tools and human wit to establish the correctness of the optimisation, eliminating several bugs. An overview of the actual verification process is presented in [2] and a detailed account including full specifications in [3]. The current paper is organised as follows: Section 2 describes TCAP and its optimisation, Section 3 describes how the correctness of the optimisation was approached, and Section 4 presents the results of the verification. Finally, Section 5 gives the conclusions and pointers to future research.

2 Optimising TCAP

Transaction Capabilities Application Part (TCAP) is a protocol supporting advanced intelligent services in mobile telecommunication networks. Examples of these services are the allocation of routing numbers associated with toll-free numbers, the identification of calling card users, and the identification, authentication, and roaming of GSM phone users. The TCAP standard is laid down as part of the Signalling Stack No. 7 [14].

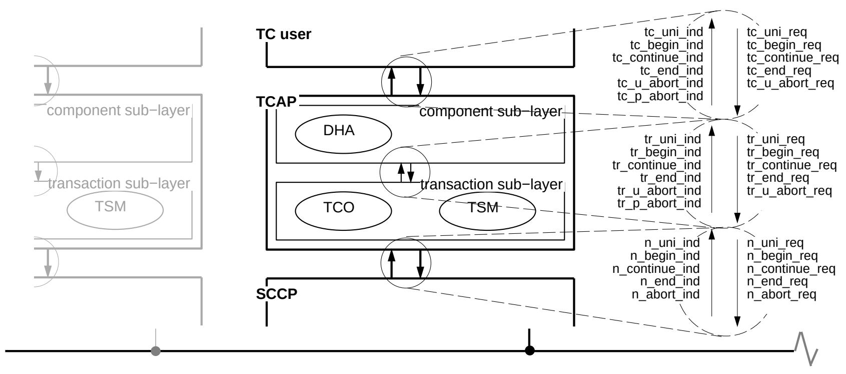

Figure 1: The dialogue-handling facilities of TCAP

The TCAP dialogue handling facilities consist of either one *unidirectional* request or multiple requests over a connection which is explicitly set up and torn down. The former case consists of one unidirectional request being issued, while the latter consists of setting up a dialogue which is then used by an arbitrary number of *continue dialogue* requests from both sides until it is either gracefully ended or aborted.

TCAP is originally specified in terms of five finite state machines organised in two communicating layers. The *component sublayer* consists of the Dialogue Handling block (DHA), and the Component Handling block (CHA), which is further decomposed into a Component Coordinator (CCO) and an Invocation State Machine (ISM). The *transaction sublayer* consists of a Transaction State Machine (TSM) and the Transaction Coordinator (TCO). The dialogue handling facilities of TCAP studied in this paper only cover the blocks (or finite state machines) DHA, TCO and TSM (Figure 1). The original protocol is specified in the Specification and Description Language SDL [13] (for a sample fragment, see Figure 2).

TCAP behaves as follows. In the protocol stack, TCAP is located between the higher-level *transport capabilities user* (TC-USER) and the lower-level *Signaling Connection Control Part* (SCCP), which provides the interface to the actual network. Requests from TC-USER to TCAP eventually result in requests to SCCP which travel across the network to show up at the receiving end as indications for SCCP, passed upward to the receiving TCAP to be, finally, passed upward to the receiving TC-USER. Internally in TCAP, messages are passed between the state machines mentioned earlier.

The crux of the optimisation is to merge the two sublayers, making communication between the two obsolete. The resulting protocol consists of just two state machines, i.e. DHA/TCO and TSM, the former of which is a merge of the original DHA and TCO. The state machine DHA/TCO is obtained by composing the actions from DHA which result in the sending of a message to the reactions of TCO on receiving this message. For instance, in processing a unidirectional request from TC-USER, DHA builds a record, requests and processes components and assembles and sends a request to TCO. Upon receiving this message, TCO assembles a unidirectional message and passes it down to the SCCP network. The merge DHA/TCO, on processing a unidirectional message, executes the same build, request and process actions, but instead of sending a request to TCO it directly continues with assembling and passing down the unidirectional message. The state machine TSM is a slightly modified version of the original TSM. The optimisation is documented as an informal flow chart [15], added for illustrative purpose in Figure 3.

The question of whether the optimisation of TCAP is correct can be paraphrased as "are the SDL specification, partly shown in Figure 2, and Figure 3 equivalent?" To tackle this question, first the notion of equivalence needs to be defined and, second, it is to be checked. These issues are addressed in the next section.

3 Approach

An equivalence relation useful for establishing the soundness of the optimisation of TCAP must satisfy a number of requirements. First, it must facilitate the abstraction of non-observable actions, to effectively hide all internal communications between the finite state machines. This holds in particular for those between DHA and TCO, which are missing in the optimisation. Second, it must be possible to relax the equivalence to apply only to situations satisfying certain conditions, to cover the fact that the optimisation is only correct under the condition that the protocol is correctly used. Third, it must be possible that equivalent systems differ in the order in which certain actions are executed. This notion of commutativity is introduced to cover the fact that the order in which certain pairs of actions are executed, such as freeing a dialogue id and freeing a transaction id, is

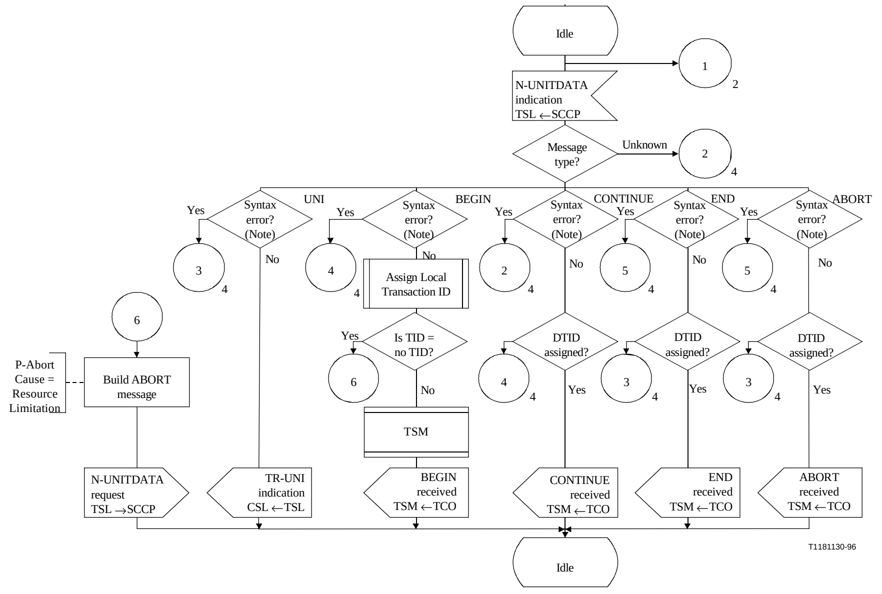

Figure 2: Part of the original design [14]

irrelevant, although the optimisation may choose one fixed order.

A well-understood vehicle for defining equivalence is provided by labeled transition systems, which formalise a process as moving from state to state by executing actions. The existence of a broad range of fundamental research and practical tools renders this an attractive choice. The restriction of the scope of automated support to finite systems plays no role here, since infinite behaviour is no issue in the context of TCAP.

Definition 1 (labeled transition system) *A labeled transition system* $\mathsf{S} = \langle S, Act, \to, s_0\rangle$ *consists of a set of states* S*, a set of actions* Act*, a transition relation* $\to : S \times Act \times S$ *and an initial state* $s_0 \in S$

Abstraction of non-observable actions is well-studied, especially in process algebra where these are formalised as 'τ-steps'. A common equivalence relation covering this abstraction is branching bisimulation [8]; it is supported by tools like CÆSAR/ALDÉBARAN [6]. Informally speaking, two systems are branching bisimular if each state of the first system corresponds to a state of the other system such that each action that can be executed in the first state can also be executed in the other state, possibly preceded and succeeded by a number of τ steps, after which both systems end up in corresponding states.

Definition 2 (branching bisimulation) *Two labeled transition systems* $\mathsf{S}_1 = \langle S_1, Act, \to_1, s_1\rangle$ *and* $\mathsf{S}_2 = \langle S_2, Act, \to_2, s_2\rangle$ *are branching bisimular iff there exists a relation* $R : S_1 \times S_2$ *which satisfies the following:*

- $s_1 R s_2$
- *if* $t_1 R t_2$ *and* $t_1 \overset{a}{\to}_1 t_1'$ *then* $a = \tau$ *and* $t_1' R t_2$*, or* $t_1 R u_2$*,* $t_1' R u_2'$*,* $t_1' R t_2'$ *and* $t_2 \overset{\tau}{\to}{}_2^* u_2 \overset{a}{\to}_2 u_2' \overset{\tau}{\to}{}_2^* t_2'$*, for states* $u_2, u_2', t_2' \in S_2$
- *if* $t_1 R t_2$ *and* $t_2 \overset{a}{\to}_2 t_2'$ *then* $a = \tau$ *and* $t_2' R t_2$*, or* $u_1 R t_2$*,* $u_1' R t_2'$*,* $t_1' R t_2'$ *and* $t_1 \overset{\tau}{\to}{}_1^* u_1 \overset{a}{\to}_1 u_1' \overset{\tau}{\to}{}_1^* t_1'$*, for states* $u_1, u_1', t_1' \in S_1$

This notion of equivalence has a syntactical counterpart in process algebras like the Algebra of Communication Processes (ACP [5], see [7] for a recent introduction). ACP processes are defined in terms of atomic *actions* by means of operators, of which *sequential composition* ('$\cdot$'), *alternative choice* ('$+$') and *parallel merge* ('$\|$') are the most important. The semantics of ACP can be defined in two ways. The *operational semantics* associates with each process a labeled transition system, extending the definition of an equivalence like branching bisimulation to apply to processes. The *axiomatic semantics* defines equivalence of processes in terms of a set of axioms, like $x + y = y + x$ for the commutativity of alternative choice. For branching bisimulation, an axiomatisation is presented in [8].

The second requirement, conditional equivalence, is best accounted for by putting the system in parallel with an environment that restricts its behaviour to those situations

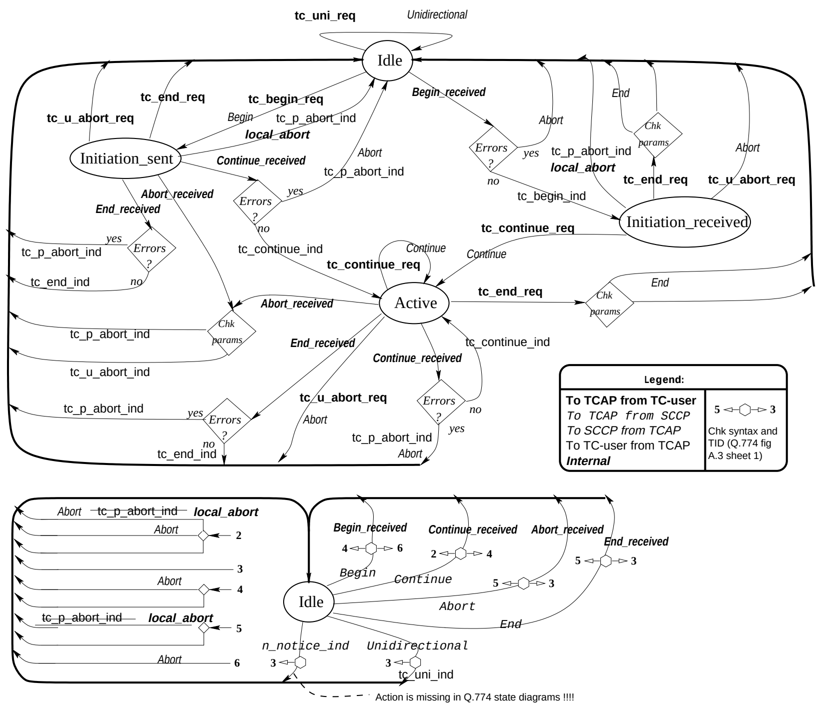

Figure 3: The optimised design [15]

satisfying the conditions. More concrete, the TCAP specification is extended with an environment which ensures that the protocol is used correctly, excluding erroneous situations such as a dialogue that is aborted though it has not been set up properly. It is worth noting that this technique is independent from the equivalence relation used.

The last requirement, commutativity of actions, is easiest characterised syntactically by adding, for each pair of actions a and b which are allowed to commute, the axiom $a \cdot b = b \cdot a$ to the axiom system which covers the equivalence relation used. A semantical characterisation, however, is far from trivial. What comes closest is the notion of partial order reduction [9], which exploits the fact that *independent actions* can be executed in either order, both resulting in the same state. The difference between independency and commutativity, however, is that both orders of independent actions do yield equivalent states, whereas both orders of commutative actions yield equivalent states by definition, i.e. through extending the equivalence relation.

The twelve pairs of actions which are known to be commutative in TCAP are given in [3]. The commutativity of these was not given beforehand, but in the verification process it was discovered that the optimised design relied on it. The designer of the optimised TCAP acknowledged that, indeed, these actions were allowed to commute.

Definition 3 *Suppose P_1 and P_2 are* ACP *processes, then P_1 and P_2 are equivalent in the context of process Env under abstraction of a set of actions I and commutation of a set of pairs of actions C iff $\tau_I(P_1 \parallel Env) = \tau_I(P_2 \parallel Env)$ is derivable by the axioms of branching bisimulation and the axioms of commutativity, i.e. $a \cdot b = b \cdot a$ ($\langle a, b\rangle \in C$). Here, τ_I denotes the substitution of τ for the actions from I.*

This notion of equivalence, however, hampers practical verification for several reasons. The first problem is that classical process algebras, without data, are not very expressive when it comes to specification of real-life processes. Second, the axiomatic characterisation suggests a theorem-proving approach to verification, which draws heavily on the capabilities of a human verifier. The nature of TCAP, however, allows the solution to be found in a compromise.

The language μCRL is a specification language that is supported by a tool set including an efficient generator of labeled transition systems [10]. The format of the generated transition systems can be processed by a popular suite like CÆSAR/ALDÉBARAN [6], with efficient tools for equivalence checking, model checking and reduction of transition systems. Thus, on the one hand, branching bisimularity of specifications can be effectively checked, while on the other hand, axioms exist that can be applied in combination with commutativity of actions [11]. However, it is not clear yet how in general the two can be combined to obtain an operational tool for checking the equivalence relation from Definition 3.

For the specific application of the verification of the TCAP optimisation the answer was found in an interpolation-style approach. That is, in order to establish equivalence of P_1 and P_2, an intermediate process P_{12} is to be found which satisfies two properties: (1) P_1 and P_{12} are equivalent under the restriction that only the axioms of branching bisimulation are used in the derivation; (2) P_{12} and P_2 are equivalent with the restriction that only the axioms of commutation are used. Although success of this strategy is not guaranteed, for the relevant interpolation property has not been established, it appears that commutation plays a restricted role, so the strategy sketched above is well worth trying.

For an example of this approach, consider the merge of the machines TCO and DHA as described in the previous section. The following shows fragments of the relevant μCRL code, which specifies among other things that TCO, on receiving a *tr_uni_req* message, executes an *assemble_uni_message* and sends an *n_uni_req* message.

```
% Specification fragment of TCO

r_tco(tr_uni_req).
   assemble_uni_message.
   s_sccp(n_uni_req)

% Specification fragment of DHA

r_user(tc_uni_req).
    build_audt_apdu.
    request_components.
    process_components.
    assemble_tsl_data.
    s_tco(tr_uni_req).
    free_dialogue_id
```

The semantics of the parallel composition of these two machines is illustrated in Figure 4. It can be easily recognised how this composition is defined as a Cartesian product of the transition systems of its constituents. Note that the non-observable actions are represented by dashed arrows. The corresponding fragment of the optimisation is given in Figure 5.

```
% Optimisation fragment of TCO/DHA

r_user(tc_uni_req).
   build_audt_apdu.
   request_components.
   process_components.
   assemble_tsl_data.
   assemble_uni_message.
   s_sccp(n_uni_req).
   free_dialogue_id
```

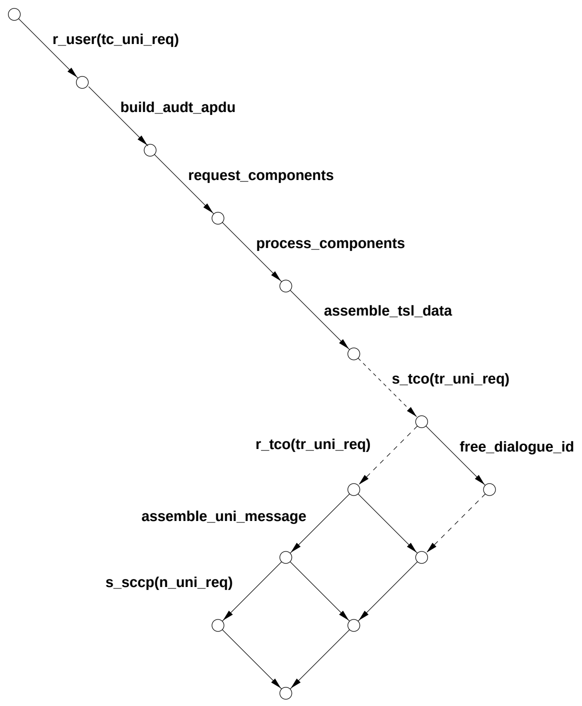

Figure 4: The original transition system

The labeled transition system shows a linear course of actions, clearly not branching bisimular with the original parallel merge. This is where commutation is essential. The freeing of the dialogue is independent from the assembling of the unidata request and the sending of these, and might as well be done before, among or after these two actions. While the optimisation fixes one order, the original allows each of the three possible orderings.

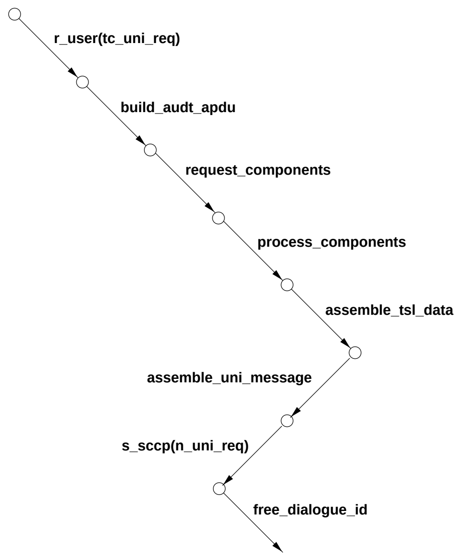

Figure 5: The optimised transition system

It is intuitively clear how to choose the intermediate specification: it consists of the optimisation with added to it all orderings occurring in the original.

```
% Intermediate fragment of TCO/DHA

r_user(tc_uni_req).
   build_audt_apdu.
   request_components.
   process_components.
   assemble_tsl_data.
   ( free_dialogue_id.
        assemble_uni_message.
        s_sccp(n_uni_req)
    +assemble_uni_message.
        free_dialogue_id.
        s_sccp(n_uni_req)
    +assemble_uni_message.
        s_sccp(n_uni_req).
        free_dialogue_id)
```

Now it can be verified that the original specification fragment is branching bisimular to the intermediate fragment, by applying the relevant tool from CÆSAR/ALDÉBARAN to the labeled transition systems of the original and the intermediate TCAP shown in Figure 6.

Thus, by applying commutativity of the freeing of the dialogue id with the assembling of the unidirectional request and the sending of these, by simple theorem proving it can be verified that the intermediate fragment is equivalent to the optimisation.

Finding the intermediate specification requires a good deal of inventiveness and trial and error, especially since the commuting pairs are not given beforehand, but have to be discovered in the process. Moreover, bugs in the optimisation severely obscure a clear view on the notion of commutativity. Nevertheless, a fruitful combination of tool support and human wit allowed the verification to be completed.

4 Results

Both the original and the optimised TCAP were specified in μCRL and, using trial and error, an intermediate TCAP specification was found which made it possible to complete the verification as outlined in the previous section. The full specifications, each covering about 700 lines of code, including the specification of the TCAP environment are included in [3] and an overview of the actual verification process is given in [2]. This section focuses on the results of the verification.

The sizes of the transition systems generated and the reductions modulo branching bisimulation are shown in Table 1. This table clearly illustrates two characteristics of TCAP. First, the sizes of the transition systems are modest, which really helped in the trial and error process of

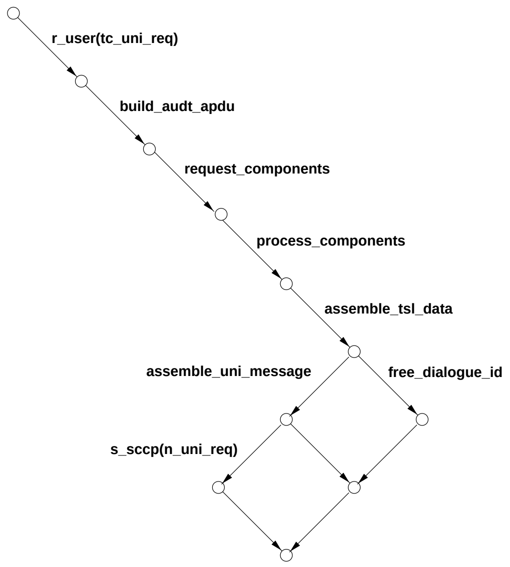

Figure 6: The intermediate transition system

localising bugs and commutative pairs in the intermediate specification. Second, the original TCAP has a larger transition system than the optimisation, which can be attributed to the difference in non-determinism between the two.

The verification revealed two bugs in the design of the optimisation. The first bug consists of the automaton DHA/TCO handling a so-called *local abort* message in the wrong state. Code inspection revealed that this bug was not just present in the μCRL specification and in the informal design from Figure 3: it had survived the implementation process and showed up in the ERLANG code.

The second bug consists of the optimisation being able to free a transaction id, by issuing a *free tid* command, *after* it terminates itself by means of a *discard receive message* command. Of course, the only correct order to execute the commands is to first free the transaction id and then to terminate: doing it the other way around means that after termination there is no process left to free the id. This bug, which only shows up in certain circumstances, manifests itself as a memory leak in the implementation.

Of these bugs, the first one would have showed up early in a test, the optimisation misbehaving consistently. Also, it would have been easily located as a code fragment inserted at the wrong place. The second bug, however, does not manifest itself necessarily in terms of observable system behaviour; memory leaks are typically observed using runtime analysis tools. Both noticing and diagnosing this bug would have been a non-trivial task in a testing situation.

Impressive as this result may seem, it must be admitted that finding this memory leak in the optimised design

specification		states	transitions
original	generated	958	2012
	minimised	187	358
intermediate	generated	829	1981
	minimised	187	358
optimised	generated	462	822
	minimised	159	266

Table 1: The sizes of the labeled transition systems

is not very useful in the current context. Although the design shows a memory leak, the ERLANG implementation does not, since the ERLANG language has its own garbage collection. However, if the design with this bug is ever implemented in another language without similar garbage collection then a memory leak is inevitable.

5 Conclusions and future research

This paper studied the correctness of an optimised design of TCAP by formalising the original and the optimised design in μCRL and checking equivalence of the two. The equivalence relation used was introduced as a relaxation of branching bisimulation which allows certain actions to commute. Checking this relation required finding an intermediate specification, and establishing the branching bisimularity of the original and the intermediate TCAP using the μCRL and the CÆSAR/ALDÉBARAN tool sets, and proving the intermediate and the optimised TCAP commutation equivalent by simple manual theorem proving.

The verification revealed two bugs, one of which would not have been easily found by testing. Therefore, the verification method used has a surplus value with respect to a testing approach. However, this method is expensive in terms of human wit, needed to find the required intermediate specification, which is a non-trivial task in the presence of bugs.

This price could be reduced if a suitable formalisation of commutation can be found, including tool support. Suitable algorithms may have a high complexity, since somehow large numbers of permutations will have to be considered. This is where future research should be aimed at. Other research areas that can speed up verification approaches like this one for TCAP are the automatic translation of both ERLANG and SDL into μCRL.

The added value of the current study is that now Ericsson has at its disposal a correct redesign of TCAP which can be used as a solid basis for implementations. Although testing remains necessary, it is good to know that the rationale underlying the optimisation is sound. Finally, the theory of the algebra of communicating processes is enriched with a new equivalence relation, which is clearly justified by the study of the correctness of an optimised TCAP.

Acknowledgement

We would like to thank Wan Fokkink, Jaco van de Pol and Arno Wouters for contributing a number of improvements to this text, and Hans Nilsson for entrusting his design and implementation of TCAP to us, for his open mind and valuable feedback, without which the verification process would have failed in an early stage of the project.

References

[1] J. Armstrong, R. Virding, C. Wikström, and M. Williams. *Concurrent programming in Erlang*. Prentice Hall, 2nd edition, 1996.

[2] T. Arts and I. A. van Langevelde. How μCRL supported a smart redesign of a real-world protocol. In S. Gnesi and D. Latella, editors, *Proceedings of the Fourth International ERCIM Workshop on Formal Methods for Industrial Critical Systems*, volume I, pages 31–53, July 1999.

[3] T. Arts and I. A. van Langevelde. Verifying a smart design of TCAP; a synergetic experience. Technical Report SEN-R9910, CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands, April 1999.

[4] J. A. Bergstra, J. Heering, and P. Klint, editors. *Algebraic specification*. ACM Press frontier series. Addison-Wesley, 1989.

[5] J. A. Bergstra and J. W. Klop. Algebra of communicating processes with abstraction. *Theoretical Computer Science*, 37(1):77–121, 1985.

[6] J.-C. Fernandez, H. Garavel, A. Kerbrat, R. Mateescu, L. Mounier, and M. Sighireanu. CADP (Cæsar/Aldébaran development package): A protocol validation and verification toolbox. In R. Alur and T. A. Henzinger, editors, *Proceedings of the 8th Conference on Computer-Aided Verification*, volume 1102 of *LNCS*, pages 437–440. Springer Verlag, Aug. 1996.

[7] W. Fokkink. *Introduction to Process Algebra*. Texts in Theoretical Computer Science (an EATCS Series). Springer Verlag, Jan. 2000.

[8] R. van Glabbeek and W. P. Weijland. Branching time and abstraction in bisimulation semantics. *Journal of the ACM*, 43(3):555–600, 1996.

[9] P. Godefroid and D. Pirottin. Refining dependencies improves partial-order verification methods. In *Proceedings of the 5th Conference of Computer-Aided Verification*, volume 598 of *Lecture Notes in Computer Science*, pages 438–449. Springer Verlag, 1993.

[10] J. F. Groote, B. Lisser, and A. G. Wouters. *A Language and Tool Set to Study Communicating Processes with Data*. CWI, `http://www.cwi.nl/~mcrl`, Feb. 1999.

[11] J. F. Groote and A. Ponse. Proof theory for μCRL: a language for processes with data. In D. Andrews, J. Groote, and C. Middelburg, editors, *Proceedings of the International Workshop on Semantics of Specification Languages*, Workshops in Computing, pages 231–250. Springer Verlag, 1993.

[12] J. F. Groote and A. Ponse. The syntax and semantics of μCRL. In A. Ponse, C. Verhoef, and S. van Vlijmen, editors, *Algebra of Communicating Processes (ACP'94)*, Workshops in Computing, pages 26–62. Springer Verlag, 1995.

[13] International Telecommunication Union, `http://www.itu.int`. *CCITT specification and description language (SDL)*, 03/93 edition, March 1993.

[14] International Telecommunication Union, `http://www.itu.int`. *Signalling System No. 7 – Transaction capabilities procedures*, 03/93 edition, March 1993.

[15] H. Nilsson. *An optimised* TCAP *design (internal document)*. Ericsson, Computer Science Laboratory, 1998.