Verification of a Revised WAP Wireless Transaction Protocol[*]

Steven Gordon[1], Lars Michael Kristensen[2]**, and Jonathan Billington[2]

[1] Institute for Telecommunications Research,
University of South Australia
[2] Computer Systems Engineering Centre,
University of South Australia
{Steven.Gordon,Lars.Kristensen,Jonathan.Billington}@unisa.edu.au

Abstract. The Wireless Transaction Protocol (WTP) is part of the Wireless Application Protocol (WAP) architecture and provides a reliable request–response service. The state space method of Coloured Petri Nets has been used to analyse a revised version of WTP, to gain a high level of confidence in the correctness of the design. Full state space analysis allows us to prove properties of the protocol for maximum values of the retransmission counters used in GSM networks (values are 4). However, the size of the state space grows rapidly as the maximum counter values are increased. We apply the sweep-line method to take advantage of the progress present in the protocol, notably the progression through major states of the protocol entities, and the increasing nature of the retransmission counters. The sweep-line method allows us to prove properties of the protocol for larger counter values, including those used in Internet Protocol (IP) networks (where the maximum values are 8). As a result, verification of WTP can be performed for the two most important networks (GSM and IP), the ones for which the WAP standard gives recommended maximum values for the retransmission counters.

1 Introduction

Coloured Petri nets (CPNs) [14] have been used extensively to model and analyse distributed systems [2, 14]. Several factors have contributed to the popularity of CPNs, including the compact manner in which systems can be modelled, and the tool support in the form of Design/CPN [7]. The motivation for modelling and analysing the Wireless Transaction Protocol (WTP) [21] with CPNs is to gain a high level of confidence in the design of the protocol. This is achieved by following a protocol engineering methodology [3], of which the main step is to verify that the *protocol specification* is a faithful refinement of the *service specification*. For WTP, this begins by creating a CPN model of the Transaction Service, from which the possible set of sequences of service primitives is generated by viewing the state space as a Finite State Automaton (FSA). This is the Transaction

[*] The work of Lars M. Kristensen and Jonathan Billington has been supported by an Australian Research Council (ARC) Discovery Grant (DP0210524).
** Supported by the Danish Natural Science Research Council.

Service language. State space analysis of the Transaction Protocol CPN allows several dynamic properties to be proved (e.g. absence of deadlocks, livelocks). The events in the protocol state space that do not represent service primitives are then abstracted out, and the resulting protocol language is compared to the Transaction Service language. The two must be language equivalent for the protocol to refine the service. We apply the FSM tool [1] to conduct the language comparison as there is no support for this in Design/CPN.

This paper has arisen from our research on the verification of WTP [8–10]. An initial CPN model and analysis results of the Transaction Service were published in [8]. This model has since been updated to enforce end-to-end behaviour of the service and allow abort primitives to be submitted after a successful transaction. Similarly, an initial CPN model of the Transaction Protocol was presented in [9], with analysis results revealing three errors in the protocol. Improvements have been made to produce a simpler and clearer model, with the main change being only one transaction is modelled. Analysis of this improved model identified further errors, and changes to the WTP Specification [21] are suggested in [10], leading to a *Revised Transaction Protocol*. The contribution of this paper is the verification and analysis of the Revised Transaction Protocol, including comparison with the Transaction Service. Details of the identified errors and suggested changes leading to the Revised Transaction Protocol are not the focus of this paper. The revised protocol and CPN models will be described only to the level of detail necessary to understand the analysis approach and results.

The state space analysis of the Transaction Protocol presents two practical challenges: the explosion in the number of states and the need to perform the analysis for different parameters. For the latter problem, our objectives are to prove the desired properties of the protocol for practical values of parameters, two of which are the recommended values of parameters in GSM networks and in networks using the Internet Protocol (IP). Several methods have been developed for alleviating the state explosion problem in CPNs, e.g. symmetries [14], equivalence classes [14], and stubborn sets [18, 19] and used in various applications (e.g. [15, 16]). In this paper we apply the recently developed sweep-line method [5, 6] in the verification of WTP [21]. The sweep-line method exploits the progress present in a system to explore the full state space while storing only small fragments of the state space in memory at a time. As demonstrated by some initial case studies [6] this can lead to a significant reduction in peak memory usage. The sweep-line method is used for the values of the parameters of WTP, where full state spaces (state spaces in their most basic form) could not be generated with the available computing power. This paper describes how the sweep-line method is used, and discusses the relevance of the results for WTP, and for protocols in general.

The sweep-line method is related to the bit-state hashing [11] and state space caching [13] methods. For our verification purposes we consider the sweep-line method to be the most appropriate reduction method. The reason is that the bit-state hashing method does not guarantee full coverage of the state space which means that it cannot (in general) be used to obtain the full protocol language.

The state-space caching method does explore the full state space which means that the protocol language could be obtained. However, the state space caching method may visit a state several times which means that the FSA to be input into the FSM tool would be unnecessarily large.

The remainder of this paper is organised as follows: Section 2 provides background information on WAP, while Sect. 3 introduces the protocol design concepts central to the verification task. The Transaction Service and Protocol are described in Sect. 4 and 5, respectively. The full state space analysis of the protocol is presented in Sect. 6, along with the desired properties. Sections 7 to 10 present the application of the sweep-line method to the verification of WTP. Section 11 concludes with a summary and areas of future work.

2 Wireless Application Protocol

The Wireless Application Protocol (WAP) [20] is an architecture proposed by the WAP Forum [20] to overcome the difficulties of providing Internet services in wireless networks. The existing Internet protocols have been designed as a method for communicating across different physical, wired networks. However, several design assumptions no longer hold for wireless networks because of two main differences: wireless networks typically have higher error rates, lower bandwidth, and more frequent disconnections than their wired counterparts; and the devices typically used in wireless, mobile networks (e.g. mobile phones, PDAs) have smaller displays, shorter battery life, less powerful CPUs and alternative input devices than devices commonly used in wired networks (e.g. PCs). The objectives of the WAP architecture design are to take advantage of the existing knowledge and infrastructure in developing Web applications, while using protocols optimised to run over wireless links.

WAP is designed in layers as shown (shaded grey) in Fig. 1. There is an application layer at the top and then four protocol layers. The *Application layer* includes specification of a mark-up language suitable for displaying information on small screens, an accompanying scripting language, and a framework for making use of telephony features in the device and network infrastructure.

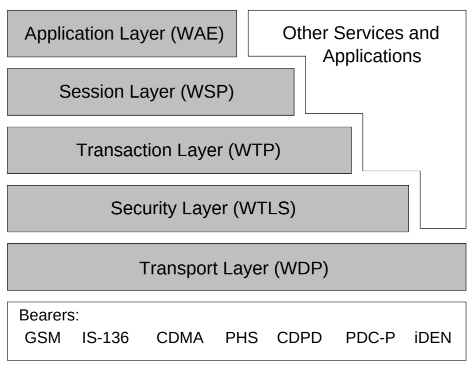

Fig. 1. WAP Architecture.

Together, the components form the Wireless Application Environment (WAE). The *Session layer* defines the Wireless Session Protocol (WSP) for creating a connection-less or connection-oriented session between a client and server. A connection-oriented session includes setup and tear-down, data push, and session suspend and resume. The *Transaction layer* defines the Wireless Transaction Protocol (WTP) which has three classes of service: an unreliable one-way request from the Initiator to the Responder (Class 0); a reliable one-way request (Class 1); and a reliable request by the Initiator and a reliable response from the Responder (Class 2). The *Security layer* defines the Wireless Transport Layer Security (WTLS) which provides applications with privacy, data integrity and authentication. This is an optional layer in the architecture. The *Transport layer* defines the Wireless Datagram Protocol (WDP) which provides a datagram service. WDP includes mappings to a range of supported bearer services so that the upper layer can operate independently of these bearers. The bearers supported include: GSM Short Message Service (SMS) and General Packet Radio Service (GPRS), CDMA Circuit Switched Data, Cellular Digital Packet Data (CDPD) and several proprietary protocols.

3 Protocol Design Concepts

Protocol engineering encompasses the design of communication architectures and protocols using formal methods [4]. We apply a protocol engineering methodology [3] to the analysis of WTP, so that a high level of confidence in the correctness of the design can be achieved. The methodology is based on a fundamental property of communication architectures: functionality is separated into a layered (or hierarchical) structure. This is illustrated in the WAP architecture of Fig. 1, where six layers are evident. The Open Systems Interconnection (OSI) [12] is an example of separation of functionality into layers. Most of the concepts used in our analysis of WTP are based on OSI [12]. Layering is achieved by describing two parts: the *service* the layer provides to the upper layer, and the *protocol* used within the layer to provide that service.

The service is described using *primitives* which convey information between service users. Service users submit primitives to the service provider and the provider delivers primitives to the users. The information conveyed by the primitive is given by its name, type and parameters. The four types of primitives are: request; indication; response and confirm. A complete service definition should specify all possible sets of primitive sequences that can occur between users. This is referred to as the *service language*.

The protocol in a layer describes the mechanisms for communicating between two (or more) protocol entities (PEs). The PEs communicate by sending *protocol data units* (PDUs) via the service provider of the layer below. The protocol describes: the actions each PE takes upon submission of a primitive by a service user; events that cause primitives to be delivered to users; and local events within each PE to ensure the service is provided.

To take advantage of the separation of functionality into layers, the protocol specification must provide the service described in the service specification.

Proving this is true is known as verification and is the major focus of our analysis of WTP. Using CPNs, the state spaces of the service CPN and protocol CPN represent the possible sequences of events that can occur in the service and protocol, respectively. By treating both state spaces as FSA, and removing all events that do not correspond to primitives occurring, we can compare the two corresponding languages to determine if the protocol preserves the sequences of primitives defined in the service.

4 Transaction Service Specification

Our modelling and verification of WTP focuses on the Class 2 service provided by the Transaction layer which is: a reliable request from the *Initiator* and a reliable response from the *Responder*. The Class 2 Transaction Service defines the possible service primitives that can be submitted by or delivered to a user. There are three types of service primitives:

1. TR-Invoke: Initiates a new transaction. The types request (req), indication (ind), response (res) and confirm (cnf) are allowed.
2. TR-Result: Sends back a result of a previously initiated transaction. The req, ind, res and cnf types are allowed.
3. TR-Abort: Aborts an existing transaction. Only req and ind types are allowed.

The WTP Specification [21] describes the parameters for these primitives and gives a table that indicates when one primitive can follow another primitive. We explain the general concepts of a transaction using two sequences of primitives.

Figure 2(a) is a time sequence diagram that shows a possible primitive sequence representing a successful transaction. On the left the primitives submitted by and delivered to the Initiator user are shown. Similarly, the primitives seen by the Responder user are shown on the right. Dependencies between primitives are shown by dashed lines in the area representing the Transaction Service provider.

The Initiator user starts a transaction by submitting a TR-Invoke.req primitive. This is the only primitive that starts a transaction. A TR-Invoke.ind primitive is delivered to the Responder user, indicating that a request has been made. The Responder user then chooses to acknowledge the request, by submitting a TR-Invoke.res primitive. The acknowledgment, via the TR-Invoke.cnf primitive, indicates to the Initiator user that the Responder user has received the request.

Once the Responder user processes the request, (once completed) the user sends the result via the TR-Result.req primitive. The result is delivered to the Initiator user (TR-Result.ind), which then acknowledges its receipt (TR-Result.res). Finally receipt of delivery is confirmed (TR-Result.cnf) to the Responder user. The transaction has now been successfully completed.

There are several other sequences that represent a successful transaction. One scenario occurs when the Responder user does not explicitly acknowledge the invocation request (see Fig. 2(b)). Instead, the sending of the TR-Result.req primitive, implicitly acknowledges that the invocation has been received. This is a more efficient scenario in terms of messages than that in Fig. 2(a). However,

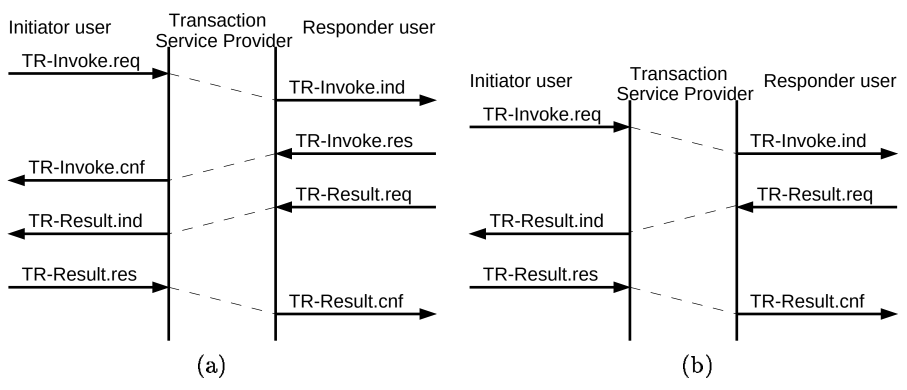

Fig. 2. Example service primitive sequences for UserAck On (a) and UserAck Off (b).

the sequence in Fig. 2(a) is allowed because it is required for use by the Wireless Session Protocol [20]. When it is required, the Initiator user sets a primitive parameter, UserAck, to On. With UserAck On, the users must acknowledge with the response primitives (TR-Invoke.res and TR-Result.res). When UserAck is Off, the response primitives may or may not be submitted.

We have presented two example primitive sequences illustrating successful transactions. There is also the possibility of transactions being aborted by either user or the Transaction Service provider. For further details on the Transaction Service, we refer the reader to [8, 10, 21].

The Transaction Service presented in the WTP Specification only specifies the possible primitive sequences from each user's point of view. There is no information describing how the two users coordinate with each other (although for some primitive sequences it is obvious). In both [8] and [10] we have discussed several shortcomings of the WTP Specification, which includes the absence of information regarding the global sequences of primitives. A CPN model of the Transaction Service has been created [10] with the intent of providing an unambiguous specification of the primitive sequences, and also to allow the automatic generation of the global set of sequences. For the full presentation of this model the reader is referred to [10]. The end result is the generation of two Transaction Service languages: one with UserAck Off and the other with UserAck On.

When UserAck is Off, there are 182 valid sequences of service primitives, the shortest sequence consists of two primitives and the longest consists of eight primitives. Figure 3 shows the FSA representing this language. The FSA was obtained by converting the Design/CPN state space representation into a text format used by the FSM tool [1], hide (make invisible) the non-primitive events, and minimise the FSA. The primitives are abbreviated using the first letter of the primitive name (I for Invoke, R for Result, and A for Abort) and the primitive type (req, ind, res, cnf). Primitives seen by the Initiator user are in upper case, while the Responder user primitives are in lowercase. The initial state of the FSA is represented by node 0. Acceptance (or halt) states are indicated using double circles. When UserAck is On, the service language is a subset (with only 130 sequences) of the UserAck Off language. The reason for this is that the use of

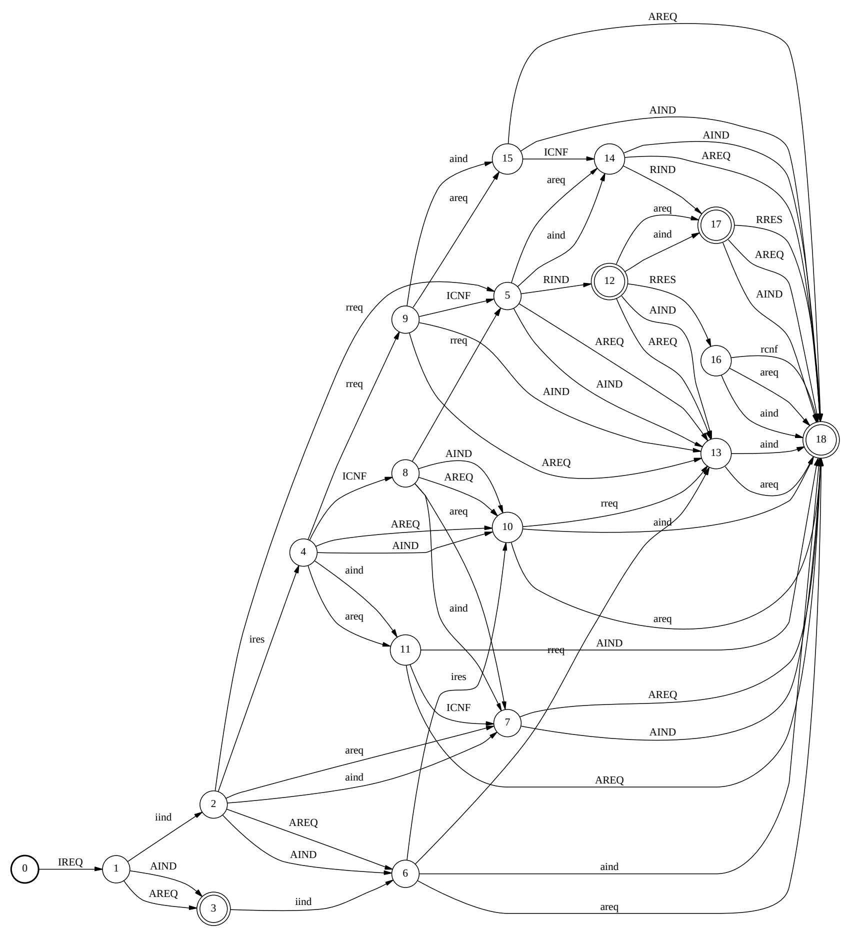

Fig. 3. Transaction Service language with UserAck Off.

the response primitive is optional with UserAck Off. The two languages are used as the baseline for determining if the protocol refines the Transaction Service.

5 Revised Transaction Protocol Specification

The Transaction Protocol describes the operation of the two protocol entities, Initiator and Responder, in terms of PDUs that are used and the actions to be taken when events occur. In Sect. 5.1 we give a brief overview of the important features of the protocol [21], especially those that impact the analysis results. Section 5.2 summarises the errors and the changes that lead to the Revised Transaction Protocol, which is analysed using full state spaces in Sect. 6 and using the sweep-line method in Sect. 7. Parts of the CPN model of the Revised Transaction Protocol are presented in Sect. 5.3.

5.1 Overview of Protocol

The Transaction Protocol defines the procedures for the Initiator PE and Responder PE to communicate in order to provide the Transaction Service. There are four types of PDUs sent between the PEs:

Invoke: Sent by the Initiator PE to start a transaction.
Result: Sent by the Responder PE to return the result.
Ack: Sent by either PE to acknowledge the Invoke PDU or Result PDU.
Abort: Sent by either PE to abort the transaction.

The PDUs are sent via the Transport Service Provider which in the WAP architecture is a datagram protocol. It is assumed re-ordering, losses and duplication of PDUs can occur in the Transport Service Provider. Each PDU has a set of parameters. The important parameters (in terms of the analysis results), along with the general procedure for a successful transaction, will become apparent from the following example (see Fig. 4).

Figure 4 is a time sequence diagram that shows: the primitives submitted by and delivered to the users; events that take place at the PEs (which are represented by the two vertical lines); and the PDUs that are transmitted via the Transport Service Provider. Time increases from top to bottom. The first event shown is the Initiator user submitting a TR-Invoke.req primitive to start a transaction. We assume the UserAck parameter is set to On.

Upon submission of the TR-Invoke.req primitive, the Initiator PE sends an Invoke PDU via the Transport Service Provider. A common parameter to every PDU is a *transaction identifier* (TID). The TID of PDUs in one transaction are identical. When the Initiator user initiates a new transaction (which can occur before the previous transaction has been completed) a new TID is given to the PDUs. The WTP Specification describes how the TID parameter is used to ensure PEs do not confuse which transaction the PDUs belong to.

Upon receipt of the Invoke PDU, the Responder PE delivers a TR-Invoke.ind primitive to the Responder user. The Responder user then submits a TR-Invoke.res

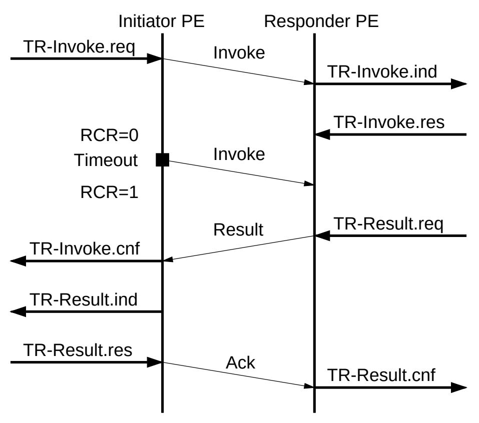

Fig. 4. Example sequence of protocol events for successful transaction.

primitive, acknowledging the invocation. Note that the Responder PE does not immediately send an acknowledgment to the Initiator PE (to reduce the number of PDUs sent across the wireless link). Instead the Responder PE waits for the TR-Result.req from the Responder user. However, in the meantime, as the Initiator PE has not received an acknowledgment of the invocation, it assumes the Invoke PDU was lost, and therefore re-transmits. The re-transmission occurs after a timer at the Initiator PE expires. The Initiator PE also increments a counter, called the *Retransmission Counter* (RCR), which limits the PE to $RCRI_{\max}$ retransmissions, after which the transaction will be aborted.

As the Responder PE has already received an Invoke PDU, it discards the re-transmitted Invoke PDU. When the Responder user submits the TR-Result.req primitive, a Result PDU is sent to the Initiator PE. Upon its receipt, the TR-Invoke.cnf and TR-Result.ind primitives are delivered to the Initiator user (i.e. the user now has the result). The last step in the transaction is the acknowledgment of the result. The Initiator user submits the TR-Result.res primitive, the Initiator PE sends an Ack PDU and upon its receipt a TR-Result.cnf primitive is delivered to the Responder user. The transaction has now been successfully completed. Note that the Responder PE also maintains a retransmission counter which it uses to re-transmit the Result PDU if it has not received an acknowledgment before a certain timeout period. The number of retransmissions in the Responder is limited to a number denoted by $RCRR_{\max}$.

The example in Fig. 4 has shown the general procedure for a transaction. There are many other scenarios, due to other features being used such as: multiple transactions can be underway at the same time; transactions can be aborted either by the users or the service provider; PDUs can be received out of order, or not received at all; and the UserAck option is set to Off.

The procedures for the Transaction Protocol's operation are mainly described in [21] using a set of state tables. There is a table for each state of each PE, and each entry (row) in the table comprises four items (columns): the event, such as the submission of a primitive by a user or timeout at the PE; conditions on the event; actions to be taken; and the next state of the PE. These state tables are used as the basis of the CPN model of the Transaction Protocol.

5.2 Errors in the Transaction Protocol

The approach to analysing WTP is to model the protocol described in the WTP Specification [21], compare the language to the Transaction Service language (and prove other properties, which are discussed in Sect. 6), and incrementally fix any errors present. In [9] three errors were presented, and fixes proposed. Since then a number of other errors have been discovered in the Transaction Protocol [10]. We summarize the two main errors here.

1. The Ack PDU and Result PDU are ambiguous, in that the receiving PE cannot be certain if the PDU indicates that the peer user has acknowledged (submitted a response primitive), or only the peer PE has acknowledged (i.e.

the response primitive has not been submitted). Both PDUs require an extra field to explicitly indicate their meaning.

2. After a transaction has been completed, the Responder PE may receive a re-transmitted Invoke PDU or Ack PDU after certain messages have been lost, and wrongly accept it to be part of a new transaction. The mechanism for checking the validity of Invoke PDUs must be prevented from taking place until a set time after a transaction has completed (instead, the Invoke PDUs are discarded). After that time, the Responder PE will know that any Invoke PDUs will be for new transactions.

These two errors do not change the general procedure for a transaction as described in Sect. 5.1. Further details on the errors and proposed solutions can be found in [10]. This motivated the development of a revised Transaction Protocol to eliminate these errors. To see if the revised protocol was correct, we revised the previous CPN model accordingly.

5.3 Revised Protocol Specification CPN

The CPN of the Revised Transaction Protocol comprises four hierarchical levels (see Fig. 5), starting with a single page (TR_Protocol) giving an overview of the two PEs and the communication channel. The communication channel is modelled as a place for either direction of communication: InitToResp and RespToInit. To limit the size of the state space, a bound has been put on each of the two places InitToResp and RespToInit for the number of PDUs that can be in transit between the Initiator and the Responder. When the bound is larger than 1, re-ordering of PDUs in the Transport Service Provider is possible. Moreover, PDUs may be lost in transmission. One limitation of the model is that we currently do not consider duplication of PDUs. Another is the omission of the segmentation and re-assembly feature (which is optional in the protocol). The CPN is also simplified by considering just a single transaction, instead of multiple, concurrent transactions. This is justified because using TIDs for PDUs the transactions are independent, provided that PDUs have a maximum packet lifetime (which is assumed in the specification).

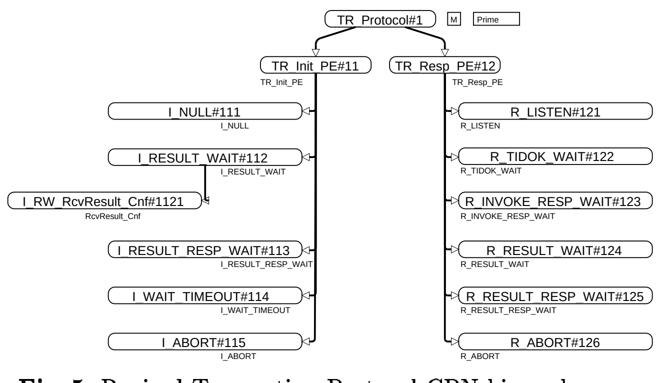

Fig. 5. Revised Transaction Protocol CPN hierarchy page.

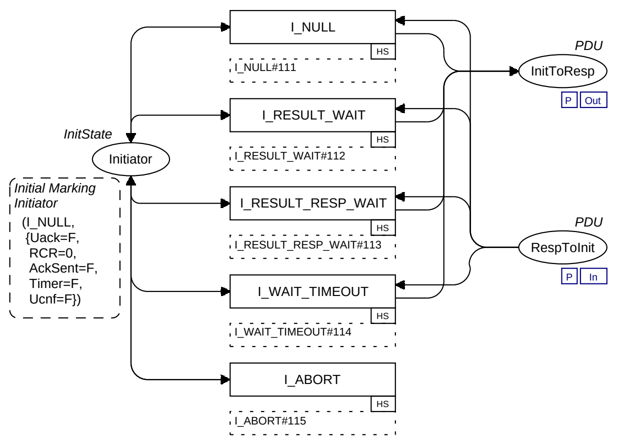

Fig. 6. Revised Transaction Protocol CPN Initiator PE page.

There are two second level pages, TR_Init_PE and TR_Resp_PE, which show the major states of each PE. Figure 6 shows the Initiator PE page. The two communication places are shown on the right, substitution transitions represent the procedures that occur in the set of major states, and the place Initiator maintains state information for the PE. The initial marking of Initiator shows the Initiator PE is in the I_NULL state and gives the initial values of the state variables (e.g. RCR=0). The general procedure for the Initiator PE during a single transaction is to traverse through the states in the following order: I_NULL, I_RESULT_WAIT, I_RESULT_RESP_WAIT, and I_WAIT_TIMEOUT. When the transaction is completed, the Initiator PE returns to the I_NULL state with certain of its other state variables set to special values. The state I_ABORT models a Transport Service Provider initiated abort. The Responder PE has a similar page, where its major states are represented by substitution transitions.

Each transition on the TR_Init_PE page (Fig. 6) is decomposed into another CPN page. These third level pages model the entries in the state tables given in the WTP Specification. Figure 7 shows the I_NULL page for the Initiator PE. The general approach is to have a transition for every entry in the state tables given in [21]. Each transition has an input arc from the Initiator place to ensure it can only be enabled when the Initiator PE is in the correct state and the state variables meet the conditions of the state table entry. Transitions model primitives being submitted by the user, primitives being delivered to the user (the only fourth level page models a special case of this), timeouts, the receipt of PDUs and the sending of PDUs. The full Transaction Protocol CPN (original and revised) can be found in [10] for the case when the medium does not lose or duplicate PDUs.

6 Full State Space Analysis

The verification of the Revised Transaction Protocol using full state spaces involves proving four desired properties (presented in Sect. 6.1), then obtaining

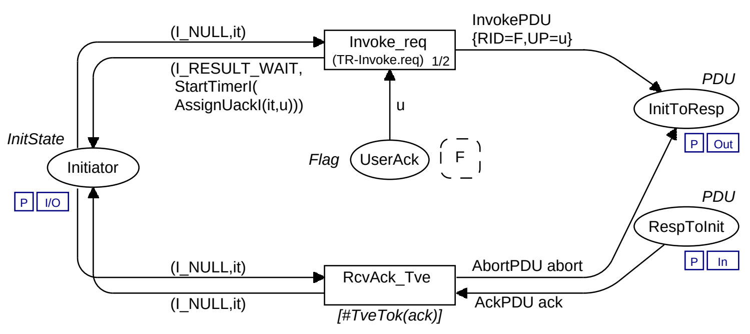

Fig. 7. Revised Transaction Protocol CPN I_NULL page.

the protocol primitive language from the state space and comparing it to the Transaction Service language. As there are several input parameters for the protocol this process is repeated for a set of parameter values. The state space and language analysis results are summarised in Sect. 6.2.

6.1 Desired Properties of the Revised Transaction Protocol

We have defined [10] four desired properties of the Revised Transaction Protocol that should hold to give a high level of confidence in the correctness of the design. The first, and most important, is the faithful refinement of the Transaction Service. The state space of the protocol is computed and converted to a textual representation using Design/CPN [7], which is then used as input to FSM [1] to obtain the protocol language. The Transaction Protocol language and the Transaction Service language are then compared using FSM, and they must be identical for this property to be true. FSM implements standard algorithms for determinisation and comparison of minimal FSAs to check language equivalence.

The three other properties are: successful termination; absence of livelocks and absence of unexpected dead transitions. We have specified the expected dead markings (called terminal markings) as a predicate on the markings, and for successful termination the state space must only contain these terminal markings. No deadlocks (i.e. undesired dead markings) can be present. The strongly connected components (SCC) graph of the Transaction Protocol is used to determine the absence of livelocks. We define livelocks as terminal strongly connected components of the SCC graph with more than one node or one or more arcs. For some sets of parameters several features of the protocol may not be activated. As a result, transitions modelling these features will be dead. We have identified such transitions, and for the fourth property to be true, no other transitions can be dead.

6.2 Analysis Results

There are four parameters used in the Revised Transaction Protocol CPN: $RCRI_{\max}$, the maximum value of the counter RCR used by the Initiator PE;

$RCRR_{\max}$, the maximum value of the RCR used by the Responder PE; and *UserAck*, which indicates whether the transaction has UserAck On or Off. Finally, there is the bound on the places modelling the communication channel. We will denote this bound by C_{chan}. Ideally, the properties would be verified independently of the parameters. However, our goal is to investigate configurations that are likely to be used in practice. The WTP Specification gives several suggested values for the counter parameters: $RCRI_{\max}$ and $RCRR_{\max}$ both 8 for IP (Internet Protocol) networks; and 4 in GSM SMS networks [21]. Here we will consider the configuration for which $RCRI_{\max} = RCRR_{\max}$ assuming that the Initiator and Responder work across the same network. All results presented in this paper were obtained on a PIII 1GHz PC with 512 Mb of memory.

Table 1 shows the experimental results obtained for values of the retransmission counters up to 5 which include those recommended for GSM SMS networks. The Config column gives the values of the parameters considered written in the form $X - Y$ where X specifies $RCRI_{\max}$, and Y specifies whether UserAck is On (indicated by T) or Off (indicated by F). The bound on the communication channel C_{chan} was in all cases set to 2 to consider re-ordering of PDUs. The Nodes and Arcs columns give the number of nodes and arcs in the full state space, respectively. The G-Time column gives the time for generation of the state space and for checking the properties of absence of deadlocks, absence of livelocks, and absence of unexpected dead transitions. The generation time is written in the form *h*:*mm*:*ss* where *h* is hours, *mm* is minutes and *ss* is seconds. The time used for verification of the three properties was in all cases insignificant compared to the state space generation time.

The C-Time column gives the time for checking (using FSM) whether the protocol and the service languages are identical. It is worth observing that the time used for language comparison was in all cases significantly less than the time used for state space generation in Design/CPN. In all cases, the deterministic FSA computed from the state space was smaller than the full state space. This is the main reason why comparison of the languages is tractable in this case.

For all configurations in Table 1, the protocol and service languages were equivalent, and the protocol successfully terminated and did not include livelocks or unexpected dead transitions. The results are only given for $RCRI_{\max}$ and $RCRR_{\max}$ up to 5 because Config 6-F could not be calculated using full state space analysis on the machine used. Configs 6-T and 7-T could be calculated, and the properties were proven correct. Therefore, five is the highest value of the counters for which the protocol can be verified with both UserAck On and Off.

Table 1. Experimental results for full state space analysis.

Config	Nodes	Arcs	G-Time	C-Time	Config	Nodes	Arcs	G-Time	C-Time
1-T	1,838	9,587	0:00:03	0:00:01	1-F	4,232	32,686	0:00:10	0:00:01
2-T	10,333	58,286	0:00:30	0:00:02	2-F	24,905	149,792	0:01:52	0:00:03
3-T	30,978	178,329	0:02:15	0:00:08	3-F	74,017	453,010	0:10:43	0:00:39
4-T	65,873	380,825	0:07:06	0:00:21	4-F	154,231	945,127	0:38:15	0:00:46
5-T	173,343	654,842	0:16:33	0:00:47	5-F	262,442	1,605,984	1:36:38	0:01:43

7 Sweep-Line Analysis

There is an intuitive presence of progress in the Transaction Protocol from the start of the transaction towards the transaction either being successfully completed or aborted. This progress is (for instance) explicit in the Initiator PE that starts in its initial state (I_NULL) and progresses through its intermediate internal states (I_RESULT_WAIT, I_RESULT_RESP_WAIT, I_WAIT_TIMEOUT) (see Fig. 6) and eventually ends in the I_NULL state with its state variables set to special values. We will denote the terminating state of the Initiator by I_TERM. There is a similar presence of progress in the Responder PE. In addition to the progress in the internal states of the two protocol entities, there is also progress due to the retransmission counters in the two protocol entities as the values of these increases as the transaction proceeds.

The progress exhibited by the Transaction Protocol is also reflected in the state space of the CPN model. Figure 8 shows the initial fragment of the state space. To simplify the figure we have omitted states in this initial fragment resulting from user and provider initiated abort. The states have been organised into four layers based on how far the system has *progressed* according to the internal state of the two protocol entities. For example, layer 1 (the top most layer) contains the states where the Initiator PE is in state I_NULL and the Responder PE is in state R_LISTEN. The initial marking represented by node 1 is the only marking in this layer. Nodes 2-4 constituting layer 2 are the states where the Initiator PE is in state I_RESULT_WAIT and the Responder PE is in R_LISTEN. Hence, the lower the layer, the further the system has progressed.

The key observation to make is that progress in the Transaction Protocol manifests itself by the property that a marking in a given layer has successor markings either in the same layer or in some lower layer, but never in an upper layer. The idea underlying the sweep-line method [6] is to exploit such progress by deleting markings on-the-fly during state space exploration. The deletion is done such that the state space exploration will eventually terminate and upon termination all reachable markings will have been explored exactly once.

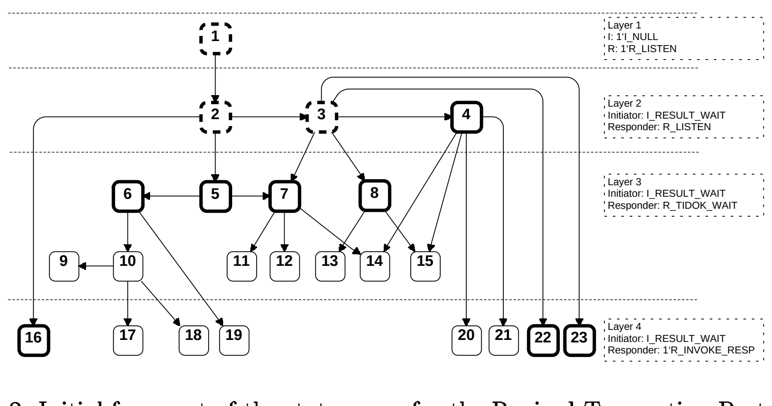

Fig. 8. Initial fragment of the state space for the Revised Transaction Protocol.

To illustrate how the sweep-line method operates, consider Fig. 8 and assume that it represents a snapshot taken during conventional state space exploration. Dashed nodes are *fully processed* markings (i.e. markings that are stored in memory and all their successor markings have been calculated). Nodes with a thick solid black border are *unprocessed* nodes (i.e. nodes that are stored in memory, but their successor markings have not yet been calculated). Nodes with a thin solid black border are markings that have not yet been calculated.

If the state space exploration algorithm processes markings according to the progress of the protocol they correspond to, node 4 will be the marking among the unprocessed markings that will be selected for processing next. This will add nodes 14, 15, 20, and 21 to the set of stored markings and mark these as unprocessed. At this point it can be observed that it is not possible from any of the unprocessed markings to reach one of the markings 1, 2, 3, and 4. The reason is that nodes 1, 2, 3, and 4, represent markings where the protocol has not progressed as far as in any of the unprocessed markings. Hence, it is safe to delete nodes 1, 2, 3, and 4, as they cannot possibly be needed for comparison with newly generated markings when checking (during the state space exploration) whether a marking has already been visited. In a similar way, once all the markings in the third layer have been fully processed these nodes can be deleted from the set of nodes stored in memory. Intuitively, one can think of a *sweep-line* being aligned with the highest layer (seen from the top) that contains unprocessed markings. During state space exploration, unprocessed markings are selected for processing in a least-progress first order causing the sweep-line to move downwards. Markings will thereby be added in front of the sweep-line and deleted behind the sweep-line.

8 Progress Measure Specification

The progress exploited by the sweep-line method is formally captured by a *progress measure*. In this section we formally specify the progress measure for the Revised Transaction Protocol based on the intuition presented in Sect. 7.

A *progress measure* [6] consists of a *progress mapping* assigning a *progress value* to each marking, and a *partial order* $(O, \sqsubseteq)$ on the markings of the system. A partial order $(O, \sqsubseteq)$ consists of a set O and a relation $\sqsubseteq \subseteq O \times O$ which is reflexive, transitive, and antisymmetric. Moreover, the partial order is required to preserve the reachability relation of the system. The definition of progress measure below is identical to Def. 1 in [6] except that we give the definition in a Petri Net formulation. In the definition, M_0 denotes the initial marking, $[M_0\rangle$ denotes the set of markings reachable from the initial marking, and $\mathbb{M}$ denotes the set of all markings. If a marking M_1 is reachable from a marking M_2 via some occurrence sequence we write $M_1 \rightarrow^* M_2$. In particular $M \rightarrow^* M$ for all markings $M \in \mathbb{M}$.

Definition 1. *A* ***progress measure*** *is a tuple* $\mathcal{P} = (O, \sqsubseteq, \psi)$ *such that* $(O, \sqsubseteq)$ *is a partial order and* $\psi : \mathbb{M} \rightarrow O$ *is a progress mapping from markings into* O *satisfying:* $\forall M, M' \in [M_0\rangle : M \rightarrow^* M' \Rightarrow \psi(M) \sqsubseteq \psi(M')$. □

It is worth noting that the definition of progress measure implicitly states that for all $M \in [M_0\rangle : \psi(M_0) \sqsubseteq \psi(M)$, i.e., the initial marking is minimal among the reachable markings with respect to the progress measure. The requirement that the progress measure preserves the reachability relation can be verified fully automatically during the sweep-line state space exploration since each arc of the state space is explored. From Def. 1 it follows that the sweep-line method can handle state space containing cycles, but all states on the cycle are required to have the same progress measure.

For specifying Transaction Protocol progress measure, we introduce some notation: For a reachable marking M, $\mathrm{RCRI}(M)$ denotes the value of the Initiator PEs retransmission counter in M, $\mathrm{RCRR}(M)$ denotes the value of the Responder PEs retransmission counter in M, $\mathrm{Is}(M)$ denotes the Initiator PE's internal state in M, and $\mathrm{Rs}(M)$ denotes the Responder PE's internal state in M. To capture the progress from the internal states, we define a mapping ψ_{Is} that enumerates the Initiator PE's states according to the progress it represents.

$$\psi_{\mathrm{Is}}(M) = \begin{cases} 0 \text{ if } \mathrm{Is}(M) = \mathsf{I\_NULL} \\ 1 \text{ if } \mathrm{Is}(M) = \mathsf{I\_RESULT\_WAIT} \\ 2 \text{ if } \mathrm{Is}(M) = \mathsf{I\_RESULT\_RESP\_WAIT} \\ 3 \text{ if } \mathrm{Is}(M) = \mathsf{I\_WAIT\_TIMEOUT} \\ 4 \text{ if } \mathrm{Is}(M) = \mathsf{I\_TERM} \end{cases}$$

A similar mapping (denoted ψ_{Rs}) is defined enumerating the Responder PE's internal state according to progress. We have not assigned any value to I_ABORT (see Fig. 6) since it only represents the event of an abort and not an actual state of the Initiator PE.

We define two mappings ψ_{RCRI} and ψ_{RCRR} to capture progress in terms of the retransmission counters in the Initiator PE and the Responder PE, respectively. The definitions are: $\psi_{\mathrm{RCRI}}(M) = \mathrm{RCRI}(M)$ and $\psi_{\mathrm{RCRR}}(M) = \mathrm{RCRR}(M)$.

In addition to the progress due to retransmission counters and internal states of the protocol entities, we will also exploit that when both protocol entities have terminated, the sum of the number of tokens on the two places InitToResp and RespToInit is decreasing (see Fig. 6). The mapping ψ_{CHAN}, capturing progress in terms of the decreasing number of tokens on the communication channel, is defined below. For a place p and a marking M, we denote by $M(p)$ the marking of p in M, and by $|M(p)|$ the number of tokens on p in M.

$$\psi_{\mathrm{CHAN}}(M) = 2 * C_{\mathsf{chan}} - (|M(\mathsf{InitToResp})| + |M(\mathsf{RespToInit})|)$$

Having identified the sources of progress, we can now define the full progress mapping ψ_{WTP} for the Transaction Protocol as a 5-dimensional vector:

$$\psi_{\mathsf{WTP}}(M) = \begin{cases} (\psi_{\mathrm{Is}}(M), \psi_{\mathrm{Rs}}(M), \psi_{\mathrm{RCRI}}(M), \psi_{\mathrm{RCRR}}(M), \psi_{\mathrm{CHAN}}(M)) \\ \qquad\qquad\qquad\qquad \text{if } \mathrm{Is}(M) = \mathsf{I\_TERM} \\ \qquad\qquad\qquad\qquad \wedge\ \mathrm{Rs}(M) = \mathsf{R\_TERM} \\ (\psi_{\mathrm{Is}}(M), \psi_{\mathrm{Rs}}(M), \psi_{\mathrm{RCRI}}(M), \psi_{\mathrm{RCRR}}(M), 0) \qquad \text{otherwise} \end{cases}$$

For the definition of the partial order $\sqsubseteq_{\mathsf{WTP}}$ between the progress values, we view the progress values as *progress vectors* in $\mathbb{Z}^5$ and consider one progress vector $\psi_{\mathsf{WTP}}(M_1) = (a_1, a_2, a_3, a_4, a_5)$ to be less than or equal to a progress vector $\psi_{\mathsf{WTP}}(M_2) = (b_1, b_2, b_3, b_4, b_5)$ if and only if it is possible to obtain $\psi_{\mathsf{WTP}}(M_2)$ from $\psi_{\mathsf{WTP}}(M_1)$ by increasing the progress values (coordinates) in $\psi_{\mathsf{WTP}}(M_1)$. Formally the ordering is defined as: $\psi_{\mathsf{WTP}}(M_1) \sqsubseteq_{\mathsf{WTP}} \psi_{\mathsf{WTP}}(M_2)$ if and only if $a_i \leq b_i$ for all $1 \leq i \leq 5$. It is straightforward to see that $\sqsubseteq_{\mathsf{WTP}}$ is a partial order. That the tuple $(\psi_{\mathsf{WTP}}, (\sqsubseteq_{\mathsf{WTP}}, \mathbb{Z}^5))$ is indeed a progress measure, i.e., also preserves the reachability relation according to Def. 1, is checked fully automatically by the Design/CPN sweep-line library [6] that we apply for the state space exploration.

9 Implementing the Progress Measure

In this section we show how the progress measure ψ_{WTP} in the previous section can be implemented and provided as input to the sweep-line library [6]. The sweep-line library is implemented on top of Design/CPN [7], and the STANDARD ML (SML) [17] programming language is available to the user for specifying progress measures. The user provides a progress measure to the tool by writing an SML function mapping a marking into an integer. SML support for unbounded integers (integers which cannot overflow) is exploited as the co-domain for progress mappings. The ordering on progress values is the usual total ordering on integers.

The first step when implementing ψ_{WTP} is to overcome the limitation that the sweep-line library currently only supports progress measures where the progress values are integers and the ordering is the total ordering on integers. The progress values for ψ_{WTP} are vectors, and the ordering $\sqsubseteq_{\mathsf{WTP}}$ is a partial order. Hence we need to define a (preferably injective) mapping from $\mathbb{Z}^5$ to $\mathbb{Z}$ and embed the partial order $\sqsubseteq_{\mathsf{WTP}}$ into the total ordering $(\leq)$ on integers. A simple way to do this is to interpret the vector $\psi_{\mathsf{WTP}}(M) = (a_1, a_2, a_3, a_4, a_5)$ as a number in base n, where n is larger than any value that can be assumed by the components. Hence, we can set $n = \max(2{*}C_{\mathsf{chan}}, 4, RCRI_{\max}, RCRR_{\max}) + 1$. The embedded progress measure $\hat{\psi}_{\mathsf{WTP}}$ can be defined as:

$$\hat{\psi}_{\mathsf{WTP}}(M) = \psi_{\mathrm{IS}}(M) * n^0 + \psi_{\mathrm{RS}}(M) * n^1 + \psi_{\mathrm{RCRI}}(M) * n^2 + \psi_{\mathrm{RCRR}}(M) * n^3 + \psi_{\mathrm{CHAN}}(M) * n^4$$

Another possibility would be to define the embedded progress values as the Euclidean distance from $(0,0,0,0,0)$ to the point $\psi_{\mathsf{WTP}}(M)$. Also, the weight given to the different components can be set differently. In the rest of this paper we used the embedding specified above. It is left as future work to compare the alternatives. The embedded progress measure can be implemented in 20 lines of SML code and input into the sweep-line library. It should be noted that by embedding the progress measure into the set of integers (i.e. making it 1-dimensional) we "lose" some of the progress in the system compared to being able to handle the 5-dimensional progress measure directly.

10 Sweep-Line Experimental Results

The sweep-line method supports on-the-fly verification of safety and reachability properties. Checking for dead markings and identifying dead transitions is straightforward with the sweep-line method since all nodes and arcs of the state space are visited. The sweep-line library has direct support for checking that all dead markings satisfy a certain marking predicate, and for identifying the dead transitions. Hence, checking for absence of deadlocks and that only the desired transitions are dead is fully supported by the sweep-line library. The absence of livelocks can also be checked with the sweep-line method, but is currently not supported by the sweep-line library. Hence, for the configurations of WTP where verification has been done using the sweep-line method, we have not been able to check the absence of livelock property. To compare the protocol and the service languages, we write the FSA corresponding to the state space into a file *during* the state space exploration. This file can then be used as input to the FSM tool. This means that the full state space is represented in the FSM tool, but it still saves memory in practice since the information related to a node and an arc is reduced to an integer, whereas in the current implementation of the state space method in Design/CPN, a state takes up more space than an integer.

Table 2 shows the experimental results obtained with the sweep-line method for different configurations of WTP with values for $RCRI_{\max} = RCRR_{\max}$ up to 8, and $C_{\mathsf{chan}} = 2$. The Full State Space column lists (for comparison) the number of nodes in the full state space and the time for full state space generation when possible. A "-" is an entry that indicates that the corresponding number could not be obtained. The Sweep-Line Method columns list the peak number of markings stored with the sweep-line method. The number in parentheses after the peak number of markings gives (in percentage) the peak number of markings divided by the number of markings in the state space. The GC-Time column gives the time used by the sweep-line method for garbage collection of states and the E-Time column gives the total time for the exploration of the state space, including the time spent in writing the FSA into a file. The Threshold column gives the garbage collection threshold used for the state space exploration. The sweep-line library uses a simple algorithm for initiating garbage collection during the sweep: whenever n new markings have been added to the state space, garbage collection is initiated. The C-Time column lists the time used by the FSM tool to compare the protocol and the service languages. It can be seen that the use of the sweep-line method reduces the peak number of nodes stored during state space exploration ranging from around 44 % to 21 %. Also, the total calculation time using the sweep-line method is significantly shorter than using the full state space analysis due to the fewer number of markings that a new marking is compared against. All comparisons revealed that the protocol and services languages were equivalent. It is also interesting to observe that as the configurations are approaching counter values of 8, the language comparison starts to become the bottleneck rather than the generation of the state space.

Table 2. Experimental results for sweep-line analysis.

Config	Full State Spaces		Sweep-Line Method				
	Nodes	Time	Peak Nodes	GC-Time	E-Time	Threshold	C-Time
1-T	1,838	0:00:03	762 (41.4)	0:00:01	0:00:04	250	0:00:01
2-T	10,333	0:00:30	3,873 (37.5)	0:00:02	0:00:26	1500	0:00:01
3-T	30,978	0:02:15	9,982 (32.2)	0:00:10	0:01:29	3000	0:00:08
4-T	65,873	0:07:06	17,543 (26.6)	0:00:09	0:01:35	4500	0:00:20
5-T	113,343	0:16:33	27,737 (24.5)	0:00:59	0:06:04	6000	0:00:46
6-T	172,657	0:32:45	39,552 (22.9)	0:01:56	0:09:50	7500	0:01:29
7-T	243,765	1:07:52	53,985 (22.1)	0:02:56	0:14:21	10000	0:02:31
8-T	326,667	-	70,037 (21.4)	0:04:25	0:20:13	12500	0:06:00
1-F	4,232	0:00:10	1,857 (43.9)	0:00:01	0:00:11	500	0:00:01
2-F	24,905	0:01:52	9,099 (36.5)	0:00:08	0:01:17	3000	0:00:04
3-F	74,017	0:10:43	23,445 (31.7)	0:00:38	0:04:21	6000	0:00:16
4-F	154,231	0:38:15	41,145 (26.7)	0:01:48	0:09:54	9000	0:00:47
5-F	262,442	1:36:38	63,627 (24.2)	0:04:00	0:18:48	12000	0:01:46
6-F	397,583	-	89,970 (22.6)	0:07:15	0:29:52	15000	0:04:36
7-F	559,604	-	121,333 (21.7)	0:10:26	0:43:04	20000	0:16:16
8-F	748,505	-	160,091 (21.4)	0:14:40	0:59:09	25000	2:36:07

11 Conclusions

We have applied the state space method for Coloured Petri Nets and language comparison techniques as implemented in the FSM tool for the verification of WTP focusing on checking that the revised version of WTP conforms to the service specification. Full state spaces were used to verify WTP for smaller configurations and the sweep-line method was applied to obtain the state spaces for the larger configurations. We have now verified WTP for the two configurations recommended in the WTP Specification, giving us a high level of confidence in the correctness of the design. This illustrates the applicability of formal methods to identify and fix errors in communication protocols, and prove properties for non-trivial and practically relevant scenarios. The use of the sweep-line method made it possible to reduce peak memory consumption of Design/CPN to about 20 % of the full state space. This allowed us to obtain the state spaces for the configurations of interest. We have shown how control flow and retransmission counters in protocols constitute a source of progress that can be exploited to obtain a progress measure for the sweep-line method. This technique is applicable in general for verification of protocols using the sweep-line method.

A disadvantage of our current approach is that the comparison between the service and the protocol language is not done on-the-fly during the state space exploration. We have to write the FSA corresponding to the protocol language into a file such that it could be compared with the service language using FSM. Future work will investigate whether a method exists for the protocol language to be compared with the service language during the state space exploration with the sweep-line method.

References

[1] AT&T. FSM Library. Web site: http://www.research.att.com/sw/tools/fsm.

[2] J. Billington, M. Diaz, and G. Rozenberg (Eds.) *Application of Petri Nets to Communication Networks.* LNCS 1605. Springer-Verlag, 1999.

[3] J. Billington, M. C. Wilbur-Ham, and M. Y. Bearman. In *Protocol Specification, Verification and Testing V*, pages 59–70. Elsevier Science Publishers, Amsterdam, New York, Oxford, 1986.

[4] J. Billington. Formal Specification of Protocols: Protocol Engineering. In *Encyclopedia of Microcomputers*, pages 299–314. Marcel Dekker, New York, NY, 1991.

[5] S. Christensen, L. M. Kristensen, and T. Mailund. *Design/CPN Sweep-Line Method Library.* Department of Computer Science, Aarhus University, Aarhus, Denmark, 2001. To appear, http://www.daimi.au.dk/designCPN/.

[6] S. Christensen, L. M. Kristensen, and T. Mailund. A Sweep-Line Method for State Space Exploration. In *Proc. of TACAS 2001*, pages 450–464, LNCS 2031, Springer-Verlag, 2001

[7] CPN Group. Design/CPN Online. http://www.daimi.au.dk/designCPN/.

[8] S. Gordon and J. Billington. Modelling the WAP Transaction Service using Coloured Petri nets. In *Proc. of MDA 1999*, pages 105–114. LNCS 1748, Springer-Verlag, 1999.

[9] S. Gordon and J. Billington. Analysing the WAP Class 2 Wireless Transaction Protocol using Coloured Petri nets. In *Proc. of ICATPN 2000*, pages 207–226, LNCS 1825. Springer-Verlag, 2000.

[10] S. D. Gordon. *Verification of the WAP Transaction Layer using Coloured Petri Nets.* PhD thesis, University of South Australia, Australia, Nov. 2001.

[11] G. J. Holzmann. *Design and Validation of Computer Protocols.* Prentice-Hall, Englewood Cliffs, NJ, 1991.

[12] ITU. Information Technology—Open Systems Interconnection—Basic reference model: The basic model. ITU-T Recommendation X.200, July 1994.

[13] C. Jard and T. Jeron. Bounded-memory Algorithms for Verification On-the-fly. In *Proc. of CAV'91*, pages 192–202, LNCS 575. Springer-Verlag, 1991.

[14] K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volumes 1 to 3.* Monographs in Theoretical Computer Science. Springer-Verlag, Berlin, 1997.

[15] J. B. Jørgensen and L. M. Kristensen. Computer Aided Verification of Lamport's Fast Mutual Exclusion Algorithm using Coloured Petri nets and Occurrence Graphs with Symmetries. *IEEE Trans. Parallel and Dist. Sys.*, 10(7):714–732, July 1999.

[16] L. Lorentsen and L. Kristensen. Modelling and analysis of a Danfoss Flowmeter System. In *Proc. of ICATPN'2000*, pages 346–366, LNCS 1825. Springer-Verlag, 2000.

[17] J. D. Ullman. *Elements of ML Programming.* Prentice Hall, Englewood Cliffs, NJ, 2nd edition, 1998.

[18] A. Valmari. Stubborn Sets for Reduced State Space Generation. In *Advances in Petri Nets 1990*, pages 491–515, LNCS 424. Springer-Verlag, 1990.

[19] A. Valmari. Stubborn Sets of Coloured Petri Nets. In G. Rozenberg, editor, *Proc. of ICATPN'91*, pages 102–121, 1991.

[20] WAP Forum. Wireless Application Protocol.
Specifications available via: http://www.wapforum.org/.

[21] WAP Forum. WAP Wireless Transaction Protocol Specification. June 2000 Conformance Release. Available via: http://www.wapforum.org/, 19 Feb. 2000.