Analysing the WAP Class 2 Wireless Transaction Protocol using Coloured Petri Nets

Steven Gordon and Jonathan Billington

Cooperative Research Centre for Satellite Systems,
University of South Australia,
Mawson Lakes SA 5095, Australia
{sgordon,jb}@spri.levels.unisa.edu.au

Abstract. Coloured Petri nets (CPNs) are used to specify and analyse the Class 2 Wireless Transaction Protocol (WTP). The protocol provides a reliable request/response service to the Session layer in the Wireless Application Protocol (WAP) architecture. When only a single transaction is considered occurrence graph and language analysis reveals 3 inconsistencies between the protocol and service specification: (1) the initiator user can receive two TR-Invoke.cnf primitives; (2) turning User Acknowledgement on doesn't always provide the User Acknowledgement service; and (3) a transaction can be aborted without the responder user being notified. Based on the modelling and analysis, changes to WTP have been recommended to the WAP Forum^{SM}.

1 Introduction

Petri nets are a proven technique for the design and verification of communication protocols [4, 15]. They provide the ability to: model at different levels of abstraction; capture the concurrent behaviour inherent in communication protocols; and formally analyse them. This can give a high degree of confidence in the protocol specification and design, which is important given that protocols are an integral part of the telecommunications and computing infrastructure. This has become more important with the emergence of e-commerce applications.

In this paper we present an initial analysis of the Class 2 Wireless Transaction Protocol (WTP) using Coloured Petri nets. To provide context, we introduce the Wireless Application Protocol, which includes WTP as part of its architecture.

1.1 Wireless Application Protocol

The Wireless Application Protocol (WAP) [24] defines an architecture that aims to support the provision of Internet and advanced information services to mobile users via a wide range of predominantly hand-held devices. An example application of WAP is to perform Web browsing on a mobile phone.

WAP is designed to take into account the limitations of the devices and the wireless data networks they utilise. The architecture specification is defined by

the WAP Forum^{SM}, an industry consortium of wireless service providers, device manufacturers, software companies and infrastructure and content providers.

WAP is based on the World Wide Web (WWW) programming model: requests are made to a server, via a gateway, which generates an appropriate response (e.g. the content of a Web page). This allows the experience and tools used in Web applications to be carried over to WAP applications. The gateway, which is not part of the WWW programming model, is used in WAP to perform content encoding/decoding and protocol conversion between the WAP stack and the Internet's transport protocol TCP/IP. Hence the gateway would be located at a base station – it is the interface between the wireless network (WAP) and the wired network (TCP/IP).

To provide a scalable and extensible architecture, WAP is designed in layers[1] (Fig. 1). From the bottom, the transport layer operates over a wide range of wireless bearer services. These include the GSM Short Message Service and General Packet Radio Service, CDMA Circuit Switched Data, Cellular Digital Packet Data (CDPD), and several proprietary protocols. There are three other protocol layers, security, transaction and session, and an application layer, the Wireless Application Environment (WAE). WAE defines a Wireless Markup Language (WML) and an accompanying scripting language (WMLScript) that are optimised to suit the limited display sizes of the browsing devices and low wireless link capacities.

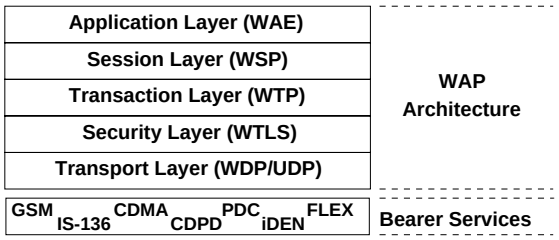


Fig. 1. WAP architecture

1.2 Previous Work

Formal methods can be applied to nearly all steps in the protocol engineering methodology – from the high level architecture design to automatic implementation and conformance testing [3]. We are interested in the analysis of the service and protocol specifications, and verification of the protocol against the service. The first phase, modelling and analysis of the Transaction service [11], included generation of the service language, i.e. the sequences of primitive events at both

[1] The specifications for each layer of the WAP architecture are available via `www.wapforum.org`. Throughout this paper we refer to WAP Version 1.1. However for WTP, Draft Version 11-June-1999 is the basis of the analysis. This has been accepted as WTP Version 1.2.

service user interfaces. In this paper we focus on analysing the Transaction protocol and verifying its conformance to the service via a comparison of the service languages.

Transport protocols, such as TCP [20] and the OSI Transport Protocol [13], have been formally analysed using various techniques (e.g. Petri nets [2, 8, 17], Estelle [6], and systems of communicating machines [16]). However, formal analysis of transaction protocols (which can be thought of as reliable transport protocols, optimised for transactions) is limited. The only other work we are aware of is the analysis of Transaction/TCP (T/TCP) [5], a protocol designed for providing an efficient transaction service as well as stream-oriented data transfer (i.e. TCP) in the Internet. T/TCP has been specified using timed and untimed automaton models [22], and demonstrates that T/TCP doesn't provide the same service as TCP. Follow on work [23] has shown the dependence of T/TCP on accurate clocks for transaction protocols to provide efficient, reliable transactions.

Although some features are similar, there are substantial differences between the service and protocols of WTP and T/TCP to warrant the formal verification of WTP. We use Coloured Petri nets (CPNs) [14] to specify and analyse the functional properties of WTP. Previous experience and tool support (i.e. Design/CPN [18]) for modelling, simulation and (functional and performance) analysis makes CPNs a suitable choice for this task. To obtain manageable occurrence graphs, the analysis is done under restricted conditions (most notably only a single transaction is analysed). Three inconsistencies between the service and protocol specifications are detected, identifying areas for improvement of the WTP design.

1.3 Overview

The remainder of this paper is organised as follows: Section 2 introduces the Transaction service and protocol. Section 3 presents the Transaction service language. Section 4 describes the design of the protocol CPN and analysis results are given and discussed in Section 5. Section 6 concludes with a summary and areas of future work.

2 Wireless Transaction Protocol

The Wireless Transaction Protocol (WTP) [25] provides 3 classes of service:

- Class 0: unreliable invoke message with no result message. This is the same datagram service as provided by the Transport layer. It is included so a datagram can be sent during a session. The User Datagram Protocol (UDP) [19] or Wireless Datagram Protocol (WDP) are recommended to be used if applications require a datagram service.
- Class 1: reliable invoke message with no result message. This can be used to provide push functionality in, for example, the Wireless Session Protocol (WSP). Within the context of an existing session the server can push data to the client, which the client then acknowledges.

– Class 2: reliable invoke message with one reliable result message. This is the basic transaction service.

This section briefly describes the Class 2 Transaction service and protocol.

2.1 WTP Service

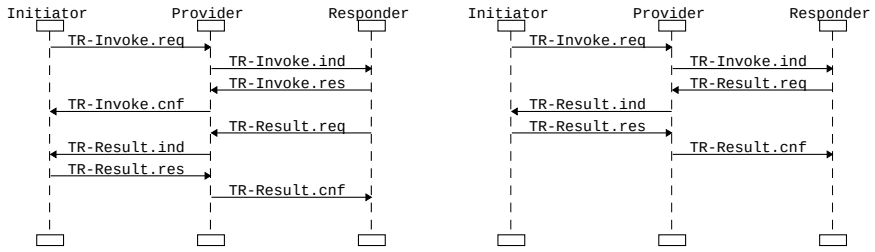
The WTP service primitives and the possible types are: TR-Invoke – req (request), ind (indication), res (response), cnf (confirm); TR-Result – req, ind, res, cnf; TR-Abort – req, ind. A transaction is started by a user issuing a TR-Invoke.req primitive. This user becomes the initiator of the transaction and the destination user becomes the responder. The responder must start with a TR-Invoke.ind. Table 1 shows the primitives that may be immediately followed by a given primitive at either end point. For example, at the initiator a TR-Invoke.req can be followed by a TR-Invoke.cnf, TR-Result.ind, TR-Abort.req or TR-Abort.ind. Further details on the service can be found in [25].

Table 1. Primitive sequences for WAP Transaction Service at each end point

	TR-Invoke				TR-Result				TR-Abort	
	req	*ind*	*res*	*cnf*	*req*	*ind*	*res*	*cnf*	*req*	*ind*
TR-Invoke.req										
TR-Invoke.ind										
TR-Invoke.res		X								
TR-Invoke.cnf	X									
TR-Result.req		X*	X							
TR-Result.ind	X*			X						
TR-Result.res						X				
TR-Result.cnf					X					
TR-Abort.req	X	X	X	X	X	X	X			
TR-Abort.ind	X	X	X	X	X	X	X			

Note: the primitive in each column may be immediately followed by the primitives marked with an X. Those marked with an X* are not possible if the User Acknowledgement option is used.

Each of the primitives has several mandatory and optional parameters. The TR-Invoke request and indication must include both source and destination addresses and port numbers. Other parameters are: User Data, Class Type, Exit Info, Handle, Ack Type and Abort Code. Of special significance is Ack Type. This parameter is used to turn on or off the User Acknowledgement (User Ack) feature. When on, an explicit acknowledgement of the invoke is necessary (i.e. TR-Invoke.res and TR-Invoke.cnf). Otherwise, the result may implicitly acknowledge the invoke. Fig. 2 gives two example sequences of primitive exchanges for a successful transaction. The sequence in Fig. 2(b) is not possible if User Ack is on.



(a) – Explicit acknowledgement (b) – Implicit acknowledgement

Fig. 2. MSC of service primitives for successful transaction

2.2 Protocol Features

Messages sent between peer protocol entities are called Protocol Data Units (PDUs) [12]. There are four primary PDUs used in the Transaction protocol: Invoke, Result, Ack and Abort. (There are 3 other PDUs available if the optional segmentation and re-assembly feature is used – this is discussed later.) Each PDU contains an integral number of octets and consists of a header, containing a fixed and variable component, and the data, if present. The relevant header fields of PDUs are given when describing the CPN model (Section 4.6).

The procedure for normal message transfer in a transaction involves 5 steps:

1. Upon receipt of a TR-Invoke.req from the user, the initiator transaction protocol entity sends an Invoke PDU to the responder. The initiator starts a retransmission timer and waits for a response.
2. Upon receipt of the Invoke PDU the responder sends a request to its user (TR-Invoke.ind) and waits for a result. The PDU includes a Transaction Identifier (TID) which is used for the remaining PDUs in the transaction. This allows several transactions to be processed concurrently. The protocol provides a mechanism for dealing with the receipt of delayed PDUs, i.e. PDUs that contain unexpected TIDs [25]. If the TID is expected, the responder can notify the user (i.e. give a TR-Invoke.ind) and proceed with the transaction. The receipt of a PDU with an unexpected TID initiates a handshake to verify if the delayed PDU should be processed or not. This involves two steps:
   (a) The responder sends an Ack PDU with a verification flag set. This asks the initiator if it has an outstanding transaction.
   (b) The initiator sends an Ack PDU with the verification flag set if the transaction is in progress, otherwise an Abort PDU is sent which indicates to the responder that the TID can be ignored.
3. While waiting for the result (from the user), the responder may send a "hold on" Ack PDU to the initiator if the responding user is taking too long to acknowledge the Invoke PDU. Then the initiator knows not to retransmit the Invoke PDU. In this case a TR-Invoke.cnf is generated at the initiator.
4. A TR-Result.req primitive from the user allows the responder to send the Result PDU. Upon receipt of the Result PDU by the initiator a TR-Invoke.cnf (if not already sent) and TR-Result.ind are passed to the user.

5. The initiator acknowledges the result by sending an Ack PDU to the responder. The initiator must either wait for a timeout before removing any transaction information or save the transaction history so it can handle a retransmission if necessary.

Reliability in WTP is provided by retransmission of PDUs until acknowledgements are received. A timer and retransmission counter are used so when the number of timeouts (and retransmissions) reaches a maximum value, the transaction is aborted. There is a maximum value for retransmitting any PDU (RCR_MAX) and for retransmitting acknowledgements (AEC_MAX). For example, assuming RCR_MAX is 1, the initiator starts a timer after sending the Invoke PDU and if no response has been received when the timer expires the PDU is retransmitted and RCR is set to 1. If again no response is received before the timeout, the initiator will abort the transaction (as RCR = RCR_MAX).

Other features of WTP include: concatenation and separation of PDUs into one service data unit, transmission of protocol parameters via transport information items (TPIs) and an optional protocol feature for segmenting and re-assembling PDUs into multiple packets.

The protocol operation is described by a set of state tables in the WTP Specification [25]. The initiator and responder each have tables representing states they can be in. Each table has an event, a condition, a set of actions and the next state the entity enters. The actions have been assumed to be ordered and sequential top to bottom. In Table 2, for example, while in the LISTEN state, if the responder receives an Invoke PDU with a valid TID and User Ack is not used (U/P==false), it will generate a TR-Invoke.ind to the user, then start the timer for waiting for a response, and set the Uack variable to false. The resulting state will be INVOKE RESP WAIT.

Table 2. State table entry for responder in LISTEN state

Responder LISTEN			
Event	**Condition**	**Action**	**Next State**
RcvInvoke	Class == 2|1	Generate TR-Invoke.ind	INVOKE RESP
	Valid TID	Start timer, A	WAIT
	U/P==False	Uack = False	

3 Transaction Service Specification

The WAP Transaction Service has been modelled and analysed using Coloured Petri nets [11]. The model allows us to locate deficiencies in the service specification and generate it's language, the possible sequences of primitives between the service user (Session layer) and the service provider (Transaction layer). For the purpose of comparison with the protocol (see Section 5.2), the Transaction Service language from [11] is presented in Fig. 3.

The Transaction Service language has 21 nodes and 74 arcs. There are four halt states in the language: nodes 6, 16, 17 and 19 (shown in bold). Node 6

represents the case when the initiator's TR-Invoke.req is immediately followed by an abort. Node 16 represents the case when the initiator has finished and the responder has also finished or aborted. Nodes 17 and 19 represent the cases when the transaction is aborted. The primitives between the following nodes are not possible when User Ack is turned on: (2,3), (2,7), (2,13), (12,14), (8,17), (3,20). In addition, the primitives that were between 2 and 13 are now between 2 and 9.

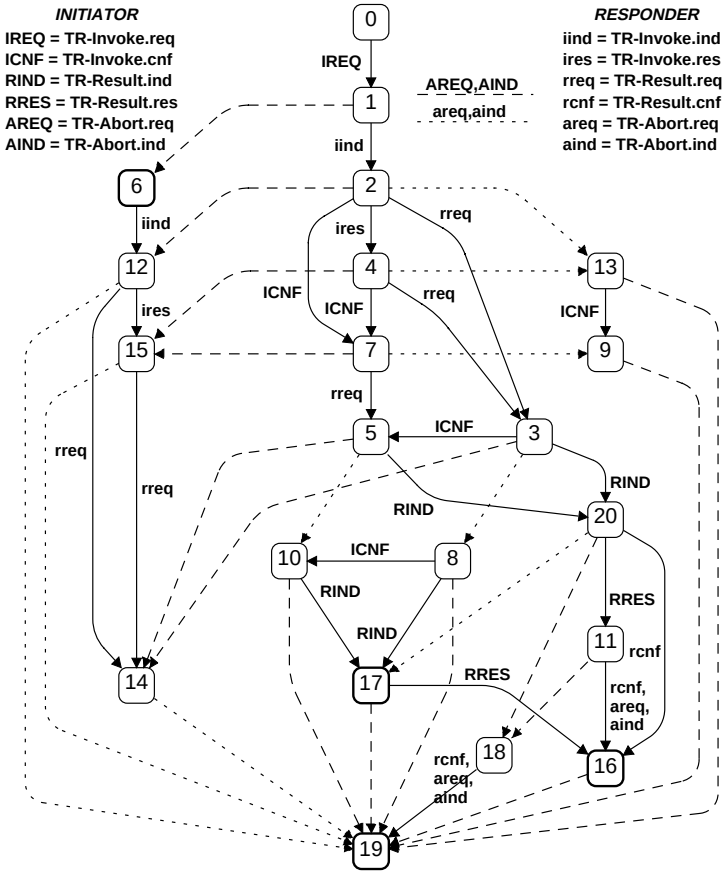


Fig. 3. Transaction Service language

4 Transaction Protocol Specification

To verify the operation of WTP, a CPN model of the protocol specification has been developed [9]. It consists of 12 pages, 55 transitions, 9 places and 21 colour

sets. The necessary components for protocol verification, as shown in Fig. 4, are modelled (some at an abstract level). This enables us to gain further insight into the protocol operation, verify general properties, and compare it with the service specification. This Section describes the design decisions and assumptions made in the modelling process, using selected parts of the CPN for explanation. A complete CPN model [9] of the protocol is too large to include in this paper. The aim is to present the main ideas behind the model and to present the analysis results.

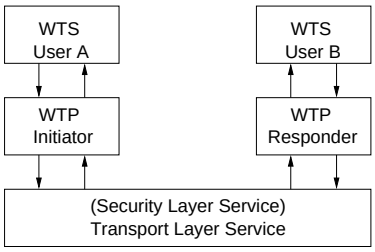


Fig. 4. Block diagram for protocol verification

4.1 Net Structure

The WTP Specification [25] defines the procedures of the protocol using state tables (as described in Section 2.2). We chose to model each table as a separate CPN page. The advantages of this approach are: it's relatively easy to transfer the table information into a CPN (see Section 4.2); the direct relationship between our CPN and WTP simplifies validation of the model by us and others; and the repetitive structure allows a manageable graphical layout of the net. A disadvantage is a lack of hierarchy in the model, and, hence, the ability to view the protocol operation from a higher level of abstraction. We feel the use of non-CPN graphics with the model (i.e. message sequences charts in Design/CPN) can compensate for this.

There are 11 top-level pages in the model as shown in Fig. 5. All pages are connected via fusion places. The first letter of the page name indicates whether it is the *I*nitiator or *R*esponder, while the remainder indicates the state the page represents. The twelfth page, I_RESULT_WAIT_RcvResult, a sub-page of I_RESULT_WAIT, is explained in Section 4.7.

4.2 Page Structure

Each CPN page is based on the following structure (see Fig. 6 for an example):

- Two fusion places, InitToResp and RespToInit, to represent the medium for transporting messages between the initiator and responder, and vice versa, respectively. The medium is further described in Section 4.5.
- A fusion place that stores the current state of a protocol entity (i.e. the initiator or responder), and the context associated with that state for each

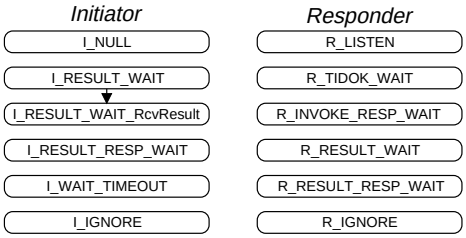


Fig. 5. Protocol CPN hierarchy page

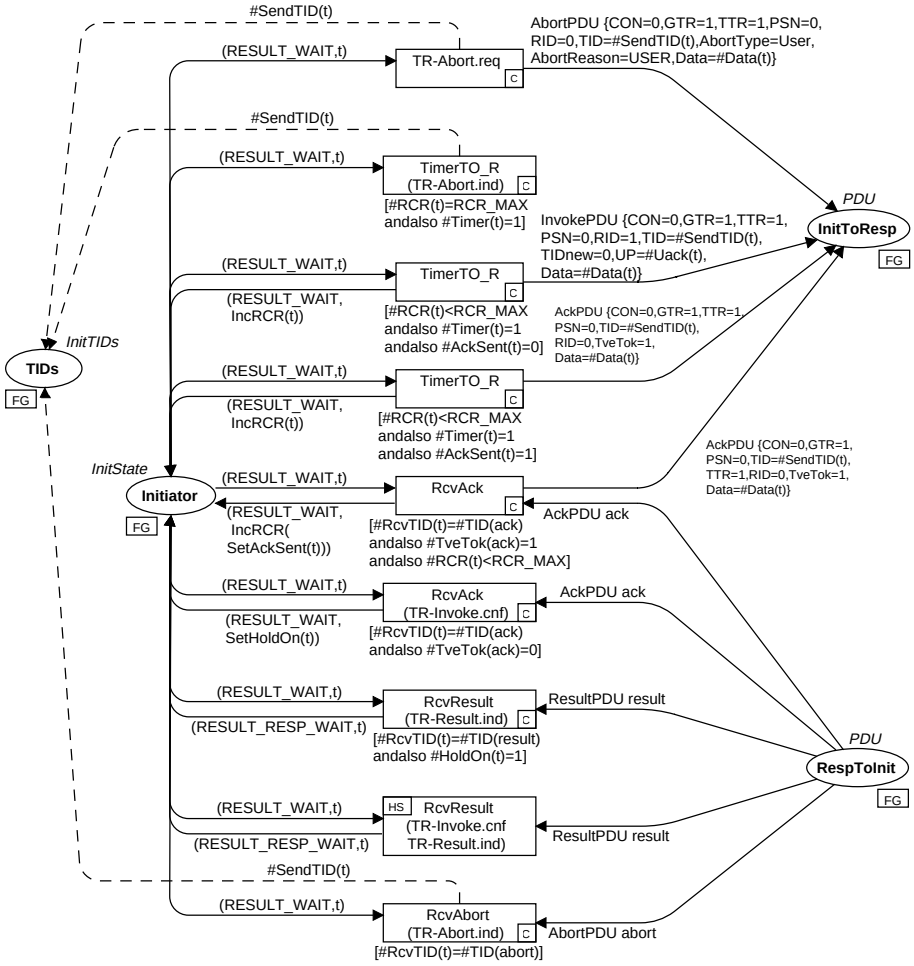


Fig. 6. Initiator in RESULT WAIT state

transaction. The initiator therefore has a place Initiator and the responder a place Responder. The representation of the state (and transaction context) is explained in Section 4.3.
- Transitions that represent events and actions from the state tables. For all top-level pages (except I_NULL – see Section 4.4), every transition has an input arc from the state place. This restricts transitions on a page to only be enabled when the entity is in the state represented by that page. Along with guards, input arcs from other places (e.g. InitToResp) can be thought of as conditions of the events. The output arcs represent an action occurring and/or state changing.
- Some pages have additional places to keep track of counter values etc. For example, in Fig. 6 place TIDs stores all transaction identifiers not in use.

The WTS users (Fig. 4) are modelled at an abstract level via the primitive events. Actions in the protocol entities that represent a submit (or deliver) of a primitive from (or to) a user are indicated by the primitives as labels on the relevant transitions. For example, in Fig. 6 there are six transitions (the top two and the bottom four) that indicate communication with User A via primitives.

4.3 State of Protocol Entities

WTP supports multiple, asynchronous transactions. In the CPN, this is modelled by storing the state and context of each transaction in progress at the initiator and the responder (in places Initiator and Responder, respectively). The transactions are differentiated by their TIDs. The state of a transaction corresponds with a state table in WTP:

```
color States = with WAIT_TIMEOUT | RESULT_WAIT | RESULT_RESP_WAIT |
     LISTEN | TIDOK_WAIT | INVOKE_RESP_WAIT;
```

The initiator has 4 states (its initial state (see Section 4.4) and the first 3 colours) and the responder has 5 states (the last 5 colours).

The context (TransData record) stores variables associated with each transaction. The variable t (in Figs. 6 and 8) is of this type.

```
color TransData = record
 SendTID:Uint16 * (* TID to send – 0..TID_MAX *)
 RcvTID:Uint16  * (* TID expected to receive – 0..TID_MAX *)
 HoldOn:Flag    * (* True if HoldOn ack received – 0/1 *)
 Uack:Flag      * (* True if User Ack requested – 0/1 *)
 AckSent:Flag   * (* True if Ack(TIDok/TIDve) sent – 0/1 *)
 RCR:RCR_c      * (* Retransmission Counter – 0..RCR_MAX *)
 AEC:AEC_c      * (* Ack Expiration Counter – 0..AEC_MAX *)
 Data:Counter   * (* Data – int *)
 Timer:Flag;      (* True if Timer on – 0/1 *)
```

The first five entries correspond to variables used by WTP at both the initiator and responder. RCR and AEC are counters used by WTP. Data is used to give each transaction a unique identifier in case of errors (see Section 4.4). As there is no time in the model, Timer has been introduced to indicate whether the timer is on (and hence a timeout can occur) or off.

We have assumed the counters can never be greater than their maximum value. This may seem obvious but an action of one entry in the Initiator RESULT WAIT state table increments RCR, but there is no condition stating RCR < RCR_MAX. We have introduced this condition, as shown in Table 3 (the change is italicised).

Table 3. State table action with new condition limiting RCR

Initiator RESULT WAIT			
Event	**Condition**	**Action**	**Next State**
RcvAck	TIDve	Send Ack(TIDok)	RESP WAIT
	Class=2|1	Increment RCR	
	RCR<RCR_MAX	Start timer, R [RCR]	

The places Initiator and Responder are typed as follows:

color InitState = **product** States * TransData;
color RespState = **product** States * TransData;

Therefore, if there were two transactions in progress for example, the marking of Initiator may be:

1‘(WAIT_TIMEOUT, {SendTID=0, RcvTID=2, ...}) +
1‘(RESULT_WAIT, {SendTID=1, RcvTID=3, ...})

4.4 Initialisation

The page I_NULL (Fig. 7) represents the state of the initiator before a transaction has begun. The state is modelled implicitly i.e. there is no colour NULL in the colour set States. Instead, the user can issue a TR-Invoke.req as many times as there are initial tokens in User. These tokens are parameters for each transaction. In Fig. 7 only 1 transaction can be initiated. Neither the continue (CON) option (used to identify TPIs) nor the User Ack are used.

Place TIDs stores all TIDs that are not outstanding at the initiator. This is necessary in the model to detect when old PDUs (i.e. those with a TID that is not outstanding) are received by the initiator. Place Data maintains an integer counter. This is used to generate the next TID (which is maintained as the variable GenTID in WTP), and also generate a unique data value for each transaction. Because the TID values wrap (i.e. in erroneous conditions, two transactions may have the same TID), the data value will be necessary to differentiate transactions when analysing the effect of errors on the protocol.

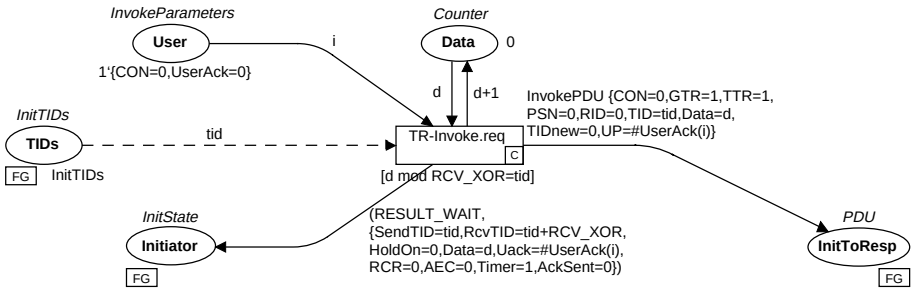


Fig. 7. Initiator in NULL state

The occurrence of TR-Invoke.req sends an InvokePDU to InitToResp and places the initiator into the RESULT_WAIT state.

Fig. 8 is the CPN page for the responder in the LISTEN state (i.e. it is waiting to receive invocations from the initiator). The three common fusion places are present (InitToResp, RespToInit and Responder), with Responder initialised to AllListen. This is a constant that says the responder can accept n asynchronous invocations, where n is double the window size at the responder. Place LastTID keeps track of the TID of the last Invoke PDU received (to store only one TID is an assumption of our current model – in practice an array of TIDs may be stored to allow Invoke PDUs to be received out-of-order). Note for the model we have set TID_MAX to 3, and for a successful receipt of an Invoke PDU, LastTID is initialised as 1. The LastTID, along with the TID of the current Invoke PDU received, is used by the windowing mechanism (implemented by TIDTest() in the guards of the two RcvInvoke transitions) to determine if Invokes are received as expected. The occurrence of the top RcvInvoke indicates the TID is expected and the transaction can proceed. The bottom RcvInvoke indicates an Invoke has been received out-of-order and a verification must occur with the initiator.

4.5 Transport Medium

From Fig. 4 the security service is provided to WTP. However, this is an optional layer, and does not alter the service (in terms of primitives) of the underlying Transport layer. Therefore, the CPN model of the transport medium must reflect the properties of the Transport layer protocols, namely WDP or UDP [19]. These datagram protocols cannot guarantee in-order delivery, removal of duplicates or loss-free delivery of messages.

Initially, the transport medium has been modelled as a single place for each direction of communication (InitToResp and RespToInit). This models the unordered characteristics of the medium. The capacity of the medium is assumed infinite. Errors in the transport medium (loss of PDUs, duplicates) have not been modelled. This is an area of future work.

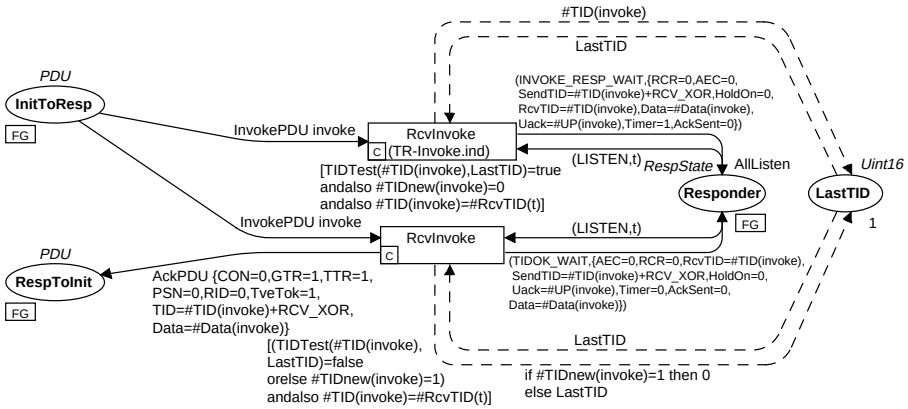


Fig. 8. Responder in LISTEN state

4.6 Protocol Data Units

PDUs are modelled as records, with entries corresponding to header fields defined in WTP. Only fields that may affect the protocol operation are included (e.g. the version field is excluded). The Invoke PDU definition is:

```
color InvokePDU_c = record
 CON:Flag       * (* Continue – 0/1 *)
 GTR:Flag       * (* Group Trailer – 0/1 *)
 TTR:Flag       * (* Transmission Trailer – 0/1 *)
 PSN:PSN_c      * (* Packet Sequence Number – 0..255 *)
 RID:Flag       * (* Retransmission Indicator – 0/1 *)
 TID:Uint16     * (* Transaction Identifier – 0..TID_MAX *)
 Data:Counter   * (* Data – int *)
 TIDnew:Flag    * (* Indicates wrapping of TID – 0/1 *)
 UP:Flag;         (* True if User Ack on – 0/1 *)
```

The first field (CON) is used in identifying TPIs and the next three are concerned with segmentation and re-assembly of PDUs. Although these protocol features (along with concatenation and separation) are not modelled, the fields are included for future use. The first 7 fields are common to all PDUs. The Result PDU has no more fields. The fields of the other PDUs are:

```
color AckPDU_c = record
 ...
 TveTok:Flag;       (* TID verification/TID Ok – 0/1 *)
color AbortPDU_c = record
 ...
 AbortType:AbortType_c  *        (* Provider/User *)
```

AbortReason:AbortReason_c; *(* UNKNOWN, PROTOERR, ... *)*

The colour set of the transport medium places is:

color PDU = **union** InvokePDU:InvokePDU_c + ResultPDU:ResultPDU_c +
AckPDU:AckPDU_c + AbortPDU:AbortPDU_c;

4.7 Correspondence to Primitives

The actions specified in the WTP state tables have been modelled as atomic events. That is, for example, the generation of TR-Invoke.ind, then starting the timer and then setting Uack to False in Table 2 is modelled as the occurrence of transition RcvInvoke (TR-Invoke.ind) in Fig. 8. This allows each primitive event to be directly associated with an individual transition, and hence an arc in the OG. This is important when performing analysis because to compare the protocol and service specifications it is necessary to select arcs in the OG that correspond to primitive events (see Section 5.2). There is one exception to this case, when an action in the initiator RESULT_WAIT state table generates two primitives (Table 4). If this was modelled as a single transition, we wouldn't be able to differentiate between the two primitives when selecting arcs in the OG. Therefore, this action is decomposed into a sub-page (denoted by HS in the transition RcvResult in Fig. 6) that contains a transition for each primitive (Fig. 9).

Table 4. State table action that generates two primitives

Initiator RESULT WAIT			
Event	**Condition**	**Action**	**Next State**
RcvResult	Class == 2	Stop timer	RESULT RESP WAIT
	HoldOn==False	Generate TR-Invoke.cnf	
		Generate TR-Result.ind	
		Start timer, A	

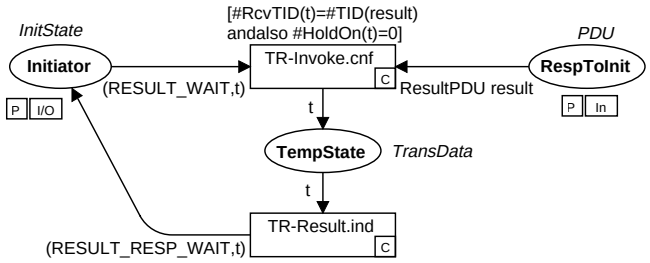


Fig. 9. Initiator in RESULT_WAIT state - sub-page for primitives

5 WTP Analysis

Occurrence graph analysis has been used to locate any undesired behaviour in the protocol specification. The OG has been reduced to a language of possible primitive events in a similar manner as the Transaction Service language [11]. A comparison of the two languages reveals errors in the protocol specification. This Section reports on these analysis results and suggests how the errors may be eliminated.

5.1 Protocol Occurrence Graph Analysis

The protocol CPN has been analysed using a single transaction. Also the maximum counter values (RCR_MAX and AEC_MAX) have been set to 1 or 2. This allows us to verify the basic operation of the protocol, as we can obtain a reasonable sized OG. The OG results with and without user acknowledgement are shown in Table 5.

Table 5. Protocol specification OG results

$OGNo.$	RCR_Max	AEC_Max	$LastTID$	$UserAck$	$Nodes$	$Arcs$	$Time(s)$	TS	DTI
1	1	1	1	Off	1634	6472	34	1	10
2	1	1	1	On	2321	9454	56	1	8
3	1	2	1	Off	1634	6472	33	1	10
4	1	2	1	On	3350	14255	104	1	4
5	2	1	1	On	31290	165257	5950	1	4
6	2	2	1	Off	19083	96356	5680	1	6
7	2	2	1	On	50491	278591	17156	1	4
8	1	1	0	On	542	2028	7	1	8

Note: TS = Terminal States, DTI = Dead Transition Instances.

The protocol terminated correctly in all cases (i.e. the single terminal state was desired). There were dead transition instances present, but they were expected. They were caused by features being modelled but not exercised, due to initial conditions. The transitions were related to the following features: User Ack; TID verification; and receiving incorrect PDUs. When User Ack is off, AEC is not used. Hence, as they only differ by the maximum AEC counter, OG number 1 and OG number 3 are identical.

OG No. 8 is the result of analysing the case of TID verification occurring. With the receipt of an Invoke PDU with TID=0 the initial value of $LastTID$ (0) forces a verification. The OG is smaller than the other cases (i.e. OG No. 2) because the receipt of the Ack PDU for verification by the initiator, implicitly acknowledges the Invoke PDU, disallowing further re-transmissions (note that essentially this means sequences leading to two TR-Invoke.cnf primitives, as discussed in the following Section, are not possible).

It should be noted that using a single transaction does not test several important features of the protocol (e.g. the windowing mechanism with sequence numbers). Analysing multiple transactions is an area for future work.

5.2 Comparison of Service and Protocol Languages

To generate the protocol language the OG is treated as a finite state automata (FSA). The sequence of primitives is of interest, so all binding elements that do not correspond to primitive events are treated as empty transitions in the FSA. A common FSA reduction technique [1] is applied to obtain the minimised deterministic FSA, or the Transaction Protocol language.

The languages have been obtained for OG No. 1, 2, 3, 4 and 8. The other OGs are too large to minimise using our current tools [7, 21]. In the following we concentrate on the languages obtained from OG No. 1 and 2 (User Ack off and on, respectively). The statistics of these languages (and for comparative purposes, the service languages from [11]) are given in Table 6.

Table 6. Transaction service and protocol FSA results

	UserAck	*Nodes*	*Arcs*	*Halts*	*Sequences*	*Longest*	*Shortest*
Service	Off	21	74	4	450	9	2
Protocol	Off	39	119	5	527	10	2
Service	On	21	69	4	194	9	2
Protocol	On	45	133	7	334	10	2

There are sequences possible in the protocol language, but not defined in the service. These identify errors in the protocol specification. For both User Ack cases, an error occurs when a second TR-Invoke.cnf can be generated at the initiator. More specifically, in relation to Fig. 3, TR-Invoke.cnf primitives can follow nodes 5, 7, 9 and 10. By tracing the sequences back to the OG, we have identified all possible sequences of protocol actions that result in this error (there are 174). A scenario showing the error is given in Fig. 10.

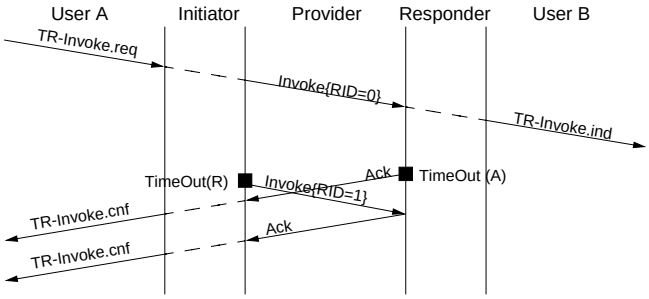


Fig. 10. Error scenario: Two TR-Invoke.cnf primitives

Closer investigation of some of the error sequences in the protocol CPN indicates the retransmission of an Ack PDU by the responder when it receives

a retransmitted Invoke PDU while in the RESULT_WAIT state results in the second TR-Invoke.cnf primitive. The Ack PDU is retransmitted by the responder to cope with the situation when the first Ack PDU is lost. In that case the first TR-Invoke.cnf would not be generated, and everything would proceed correctly. Therefore to remedy the error, on receipt of the second Ack PDU (i.e. after a TR-Invoke.cnf has been issued) WTP should not re-issue a TR-Invoke.cnf and can discard (and log) the Ack PDU.

When User Ack is used there are two other inconsistencies with the service specification:

1. A TR-Result.req may immediately follow a TR-Invoke.ind at the responder. That is, the arcs between nodes 2 and 3, and 12 and 14 are present in the protocol language however, as stated in Section 3 and easily seen in Table 1, they are not present in the service language (Fig. 3) when User Ack is on. By restricting a TR-Result.req to only be issued when User Ack is off, as shown in Table 7, the error can be removed. With this change made to the protocol CPN, the language obtained from the OG disallows the extra TR-Result.req primitives, as required.

Table 7. State table entry with new condition restricting TR-Result.req

Responder INVOKE RESP WAIT			
Event	**Condition**	**Action**	**Next State**
TR-Result.req	*Uack==False*	Reset RCR Start timer R[RCR] Send Result PDU	RESULT RESP WAIT

2. There are inconsistent halt states in the protocol language. In relation to Fig. 3, node 12 would be a halt state. For example, Fig. 11 shows a sequence that constitutes a transaction. After the number of timeouts at the responder reaches the maximum, the transaction is aborted. The retransmitted Invoke PDU initiates a TID verification which fails. At the end of the sequence the initiator is in the NULL state and the responder in the LISTEN state, indicating that both entities have discarded any state information for that transaction. However, the responder user has been issued a TR-Invoke.ind but no other primitive to indicate the end of the transaction. A solution is to issue a TR-Abort.ind primitive to the responder user when a transaction is aborted due to a timeout (and the responder goes back to the LISTEN state).

6 Conclusions

Initial Coloured Petri net analysis of the WAP Class 2 Wireless Transaction Protocol has revealed discrepancies between the service specification and the

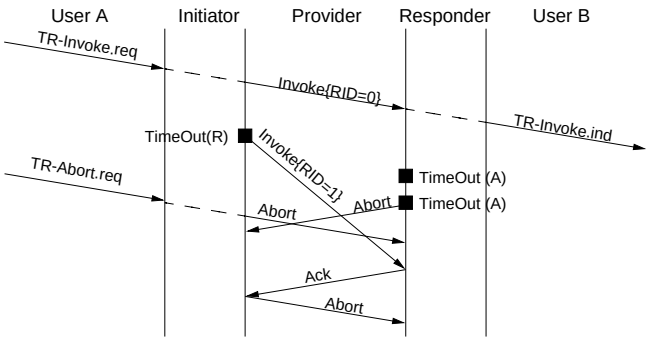


Fig. 11. Error scenario: Invalid halt state

protocol specification. This can be used for improving the specification and design of WTP.

WTP provides three classes of service, the most commonly used (Class 2) being a reliable request/response service. The service specification has been modelled using Coloured Petri nets and the Transaction Service language presented in [11]. The focus of this paper has been outlining the protocol specification CPN model and presenting the initial analysis results. In summary, we have:

- Presented a CPN model of the protocol, which was developed based on the WTP state tables. The model was developed to be able to analyse protocol features using a modular approach. Not all features are currently modelled.
- Using occurrence graph analysis, shown that, when a single transaction is possible and under restricted conditions on the counter limits, no deadlocks occur in the protocol and the dead transitions are as expected.
- Generated the protocol language, and compared it to the Transaction Service language. This identified the following discrepancies between the two:
  1. Two TR-Invoke.cnf primitives could be generated at the initiator.
  2. The User Ack service is not always provided when user acknowledgement is on (i.e. a TR-Request.req can follow a TR-Invoke.ind).
  3. When User Ack is on a transaction can finish without the responder user being notified.
- Proposed the following changes to WTP:
  1. For all actions when the initiator increments RCR, a condition should restrict RCR $<$ RCR_MAX. The specific change to the state tables is given in Table 3.
  2. To disallow two TR-Invoke.cnf primitives, the initiator should discard a second Ack PDU after already issuing a TR-Invoke.cnf, and not issue the second primitive.
  3. Introduce a condition on the responder user issuing a TR-Result.req, as given in Table 7, to disallow the primitive following a TR-Invoke.ind when User Ack is on.

4. Generate a TR-Abort.ind when the the Responder aborts due to a time-out in the INVOKE_RESP_WAIT state so the user is notified of the end of the transaction.

These changes have been submitted to the WAP Forum [10].

Further analysis is required to be confident of WTPs correctness and performance. Firstly, the operation of the protocol under more complex conditions (multiple transactions, errors) needs to be verified. This work is in progress, and it is likely other analysis techniques (e.g. state space reduction) will be required. This analysis will also allow us to further evaluate the implications of the errors found and their proposed solutions. Finally, it will be necessary to analyse the performance of the protocol, either using the existing CPNs, or some other tools and techniques (e.g. OPNET).

Acknowledgements

This work was carried out with financial support from the Commonwealth of Australia through the Cooperative Research Centres Program.

References

[1] W. A. Barret and J. D. Couch. *Compiler Construction: Theory and Practice.* Science Research Associates, 1979.

[2] M. Y. Bearman, M. C. Wilbur-Ham, and J. Billington. Specification and analysis of the OSI Class 0 Transport Protocol. In J. M. Bennet and P. Pearcey, editors, *Proc. of the 7th Int. Conf. on Computer Communications*, pages 602–607, Sydney, Australia, 30 Oct. - 2 Nov. 1984. North Holland Publishers.

[3] J. Billington. Formal specification of protocols: Protocol engineering. In *Encyclopedia of Microcomputers*, pages 299–314. Marcel Dekker, New York, NY, 1991.

[4] J. Billington, M. Diaz, and G. Rozenberg, editors. *Application of Petri Nets to Communication Networks: Advances in Petri Nets.* LNCS 1605. Springer-Verlag, Berlin, 1999.

[5] R. Braden. T/TCP - TCP extensions for transactions functional specification. IETF RFC 1644 URL: ftp://ftp.isi.edu/in-notes/rfc1644.txt, July 1994.

[6] S. Budkowski, B. Alkechi, M. L. Benalycherif, P. Dembinski, M. Gardie, E. Lallet, J. P. Mouchel La Fusse, and Y. Soussi. Formal specification, validation and performance evaluation of the Xpress Transfer Protocol. In A. Danthine, G. Leduc, and P. Wolper, editors, *Protocol Specification, Testing and Verification, XIII*, pages 191–206, Liege, Belgium, May 1993.

[7] B. Cheong and S. Gordon. *Automata Reduction Tool (ART): System Manual.* Institute for Telecommunications Research, University of South Australia, 25 February 1999.

[8] J. C. A. de Figueiredo and L. M. Kristensen. Using Coloured Petri nets to investigate behavioural and performance issues of TCP protocols. In K. Jensen, editor, *Proc. of the 2nd Workshop on the Practical Use of Coloured Petri Nets and Design/CPN*, pages 21–40, Aarhus, Denmark, 13-15 October 1999. Department of Computer Science, Aarhus University. PB-541.

[9] S. Gordon and J. Billington. Modelling and analysis of the WAP class 2 Wireless Transaction Protocol using Coloured Petri nets. Draft technical report, Institute for Telecommunications Research, University of South Australia, Adelaide, Australia, November 1999.

[10] S. Gordon and J. Billington. WAP Forum Input Document: Inconsistencies in the Wireless Transaction Protocol. Submitted to WAP Forum 19 November 1999.

[11] S. Gordon and J. Billington. Modelling the WAP Transaction Service using Coloured Petri nets. In H.-V. Leong, W.-C. Lee, B. Li, and L. Yin, editors, *Proc. of the 1st Int. Conf. on Mobile Data Access*, LNCS 1748, pages 109–118, Hong Kong, 16-17 December 1999. Springer-Verlag.

[12] ITU. Information Technology–OSI–Basic reference model: The basic model. ITU-T Recommendation X.200 | ISO/IEC 7498, July 1994.

[13] ITU. Information Technology–OSI–Protocol for providing the connection-mode transport service. ITU-T Recommendation X.224 | ISO/IEC 8073, November 1995.

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

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

[16] G. M. Lundy and R. C. McArthur. Formal model of a high speed transport protocol. In R. J. Linn and M. U. Uyar, editors, *Protocol Specification, Testing and Verification, XII*, pages 97–111, Lake Buena Vista, FL, June 1992.

[17] H. Mehrpour and A. E. Karbowiak. Modelling and analysis of DOD TCP/IP protocol using numerical Petri nets. In *Proc. of IEEE Region 10 Conf. on Computer Communication Systems*, pages 617–622, Hong Kong, September 1990.

[18] Meta Software Corporation. *Design/CPN Reference Manual for X-Windows, Version 2.0*. Meta Software Corporation, Cambridge, MA, 1993.

[19] J. Postel. User Datagram Protocol. IETF RFC 768 URL: ftp://ftp.isi.edu/in-notes/rfc768.txt, August 1980.

[20] J. Postel. Transmission Control Protocol. DARPA Internet program protocol specification. Information Sciences Institute, University of Southern California, Marina del Ray, CA, September 1981. IETF RFC 793 URL: ftp://ftp.isi.edu/in-notes/rfc793.txt.

[21] D. Raymond and D. Wood. *User's Guide to Grail*. Dept. of Computer Science, University of Western Ontario, London, Canada, March 1996. Version 2.5.

[22] M. A. Smith. Formal verification of communication protocols. In R. Gotzhein and J. Bredereke, editors, *Formal Description Techniques IX: Theory, Applications, and Tools*, pages 129–144. Chapman & Hall, London, UK, October 1996.

[23] M. A. Smith. Reliable message delivery and conditionally-fast transactions are not possible without accurate clocks. In *Proc. of the 17th Annual ACM Symposium on Principles of Distributed Computing*, Puerto Vallarto, Mexico, June 1998.

[24] WAP Forum. Wireless Application Protocol Architecture Specification. Available via http://www.wapforum.org/, 30 April 1998.

[25] WAP Forum. Wireless Application Protocol Wireless Transaction Protocol Specification. Available via http://www.wapforum.org/, 11 June 1999.