Modelling the WAP Transaction Service 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. The Wireless Application Protocol (WAP) is an architecture designed to support the provision of wireless Internet services to mobile users with hand-held devices. The Wireless Transaction Protocol is a layer of WAP that provides a reliable request/response service suited for Web applications. In this paper Coloured Petri nets are used to model and generate the possible primitive sequences of the request/response Transaction Service. From the results we conclude that the service specification lacks an adequate description of what constitutes the end of a transaction. No other deficiencies were found in the Transaction Service.

1 Introduction

As wireless technologies advance, efficient access to Internet and advanced information services is becoming an important requirement from the perspective of mobile users. Currently, characteristics of wireless networks (e.g. low and varying bandwidths, drop-outs) and terminal devices (e.g. low power requirements, small displays, various input devices) limit the quality of these services for the mobile user. Therefore there is a need for existing protocols designed for use in the fixed network to be refined, and when necessary new protocols created that alleviate some of these limitations. The Wireless Application Protocol (WAP) [5] defines a set of protocols that aim to do this. In particular, the Wireless Transaction Protocol [6] provides a request/response service that is suited to using Web applications from hand-held devices such as mobile phones.

As with any new communication protocol, it is important to ensure the correctness of the Wireless Transaction Protocol. In this paper, Coloured Petri nets (CPNs) [8] are used to model the WAP Class 2 Transaction Service and generate it's language, the possible sequences of events between the users of the service and the service provider [2]. This can help identify any deficiencies in the current service specification. It is also the first step in the verification of the Wireless Transaction Protocol design. The use of formal methods is important because ensuring the correctness of a complex protocol is seldom possible via other design approaches. High-level Petri nets are a suitable formal method for the design of communication protocols because of their ability to express concurrency, non-determinism and system concepts at different levels of abstraction.

They have been used to analyse various protocols [3]. CPNs are a popular form of high-level Petri nets that have extensive tool support [4, 9] for the design of systems, including protocols.

2 Wireless Transaction Protocol

The Wireless Application Protocol (WAP) architecture comprises 5 layers: transport, security, transaction, session and application. The Wireless Transaction Protocol (WTP) [6] provides 3 classes of service to the session layer: Class 0 – unreliable invoke message with no result message; Class 1 – reliable invoke message with no result message; and Class 2 – reliable invoke message with one reliable result message (this is the basic transaction service). This section describes the Class 2 Transaction Service in more detail.

Layer-to-layer communication is defined using a set of service primitives [7]. For the Transaction Service, the primitives occur between the WTP user and the WTP service provider . The sequences of primitives describe how WTP provides the Transaction 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; and 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 given primitives at the initiator and responder interfaces. 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. There is no information given regarding the global behaviour of the service in the WAP specification [6].

Table 1. Primitive sequences for WAP Transaction Service

	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 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 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.

3 Coloured Petri Nets

CPNs are a class of high-level nets that extend the features of basic Petri nets. The net consists of two types of nodes, *places* (ellipses) and *transitions* (rectangles), and directional arcs between nodes. An input arc goes from a place to a transition and an output arc vice versa. Places are typed by a *colour* set. For example, in Fig. 1 place Initiator has the colour set State. Places may be marked by a value from the colour set. These are known as *tokens*. The collection of tokens on a place is called it's *marking*, and the marking of the CPN comprises the markings of all places. Transitions and arcs can also have inscriptions which are expressions that, along with the tokens in places, determine whether a transition is *enabled*. A transition is enabled if sufficient tokens exist in each of its input places (as determined by the input arc inscriptions), and the transition inscription, or *guard*, (given in square brackets) evaluates to true.

In Fig. 1, TR-Invoke.req is enabled because NULL (the initial marking given to the right of the place) is in the only input place and it is also the arc inscription, and there is no guard shown for the transition (which implies the guard is always true). A subset of the enabled transitions can *occur*. The occurrence of a transition destroys the necessary tokens in the input places and creates new tokens in the output places, as given by the expressions on the arcs. The occurrence of TR-Invoke.req replaces NULL with INVOKE_WAIT in Initiator and creates Invoke in place InitToResp. When variables are used in arc inscriptions or guards, the values they are bound to on occurrence of a transition give, together with the transition name, a *binding element*.

The CPNs in this paper were edited, simulated, and partly analysed using Design/CPN [4]. Design/CPN allows the CPN to be drawn on separate pages to increase the readability of the net. One technique used to combine the different pages is known as *fusion places* which are copies of a place. Design/CPN may be used to interactively or automatically simulate the net, or to create an *occurrence graph*. An occurrence graph (OG) is a graph with nodes and arcs representing net markings and binding elements, respectively. A complete OG represents all possible states the CPN can reach. In Design/CPN, queries can be made on the OG to determine dynamic properties of the CPN (e.g. deadlocks, live-lock, bounds on places). An OG can also be viewed as a finite state automata (FSA) which, with appropriate analysis techniques can be used to give the language accepted by the CPN (where the binding elements are the alphabet), which in our case is the Transaction Service language.

4 Transaction Service CPN

4.1 Modelling Assumptions

The aim of modelling the Transaction Service is to generate the service language. That is, the possible sequences of service primitives between the user and provider are of major interest. With this in mind, several assumptions can be made to simplify the model.

The primitive parameters have no effect on the sequences of primitives. Therefore each primitive is modelled as a message (e.g. Invoke) which represents the primitive type and its parameters (e.g. Invoke{SrcAdr, DestAdr, ... }).

Only the general case when User Acknowledgement is off is modelled. As the possible primitives when User Ack is on is a subset of this general case (i.e. two of the primitive sequences are not allowed), the language will be a subset of the language generated from the model. This is straightforward to obtain.

The channel between initiator and responder can be separated into two directions of flow. The channel does not guarantee ordering of messages, hence each direction can be modelled as a single place. The modelling of the channel has also been done with the analysis techniques in mind. The OG calculated can be viewed as a FSA, which in turn can be minimised using a standard reduction technique [1]. Knowing this, it is possible to allow the reduction technique to handle functionality that would otherwise be necessary in the model. This is explained in detail in Sect. 5.1 after the model is presented.

4.2 CPN Model

The CPN model of the WAP Transaction Service has four separate pages (Fig. 1 to 4), representing an invocation, result, user abort and provider abort. Each page has the same structure:

- Two fusion places called Initiator and Responder representing the states of the initiator and responder, respectively. These places are typed by the colour set State:
  color State = **with** NULL | INVOKE_WAIT | INVOKE_READY | WAIT_USER | RESULT_WAIT | RESULT_READY | FINISHED | ABORTED;
- Two fusion places called InitToResp and RespToInit representing the communication channels from initiator to responder, and from responder to initiator, respectively. These places are typed by the colour set Message:
  color Message = **with** Invoke | Ack | Result | Abort;
- Transitions that represent the sending and receiving of the different primitive types by the user. Note that each transition has a boxed 'C' underneath. This indicates a *code segment* is used. Code segments are CPN ML code that are executed by Design/CPN when the associated transition occurs [10]. This allows, for example, auxiliary graphics to be drawn as the net is simulated. For the Transaction Service CPN code segments are used to draw message sequence charts (MSC) (e.g. Fig. 5(a)).

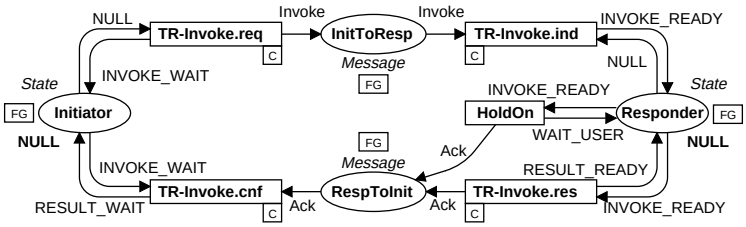


Fig. 1. TR-Invoke primitive sequence CPN

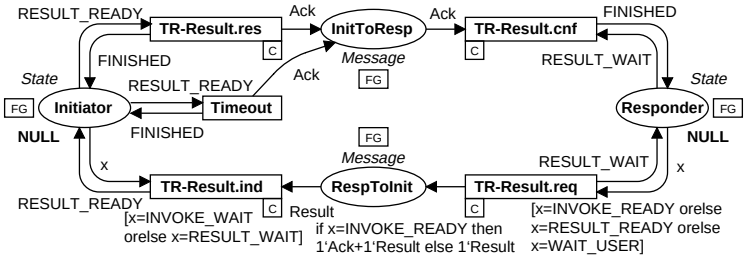


Fig. 2. TR-Result primitive sequence CPN

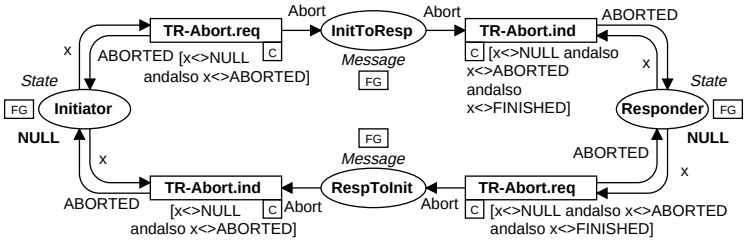


Fig. 3. User TR-Abort primitive sequence CPN

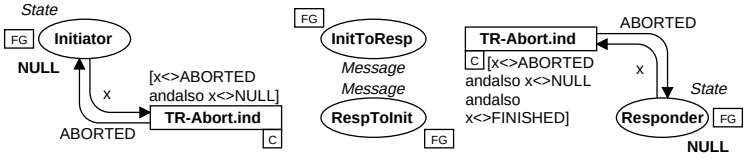
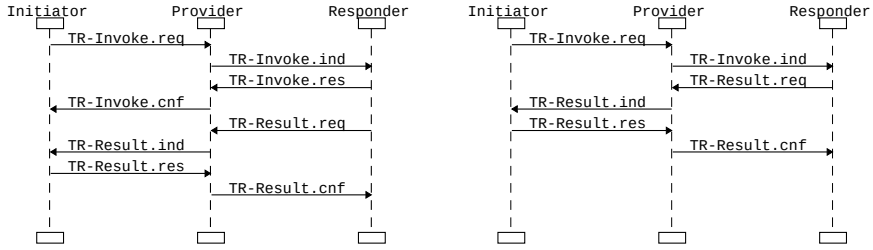


Fig. 4. Provider TR-Abort primitive sequence CPN

In addition, the Invoke and Result pages have transitions (HoldOn and Timeout, respectively) that indicate an interaction by the service provider which is not seen by the user (and therefore no primitive occurs). The occurrence of HoldOn, for example, indicates a timer has expired at the responder because the user is taking too long to generate the result. A message is sent to the initiator so that it will hold on until the responder user has generated the result. Finally, there is also a variable that can take any value from State:

var x:State;

Fig. 1 models the sequence of TR-Invoke primitives. The initial marking of both Initiator and Responder is NULL. In this marking the first and only transition that can occur is TR-Invoke.req. It follows that a possible transition occurrence sequence is the four TR-Invoke primitive types in order (i.e. request, indication, response then confirm). This would put the initiator into state RESULT_WAIT and the responder into RESULT_READY. From the TR-Result page (Fig. 2), again the TR-Result primitive transitions could occur in order. Both the initiator and responder would be in the FINISHED state. This sequence represents a successful transaction with explicit acknowledgement. The message sequence chart is shown in Fig. 5(a).



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

Fig. 5. MSC of service primitives for successful transaction

The Transaction Service does not require explicit acknowledgement of the TR-Invoke.req primitive by the responder (recall User Acknowledgement is assumed to be off). Instead by sending a TR-Result.req after receiving a TR-Invoke.ind, the responder can implicitly acknowledge the invocation. In Fig. 1 when the initiator is in the INVOKE_WAIT state and responder in INVOKE_READY both TR-Invoke.res and TR-Result.req are enabled. If TR-Result.req occurs (x is bound to INVOKE_READY) the Result message is sent to, and can be acknowledged by the initiator. The MSC for this sequence is shown in Fig. 5(b).

The previous two sequences of transitions were examples of successful transactions. However, as shown in Table 1, a TR-Abort.req or TR-Abort.ind from the initiator or responder can follow any primitive except themselves and, in the case of the responder, TR-Result.cnf (because the responder has successfully completed the transaction). Transaction aborts are modelled on two separate

pages: one for user initiated abort (Fig. 3) and the other for provider initiated abort (Fig. 4). The aborts are symmetric – they can come from either initiator or responder. A separate page is used for the provider abort because the TR-Abort.ind does not require a message from either user. The channel places are shown to be consistent with the other pages – they are not necessary.

5 Analysis

The OG generated by Design/CPN for the Transaction Service CPN contains 85 nodes and 206 arcs. There were 28 different terminal states. The graph can be viewed as a FSA with the binding elements (essentially the service primitives) as the input language. Using a standard reduction technique [1], the minimised FSA gives a compact description of the possible sequences of primitives, or service language. This section explains how the minimisation of the OG can remove complexity from the model and presents the Transaction Service language.

5.1 Analysis Assumptions

The design of the model took into account the analysis techniques that would be applied (i.e. the FSA minimisation). In particular, it was expected that multiple terminal markings would be generated that were only differentiated by the markings of the places connecting the initiator to responder (InitToResp and RespToInit). Extra transitions could have been used in the model to remove all tokens from the these places once the initiator and responder had FINISHED or ABORTED. This would ensure only a single terminal marking was generated for these cases. However it was decided to let the FSA minimisation handle the extra terminal markings (it effectively merges all terminal markings into one) so the models could remain free of any "cleanup" transitions.

By treating the OG as a FSA, the introduction of halt states (states that indicate a possible end of a primitive sequence) was also possible. A halt state may or may not lead to other states. As well as all terminal markings being halt states, nodes were defined as halt states if they satisfied either of the following conditions:

- The marking of Initiator is ABORTED and the marking of InitToResp is Invoke. This represents the special case when a TR-Invoke.req is followed by a TR-Abort.req (by the user) or TR-Abort.ind (by the provider). This is a feasible halt state because the provider may not be able to notify the responder due to, for example, network failure. In this case the sequence of primitives is complete.
- The marking of Initiator is FINISHED and the marking of Responder is FINISHED or ABORTED. Although there is no mention of this in the service specification [6], we have assumed that when the initiator has acknowledged the result by issuing a TR-Result.res primitive (and the responder has finished or aborted) the transaction may be complete. However, aborts at the initiator are still possible.

5.2 Transaction Service Language

Fig. 6 shows the Transaction Service language obtained from the minimisation of the OG. There are 21 nodes and 74 arcs. For clarity, abbreviations of the service primitives are given for the arc labels. The first letter of each label represents the service primitive (*I*nvoke, *R*esult, *A*bort). The following three letters represent the primitive type (*req*uest, *ind*ication, *res*ponse, *conf*irm). In addition the initiator primitives are given in uppercase and the responder primitives in lowercase. Multiple arcs between two nodes are drawn as one with labels separated by commas. The combinations of TR-Abort.req and TR-Abort.ind from the initiator and responder are drawn as dashed and dotted lines, respectively.

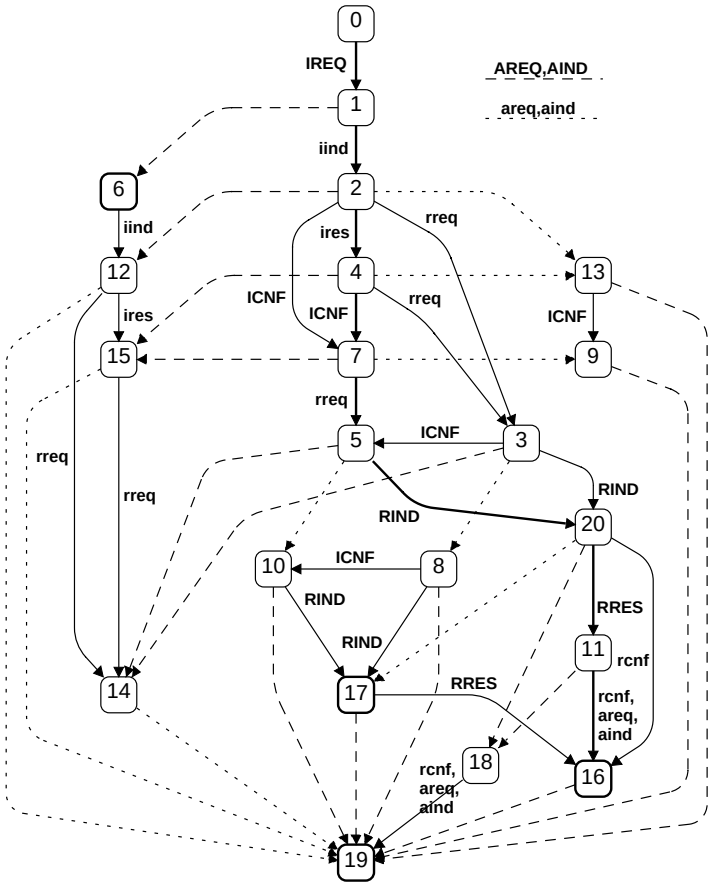


Fig. 6. Transaction Service language

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 im-

mediately 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.

Further analysis reveals there are 450 possible sequences of primitives. The shortest sequences are 2 primitives (TR-Invoke.req followed by TR-Abort.req or TR-Abort.ind at the initiator) and the longest sequences are 9 primitives (e.g. a successful transaction (shown as bold arcs – this corresponds to the sequence shown in Fig. 5(a)) followed by a TR-Abort.req from the initiator).

The primitives between the following nodes are not possible when User Acknowledgement 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.

From the service language it is unclear why the initiator would issue a TR-Abort.req (e.g. node 16 to node 19 in Fig. 6) after it had acknowledged the result with a TR-Result.res primitive (e.g. node 20 to node 11). However, an examination of the protocol specification provides an explanation. Transaction information is saved after the initiator has sent the last acknowledgement in case retransmissions are necessary. By issuing a TR-Abort.req after a TR-Result.res, the transaction state information is released. Otherwise, a timeout will release the information.

6 Conclusions

We have described, modelled and analysed the WAP Class 2 Transaction Service in a first step towards verifying the Wireless Transaction Protocol. WTP utilises the datagram service (Transport layer) in the WAP architecture and provides a reliable request/response service to the upper layers.

Coloured Petri nets were used to model the Transaction Service and generate the occurrence graph. The knowledge of the analysis techniques used allowed several assumptions to be made that simplified the model. Halt states were introduced and the OG was treated as a finite state automata and reduced to obtain the service language.

The Transaction Service language generated provides a complete set of service primitive sequences, when taking both ends of the transaction into account (i.e. both initiator and responder). This global behaviour is not described in the WAP specification. From the modelling and analysis, two questions not fully answered in the service specification arose:

1. What constitutes the end of a transaction?
2. Why is a TR-Abort.req primitive possible after a TR-Result.res from the initiator user?

Answers were obtained from examining the operation of the protocol in more detail. A transaction may be considered complete if either:

1. both initiator and responder have aborted,

2. a TR-Invoke.req at the initiator is followed immediately by an abort, and the provider hasn't notified the responder (e.g. due to network failure),
3. the initiator has acknowledged the result and the responder has either received the acknowledgement or aborted.

In the final case, it is still possible for the initiator to issue a TR-Abort.req to clear transaction state information. The need to understand the protocol operation is a shortcoming of the specification – the service should be described independently of the protocol. No other deficiencies have been found in the service specification.

The Transaction Service language can be used as a basis for verifying that the Wireless Transaction Protocol conforms to the service specification. The next step to achieve this is to model the operation of the protocol in detail. This work is in progress. An incremental approach is being used so different features can be modelled and analysed. The desired results are to generate an OG from which properties of the protocol can be derived (e.g. presence of deadlocks). Then the protocol language can be generated and compared to the service language.

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] J. Billington. Abstract specification of the ISO Transport service definition using labelled Numerical Petri nets. In H. Rudin and C. H. West, editors, *Protocol Specification, Testing, and Verification, III*, pages 173–185. Elsevier Science Publishers, Amsterdam, New York, Oxford, 1983.

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

[4] Meta Software. *Design/CPN Reference Manual, Version 2.0.* 1993.

[5] WAP Forum. Wireless application protocol architecture specification. Available via: http://www.wapforum.org/, Apr. 1998.

[6] WAP Forum. Wireless application protocol wireless transaction protocol specification. Available via: http://www.wapforum.org/, Apr. 1998.

[7] ISO/IEC. *Information Technology - Open Systems Interconnection - Basic Reference Model - Conventions for the Definition of OSI Services.* 10731. 1994.

[8] K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use*, Vol. 1-3. Springer-Verlag, Berlin, 1997.

[9] K. Jensen, S. Christensen, and L. M. Kristensen. *Design/CPN Occurrence Graph Manual, Version 3.0.* Department of Computer Science, Aarhus University, Aarhus, Denmark, 1996.

[10] L. M. Kristensen, S. Christensen, and K. Jensen. The practitioner's guide to Coloured Petri nets. *Int J Software Tools for Technology Transfer*, 2(2):98–132, 1998.