World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:11, No:10, 2017

Modeling and Analyzing the WAP Class 2 Wireless Transaction Protocol Using Event-B

Rajaa Filali, Mohamed Bouhdadi

***Abstract*—**This paper presents an incremental formal development of the Wireless Transaction Protocol (WTP) in Event-B. WTP is part of the Wireless Application Protocol (WAP) architectures and provides a reliable request-response service. To model and verify the protocol, we use the formal technique Event-B which provides an accessible and rigorous development method. This interaction between modelling and proving reduces the complexity and helps to eliminate misunderstandings, inconsistencies, and specification gaps. As result, verification of WTP allows us to find some deficiencies in the current specification.

***Keywords*—**Event-B, wireless transaction protocol, refinement, proof obligation, Rodin, ProB.

I. Introduction

THE WTP is one of the protocols defined by the WAP Forum [1]. It is a layer of the WAP that provides a reliable request/response service suited for Web applications from hand-held devices such as mobile phones.

To ensure the correctness of the WTP, many of formal methods have been applied such as Petri nets [2] and SPIN [3]. In this paper, we use Event-B method [4], [5] to model and verify WTP, focusing on the Class 2 Transaction service and protocol.

Event-B is a formal modeling method for developing systems via step-wise refinement [6], based on first-order logic [7], the modeling process starts with an abstraction of the system and then during refinement levels, features of the system are modeled, and the goals are achieved in a detailed way. The event-B is one of the methods used early to prove communication protocols [8], [9].

The use of Event-B method allows us to prove properties of the protocol, and these properties can be automatically (or interactively) proved through proof obligations [10] generated from Rodin platform [11] and its plug-in ProB [12], such as deadlock freeness, order of exchanged messages, and some business requirements.

The contributions of this paper are divided into three different areas: create the model, identify functional and non-functional properties, verification of modeled properties.

From the results of our modeling and verification of WTP, we identify some deficiencies in the current specification: (1) the initiator can abort the transaction without the responder user being notified; (2) when the responder receives the last acknowledgement, the transaction must not be aborted; and (3) when the timer expires (RCR=RCR_MAX), both initiator and responder must abort the transaction.

The reminder of the paper is organized as follows. Section II gives a brief overview of Event-B. Section III provides the requirements of the WTP protocol which are informally defined. In Section IV, the formal development is presented using Event-B. Finally, a conclusion is presented to summarize the main outcomes of this research.

R. Filali and M. Bouhdadi are with the LMPHE laboratory University of Mohammed V, Faculty of sciences 4 Street Ibn Batouta, PB 1014 RP, Rabat Morocco (e-mail: rajaafilali@ gmail.com, bouhdadi@ gmail.com).

II. Event-B Modeling Approach

Event-B is a formal method for system-level modelling and analysis. Key features of Event-B are the use of set theory as a modelling notation, the use of refinement to represent systems at different abstraction levels, and the use of mathematical proof to verify consistency between refinement levels. Event-B, a variant of B method [13], was designed for developing distributed systems. In Event-B, the events consist of guarded actions occurring spontaneously rather than being invoked. The invariants state the properties that must be satisfied by the variables and maintained by the activation of the events.

Event-B models are organized in terms of two basic components: contexts and machines.

- Contexts specify the static part of a model. They made of a list of distinct carrier sets, constants, axioms and theorems
- Machines specify the dynamic part of the system. They may contain variables defining the state of a machine, invariants constraining that state, and events (describing possible state changes). Each event is composed of a set of guards and a set of actions. Guard states the necessary conditions under which an event may occur, and actions describe how the state variables evolve when the event occurs.

From a given model M1, a new model M2 can be built as a refinement of M1. In this case, model M1 is called an abstraction of M2, and model M2 is said to be a concrete version of M1. A concrete model is said to refine its abstraction. Each event of a concrete machine refines an abstract event or refines skip. An event that refines skip is referred to as a new event since it has no counterpart in the abstract model.

A key concept in Event-B is proof-obligation (PO) capturing the necessity to prove some internal properties of the model such as typing, invariant preservation by events, and correct refinements.

The Rodin is the tool of the Event-B. It allows formal Event-B models to be created with an editor. It generates proof obligations that can be discharged either automatically or

interactively. Rodin is modular software, and many extensions are available. These include alternative editors, document generators, team support, and extensions (called plug-ins) some of which include support decomposition and records.

A. Transaction Service

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: Initiates a new transaction the type req (request), ind (indication), res (response), cnf (confirm) are allowed.

TR-Result: Sends back a result of a previously initiated transaction. The req, ind, res, and cnf types are allowed.

TR-Abort: Aborts an existing transaction. The only req and ind are allowed.

B. Transaction Protocol

The transaction protocol defines the procedures for the initiator PE and responder PE to communicate in order to provide the transaction service. The messages sent between peer protocol entities are called Protocol Data Units (PDUs). There are four primary PDUs used in the Transaction protocol:

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.

Fig. 1 models an example sequence of primitives. This sequence shows the TR-Init-User making a request (TR-Invoke.req) which is delivered to the TR-Resp-User (TR-Invoke.ind). The TR-Resp-User confirms that the request was received (TR-Invoke.res) which is in turn delivered to the TR-Init-User (TR-Invoke.cnf). After confirming the receipt of the request, TR-Resp-User sends the result (TR-Result-req), which is delivered to the TR-Init-User (TR-Result.ind). Finally, the TR-Init-User confirms the receipt of the result (TR-Result.res), resulting in a TR-Result-cnf primitive being delivered to the TR-Resp-User.

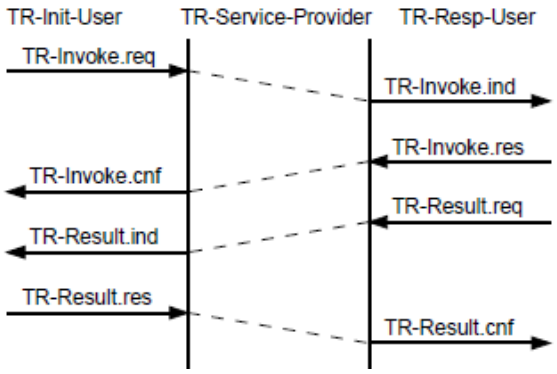

Fig. 1 Example sequence of protocol events for successful transaction

III. Formal Modeling of the WTP Protocol Using Event-B

A. Initial Model

The initial model is presented as follows:

The context is made of two sets Messages and PrimitiveTypes. The set "Messages" represents the message type exchanges between the initiator and responder, whereas the set "PrimitiveTypes" represents the possible types of the WTP service primitives.

SETS

PrimitiveType

Messages

AXIOMS

axm1: partition (PrimitiveType, {req}, {ind}, {res}, {cnf})

axm2: partition (Messages, {Invoke}, {Ack}, {Result}, {Abort})

In the machine, we first define some variables:

The variables are called "InitToResp" and "RespToInit" representing the communication channels from initiator to responder and from responder to initiator, respectively. These variables are typed as subset of Messages.

The two variables "InvokePrimitive" and "ResultPrimitive": they describe the different primitive types of the service primitive TR-Invoke and TR-result, respectively. These two variables are typed as subset of PrimitiveTypes.

"GenTID" and "SendTID" represent the ID transaction (TID) to use for the next transaction and the TID value to send in all PDUs in this transaction, respectively.

GenTID must be incremented by one for every initiated transaction.

VARIABLES

InitToResp

RespToInit

InvokePrimitive

ResultPrimitive

GenTID

SendTID

INVARIANTS

inv1 : InitToResp $\subseteq$ Messages

inv2 : RespToInit $\subseteq$ Messages

inv3 : ResultPrimitive $\in$ PrimitiveType

inv4 : InvokePrimitive $\in$ PrimitiveType

inv5 : SendTID $\in \mathbb{N}$

inv6 : GenTID $\in \mathbb{N}$

Finally, we define the events of our abstract model:

TR-Invoke.req: when the initiator initiates a new transaction, the invoke message is sent to the Responder.

Rcv-Invoke: the responder receives the message invoke, and generates the primitive type TR-Invoke.ind.

TR-Invoke.res: The Responder waits for the invoke message to be processed, the acknowledgement is sent to prevent the Initiator from re -transmitting the invoke message.

InitRcv_ACK: the initiator receives the acknowledgement from the responder,

TR-Result.req: The result is sent to the Initiator by the

responder
Rcv-Result: the initiator receives the message result, and generates the primitive type TR-result.ind.
TR- Result.res: The Initiator acknowledges the received result message.
RespRcv_ACK: the responder receives the acknowledgement from the initiator
RespRcv_ACK: the responder receives the acknowledgement from the initiator

TR-Invoke-req
WHEN
grd1: Invoke ∉ InitToResp
grd2: Invoke ∉ RespToInit
THEN
act1: InitToResp:= InitToResp ∪ {Invoke}
act2: SendTID:=GenTID
act3: GenTID:=GenTID+1
act4: InvokePrimitive:=req
END
Rcv-Invoke
WHEN
grd1: Invoke ∈ InitToResp
grd2: InvokePrimitive=req
THEN
act1: InvokePrimitive := ind
act2: InitToResp:=InitToResp \ {Invoke}
act3: RespToInit:=RespToInit ∪ {Invoke}
END
TR-Invoke.res
WHEN
grd1: Ack ∉ RespToInit
grd2: InvokePrimitive=ind
THEN
act1: RespToInit:= RespToInit ∪ {Ack}
act2: InvokePrimitive:=res
END
Rcv_Invoke.ACK
WHEN
grd1: Ack ∈ RespToInit
grd2: InvokePrimitive=res
THEN
act1: InvokePrimitive:= cnf
act2: RespToInit:=RespToInit \ {Ack}
END
TR-Result-req
WHEN
grd1: Result ∉ RespToInit ∧ Result ∉ InitToResp
grd2: InvokePrimitive=cnf ∨ InvokePrimitive=ind
THEN
act1: RespToInit:= RespToInit ∪ {Result}
act2: ResultPrimitive:=req
END
Rcv-Result
WHEN
grd1 : Result ∈ RespToInit
grd2 : ResultPrimitive=req
THEN
act1 : ResultPrimitive:=ind
act2 : RespToInit:=RespToInit \ {Result}
act3 : InitToResp:=InitToResp ∪ {Result}
END
TR-Result.res
WHEN
grd1 : Ack ∉ InitToResp
grd2 : ResultPrimitive = ind
THEN
act1 : InitToResp:=InitToResp ∪ {Ack}
act2 : ResultPrimitive:=res
END
Rcv_Result.ACK
WHEN
grd1 : Ack ∈ InitToResp
grd2 : ResultPrimitive=res
THEN
act2 : ResultPrimitive:=cnf
act3 : RespToInit:=RespToInit ∪ {Ack}
act4 : InitToResp:=InitToResp \ {Invoke}
END

B. First Refinement (Re-Transmission until Acknowledgment)

In this refinement, we introduce the Re-transmission until Acknowledgement procedure; it is used to guarantee reliable transfer of data from one WTP provider to another in the event of packet loss. For this, we define two variables: R as the re-transmission timer and RCR as the re-transmission counter. We also add Temp_R as Boolean variable for whether the timer is hold or not.

The variables R and RCR should not exceed the constants R_DEFAULT and RCR_MAX, respectively.

When the message (invoke or result) has been sent, the retransmission timer started and the re-transmission counter is set to zero (we refine the abstract events "TR-Invoke-req" and "TR-Result-req" by adding Temp_R=true and RCR=0 as actions).

If a response (acknowledgement) has not been received when the retransmission timer expires, the retransmission counter is incremented by one, the message retransmitted and the retransmission timer restarted (we add two new events "re-send_Invoke" and "re-send_Result"). The WTP provider continues to re-transmit until the number of re-transmissions has exceeded the maximum re-transmission value

When the acknowledgement has been received, the timer R must turn off (we refine the abstract events "Rcv_Invoke.ACK" and "Rcv_Result.ACK" by adding the action temp B:=FALSE).

We add also a time progression event "Clock_R" activated when the time R started.

VARIABLES
R
RCR
Temp_R
INVARIANTS
inv1 : R ∈ ℕ
inv2 : RCR ∈ 0..RCR_MAX
inv3 : Temp_R ∈ BOOL
TR-Invoke-req ≙
REFINES
TR-Invoke-req
Then
act5 : RCR:=0
act6 : Temp_R :=TRUE
Resend_Invoke ≙
WHEN
grd1 : Invoke ∈ InitToResp
grd6 : R = R_DEFAULT

grd5 : RCR< RCR_MAX
THEN
act3 : InitToResp ≔ InitToResp ∪ {Invoke}
act5 : RCR≔RCR+1
act6 : R≔0
END
Resend_result ≙
WHEN
grd1 : Result ∈ RespToInit
grd9 : R=R_DEFAULT
grd4 : RCR < RCR_MAX
THEN
act1 : RespToInit ≔ RespToInit ∪ {Result}
act3 : R≔0
act4 : RCR≔RCR+1
END
TR-Result-req ≙
REFINES
TR-Result-req
When
Then
act5 : RCR≔0
act6 : Temp_R ≔TRUE
Clock_R ≙
WHEN
grd1 : Temp_R=TRUE
THEN
act1 : R≔R+1
END

C. Second Refinement (Transaction Abort)

The transaction abort is an important refinement of this model. The aborts are symmetric; they can come from either initiator or responder. Also, when the number of timeouts (and retransmissions) reaches a maximum value, the transaction is aborted.

A new variable "AbortPrimitive" represents the different primitive types of the service primitive TR-Abort.

In this refinement, we also introduce the different states of the Initiator and Responder. For this, we define two new variables init_ste and resp_ste:

init_ste denotes the current state of Initiator.

resp_ste denotes the current state of Responder.

They typed by the set states:

partition(states, {null}, {invoke_wait}, {invoke_ready}, {wait_user}, {result_wait}, {result_ready}, {finished}, {aborted})

We are now ready to define our events:

init_Abort-req: the initiator abort the transaction by sending the "Abort" message, and it enters in "aborted" state.

resp_RcvAbort: the responder receives the abort from the initiator, and it generates the TR-Abort indication primitive.

resp_Abort-req: the responder initiates the abort by sending the Abort message to the initiator, and it enters in "aborted" state.

init_RcvAbort : the abort is received by the initiator, and the TR-Abort indication primitive is generated.

TimerTO_R: when the number of timeouts reaches a maximum value, the transaction is aborted and both initiator and responder enter in aborted state.

init_Abort-req ≙
WHEN
grd1 : init_ste≠null ∧ init_ste≠aborted
THEN
act1 : InitToResp ≔ InitToResp ∪ {Abort}
act2 : AbortPrimitive≔req
act3 : init_ste≔aborted
END
resp_RcvAbort ≙
WHEN
grd1 : AbortPrimitive=req
grd2 : Abort ∈ InitToResp
grd3 : resp_ste ≠null ∧ resp_ste≠ aborted ∧ resp_ste≠finished
THEN
act1 : AbortPrimitive≔ind
act2 : resp_ste≔aborted
END
resp_Abort-req ≙
WHEN
grd1 : resp_ste ≠null ∧ resp_ste≠ aborted ∧ resp_ste≠finished
THEN
act1 : RespToInit ≔ RespToInit ∪ {Abort}
act2 : AbortPrimitive≔req
act3 : resp_ste≔aborted
END
init_RcvAbort ≙
WHEN
grd1 : AbortPrimitive=req
grd2 : Abort ∈ RespToInit
grd3 : init_ste≠null ∧ init_ste≠aborted
THEN
act1 : AbortPrimitive≔ind
act2 : init_ste≔aborted
END
TimerTO_R ≙
WHEN
grd1 : RCR = RCR_MAX
THEN
act1 : AbortPrimitive≔ind
act2 : init_ste≔aborted
act3 : resp_ste≔aborted
END

IV. Conclusion

In this paper, we have modeled and proved the WAP Class 2 WTP using Event-B.

We have explained our approach using refinement, which allows us to achieve a very high degree of automatic proof. The model was developed to be able to verify protocol features. We have used the Rodin tool to generate the proof obligations and to discharge those obligations automatically and interactively. We have also used the ProB, Rodin plug-in, for deadlock checking.

From the results of our modeling and verification of WTP, we have identified some deficiencies in the current specification.

References

[1] WAP Forum. Wireless Application Protocol Wireless Transaction Protocol Specification. Available via http://www.wapforum.org/, 12 Jul

2001.
[2] S. Gordon and J. Billington, *Analysing the WAP class 2 wireless transaction protocol using coloured Petri nets*, Springer Berlin Heidelberg, 2000.
[3] Y. T. He, *Verification of the WAP transaction layer using the model checker SPIN*, 2003.
[4] J. R. Abrial, *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, 2010.
[5] D. Cansell, and D. Méry, *The Event-B Modeling Method: Concepts and Case Studies*. Springer, 2007.
[6] R. J. Back, *On the correctness of refinement steps in program development*, Department of Computer Science, University of Helsinki, 1978.
[7] M. Fitting, *First-order logic and automated theorem proving*, Springer Science & Business Media, 1996.
[8] R. Filali and M. Bouhdahi, *A Mechanically Proved and an Incremental Development of the Session Initiation Protocol INVITE Transaction.* Journal of Computer Networks and Communications, 2014.
[9] R. Filali, and M. Bouhdadi. *Formal verification of the Lowe modified BAN concrete Andrew Secure RPC protocol*. In RFID and Adaptive Wireless Sensor Networks (RAWSN), Third International Workshop. IEEE. pp. 18-22. 2015.
[10] S. Hallerstede, On the purpose of Event-B proof obligations, In: Abstract state machines, B and Z. *Springer Berlin Heidelberg*, pp. 125-138. 2008.
[11] C. Jones, I. Oliver, A. Romanovsky, and E. Troubitsyna, *RODIN (rigorous open development environment for complex systems),* University of Newcastle.
[12] O. Ligot, J. Bendisposto, and M. Leuschel. Debugging event-b models using the prob disprover plug-in, *Proceedings AFADL 7*, 2007.
[13] J. R. Abrial, *The B-Book: Assigning Programs to Meanings*, Cambridge University Press, 1996.