Received: March 26, 2018

INASS

International Journal of
Intelligent Engineering & Systems

http://www.inass.org/

A Mechanized Formal Refinement Proof of Modbus Communication Using Event-B Proof System

Sanae El Mimouni [1]* Mohamed Bouhdadi [1]

[1] *Laboratory of Magnetism and Physics of High Energies, Faculty of Sciences, Mohammed V University, Rabat, Morocco*
* Corresponding author's Email: sanae.elm@gmail.com

Abstract: Formal methods have been applied in the design of a great number of systems. Many protocols have been specified and verified formally. Some protocol standards are even defined by means of a formal method. This paper studies a based formal approach for testing associated properties of Modbus communication protocol with the Event-B Method. Event-B is a formal method for systems modeling, founded on set theory and predicate logic. It has the benefit of mechanized proof, and it is practicable to model a system in various levels of abstraction using refinement. Our aims are constructing a model with a clear and accurate formulation of the protocol properties and discharge all proof obligations. To satisfy these, attentive choice of invariants and machine theorems was important and eased the proof effort. A major focus of our work has been to explore the use of the Event-B method for formally specifying Modbus protocol. The supported language was sufficiently expressive and all proof obligations could be discharged. We reached a good degree of automatic proof. All interactive proofs involved a small number of steps and were straightforward to reach. The result of this approach was that we achieved a very high degree of automatic proof.

Keywords: Formal method, Modbus communication protocol, Refinement, Event-B method, Proofs, Rodin.

1. Introduction

Formal methods can be defined as mathematically based techniques, which are used for specifying and reasoning about software and hardware systems. The essence of formal methods comes down to proof: (i) formulating proof obligations in terms of formal specifications and models, (ii) verifying, via algorithmic proof search, that a designed system meets its specifications, and (iii) algorithmically synthesizing all or parts of a system so as to satisfy its specifications. Unlike traditional calculus-based engineering mathematics, formal methods rely primarily on discrete mathematics and computer science formalisms such as finite state machines.

This paper studies a based formal approach for testing associated properties for communication protocol. Communication protocols define the set of rules needed to exchange messages among communicating entities. Networked and distributed systems, built around communicating protocols, are widely used nowadays. So, it is becoming more significant that communication protocols be formally specified and verified.

One of the most popular industrial data communication protocol is Modbus, which is widely used in industrial automation, for good reasons. It is simple, inexpensive, universal and easy to use. The Modbus protocol [1] is an element of the supervisory control and data acquisition (SCADA) system, and it's the foremost usually used protocol in industrial systems, together with the oil and gas industries and power industries [2]. It is nowadays the most frequently accessible way of connecting industrial electronic devices. It has become a standard communications protocol in industry. It is employed extensively by various manufacturers throughout several industries. Modbus is usually used to transfer signals from instrumentation and control devices back to a central controller or data assembling system, such as a system that measures temperature and

humidity and transmits the results to a computer. This article describes a formal model of the Modbus protocol using the Event-B method. We have used Event-B as proof-based development method which integrates formal proof techniques for writing specifications and building the model systematically using formal refinement, the key point is to start with a very abstract model of the system under development. Details are gradually added to this first model by building a sequence of more concrete ones. This strategy eases the proof of the correctness of requirements because only a small number of proof obligations are generated at each step. The presented formalization is based on the Modbus Application Protocol Specification [1] and the Modbus over Serial Line: Specification and Implementation Guide [3].

Event-B [4,] is a formal method that applies the concept of refinement [5] in modeling; it is founded on set theory and predicate logic.

A great variety of formal specification techniques exist, some of which are general purpose (such as Z, Vienna Development Method (VDM) or Common Object-oriented Language for Design (COLD)), while others are generally used in a specific domain of application (such as Language Of Temporal Ordering Specification (LOTOS), Specification and Description Language (SDL) and Process Specification Formalism (PSF)). The mathematical theories on which these languages are based range from set theory and temporal logic to lambda-calculus and process algebra. The reason to choose Event-B, in particular, is motivated by several factors. Event-B is a simplification of Classic-B, it promotes a layered style of formal modeling, where a model is developed as chains of abstract models, and level-by-level concrete details are progressively introduced via provably correct refinement steps. This style of modeling, which called refinement, decomposing machines into small, discrete events explicitly linked to their abstractions. This encourages more incremental refinement and easier verification by the generation of more easily discharged POs (Proof Obligations), enabling the practical construction of larger systems. However, Event-B's main advantage is its flexibility, both in the notation itself and its supporting tools.

The Event-B refinement process authorizes us to gradually add implementation details while preserving functional correctness during stepwise model transformation.

Along the refinement, a set of proof obligations is discharged. The purposes of the proof obligations are to verify the consistency of a specification and to preserve the functionality from its abstract specification. It is difficult to manually generate and prove the proof obligations. Thus, the RODIN platform [6] has been built to provide an automated tool to generate and prove the proof obligations automatically or interactively. Besides, The RODIN platform can also support modeling in Event-B. Event-B, together with the RODIN platform, has been successfully applied to several practical safety-critical systems. Some concrete examples are a train controller system [7], hybrid systems [8], a spacecraft system [9], and a metro system [10]. Event-B can be regarded as a method for correct-by-construction software development. Event-B has gained widespread attention with it tool support which can be employed to specify different communication protocols as in [11], wireless communication [12], Wireless Sensor Networks [13] or in hybrid encryption technique [14].

The model verification effort and, in particular, automatic generation and proving of the required proof obligations, are significantly facilitated by the provided tool support, the RODIN platform. In this paper, we liberally used refinements. We give a great deal of attention to proofs. Consequently, we now have a specification of Modbus protocol where all proof-obligations have been discharged.

The purpose of this paper is to provide with a collection of protocol descriptions which illustrates how to use formal specification techniques such as the Event-B method in the field of communication protocols. The specifications in this paper have a level of abstraction that is appropriate for a clear understanding of the Modbus protocol without having to deal with implementation details.

This article follows a general pattern of moving from the general and abstract to the specific model. After discussing the introduction and motivation of this work, the rest of the paper is organized as follows. In Section 2,3 and 4, we recall the definition of both Modbus communication protocol and the Event-B method along with the Rodin Platform. We then present master and slave behavior. Based on this definition and properties, we define in Section 5 an approach to model the Modbus protocol. Section 6 summarizes the results and draws a conclusion.

2. Event-B method

Event-B is a formal method for specifying, modeling and reasoning about systems. Event-B is a modeling framework derived from the B method developed by Jean-Raymond Abrial. Event-B is now centered on the general notion of events. Event-B is a formal modeling method for developing systems

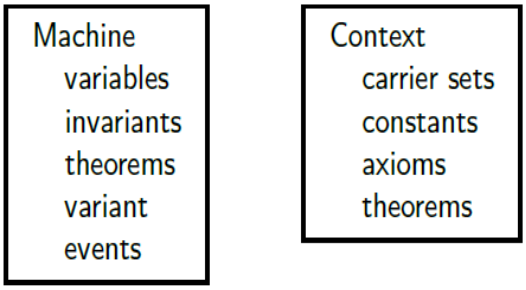

Figure. 1 Machine and context

via step-wise refinement. In Event-B, the system, and its properties are specified using set-theory and predicate logic. It uses proof and refinement to show that the properties hold as the development proceeds. Event-B models are structured in terms of two main components: contexts and machines (Fig. 1)

Contexts: Contexts specify the static part of a model and may consist of carrier sets, constants, axioms, and theorems. Carrier sets are same as types. Axioms restrain carrier sets and constants, when in fact theorems represent properties derivable from axioms. The utility of a context is to isolate the parameters of a formal model (carrier sets and constants) and their properties, which are intended to hold for all instances.

Machine: Machines contain variables modeling state data, invariants which restrict the possible values of variables, and events which change the values of variables. An event consists of guards, which must be true in order for the event to occur, and actions, in which the values of variables are changed.

There are three kinds of relationships between components of an Event-B model as shown in Fig. 2.

- A concrete machine can only "refine" at most one more abstract machine.
- A concrete context can "extend" zero, one, or several more abstract contexts.
- A machine can "see" zero, one, or several contexts.

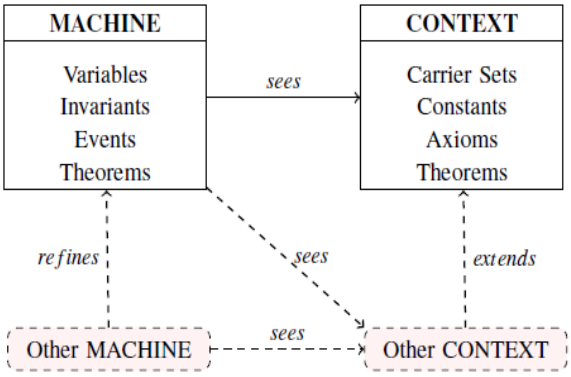

Figure. 2 Machine and context relationships

If a machine "sees" a context, then all the components like constants, sets, and axioms defined in the context and extended from other Contexts can be used by the machine.

Events may be parameterized, and in general, an event takes the form (≙ it's a default symbol of an event)

Event ≙ **any** p **where** G(p, v) **then** S(p, v) **end** (1)

Where p is the event's parameters, G(p, v) is the event's guard (the conjunction of one or more predicates), and S(p, v) is the event's action. The guard states the condition under which an event may occur, and the action represents how the state variables evolve when the event occurs. We use the short form:

Event ≙ **when** G(v) **then** S(v) **end** (2)

When the event does not have any parameters, and we write:

Event ≙ **begin** S(v) **end** (3)

A dedicated event in the form of (3) without any parameters or guard is used for initialization.

The action of an event is composed of one or more assignments of the form:

$$act := E(x, v) \quad (4)$$

or

$$act :\in E(x, v) \quad (5)$$

or

$$act:| P(x, v,a') \quad (6)$$

Where x is a variable in v. E(x, v) is an expression, and P(x, v,a') is a predicate. Assignments in Event-B may also be non-deterministic. All assignments of an action S(p, v) occur simultaneously.

Refinement: Refinement is a top-down development method and is at the core of Event-B modelling. We start by specifying the system at an abstract level and gradually refine by adding further details in each refinement step until the concrete model is achieved. A machine M0 can refine another machine M1. We call M1 the abstract machine and M0 the concrete machine. The states of the abstract machine are related to the states of the concrete machine by gluing invariants J (v; w), where v are the variables of the abstract machine and w are the variables of the concrete machine. A special case of refinement (called horizontal refinement) is when v

is kept in the refinement, i.e. v ⊆ w. Intuitively, any behavior of M0 can be simulated by a behavior of M1 with respect to the gluing invariant J (v; w).

Event-B is supported by several tools, currently, in the form a platform called Rodin [6].

3. Rodin platform

Rodin is an Eclipse-based development environment for Event-B. It is open source and provides an environment for system modeling and analyzes, including support for checking specification correctness and for refinement proofs. While constructing an Event-B program, Rodin will automatically generate a set of POs for the program under consideration. The list of proof obligations can be found in [4]. Each PO is a logical formula, whose validity implies that certain correctness properties are satisfied with the program under consideration. In Rodin, the correctness properties include:

1. The Event-B program is not in an invalid state (i.e. a state where some invariant might not hold).
2. The behavior of a concrete Event-B program will correspond to the behavior of its abstract program.

The first property is ensured by proving that the invariant is preserved and by proving the well-definedness of predicates [4]. The second one, i.e. the correspondence between abstract and concrete Event-B programs, is usually called the refinement PO. There are three kinds of POs which can be generated from Rodin to guarantee that the refinement is correct [4]:

- Guard strengthening (GRD)
- Action simulation (SIM)
- Equality of a preserved variable (EQL)

We present some important windows of the platform as follows:

- Proving Perspective (Fig. 3): It provides all proof obligations, which are automatically generated for Event-B machines. These proof obligations can be discharged automatically or interactively with hypotheses and goal windows.
- Event-B Perspective (Fig. 4): This perspective includes windows, which allow us to edit Event-B machines and contexts. If users encode incorrectly, problem windows will show the error's content.

Obligations are proved either automatically or manually. In automatic mode, Rodin applies some pre-set proof tactics made up of internal and external provers to discharge the obligations. In interactive mode, the user "orients" the proof attempts by using some easy proof steps to simplify the obligations

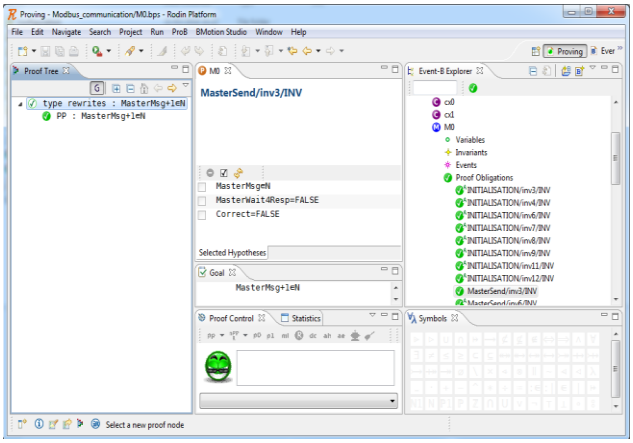

Figure. 3 The Proof Obligation Perspective: on the left, it is shown the proof tree of the selected PO, on the middle; on the top window are the hypotheses of the selected PO and just below the respective goal. Below the goal window are the buttons used to interactively discharge a PO, on the right, are the list of generated POs. Having all the POs green, it means that all the POs are discharged

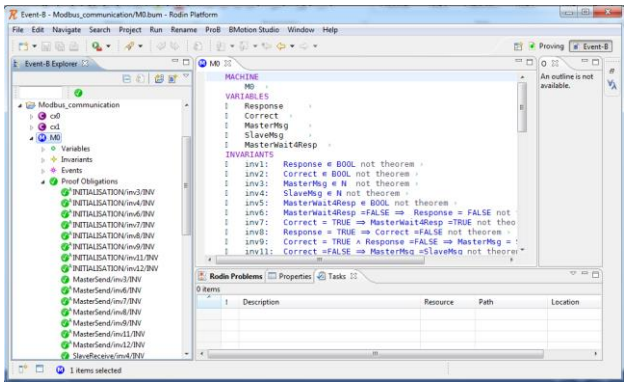

Figure. 4 The Event-B Perspective: on the left, the list of projects where the Modbus_communication project is expanded, showing several machines and a context, in the middle window, a view of a machine M0 where the sections of variables, invariants and events can be edited.

before bringing up some trusted external provers to end the proofs. As interactive proofs require hand-operated involvement, it is usually considered as some costs of developing formal models.

4. Modbus Communication

Modbus is a serial communication protocol developed by Modicon promulgated by Modicon in 1979 for employment with its programmable logic controllers (PLCs). In an easy way, it is a method applied for sending information over serial lines between electronic devices. The Modbus protocol is part of the supervisory control and data acquisition (SCADA) system, and it is the most generally used protocol in industrial systems, including the oil and gas industries and power industries.

As defined by the Modbus Organization, Inc.:

"Modbus is an application-layer messaging protocol that provides client/server communication among boards connected on different types of buses or networks. Modbus is the de facto industrial serial standard that enables automation boards to communicate"

Modbus is a request-response protocol implemented using a master-slave relationship. In a master-slave relationship, only one device (the master) can initiate transactions (queries). The opposite devices (the slaves) reply by providing the demanded data to the master, or by taking the action asked for in the query. Typically, the master is a human-machine interface (HMI) or Supervisory Control and Data Acquisition (SCADA) system and the slave is a sensor, programmable logic controller (PLC), or programmable automation controller (PAC). The master can address individual slaves or can initiate a broadcast message to all slaves.

The master node issues a MODBUS request to the slave nodes in two modes:

– Unicast mode: the master communicates a singular slave. After the reception and the process of the request, the slave resubmits a message (a 'reply') to the master. In that mode, a MODBUS transaction is composed of two messages: a request from the master, and a response from the slave.

– Broadcast mode: the master is able to transmit a request to all slaves. No reply is returned to broadcast inquiries transmitted by the master.

5. Modeling Modbus protocol

This modeling approach is based on Modbus Protocol described in [1, 3]. Essentially, we aim at constructing the model of a Modbus protocol. This protocol determines the communication taking place between a master and a slave. Both the master and slave begin in a certain state, and through the exchange of messages, they move from one state to another.

In this work, the modeling is done using Event-B and the specifications were verified with the Rodin tool [6]. The proof obligations can be satisfied either with the available automatic provers or interactively, with direct interference from the user.

5.1 Initial model

In this initial model, we just formalize what the participants can eventually do: One side sends a request to the other and waits for a response. The other side receives the request and sends a response. The first side stops waiting for a response when it receives it. For our abstract model, we use five variables which are: Response (the response channel), Correct (Whether Data is correct or not), MasterMsg (Message Number seen by the master), SlaveMsg(Message Number seen by the slave) and MasterWait4Resp(State of Master).

INVARIANTS
inv1 : Response ∈ BOOL
inv2 : Correct ∈ BOOL
inv3 : MasterMsg ∈ N
inv4 : SlaveMsg ∈ N
inv5 : MasterWait4Resp ∈ BOOL
inv6:MasterWait4Resp=FALSE ⇒ Response= FALSE
inv7:Correct=TRU⇒ MasterWait4Resp =TRUE
inv8 : Response = TRUE ⇒ Correct =FALSE
inv9:Correct = TRUE ∧ Response =FALSE ⇒ MasterMsg = SlaveMsg +1
inv10:Correct=FALS⇒MasterMsg =SlaveMsg
inv11 : MasterMsg = SlaveMsg ∨ MasterMsg = SlaveMsg +1

We define the dynamics of the system by means of three events:

- Event **INITIALISATION**: Initialize the used variables.
- Event **MasterSend**: Master Sends request.
- Event **SlaveReceive**: Slave gets request and sends a response.
- Event **MasterReceiveResponse**: Slave gets the response.

INITIALISATION ≙
BEGIN
act1 : Response := FALSE
act2 : Correct := FALSE
act3 : MasterMsg := 0
act4 : SlaveMsg := 0
act5 : MasterWait4Resp := FALSE
END

Event MasterSend ≙
WHEN
grd1 : MasterWait4Resp =FALSE
grd2 : Correct = FALSE
THEN
act1 : MasterWait4Resp := TRUE
act2 : Correct := TRUE
act3 : MasterMsg := MasterMsg +1
END

Event SlaveReceive ≙
WHEN
grd1 : Correct = TRUE

THEN
act1 : Correct := FALSE
act2 : Response := TRUE
act3 : SlaveMsg := SlaveMsg +1
END

Event MasterReceiveResponse ≙
WHEN
grd1 : MasterWait4Resp = TRUE
grd2 : Response =TRUE
THEN
act1 : Response = FALSE
act2 : MasterWait4Resp := FALSE
END

Proof Obligation: When an Event-B model is created or refined, a set of proof obligations must be discharged in order to guarantee certain properties of a model. We can define a proof obligation as a mathematical formula to be proven, in order to ensure that an Event-B component is correct. Proof obligations have a two-fold purpose [15]:

- They show that a model is sound with respect to some behavioral semantics.
- They serve to verify the properties of the model.

The proof obligations define what is to be proved for an Event-B model. These proofs concern invariant preservation, Feasibility, Fusion,…They are automatically generated by RODIN platform tool called the proof obligation generator, just to check contexts and machines texts and decide what is to prove in these texts, there are eleven rules for the proof obligation all defined and labeled inside the RODIN platform. We will just define one proof used in our model.

Invariant Establishment and Preservation (Noted as INV): An essential feature of an Event-B machine M is its invariant I(v). It shows properties that hold in every reachable state of the machine. Obviously, this does not hold a priori for any machines and invariants, and therefore must be proved. A common technique for proving an invariant property is to prove it by induction: (1) to prove that the property is established by the initialisation init (invariant establishment), and (2) to prove that the property is maintained whenever variables change their values (invariant preservation).

Invariant establishment states that any possible state after initialisation given by the after predicate K(v') must satisfy the invariant I. The proof obligation rule is as follows:

$$K(v') \vdash I(v') \qquad \textbf{(INV)}$$

The statement of the above form is called a sequent. The symbol ⊢ is named the turnstile. The part situated on the left side of the turnstile, denotes a finite set of predicates called the hypotheses (or assumptions). The part situated on the right side of the turnstile, here **I(v'),** denotes a predicate called the goal (or conclusion). The intuitive meaning of such a statement is that the goal **I(v')** is provable under the set of assumptions **K(v').** In other words, the turnstile can be read as the verb "entail," or "yield"; the assumptions **K(v')** yield the conclusion **I(v')**.

Invariant preservation makes it necessary to prove that every event occurrence re-establishes the invariants I. More precisely, for every event evt, assuming the invariants I and evt's guard G, we must prove that the invariants still hold in any possible state after the event execution given by the before-after predicate Q(x, v, v'). The proof obligation rule is as follows:

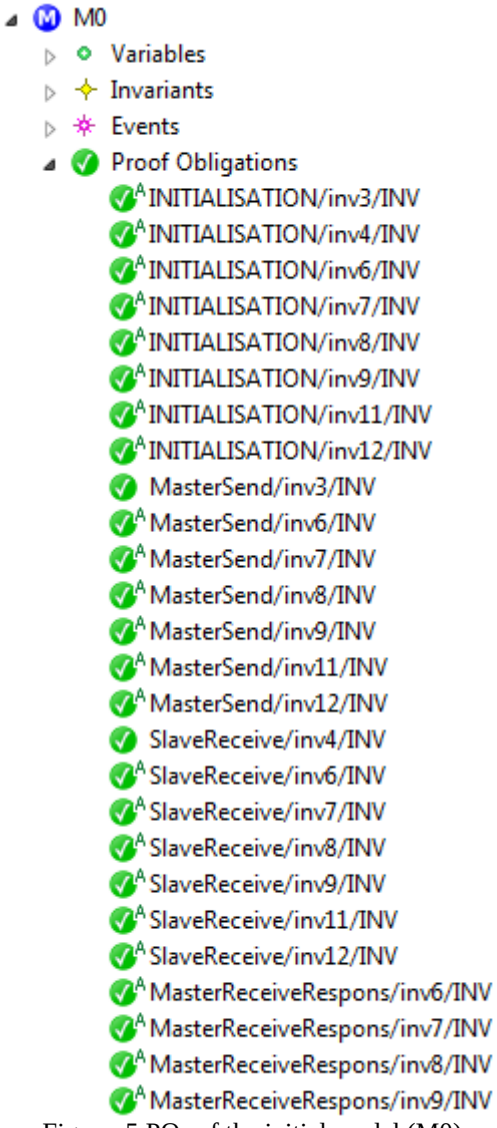

Figure. 5 POs of the initial model (M0)

$$I(v),G(x, v),Q(x,v,v') \vdash I(v') \quad \textbf{(INV)}$$

Fig. 5 shows proof obligations of the initial model (M0) the Modbus protocol. In this initial model, we have 26 invariants preservation proofs, with two of them proved interactively.

5.2 First refinement: Master behavior

For the first refinement, we introduce the behavior of Modbus master (shown in Fig. 6) into the scene whereby the slave does not interact at all. After power-up, the initial state of the master is “Idle” which means there is no pending request. A request can be exclusively transmitted in “Idle” state. The Master changes the previous state which is ”Idle”, just after transmitting a request, and won’t be able to transmit a second request at the same time. As we specify only the unicast mode in this paper, the master goes into ”Waiting for reply” state after sending the request to a slave and starts a response time-out, to prevents the Master from remaining indefinitely in ”Waiting for reply” state. Upon receiving a response, the Master examines the reply before initiating the data processing. In a situation where an error is discovered on the frame, a retry may be executed. If no response is obtained, the Response time-out expires, and an error is produced. Then the Master goes into”Idle” state, allowing a repeat of the request.

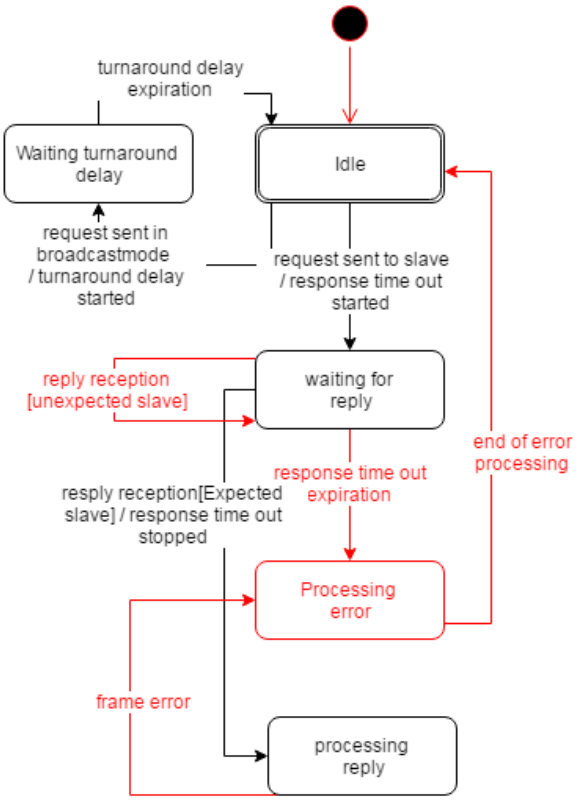

Figure. 6 Master state diagram

This behavior is modeled as follows: First, we introduce the concept of state. For this, we define a carrier set named STATE. It contains four constants (Idle, Wait4Reply, Processing_Reply, and Processing_Err) defined by axioms (axm1-axm7).

AXIOMS

axm1:partition(STATE,Idle,Wait4Reply,Processing_Reply, Processing_Err)
axm2 : Idle ≠ Wait4Reply
axm3 : Idle ≠ Processing_Err
axm4 : Idle ≠ Processing_Reply
axm5 : Wait4Reply ≠ Processing_Err
axm6 : Wait4Reply ≠ Processing_Reply
axm7 : Processing_Err ≠ Processing_Reply

For the maximum time that a request can be transmitted, we define it by constant Max request, which is a natural number and strictly positive.

AXIOMS

axm1 : Max_request ∈ N
axm2 : Max_request > 0

Then we can use three variables: state, time and retries defining respectively the state of the master, current time and the number of retries of retransmission of request.

The events: Event MasterSend and MasterReceiveRespons are refined from the previous machine.

Event MasterSend ≙
REFINES
Event MasterSend
ANY
tm
WHERE
grd1 : MasterWait4Resp =FALSE
grd2 : Correct = FALSE
grd3 : tm ∈ N
grd4 : state = Idle
THEN
act1 : MasterWait4Resp := TRUE
act2 : Correct := TRUE
act3 : MasterMsg := MasterMsg +1
act4 : state := Wait4Reply
END

Event MasterReceiveRespons ≙
REFINES
Event MasterReceiveRespons
ANY
tm
WHERE
grd1 : MasterWait4Resp = TRUE
grd2 : Response =TRUE

grd3 : tm ∈ N
grd4 : state = Processing Reply
grd5 : time ≥ time +1
THEN
act1 : Response := FALSE
act2 : MasterWait4Resp:= FALSE
act3 : state := Idle
END

We define now our new events of this refinement: The event Time which progress the time.

Event Time ≙
BEGIN
act1 : time := time +1
END

Event Master fail ^=
WHEN
grd1 : state = Idle
grd2 : retries= Max request
THEN
act1 : retries := retries +1
END

Event Master Resend ≙
WHEN
grd1 : state =Idle
grd2 : retries > Max request
THEN
act1 : state := Wait4Reply
END

Event Error ≙
ANY
tm
WHERE
grd1 : tm ∈ N
grd2 : state = Wait4Reply
grd3 : tm ≥ time+1
THEN
act1 : state := Idle
END

Proof Obligation: Fig. 7 shows proof obligations of the first refinement (M1) which describes the master behavior. In the first refinement, the Rodin Platform generates four invariants preservation proofs, with two of them proved interactively.

5.3 Second refinement: Slave behavior

In this refinement, we model the behavior of Modbus slave (shown in Fig. 8).

As like the master, the state of the slave is ”Idle” after power-up, which means there is no pending request. At the time a request is received, the slave checks the packet before performing the action requested in the packet. Several errors may occur. In case of error, a reply must be sent to the master. After receiving and processing the request, a reply must be formatted and sent to the master. Once the required action has been completed since we cover the unicast mode. If the slave detects an error in the received frame, no response is returned to the master.

In this refinement, the slave also has a state, so we will add three constants (Checking_Request, Formatting_Reply, and Processing_Request) to our previous STATE SET.

axm8: Checking_Request ∈ STATE
axm9: Processing_Request ∈ STATE
axm10: Formatting_Reply ∈ STATE

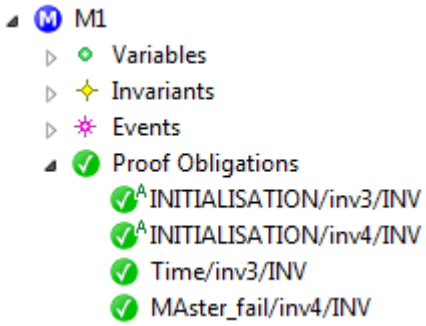

Figure. 7 POs of the first refinement (M1)

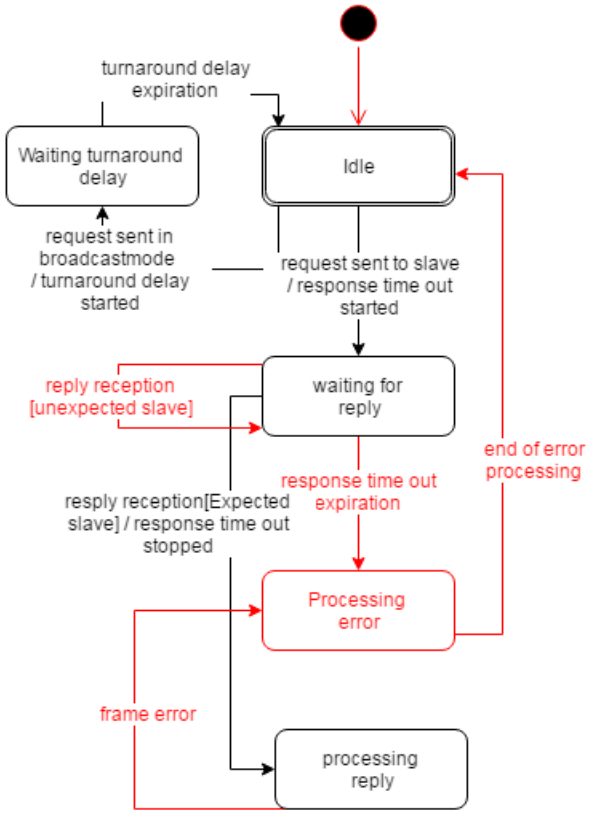

Figure. 8 Slave state diagram

In addition, we add two variables state slave and Error.

INVARIANTS

inv1: state slave ∈ STATE
inv2: Error ∈ BOOL
inv3:(state_slave=Checking_Request∨state_slave = Formatting_Reply ⇒ state_slave = Idle
inv4:state_slave=Idle⇒state_slave=Checking_Request
inv5: Error= TRUE ⇒ state_slave=Idle
inv6:state_slave = Checking_Request ⇒ state_slave= Formatting_Reply ∨ state_slave =Idle

For this refinement, we have two new events:

SlaveReceive ≙
REFINES SlaveReceive
WHEN
grd1 : Correct = TRUE
grd2 : state slave = Idle
THEN
act1 : Correct := FALSE
act2 : Response := TRUE
act3 : SlaveMsg := SlaveMsg +1
act4 : state_slave := Checking_Request
END

Error Slave ≙
WHENs
grd1: state_slave= Checking_Request ∨ state slave = Formatting_Reply
grd2: Error =TRUE
THEN
act1: state slave := Idle
END

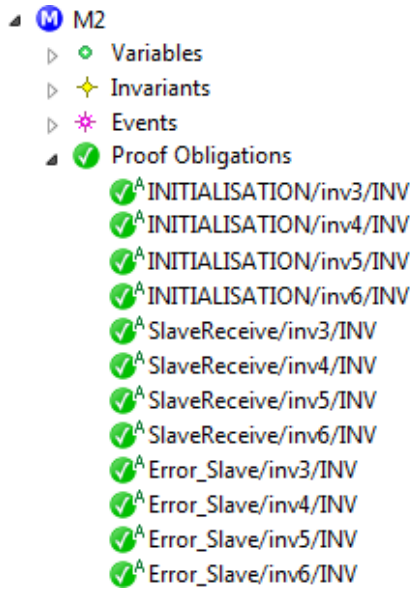

Figure. 9 POs of the second refinement (M2)

Table 1. Proof statistics for the Modbus protocol development

Model	Total POs	Automatic proof	Interactive proof
Abstract model	26	24	2
First refinement	4	2	2
Second refinement	12	11	1
Total	42	37	5

Proof Obligation: Fig. 9 shows proof obligations of the second refinement (M2) which describes the slave behavior. This refinement requires 12 proofs; all proved automatically only one of them proved interactively.

Proofs Statistics: The proof statistics for the development of the Modbus protocol is in Table 1. The complete development of the Modbus communication protocol results in 42 POs, within which 37 are proved automatically by the Rodin tool.

6. Conclusion

This paper models the Modbus protocol using the Event-B formal language. Modbus is a communication protocol that transfers information between the electronic devices in data gathering systems. It works based on the Master/Slave architecture. It means that always a device that is Master requests a data or an action from the Slave devices and Slave devices response to the Master device. This article provides an abstract model of the Modbus protocol using the Event-B formalism and then refines it to a model with more details. Which allows us to attain a very high level of automatic proof. We have used Rodin tool for writing Event-B specifications. Event-B uses the proof obligations generated by axioms and theorems to ensure the consistency among different layers. It indicates the model meets software requirements through the fully proved obligations. The presented model generates 42 proof obligations out of which 37 are discharged automatically by the prover of the tool while 5 proof obligations are discharged manually. The proof obligations generated by the model give the rigorous reasoning about the design of model. During execution of the model, all invariants are preserved which ensures that our model is correct.

As for future work, the formal verification technique applied to routing algorithms for wireless networks has been a quite unexplored field yet, and therefore there are many opportunities for new

research. Indeed, the field is in need of more specific techniques and tools.

References

[1] Modbus-IDA: *Modbus Application Protocol SpecificationV1.1a*, http://www.modbus.org.

[2] K. Stouffer and K. Kent, *Guide to Supervisory Control and Data Acquisition (SCADA) and Industrial Control Systems Security*, Recommendations of the National Institute of Standards and Technology, 2006.

[3] Modbus-IDA: *Modbus over Serial Line: Specification and Implementation Guide V1.0*, http://www.modbus.org.

[4] J.R. Abrial, *Modeling in Event-B, System and Software Engineering*, Cambridge University Press, 2010.

[5] A. S. Fathabadi, M. J. Butler, and A. Rezazadeh, "Language and tool support for event refinement structures in event-b", *Formal Aspects of Computing*, Vol.27, No.3, pp.499-523, 2015.

[6] J. R. Abrial, M. J. Butler, S. Hallerstede, T. S. Hoang, F. Mehta, and L. Voisin, "Rodin: an open toolset for modelling and reasoning in event-b", *International Journal on Software Tools for Technology Transfer*, Vol.12, No.6, pp.447-466, 2010.

[7] W. Su, J.R. Abrial, R. Huang, and H. Zhu, "From requirements to development: Methodology and example", In: *Proc. of Formal Methods and Software Engineering - 13th International Conference on Formal Engineering Methods*, pp.437-455, 2011.

[8] L. Petre and E. Sekerinski, *From Action Systems to Distributed Systems - The Refinement Approach*, Chapman and Hall/CRC, 2016.

[9] A. S. Fathabadi, A. Rezazadeh, and M. J. Butler, "Applying atomicity and model decomposition to a space craft system in event-b", In: *Proc. of NASA Formal Methods - Third International Symposium*, pp.328-342, 2011.

[10] R. Silva, "Lessons learned/sharing the experience of developing a metro system case study", *CoRR*, Vol.abs/1210.7030, 2012.

[11] S.E. Mimouni and M. Bouhdadi, "An Incremental Proof-Based Process of the NetBill Electronic Commerce Protocol", In: *Proc. of the 4th International Conference of Networked Systems*, pp.209-213, 2016.

[12] B. L. Malleswari and S. Parnapalli, "Performance of MIMO MC-CDMA for STBC Communication System Using OKHA Based Optimal Channel Estimation", *International Journal of Intelligent Engineering and Systems*. Vol.10, No.4, pp.1-10, 2017.

[13] E. H. Houssein and Y. M. Wazery, "Vortex Search Topology Control Algorithm for Wireless Sensor Networks", *International Journal of Intelligent Engineering and Systems*, Vol.10, No.6, pp.87-97, 2017.

[14] S. R. M. Halagowda, and S. K. Lakshminarayana, "Image Encryption Method based on Hybrid Fractal-Chaos Algorithm", *International Journal of Intelligent Engineering and Systems*, Vol.10, No.6, pp.221-229, 2017.

[15] S. Hallerstede, "On the purpose of event-b proof obligations", *Formal Aspects of Computing*, Vol.23, No.1, pp.133-150, 2011.