Formal Modeling of Simple Network Management Protocol using Event-B

Rajaa Filali, Sanae El Mimouni, Anas Amamou, Bahija Boulamaat, and Mohamed Bouhdadi

Abstract**—**Simple Network Management Protocol (SNMP) is a popular protocol for network management. It is used for collecting information from, and configuring, network devices.This standardization gives network administrators the ability to monitor network performance. In this paper, we highlight to analyze the correctness and authenticity of SNMP using the formal method Event-B and the Rodin Tool to verify the accuracy of our protocol's performance. Event-B is formal technique that enables user to express the problem at abstract level and then add more details in refinement step to obtain concrete specification. This interaction between modelling and proving reduces the complexity and helps in assuring that the SNMP specification is correct and unambiguous.

Keywords**—**Simple Network Management Protocol, Formal Modelling, Refinement, Event-B, Rodin

I. Introduction

SIMPLE network Management Protocol is a communication protocol, it is used to administer and manage networked devices. It can be used to manage large networks that span firewalls or embedded devices. The specifications for this protocol can be found in Request For Comments (RFC) 1157.

This article is an extended version of a conference paper that appeared as [1].

Increasingly numerous communication protocols are being employed in computer networks of various types. This increases the need of adequate software specification techniques and suitable development methods to make the system more reliable.

Rajaa Filali, *LMPHE laboratory, University of Mohammed V, Faculty of sciences, Rabat ,Morocco,(* e-mail: rajaa.filali@gmail.com).

Sanae El Mimouni, *LMPHE laboratory, University of Mohammed V, Faculty of sciences, Rabat ,Morocco,(* e-mail: sanae.elm@gmail.com).

Anas Amamou, *LMPHE laboratory, University of Mohammed V, Faculty of sciences , Rabat ,Morocco,(* e-mail: amamou.anas@yahoo.fr).

Bahija Boulamaat, *LMPHE laboratory, University of Mohammed V, Faculty of sciences, Rabat, Morocco (e-mail: boulamaatbahija@gmail.com)*

Mohamed Bouhdadi, *LMPHE laboratory, University of Mohammed V, Faculty of sciences, Rabat ,Morocco,(* e-mail: bouhdadi@fsr.ac.ma).

A number of formal approaches have been applied to model and analyze these protocols, such as Petri Nets [2,3] and State Machine [4,5]. Recently a new method Event-B [6,7] has been developed by Jean Raymond ABRIAL who has developed the B method [8] and the Z method [9].

In this paper, we use Event-B to model and prove the SNMP protocol. The most important benefit of using Event-B is its capability to use abstraction and refinement [10].

Indeed, in this approach the modeling process starts with an abstraction of the system which specifies the goals of the system. The abstract level of our Event-B model shows these goals in a very general way, and then during refinement levels, features of the protocol are modeled and the goals are achieved in a detailed way. Moreover the Rodin tool [11] permits an automated proof of the different models of the system.

The reminder of the paper is organized as follows. Section 2, gives a brief overview of Event-B. Section 3 provides the requirements which are informally defined. In Section 4, the formal development is presented. Finally, a conclusion is presented to summarize the main outcomes of this research

II. Overview of Event-B

Event-B is a formal method for specifying, modeling and reasoning about systems, especially complex systems such as an electronic circuit, an airline seat booking system, a PC operating system, a network routing program, a nuclear plant control system, a Smartcard electronic purse, etc..Event-B has evolved from classical B.

Key features of Event-B are the use of set theory as a modeling notation, the use of refinement to represent systems at different abstraction levels and the use of mathematical proof to verify consistency between refinement levels. 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. An Event-B model has two parts, context and machine. Each context specifies the static properties of the system, including sets, axioms, and constants. Each machine specifies the dynamic part of the system, including variables, invariants and events. Variables represent the current state of the system and invariants specify the global specification of the variables and system behaviors.

An event is defined by the syntax: EVENT e WHEN G THEN S END , Where G is the guard, expressed as a first-order logical formula in the state variables, and S is any number of generalized substitutions, defined by the syntax S ::= x := E(v) |x := z : |P(z). The deterministic substitution, x := E (v), assigns to variable x the value of expression E(v), defined over set of state variables v. In a non-deterministic substitution, x := z : |P(z), it is possible to choose non-deterministically local variables, z, that will render the predicate P(z) true. If this is the case, then the substitution, x := z, can be applied, otherwise nothing happens.

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 plugins) some of which include support decomposition and records.

The Rodin tool supports the application of the Event-B formal method. It provides core functionality for syntactic analysis and proof-based verification of Event-B models. Rodin also provides extension points for a range of additional plug-ins that enrich the core functionality through support for features such as model checking, model animation, graphical front ends, additional proof capabilities and code generation.

The RODIN Project was followed by the DEPLOY Project which addressed further development of the Rodin core and associated plug-ins in parallel with industrial-scale deployment of the Rodin tools. Exposing the tools to serious industrial users in DEPLOY drove the developers to implement significant improvements in performance, usability and stability of Rodin and key plug-ins such as ProB, the Theory plug-in, Camille and UML-B. Of course, as well as demanding improvements to the tool, the industrial users demanded documentation on the tool, which led to this handbook

III. Informal Description Of SNMP Protocol

The SNMP is a client/server (agent/manager) protocol. SNMP is described by a series of Request for Comments (RFCs) [12] that specifies and structures the information that is exchanged between managing and managed systems.

The **agents** (Server) reside on systems that are managed. The agent receives requests to either retrieve or change management information by referencing MIB objects. Management Information Base (MIB) objects are units of information that provide information about the system and the network to the managing system. MIB objects are referenced by the agent whenever a valid request from an SNMP manager is received.

The **manager** (Client) refers to a system that runs a managing application or suite of applications. These applications depend on MIB objects for information that resides on the managed systems. Managers generate requests for this MIB information, and an SNMP agent on the managed system responds to these requests. A request can either be the retrieval or modification of MIB information.

By accessing the MIB objects, the SNMP agent allows configuration, performance, and problem management data to be managed by the SNMP manager. This is how the agent makes network and system information available to other systems.

SNMP **traps** enable an agent to notify the management station of significant events by way of an unsolicited SNMP message.

As shown in (Fig. 1), the setup on the left shows a network management system that polls information and gets a response. The setup on the right shows an agent that sends an unsolicited or asynchronous trap to the network management system (NMS).

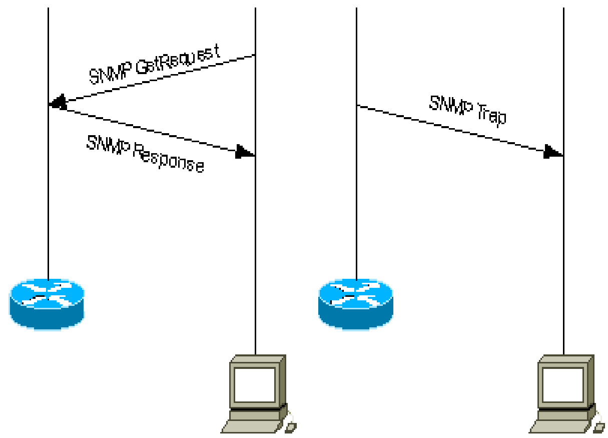

Fig. 1 The two setups of the network management system

Among the SNMP commands are specific protocol operations that facilitate in the requests and responses of managed network devices. The most basic operations include: Get, GetNext, Set, and Trap (see Fig. 2)

GetRequest: A Get message is sent by a manager to an agent to request the value of a specific OID. This request is answered with a Response message that is sent back to the manager with the data.

GetNextRequest: A GetNext message allows a manager to request the next sequential object in the MIB. This is a way that you can traverse the structure of the MIB without worrying about what OIDs to query

SetRequest: A Set message is sent by a manager to an agent in order to change the value held by a variable on the agent. This can be used to control configuration information or otherwise modify the state of remote hosts. This is the only write operation defined by the protocol..

GetResponse: This message, sent by an agent, is used to send any requested information back to the manager. It serves as both a transport for the data requested, as well as an acknowledgement of receipt of the request. If the requested data cannot be returned, the response contains error fields that can be set with further information. A response message must be returned for any of the above requests, as well as Inform messages.

Trap: A trap message is generally sent by an agent to a manager. Traps are asynchronous notifications in that they are unsolicited by the manager receiving them. They are mainly used by agents to inform managers of events that are happening on their managed devices.

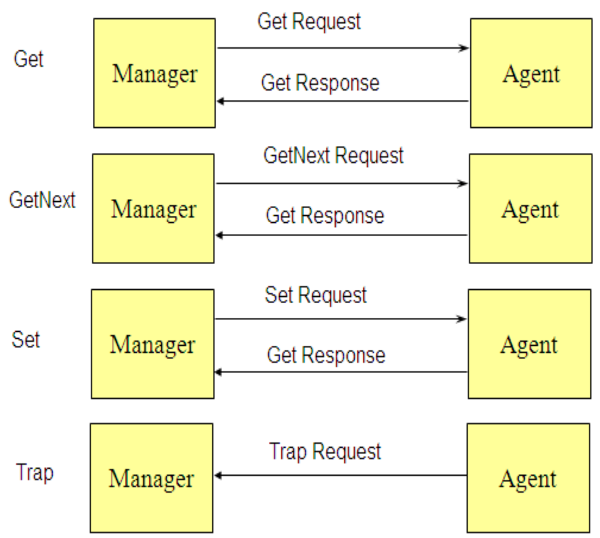

Fig. 2 The permitted operations between managers and agents

As an example: an SNMP manager requests configuration information for a particular system. The manager formats this request in a GET protocol data unit (PDU) and transmits the request to the agent using a communication service. After the manager's request has been received, the agent packages the requested MIB object information in a RESPONSE PDU and transmits it back to the manager

IV. Modeling of SNMP Protocol

A. Initial Model

The first model is the most abstract specification of the system.

We can use two variables to represent the state of the initial model: *reqt* to denote the number of requests that have been sent, and *resp* to indicate the number of responses that have been given.

We have three invariants: inv1 and inv2 denotes that the two variables *reqt* and *resp* are natural numbers. inv3 specifies that the communication is synchronous: either the number of requests is the same as the number of responses or it is greater than the number of responses by 1 in the case where a response is expected before another request can be created.

VARIABLES
reqt
resp
INVARIANTS
inv1 : reqt $\in \mathbb{N}$
inv2 : resp $\in \mathbb{N}$
inv3 : reqt=resp $\vee$ reqt=resp+1

Initially, there are no requests or responses hence both variables are initialed by 0.

INITIALISATION
act1 : resp:=0
act2 : reqt:=0

Finally, we define two events in our abstract model. An event **Manager_request** represents the sending request from the manager to the agent, starts when the number of requests and the number of responses are identical and increases the number of requests by 1. An event **Agent_response** represents the response sent from the agent to the manager, guards of this event state that the number of requests and responses are different.

Manager_request
WHEN
grd1 : reqt=resp
THEN
act1 : reqt:=reqt+1
END

Agent_response
WHEN
grd1 : reqt≠resp
THEN
act1 : resp:=resp+1
END

B. First Refinement

First, we define three carrier sets:

Requests: set of messages which can be sent by the manager, it contains three constants (GetRequest, GetNextRequest and SetRequest) defined by the axioms (axm1, axm2 and axm3).

Responses: set of responses sent by the Agent, it contains the constant GetResponse which represented by the axiom (axm4).

Notification: set of messages sent by the Agent to inform the Manager. The axiom (axm5) represent that this set contains the constant Trap.

AXIOMS
axm1 : GetRequest $\in$ Requests
axm2 : GetNextRequest $\in$ Requests
axm3 : SetRequest $\in$ Requests
axm4 : GetResponse $\in$ Responses

axm5 : Trap ∈ Notification

In this first refinement, we introduce the channels and the messages sent between the manager and the agent, because in the reality the message needs to be sent via some channel between two parties.

We add three variables **reqtChan**, **respChan** and **notifChan** which represent respectively the channel of messages sent by the manager, the channel of messages sent by the agent and the channel of messages sent by the Agent to inform the Manager.

INVARIANTS
inv1 : reqtChan ⊆ Requests
inv2 : respChan ⊆ Responses
inv3 : notifChan ⊆ Notification

We define now our events:
Manager_send_request: refining the abstract event Manager_request: the manager sends a message to the agent.
Agent_receive_request: the agent receives the request sent by the manager.
Agent_send_response refining the abstract event Agent_response: after receiving the request, the agent sends a response to the manager.
Manager_receive_response: the manager receives the response sent by the agent.
Notify: the agent can send a trap, or asynchronous notification, to the manager

Manager_send_request
REFINES
Manager_request
ANY msg WHERE
grd1 : reqt=resp
grd2 : msg ∈ Requests
grd3 : msg ∉ reqtChan
THEN
act1 : reqt≔reqt+1
act2 : reqtChan ≔ reqtChan ∪ {msg}
END

Agent_receive_request
ANY msg WHERE
grd1 : msg ∈ reqtChan
THEN
act1 : reqtChan≔ reqtChan \ {msg}
END

Agent_send_response
REFINES
Agent_response
ANY msg WHERE
grd1 : reqt≠resp
grd2 : msg ∈ Responses
grd3 : msg ∉ respChan
THEN
act1 : resp≔resp+1
act2 : respChan ≔ respChan ∪ {msg}
END

Manager_receive_response
ANY msg WHERE
grd1 : msg ∈ respChan
THEN
act1 : respChan ≔ respChan \ {msg}
END
Notify
ANY msg WHERE
grd1 : msg ∈ Notification
THEN
act1 : notiChan ≔ notiChan ∪ {msg}
END

C. Second Refinement

In this refinement, the overtime retransmission mechanism is added to ensure the correctness and the completeness of the data transmission. This means that a request from the manager may not arrive at the agent, and the agent's reply may not make it back to the manager. The manager probably wants to implement a timeout and retransmission.

We need a new constant, *T_OUT,* which is the maximum waiting time for the Manager. We add two new variables: *CurrentTime* and *time*. *CurrentTime* is a variable which stands for the current time and *time* is used to record the time when the Manager sends a message to the Agent.

INVARIANTS
inv1 : time ∈ ℕ
inv2 : CurrentTime ∈ ℕ

Concerning Events, we refine the two events *Manager_send_request* and *Manager_receive_response*. We add also two new events: Resend and Clock.

If the event *Manager_receive_response* has not happened before the set time, the event Resend will happen and the message will be resent again. For the event *Manager_receive_response* the refinement is just a superposition, time constraints are added without changing the existing expressions. If the Manager starts to send a message, it takes a propagation time to progress in the channel. If the transmission is successful, the propagation time should be shorter than *T_OUT.*

Manager_send_request
REFINES
Manager_send_request
WHERE
grd4 : CurrentTime < time+T_OUT
THEN
act4 : time≔CurrentTime

END

Manager_receive_response
REFINES
Manager_receive_response
grd3 : CurrentTime< time+T_OUT
THEN
END

Resend
REFINES
Manager_send_request
grd4 : CurrentTime> time+T_OUT
THEN
act4 : time:=CurrentTime
END

Clock
BEGIN
act1 : CurrentTime:=CurrentTime+1
END

V. Conclusion

In this paper, we have modeled and proved SNMP protocol using Event-B.

We have explained our approach using refinement, which allows us to achieve a very high degree of automatic proof. The powerful support is provided by the Rodin tool. Rodin proof is used to generate the proof obligations and to discharge those obligations automatically and interactively.

Modeling and analyzing SNMP specification using formal methods can help in assuring correctness, unambiguity, and clarity of the SNMP protocol. Since a well-defined and verified protocol specification can reduce the cost for its implementation and maintenance, modeling and analysis are important steps of the protocol development life-cycle from the point view of protocol engineering.

.

References

[1] R. Filali ,S. El Mimouni ,A Amamou,B. Boulamat and M. Bouhdahi, "Modeling of SNMP protocol in Event-B ", Proceedings of the 1st International Conference on Mathematical Methods & Computational Techniques in Science & Engineering (MMCTSE 2014), pp.208-211

[2] Woodside, C.M., "Performance Petri net analysis of communications protocol software by delay-equivalent aggregation," In Petri Nets and Performance Models, pp. 64-73, 1991.

[3] Antonidakis, E. "Conferencing protocols and petri net analysis", WSEAS Transactions on Computers, vol. 5, no 12, pp. 3112-3118, 2006

[4] Bochmann, G. "Formal Methods in Communication Protocol Design," IEEE Transactions on Communication, vol.28, pp. 624-631, 1980.

[5] Al Dallal, J., "Automatic synthesis of timed protocol specifications from service specifications." WSEAS Transactions on Computers 5.1 (2006): 105-112.

[6] Abrial, J.R., Modeling in Event-B: system and software engineering, Cambridge University Press, 2010.

[7] Li, X. B., and Zhao, F. X.. "Formal development of a washing machine controller by using formal design patterns." WSEAS International Conference. Proceedings. Mathematics and Computers in Science and Engineering. Ed. Lifent Xi. No. 3. World Scientific and Engineering Academy and Society, 2009.

[8] Abrial, J.R., The B-book: assigning programs to meaning, Cambridge University Press. 2005.

[9] Abrial, J.R., "B#: Toward a synthesis between Z and B," In: ZB 2003: Formal Specification and Development in Z and B, Springer Berlin Heidelberg, pp. 168-177, 2003.

[10] Back, R.J., On the correctness of refinement steps in program development, Department of Computer Science, University of Helsinki, 1978.

[11] Jones, C., Oliver, I., Romanovsky, A., and Troubitsyna, E., RODIN (rigorous open development environment for complex systems), University of Newcastle upon Tyne, Computing Science, 2005.

[12] Case, J., Fedor, M., Schoffstall, M., and Davin, J., "RFC 1157: Simple network management protocol (SNMP)," IETF, April, 1990