Event Ordering Using Graphical Notation for Event-B Models

Rahul Karmakar[1(✉)], Bidyut Biman Sarkar[2], and Nabendu Chaki[3]

[1] The University of Burdwan, Burdwan, India
rkarmakar@cs.buruniv.ac.in
[2] Techno International, Rajarhat, Kolkata, India
bidyutbiman@gmail.com
[3] University of Calcutta, Kolkata, India
nabendu@ieee.org

Abstract. System requirements are sometimes either too complex or undefined. Event-B is a formal modeling method and is being used increasingly to model various systems. Event-B models support atomicity decomposition and are quite useful for complex refinement structures. However, neither a Event-B model represents any explicit control flows among the events, nor does it support links between the new events during refinements. This work aims to model the Stop and Wait mechanism for an Automatic Repeat Request (ARQ) protocol to analyze the complexities due to communication errors during data re-transmissions. The limitation is the lack of control flows among the events during successive refinements. This has been graphically represented in this work and embedded with Event-B notations for the atomicity decomposition of the model. Finally, the successive refinements presented using an Event-B model, has been validated using the Rodin tool. This leads to a successful ARQ model.

Keywords: Event-B · Formal modeling · Stop and wait ARQ · RODIN tool · Atomicity decomposition · ERS diagram

1 Introduction

Model-based verification techniques describe the system behaviour in a mathematical and unambiguous fashion [1]. Event-B modeling language is devised as an extension of classical B methods, which has different applications in diverse domains. It is a step by step process of system development. We start with an abstract model and refine the model successively to meet the requirements. An Event-B model has a static component called context, where we declare all the sets, constants and axioms. The dynamic part is the machine that sees the context. A machine has variables and invariants. The state changes of a machine are defined by guards and actions, which is called event [2]. Rodin [3] is a tool support for the validation of an Event-B model. The control flow between events cannot be handled explicitly in Event-B as it does not accept ordering of the events.

K. Saeed and J. Dvorský (Eds.): CISIM 2020, LNCS 12133, pp. 377–389, 2020.
https://doi.org/10.1007/978-3-030-47679-3_32

However, it can be managed implicitly by Event-B. Event-B modeling allows to refine a model incrementally. When a new event is introduced in a refinement stage, we could not link externally between the new and the abstract event. Atomicity decomposition of a model is supported by Event-B. The relationship between the events (atomicity) is a very important aspect to maintain, when we design a large and complex system. A case study on stop and wait for the ARQ technique with atomicity decomposition of the model is presented in this paper. It is a layer 2 flow control mechanism for data communication. Similar applications are found using Event-B [4] and Petri Net [5]. The basic concept of atomicity decomposition and model decomposition are explained below.

The atomicity decomposition technique is described with a brief overview. Figure 1 represents the explicit relationship among the events A to F. Event A is the abstract event and three events B, C and D are the children of A in the tree like structure. The event B does not refine the event A and represented by the dashed line. Events C and D refine the event A and the refinement relationship is represented by the solid lines [6]. Another aspect of the diagram is that, it implies event B will execute before event C and C will execute before event D. Thus, the ordering among the events (B, C, D) is also represented by the diagram. This ordering can be established by Event-B notations. The * between A and C signifies multiple instances of C, which means event C can execute multiple times. Different constructors can be used to represent event relationships. We get some of the representations from Fig. 1. Event E and F are exclusive to each other, which is represented by XOR constructor. Constructors like AND, OR can also be used to represent the relationships between Events. We can decompose the model with graphical notations and then implement the explicit ordering using Event-B notations.

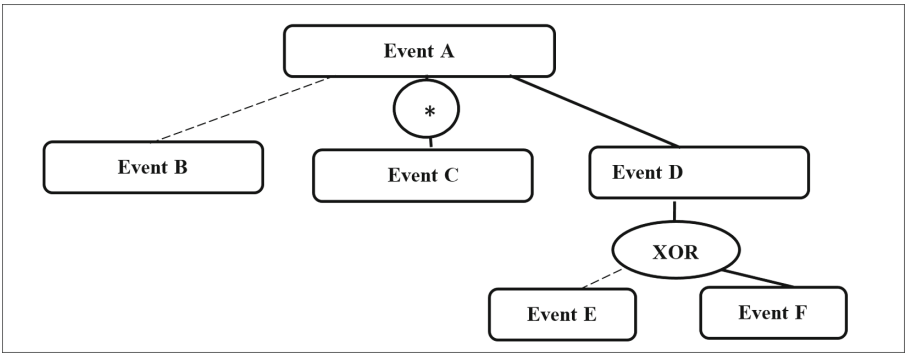

Fig. 1. Atomicity decomposition diagram.

Model decomposition technique decreases the complexity of large system and represented in Fig. 2. The large model can be divided into sub-models and refined individually. The events and variables are shared among the sub models for

distributed and concurrent systems [7]. The shared event and variable are represented in Fig. 2. There are 3 events A, B, and C. Events X and Y share the variables A and event Y and Z share the event B. The machine M is divided into two submachine M1 and M2. Both the submachines share the event Y. This paper present an Event-B modeling approach using a graphical notation introduced as in Jackson System Development JSD [6]. This decomposition structure is explained in [8,9]. We could incorporate explicit ordering between the events using JSD graphical notations. This ordering has improved and enriched the conventional Event-B modeling. Refinements are also represented by the graphical structure. These two approaches help us to design the ARQ system in a more flexible manner, especially when re-transmission takes place due to frame loss or acknowledgement message loss. The paper is structured as follows, in Sect. 2 we present an overview of stop and wait ARQ technique. Section 3 presents a brief survey of the related works. Section 4 presents the ERS of stop and wait for ARQ in detail followed by some observations in Sect. 5. The concluding remarks highlighting the potential of the proposed work in shaping up paths for future research directions are presented in Sect. 6.

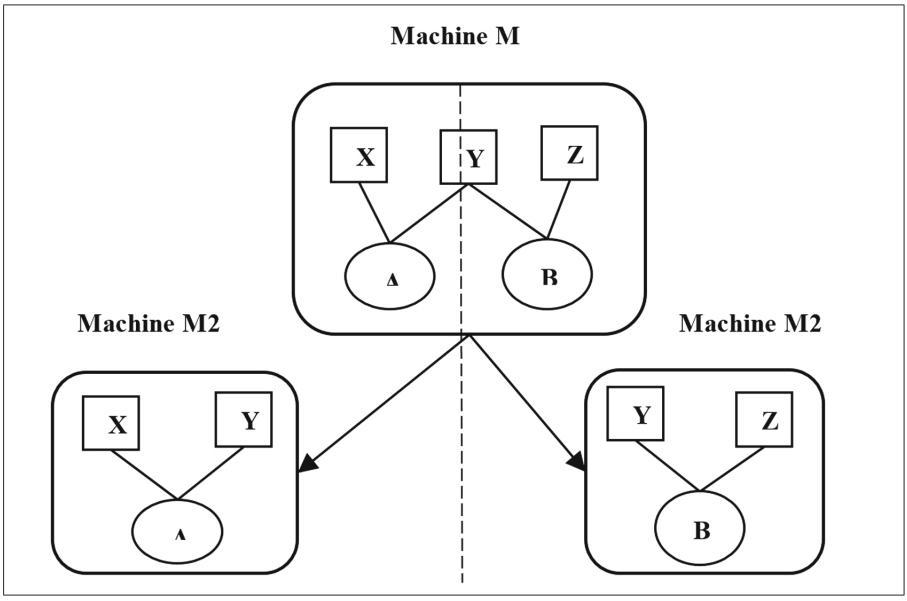

Fig. 2. Model decomposition

2 An Overview of ARQ Protocol and Requirements

Flow control is one of the important design issues in the data link layer when the speed mismatch happens between the sender and receiver. It controls the

rate of the frames transmitted to the receiver. Figure 3(a) represents the normal operation of Stop and Wait for ARQ mechanism. The sender sends a packet and waits for the acknowledgment from the receiver within a specified time to ensure the successful transmission. The control variables S and R have the current value of the frame either 0 or 1. Timeout is an important aspect of this technique. Suppose the acknowledgment is not received within the specified time due to delayed acknowledgment showed in Fig. 3(b) or lost acknowledgment showed in Fig. 3(d) then the duplicate copy of the frame is sent by the sender after the timeout. The duplicate frame is then discarded at the receiving end in case of delayed acknowledgment showed in Fig. 3(b) and the duplicate acknowledgment is discarded at the sender's side in case of acknowledgment lost showed in Fig. 3(d). The duplication is identified by the control variable values. Figure 3(c) represents how transmission takes place when a frame is lost. The sender retransmits the frame after the timeout and accepted by the receiver [10]. The whole operation is represented using ERS in Sect. 4.

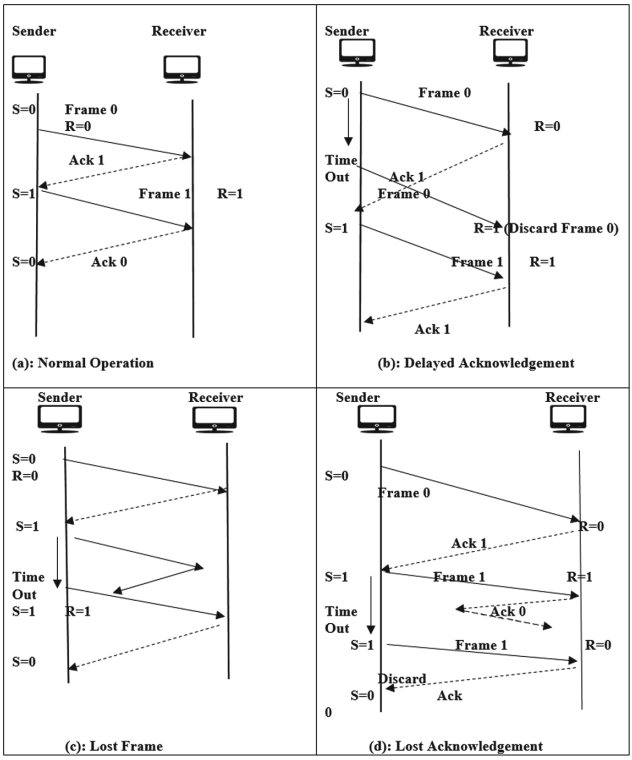

Fig. 3. Stop and Wait ARQ operation

3 Related Work

There have been works [8] proposing additional structuring to augment Event-B notation for atomicity decomposition of a complex refinement. Model decomposition is also presented using some case studies. Event Refinement Structure (ERS) was proposed by Butler [8]. It is a graphical notation based on JSD [6]. The relationships between the events are presented graphically to implement the Event-B notations. A technique is proposed to decompose a machine into sub-machines and those are refined independently. The paper addresses two important aspects of system development using Event-B i.e., atomicity decomposition and model decomposition using graphical notations. These are explained in the section of this manuscript. Atomicity decomposition of the Multimedia Protocol using Event-B is addressed in the paper [11]. It represents a protocol of media Channel System that establishes, modify and closes the channel by the communication parties. They also compared the model with the spin model checker. Further decompositions are performed using guards and events instead of sequential decomposition, which will be more useful for a complex system. No automatic model builder is used in this work. The main goal of the BepiColombo mission [7] had been to explore the planet Mercury. The whole system is controlled by the Mission-Critical-Software (MCS). The MCS controls the earth and also the device. It checks the Telecommand (TC) received from the earth and then validates the TC. TC is a control message and there are many types of TCs in the system. The atomicity, decomposition, and model decomposition are implemented using Event-B to handle this complex system. They validate the model using the RODIN tool. It shows how atomicity decomposition and model decomposition are effectively used to handle the control behavior manually. A plug-in tool Event Refinement Structure (ERS) is developed [9]. This tool automatically constructs the consistent Event-B model with control flows and refinement relationships. A context-free grammar notation Augmented Backus Normal Form (ABNF) is introduced to describe the ERS language syntax. Some of the ABNF features are flow, par, child, constructor, etc. Altogether 19 translation rules from ERS to Event-B are formed. All the constructors like a loop, logical XOR, replicator are also described. The development architecture is like; they define the ERS language specification in the Eclipse Modelling Framework (EMF) Meta-model. The source Meta-model is then transformed into the Event-B EMF target meta-model. The transformation from the target meta-model to Event-B is performed by a rule-based model-to-model transformation language called Epsilon Transformation Language (ETL). The ERS tool is then compared against the BepiColombo system [9] and the Multimedia Protocol [11]. It is found that the total proofs from the systems in [11] and [7] are substantially reduced. Here, the ERS tool does not provide a graphical environment of the ERS diagrams. The ERS diagrams are represented as an EMF model and manipulated by the EMF structure editor. More translations rules can be implemented for the ERS language. The improved version of the ERS tool provides a graphical environment for the ERS [12]. The Generic Diagram Extension Framework approach is proposed to transform ERS to Event-B. It has graphical and

validation support for the model whereas the ERS tool needs another tool for model validation. It is a Java-based tool useful for complex case studies and validation of models. Authors claimed that more translation rules can be added in the future to add application-specific guards, actions, and invariants in the ERS environment without switching to Event-B editor. Another work [4], describes the stop and wait (SAW) technique is implemented by Event-B and verified by the UPPAAL model checker. They provide the mapping between Event-B to UPPAAL. The SAW model is implemented manually by Event-B and checked using RODIN then verifies the model using UPPAAL. The authors handle the complexity of the protocol with a single machine. Different cases of retransmission of message and acknowledgment are not presented clearly. These cases can be represented by the refinement steps. We find ERS as a very useful approach to model a system requirement using Event-B, from many of the existing works that we studied. The atomicity decomposition and model decomposition can be applied to a communication protocol. Event-B is a formal method for system-level modeling and analysis. This is often used to represent a system at different abstraction levels and for formal verification of consistency between refinement levels. However, there is no formal graphical representation for complex system refinements. This paper represents the stop and wait technique with ERS and establish the explicit ordering between events. Subsequently, these graphical notations are translated to Event-B notations. This helps to handle the complex behavior of the system effectively. The comparison of the complexity and flexibility of the approach with the non-ERS design-based approach has been done for justification.

4 Atomicity Decompositions of Stop and Wait Protocol

4.1 Abstract Specification: Basic Operations of SAW

This is the abstract representation of the Stop and Wait operations. The requirement goal is to establish sequencing among the events and the refinement relationships. The graphical representation of the abstract system is represented in Fig. 4 with five events. As discussed in Sect. 1, *Sender_Send_Request*, *Receiver_Receive_Request*, *Receiver_Send_Ack* and *Sender_Receiver_Ack* refine the abstract event *Stop_and_Wait_ARQ* and these are in the sequence to complete the basic Stop and Wait ARQ operation i.e. *Sender_Send_Request* is followed by *Receiver_Receive_Request*, followed by *Receiver_Send_Ack*. The requirement properties can be established with the rules given below. The ordering and refinement relationship between two events are established by using subset property. The relation between set variable of the abstract event and the concrete variable of the refined event can be represented as:

Preceding Abstract Event (variable name is same as the event name) $\subseteq$ Succeeding Refined Event (variable name is same as the event name) It may be defined in Event-B by giving the same name of the variables with the events and established as an invariant property [7]. These properties are held while

designing the whole system. The sequencing and refinement relationships among the events are showed in Fig. 4. This can be described as the Event-B properties 1 to 4 in Table 1. Property 1 ensures that the *Sender_Send_Request* can send multiple request. Hence, the variable *Sender_Send_Request* is a subset of the set Request. Four other scenarios are shown in Fig. 5(a), (b), and 6(a), (b). The successful receiving of the request by the receiver is checked. This decomposition is represented in Fig. 5(a) and in 5(b). The *Receiver_Receive_Fail* event is used for the purpose. The event relationship can be represented by the Event-B properties 5 and 7 in Table 1. The event *Sender_Receive_Fail* is used when the sender did not receive the acknowledgment sent by the receiver represented in Fig. 6(a) and (b). This relation can be represented by Event-B properties 6 and 8 in Table 1. These two events refine the abstract event *Stop_and_Wait_ARQ* and are represented by solid lines between them. The disjoint relationship between *Receiver_Receive_Request* and *Receiver_Receive_Fail* can be established using the intersection property Event X $\cap$ Event Y $= \emptyset$ and given as Event-B property 7. Property 8 established the same relation between *Sender_Receive_Ack* and *Sender_Receive_Fail*. The four events in Fig. 4, *Sender_Send_Request*, *Receiver_Receive_Request*, *Receiver_Send_Ack* and *Sender_Receiver_Ack* refine the event Stop_ and_ Wait _ARQ. The refinement relationship between the Stop_ and_ Wait _ARQ and the *Sender_Send_Request* showed in Fig. 5(a) and 5(b) for successful sending of request from sender to receiver can be modelled using Event-B machine given below.

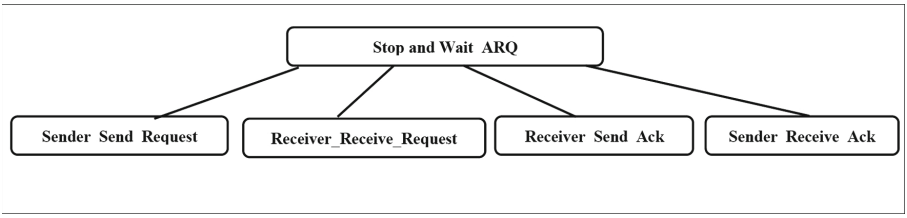

Fig. 4. Normal operations of stop and wait ARQ.

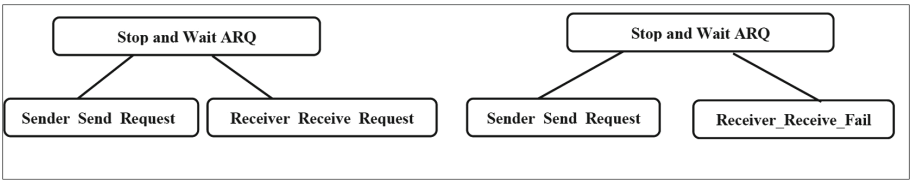

Fig. 5. (a)(b) Sender receives request or Sender receives fail.

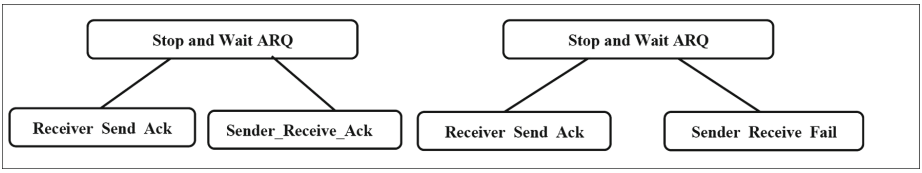

Fig. 6. (a)(b)Sender receives ACK or Sender receives ACK

Table 1. Variables and Properties

Variables	Event relationship Properties (P)
Sender_Send_Request	P1:*Sender_Send_Request*⊆Request
Receiver_Receive_Request	P2:*Receiver_Receive_Request*⊆*Sender_Send_Request*
Receiver_Send_Ack	P3:*Receiver_Send_Ack*⊆*Receiver_Receive_Request*
Sender_Receive_Ack	P4:*Sender_Receive_Ack*⊆*Receiver_Send_Ack*
Receiver_Receive_Fail	P5:*Receiver_Receive_Fail*⊆*Sender_Send_Request*
Sender_Receive_Fail	P6:*Sender_Receive_Fail*⊆*Receiver_Send_Ack*
	P7:*Receiver_Receive_Request*∩*Receiver_Receive_Fail*= ∅
	P8:*Sender_Receive_Ack*∩*Sender_Receive_Fail*= ∅

Sender_Send_Request
refines
Stop_and_Wait_ARQ
ANY
Request
WHERE
*grd*1: *Req* ∈ Request \ *Sender_Send_Request*
THEN
*act*1: *Sender_Send_Request* := *Sender_Send_Request* ∪ Request
END

The *Sender_Send_Request* event refines *Stop_and_Wait_ARQ* event. The Stop and Wait ARQ system can send and receive multiple requests and acknowledgements so Request and ACK are considered as sets and Request and Ack are used as a parameter. The guard ensures that the event *Sender_Send_Request* has not occurred with the current Request. The action of the event defines the inclusion of the new Request to the variable *Receiver_Receive_Request*. The Event relationship between *Receiver_Receive_Request* or *Receiver_Request_Fail* with Stop_ and _Wait _ARQ showed in Fig. 5(a) and (b) can be represented below.

Receiver_Receive_Request
refines
Stop_and_Wait_ARQ
ANY

Request
WHERE
$grd1$: $Request \in Sender\_Send\_Request \setminus (Sender\_Send\_Request \cup Sender\_Receive\_Fail)$
THEN
$act1$: $Receiver\_Receive\_Request := Receiver\_Receive\_Request \cup$ Request
END

The guard ensured the event $Sender\_Send\_Request$ is executed before $Sender\_Send\_Request$ or $Sender\_Receive\_Fail$ event. The events can be modelled towards successful receiving of the acknowledgment showed in Fig. 6(a) and 6(b) from Receiver to Sender. This is given below.

$Receiver\_Send\_Ack$
refines
$Stop\_and\_Wait\_ARQ$
ANY
Ack
WHERE
grd1: Ack $\in Receiver\_Receive\_Request \setminus Receiver\_Send\_Ack$
THEN
act1: $Receiver\_Send\_Ack := Receiver\_Send\_Ack \cup$ Ack
END

The $Sender\_Receive\_Ack$ event can be modelled given below with Event-B notations.

$Sender\_Receive\_Ack$
Refines
$Stop\_and\_Wait\_ARQ$
ANY
Ack
WHERE
grd1: Ack $\in Receiver\_Send\_Ack \setminus (Receiver\_Send\_Ack \cup Receiver\_Receive\_Fail)$
THEN
act1: $Sender\_Receive\_Ack := Sender\_Receive\_Ack \cup$ Ack
END

Confirm sending and receiving by the Sender and Receiver: Four new events $Sender\_Ready$, $Timer\_Set$, Req_Seq_No and $Wait\_Ack$ are introduced to complete the operation $Sender\_Send\_Request$ depicted in Fig. 7. These events will not refine the event $Sender\_Send\_Request$ and represented with dashed lines. The orderings among the events ensure the successful sending the request from sender's side to receiver. The $Receiver\_Receive\_Request$, $Receiver\_Send\_ACK$ and $Sender\_Receive\_ACK$ events are represented in

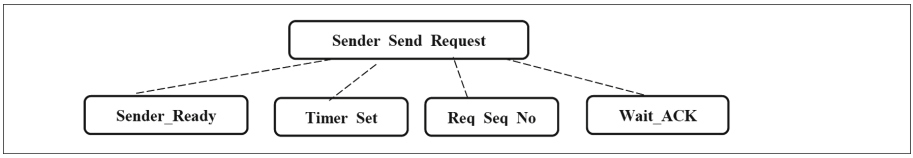

Fig. 7. Sender sends request to the receiver.

Fig. 8(a)(b) and 9. All the new events skip the refinements and their sequencing ensures the successful receiving the request by the receiver and Ack by the sender.

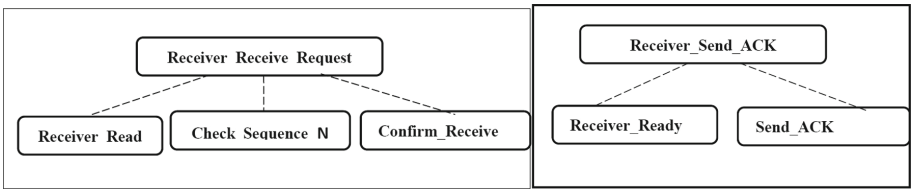

Fig. 8. (a)(b)Receiver receives the request and Receiver sends the Ack

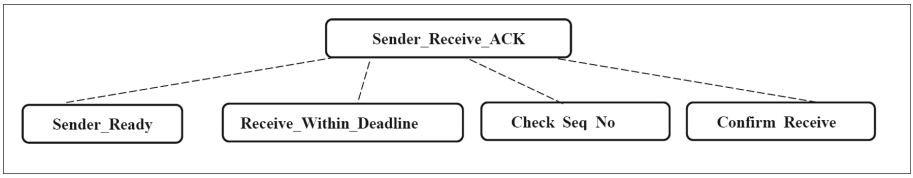

Fig. 9. Resend duplicate frame when ACK is lost (Regiment 1)

4.2 Refinement 1: Resending Request/Ack

As discussed in Sect. 2 all the resending scenarios of Request and Acknowledgement are represented with the ERS diagram as a refinement of the previous abstract model. Figure 10 represents resending of the old frame with three events *Sender_Ready*, *Not_Receive_Within_Time* and *Resend_Old_Request*. Only the *Resend_Old_Request* event refines the *Sender_Receive_Request*. This relationship between the events can be implemented by the subset relation between the concrete variable *Resend_old_Request* with the set variable *Sender_Receive_Ack*. The resending of duplicate frame, when the acknowledgment is lost from receiver to sender, is represented by the ERS notations in Fig. 11. There are four events *Sender_Ready*, *Not_Receive_Within_Time*, *Receive_Old_ACK*, and *Resend_Old_Request* which skip the refinement and the *Resend_Old_Request* refines the abstract event *Sender_Receive_ACK*. The refinement relationship

between *Resend_Old_Request* and *Sender_Receive_ACK* can be represented by the property *Resend_Old_equest* $\subseteq$ *Sender_Receive_ACK*. The resending of duplicate acknowledgment operation was discussed in Sect. 2. When a particular frame is not received in time or the duplicate frame is received, then the corresponding acknowledgment is resent. The ERS in Fig. 12 showed the operation and the relationship between events. *Receiver_Ready* and *Not_Receive_Request* or *Receive_Old_Request* events did not refine the abstract event *Receiver_Receive_Ready* and represented by dashed lines. The *Resend_ACK* event refines *Receiver_Receive_Request* and represented by the solid line. The property is represented as *Resend_ACK* $\subseteq$ *Receiver_Receiver_Request*.

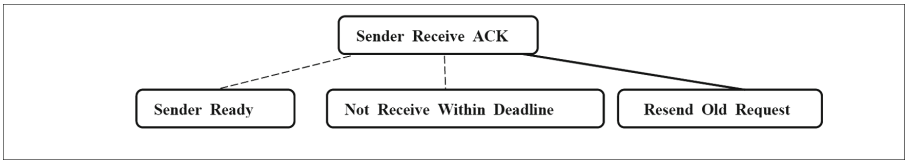

Fig. 10. Resend duplicate frame when ACK is lost (Regiment 1)

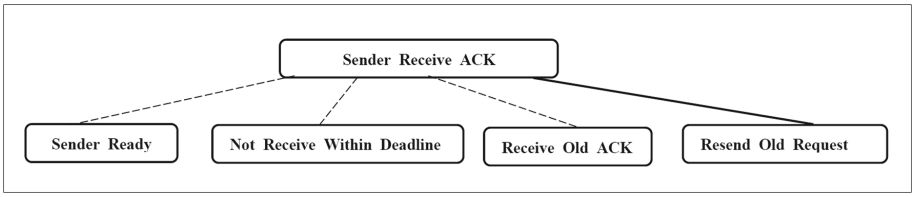

Fig. 11. Resend duplicate frame when ACK is delayed (Regiment 1)

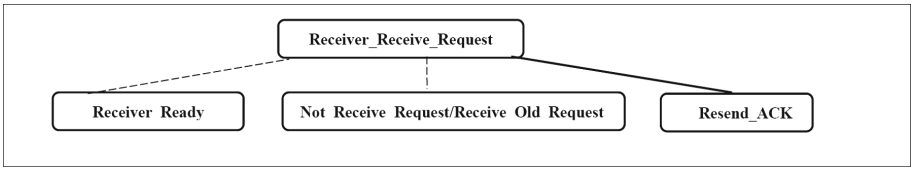

Fig. 12. Resend duplicate ACK when the frame is lost

5 Critical Observations

The atomicity decomposition is done at the stop and wait for the ARQ system in the proposed approach. The system design represents all the events and their relationship with graphical notations. This is the major contribution in this work that has eased the process of elicitation and helps to understand the system more unambiguously. This eventually helps efficient usage of formal notations to represent and design a complex system. The orderings of events are represented explicitly in this proposed methodology. All the refinement relationships between events represented before system design. Besides, in the proposed extension of the Event-B model, this technique helps to design the resending operations of frames and acknowledgments more effectively. As for example, in Fig. 10, the *Sender_Ready* event does not refine the abstract event *Sender_Receive_ACK*. The relationship is represented by the dashed lines and helps to covert the relations into Event-B notations. Relationships between events are represented by invariant properties. All the ERS structures shown in Sect. 4 are converted into Event-B notations. The model may be validated by the RODIN model checker. This technique is more flexible and less complex than the ARQ protocol designed using Petri Net because of the graphical representations of the events with the order [5]. All the Event-B notations described above will make SAW system modeling unambiguous and flexible.

6 Conclusions

The model, under evaluation, may further be decomposed into sub-models. The events and variables could be decomposed accordingly. The ERS structure not only establishes the atomicity decomposition of an Event-B model but is found to be quite useful to detect wrong atomicity decompositions. It can be detected using the proposed methodology because the system is represented graphically. An appropriate tool-support may also be developed to provide a comprehensive system to verify the event refinement structure automatically. The conversion of the ERS notations into Event-B is done manually in the proposed work. A tool may be developed in future for the automatic conversion of ERS into Event-B for complex system modeling. The translation rules can be designed effectively for the automatic development starting from a semi-formal requirements description using the graphical notation used.

References

1. Baier, C., Katoen, J.-P.: Principles of model checking. MIT Press, Cambridge (2008)
2. Abrial, J.-R.: Modeling in Event-B: System and Software Engineering. Cambridge University Press, Cambridge (2010)
3. Jastram, M., Butler, M.: Rodin User's Handbook: Covers Rodin v.2.8. CreateSpace, Scotts Valley (2014)

4. Filali, R., Bouhdadi, M.: Formal modeling and verification of time-constrained ARQ protocols with Event-B. Int. J. Eng. Technol. **8**, 1807–1816 (2016)
5. Best, E., Devillers, R., Koutny, M.: Petri Net Algebra. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-662-04457-5
6. Floyd, C.: A comparative evaluation of system development methods. In: Proceedings of the IFIP WG 8.1 Working Conference on Information Systems Design Methodologies: Improving the Practice, pp. 19–54. North-Holland Publishing Co. (1986)
7. Salehi Fathabadi, A., Rezazadeh, A., Butler, M.: Applying atomicity and model decomposition to a space craft system in Event-B. In: Bobaru, M., Havelund, K., Holzmann, G.J., Joshi, R. (eds.) NFM 2011. LNCS, vol. 6617, pp. 328–342. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-20398-5_24
8. Butler, M.: Decomposition structures for Event-B. In: Leuschel, M., Wehrheim, H. (eds.) IFM 2009. LNCS, vol. 5423, pp. 20–38. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00255-7_2
9. Salehi Fathabadi, A., Butler, M., Rezazadeh, A.: Language and tool support for event refinement structures in Event-B. Formal Aspects Comput. **27**(3), 499–523 (2014). https://doi.org/10.1007/s00165-014-0311-1
10. Forouzan, A.B.: Data Communications & Networking (sie). Tata McGraw-Hill Education, New York (2007)
11. Salehi Fathabadi, A., Butler, M.: Applying Event-B atomicity decomposition to a multi media protocol. In: de Boer, F.S., Bonsangue, M.M., Hallerstede, S., Leuschel, M. (eds.) FMCO 2009. LNCS, vol. 6286, pp. 89–104. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17071-3_5
12. Dghaym, D., Trindade, M.G., Butler, M., Fathabadi, A.S.: A graphical tool for event refinement structures in Event-B. In: Butler, M., Schewe, K.-D., Mashkoor, A., Biro, M. (eds.) ABZ 2016. LNCS, vol. 9675, pp. 269–274. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-33600-8_20