Applying Event-B Atomicity Decomposition to a Multi Media Protocol

Asieh Salehi Fathabadi and Michael Butler

University of Southampton
asf08r,mjb@ecs.soton.ac.uk

Abstract. Atomicity Decomposition is a technique in the Event-B formal method, which augments Event-B refinement with additional structuring in a diagrammatic notation to support complex refinement in Event-B. This paper presents an evaluation of Event-B atomicity decomposition technique in modeling a multi media case study with the diagrammatic notation. Firstly the existing technique and the diagrammatic notation are shown. Secondly an evaluation is performed by developing a model of a Media Channel System. A Media Channel is established between two endpoints for transferring multi-media data. Finally some extensions to the existing diagrammatic notation are proposed and applied to the multi-media case study.

Keywords: Event-B, Refinement, Atomicity Decomposition, Structured Event Refinement.

1 Introduction

Event-B [1, 2] is a formal method that uses the concept of refinement [3, 4] in modeling. Event-B modeling starts with an abstraction of a system and adds details during refinement levels in order to gain a final model close to the implementation. Moreover mathematical proofs are incorporated into Event-B to verify the correctness of refinement steps.

The most important benefit of using Event-B is its capability to use abstraction and refinement. In this approach the modeling process starts with an abstraction of the system which speifies the goals of the system. In our case study, a media channel system, establishing and modifying the established channel are the main system goals. 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 tool support is another benefit of using Event-B. The Rodin tool [5] supports proof obligation generation and automated proof. Through a refinement approach, we prove that the abstract goals concerning establishment and modification of media channels are satisfied by the detailed protocol. In the developed Event-B models of the media channel system reported here, all proofs are generated and discharged by the Rodin tool.

F.S. de Boer et al. (Eds.): FMCO 2009, LNCS 6286, pp. 89–104, 2010.

Modeling of large and complex systems can result in large and complex models and difficult proofs [6]. Refinement techniques can address this complexity. In Event-B refinement, rather than having a single large model, it is common to represent a desired outcome as an abstract atomic event and then decompose that into smaller sub-events in subsequent refinement levels. If the abstraction gaps between refinement levels are small, it means relatively small details are added in each refinement level and proof obligations would be relatively easy to discharge. Most of proof obligations are related to consistency between refinement levels, so with the small gaps these proofs become easier to discharge. This will be explained more in the next section when we introduce invariants.

Although refinement offers the advantages outlined above, the Event-B refinement method does not explicitly represent all refinement connections between abstract and concrete events. *Atomicity decomposition* diagrams provide a structuring technique which addresses this through a diagrammatic notation. The atomicity decomposition technique helps to structure refinement in Event-B. This technique is introduced in [7]. It is intended to make the standard refinement rules clearer and their application more systematic. In Event-B refinement there is no clear connection between certain actions of different refinement levels. The diagrammatic notation of atomicity decomposition shows relationships between refinement levels. In this approach usually a single event shows the goal in the abstract level, and then it is decomposed to sub-events in refinement.

The contribution of this paper is applying existing Event-B atomicity decomposition technique to a multi media case study. An evaluation of this technique in modeling the multi media system is presented. There are several contributions in this evaluation. First we will see how system goals are modeled in the abstract level with single events. Then details of the protocol are added gradually during refinement levels. For applying these details we will see how the atomicity decomposition diagrammatic notation will help us to structure refinement in an explicit way. Finally this development leads to discharge of all proof obligations using the Rodin tool-set.

In this paper after a short background about Event-B, we will explore how the diagrammatic notation for atomicity decomposition of [7] can help to structure refinement in Sect. 3. Then an incremental development of an existing multimedia protocol using this technique will be presented. In this protocol, a media channel is a point-to-point and dynamic channel, established for transferring multi-media data between two endpoints, called initiator and acceptor. In the previous paper [7] the connection between the requirements of a system and the decomposition technique was not explicitly discussed. In this paper we will see how requirements of the system are linked with levels in the atomicity decomposition diagram. The current atomicity decomposition technique provides sufficient patterns in development of media channel system in most of refinement levels. However some extensions to the diagrammatic technique are proposed and applied to the case study.

2 Event-B Background

Event-B [1, 2] is a formal method for specifying, modeling and reasoning about systems. Event-B has evolved from Classical B [8] and Action Systems [9]. Key features of Event-B are modeling and reasoning. The modeling notation is based on set theory and predicate logic. Building a model in Event-B typically starts with a very abstract level, and continues in different levels by use of refinement technique. Event-B use mathematical proof to verify consistency between refinement levels.

An Event-B model [1, 10] consists of *contexts* and *machines*. In other words, a model is made of several components of these two types. Contexts contain the static part of a model while a machines contain the dynamic part. There are various relationships between contexts and machines. A context can be "extended" by other contexts and "referenced" or "seen" by machines. A Machine can be "refined" by other machines and refers to contexts as its static part. The structure is shown in Fig. 1.

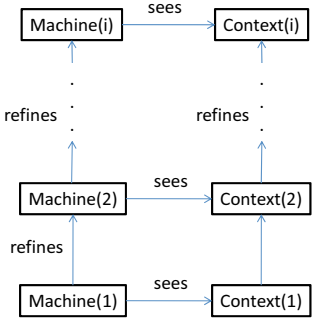

Fig. 1. Event-B Structure

Building a model usually starts with a very abstract model of the system, and then gradually details are added through several modeling steps in such a way that leads us towards a suitable implementation; this approach is called refinement [3, 4]. Thus, instead of building a single model in a flat manner, we have a sequence of models, where each of them is supposed to be a refinement of the previous.

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.

In the introduction we stated that small gaps between refinement levels results in simplicity in proof obligations. Most of proof obligations are related to consistency between refinement levels. Ensuring consistency is done by some

gluing invariants. Invariants constrain variables, and are supposed to be maintained whenever variables are changed by an event. A gluing invariant connects the abstract variables to the concrete ones. In other words, it glues the state of the concrete model to that of its abstraction. When just small changes are applied to a new level of refinement, the abstract model and the concrete model are similar, so invariants which glue the state of these two models would be simple. Therefore it can be said that small gaps between refinement levels result in simple gluing invariants, and simple gluing invariants result in simplicity in proofs related to them.

3 Atomicity Decomposition in Event-B

This section highlights the motivation for the atomicity decomposition technique and presents the technique introduced in [7] as a background to development of our case study.

Although the refinement technique in Event-B provides a flexible approach to modeling, it does not show all the relations between abstract events and concrete events. In the atomicity decomposition approach of [7], a graphical technique is proposed which is intended to make the relationships between abstract and concrete events clearer and easier to manage than simply using the standard Event-B refinement method. In this technique course-grained atomicity can be refined to more fine-grained atomicity. Sub-atomic events are treated in two ways, some refine abstract events and the others are viewed as hidden events in abstract level which refine *skip*.

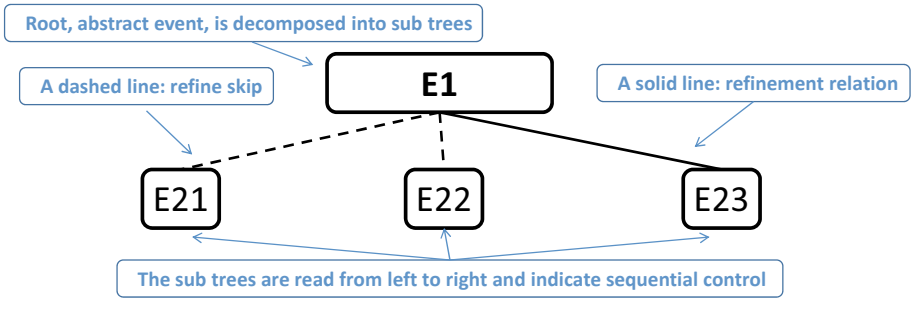

Fig. 2. Atomicity Decomposition Diagram

The tree structure notation of atomicity decomposition is illustrated in Fig. 2. In [7] this is called an event refinement diagram. The abstract atomic event, *E1* in this case, appears in the root node, which is decomposed to sub-events in the next refinement level. There is a sequential control from left to right between sub-events; in other words, the events, E21, E22, E23 in this figure, are read from left to right and are executed in this order. One important feature in the

structure is the distinction between solid lines and dashed lines. The sub-event corresponding to dashed line, E21, E22, are new events which refine *skip* in the abstract level. The child node with a solid line, E23, is a main event which should be proved to refine the abstract one, E1. The hierarchical and sequential structure is influenced by the structure diagrams of Jackson System Development (JSD) [13].

In this case, E21, E22 should execute before E23 in order to reach a state that enables event E23. This is done by some control variables in Event-B model. An Event-B model of this diagram is illustrated in Figs. 3, 4 and 5. VarE21, VarE22 and VarE23 are control variables. Event E22 is guarded by VarE21 which indicates the sequential execution of E21 and E22. Also event E23 is guarded by VarE22. Event E23 can be executed only when E22 has been executed, and event E22 can be executed only when E21 has been executed.

The possible execution traces of the model are presented here. The only event trace in abstract machine, which contains abstract event E1, is

$< E1 >$

and the execution of E21, E22, and E23 is given by the only trace of the refined model:

$< E21, E22, E23 >$

The event refinement diagram is used because it explicitly illustrates our intention that the effect achieved by E1 at the abstract level is realized at the refined level by execution of E21 followed by E22 followed by E23. In the standard Event-B method E21 and E22 are refinements of *skip* and there is no explicit connection to E1. Technically, E23 is the only event that refines E1 but the diagram indicates that we break the atomicity of E1 into events E21, E2 and E23.

```
E21   ≙
STATUS
  ordinary
WHEN
  grd1   :   VarE21 = FALSE
THEN
  act1   :   VarE21 ≔ TRUE
END
```

Fig. 3. Event-B Model Part a

Atomicity decomposition has been applied to a distributed file system in [7]. It can be used for many types of system, including sequential, concurrent and distributed systems. It is important to note that the technique of using refinement of *skip* is standard in action systems [9] and Event-B [1] and its use can also be found in Z refinement [11].

```
E21   ≙
STATUS
  ordinary
WHEN
  grd1   :   VarE21 = FALSE
THEN
  act1   :   VarE21 ≔ TRUE
END
```

Fig. 4. Event-B Model Part b

```
E21   ≙
STATUS
  ordinary
WHEN
  grd1   :   VarE21 = FALSE
THEN
  act1   :   VarE21 ≔ TRUE
END
```

Fig. 5. Event-B Model Part c

4 An Overview of Media Channel System Requirements and Multi Media Protocol

Media Channel Properties

All properties described in this section are based on a Spin model in [12]. This case study has a protocol for establishing, modifying and closing a media channel. We believe that using the atomicity decomposition technique eases understanding and development of the models.

Each Media Channel has one source, one sink, a codec type and a specific direction. A Media Channel is point-to-point and dynamic, established for transferring multi-media data.

A codec is a specific data format by which data is encoded. The codec choice in the media channel is dynamic; it means that each endpoint of the channel is allowed to change the codec in the middle of data transfer. Although each endpoint can interpret more than one codec, the source and sink of a media channel have to know which codec they are supposed to send or receive with. So any two endpoints of a media channel should have at least one common codec.

Note that in our Event-B model, we are not modeling just a single media channel, rather we are modeling a system that manages an arbitrary number

of channels simultaneously by interleaving events associated with separate channels.

4.1 Requirements for Establishing a Media Channel

Either end of a channel, sender or receiver, can attempt to open a media channel by sending an *open* signal. The other end can respond affirmatively with *openAck* (Open Acknowledge) or negatively with *close*. A media flow can be established between two media endpoints if and only if both media endpoints agree.

Each *open* signal carries the medium being requested, and a descriptor. A descriptor is a record in which an endpoint describes itself as a receiver of media. A descriptor contains an IP address, port number, and priority-ordered list of codecs that it can handle. If the endpoint does not wish to receive media, then the only offered codec is *noMedia*. Each *openAck* signal also carries a descriptor, describing the channel acceptor as a receiver of media.

A selector is a response to a descriptor. A selector is a record in which an endpoint describes itself as a sender of media. It contains the identification of the descriptor it is responding to, the IP address of the sender, and the port number of the sender. If the selecting endpoint does not wish to send media, then the selector contains *noMedia*; otherwise, it contains a single codec selected from the list in the descriptor. The only legal response to a descriptor *noMedia* is a selector *noMedia*.

After sending an *open* signal by the *initiator* side of the channel, and sending an *openAck* signal by the other side, called the *acceptor*, both endpoints have to respond to descriptors carried by the *open* and *openAck* signal, by sending a *select* signal carrying a selector. As said before, it is a rule of the protocol that a selector should be sent in order to respond to receiving a descriptor. A media channel is established by the endpoint, initiator or acceptor, which receives a real codec in a select signal. Fig. 6 illustrate a life cycle of a media channel starting with establishing the channel.

4.2 Requirements for Modifying an Established Media Channel

Modifying an established media channel may involve changing the codec and changing the port of each endpoint. At any time after sending the first selector in response to a descriptor, an endpoint can choose a new codec from the list in the descriptor, send it as a selector in a select signal, and begin to send media in the new codec. In Fig. 6, *select(sel'2)* shows this possibility.

At any time after sending or receiving *oAck*, an endpoint can send a new descriptor for itself in a describe signal. The endpoint that receives the new descriptor must begin to act according to the new descriptor. This might mean sending to a new address or choosing a new codec. In any case, the receiver of the descriptor must respond with a new selector in a select signal, if only to show that it has received the descriptor. In Fig. 6, describe(desc3) and select(sel3) illustrate this interaction. Finally at any time after sending or receiving *oAck*, an endpoint can send a new port and describe itself by a new port.

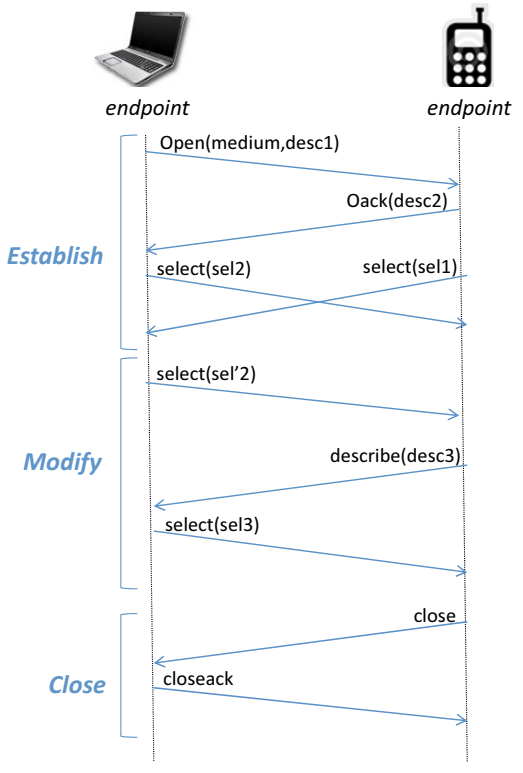

Fig. 6. Protocol of Media Channel System

4.3 Requirements for Closing an Established Media Channel

Either endpoint can close the media channel at any time by sending close, which must be acknowledged by the other end with a *closeAck* (close Acknowledge). Figure 6 illustrates the case that the acceptor side closes a channel.

5 Linking Requirements and Atomicity Decomposition

5.1 Abstract Specification

The abstract events are illustrated in an informal diagram that aids understanding, Fig. 7. It is not a formal decomposition diagram. These events happened sequentially from left to right. The circle containing "*", shows that multiple execution, zero or more, of the related event, *modify* in this case, is possible. So first a media channel is established by execution of *establishMediaChannel* event, then it can be modified zero or more times by execution of the *modify* event and then closed.

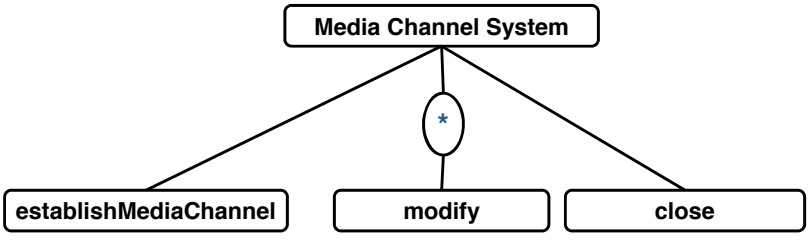

Fig. 7. Initial Model of Media Channel System

As described before, this ordering is ensured by event guards in the Event-B model. The abstract model contains a variable called *mediaChannel* containing established media channels, and a function *codec* which maps each established channel to its chosen the codec. The first event, *establishMediaChannel*, is guarded by

$ch \notin mediaChannel$

So if a channel has not been added to *mediaChannel* set, it means it has not been established then by execution of this event it would be added to *mediaChannel*:

$mediaChannel = mediaChannel \cup \{ch\}$

Events, *modify* and *close* can be executed for a channel if it was established, it is done by this guard:

$ch \in mediaChannel$

In *modify* event the codec of a channel can be changed and in *close* event, the channel is removed from *mediaChannel*.

5.2 Refinement 1: Breaking the Atomicity of Establish Media Channel

In the abstract model, we saw that a media channel is established in a single atomic step. However first phase of Fig. 6 has shown that establishing a media channel is not atomic. Instead, an open request should be sent by the initiator endpoint and should be responded to by an *openAck* signal from the acceptor side.

Following the protocol steps of Fig. 6, breaking the atomicity of establishing a media channel is outlined diagrammatically in Fig. 8. Two cases are possible. The initiator can send an open signal containing a list of codecs in a descriptor and define itself as a receiver, in this case the acceptor sends an open acknowledge signal without any codec and then selects a codec from received list in select signal. In this case the direction is from acceptor to initiator.

In the other case open signal does not contain a list of codecs and instead the acceptor sends a list of codecs in open acknowledge signal and defines itself as the receiver and the initiator selects a specific codec in select signal. The direction in this case is from initiator to acceptor. In both cases after receiving

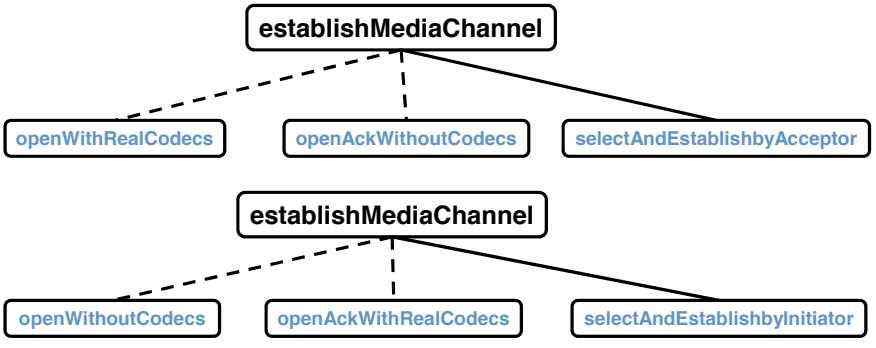

Fig. 8. Breaking the Atomicity of Establish a Media Channel

a select signal carrying a real codec, selected from the priority list of codecs of the received descriptor, the media channel can be established.

Similar to the abstract model, this sequencing the Event-B model is done by some control variables. In the first case, by execution of *openWithRealCodecs*, a channel is added to a specific set variable, *openWithCodecsSet* and the next event *openAckWithoutCodecs* can be executed only for a channel which is in *openWithCodecsSet*. The other events are encoded in a similar way.

5.3 Refinement 2: Breaking the Atomicity of Modify Media Channel

Up to this level, *modify* was considered as an atomic event that simply changes the codec of an established media channel. In this refinement we break the atomicity of the *modify* event. There are different ways of modifying the properties of an established channel. The *modify* event is decomposed to sub-events in three diagrams, presented in Fig. 9. This decomposition fulfills the modification requirements which was shown in second phase of Fig. 6.

First, in Case A, after establishing a media channel the endpoint which plays the role of the media sender can select a new codec from the list of acceptable codecs of the receiver, which has been received at the time of establishing the media channel.

In Case B, the receiver side of an established media channel, can send a new list of codecs in a *describe* signal. As described in Sect. 4, the other endpoint, has to respond to a descriptor by choosing a codec from the new list and send it via a selector.

It is shown that each endpoint, either initiator or acceptor, can describe itself with a new port by sending a describe signal carrying the new port property. It is represented as Case C.

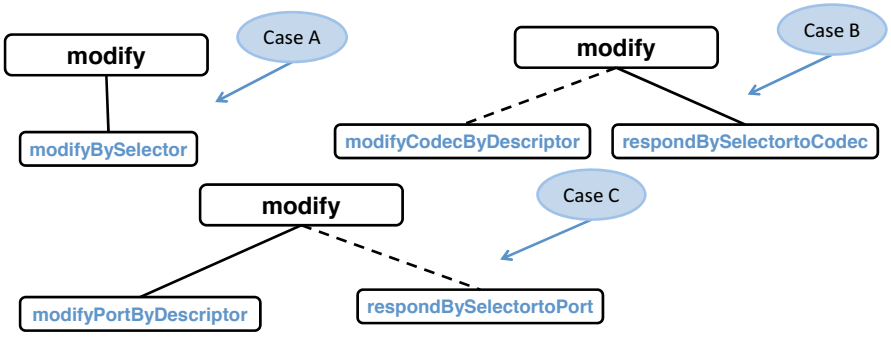

Fig. 9. Breaking the Atomicity of Modify a Media Channel

5.4 Refinement 3: Breaking the Atomicity of Close Media Channel

This is a simple refinement in which the atomicity of close is broken into two events (see Fig. 10). *CloseRequest* can be sent by each side of the channel, the sender of receiver. This figure satisfies the closing requirements shown in the last phase of Fig. 6.

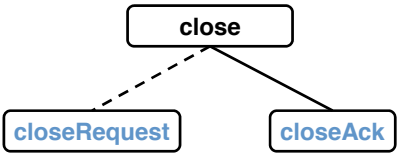

Fig. 10. Breaking the Atomicity of Close Media Channel

In Event-B model, by execution of *closeRequest* event, the channel is added to a set called *closeReqSet*, and then *closeAck* event is guarded by checking the set membership of $(ch \in closeReqSet)$.

5.5 Assessment

The refinement and atomicity decomposition technique for Event-B of [7] provides a manageable incremental development of media channel system. The overall behaviour of a media channel is modeled abstractly as three atomic events, *establish*, *modify* and *close*. Then each event has been decomposed to sub-events during refinement levels and details have been added to the model gradually. The event decomposition is presented by atomicity decomposition diagrams. The atomicity decomposition technique helps to present the relationships between an abstract atomic event and concrete sub-events in a hierarchical and sequential structure and it is specified by some guarded events in Event-B model.

Up to the fourth level of refinement of the media channel system, the basic sequential atomicity decomposition of the diagrammatic notation was sufficient for decomposing events and adding details to the model. However in the fourth level we identified some extensions to the notation that were convenient for representing further aspects of the atomicity decomposition. These are covered in the next section.

6 Extending the Diagrammatic Notation

6.1 Case Splitting Pattern

We found it convenient to introduce a diagrammatic notation to represent case splitting in a refinement. With the case splitting notation, an event is split to several sub-events is a way that execution of the abstract event is realized by execution of any of the refined events. It is presented by a circle containing an "or", as can be seen in Fig. 11. Jackson's JSD diagrams also includes case splitting [13].

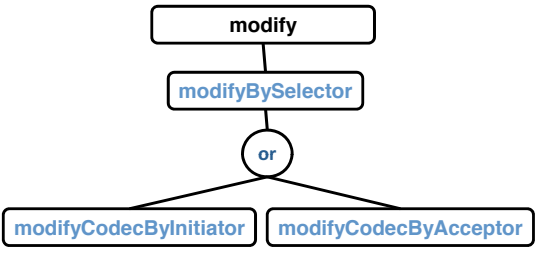

Fig. 11. Case Splitting, Level 4 of Refinement

In our model the case splitting is achieved by adding only one guard to each refined event which constrains the direction of the media channel. If the direction is from initiator to acceptor, *ItoA*, only the Initiator can modify the codec, because it has received the list of codecs belonging to acceptor, so can choose a new codec whenever it wants; and if the direction is from acceptor to initiator, *AtoI*, the one which has received the list of codecs is acceptor, so in this case it may modify the codec.

6.2 Weak Sequencing and Guard Lines

Consider the diagram in Fig. 12(a) where abstract event X is sequentially split into events *M1* and *M2* and where *M1* and *M2* are respectively further refined into sequential sub-events. Clearly, in abstract level *M2* occurs after *M1*, and in the next level *ReceiveM1* occurs after *SendM1* and *ReceiveM2* occurs after *SendM2*. Since *ReceiveM1* refines *M1* and *ReceiveM2* refines *M2*, and *M2* occurs after *M1*, clearly *ReceiveM2* occurs after *ReceiveM1*, i.e., the ordering constraint

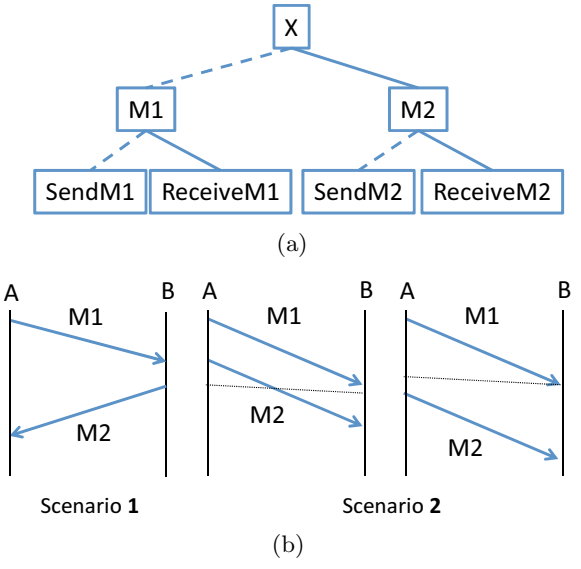

Fig. 12. Weak Sequencing interpretation Versus Strong Sequencing interpretation

between *M1* and *M2* is inherited by *ReceiveM1* and *ReceiveM2* via the solid refinement lines in Fig. 12(a).

An important question is whether there is an ordering constraint between *ReceiveM1* and *SendM2*. Two interpretations are possible. First it is possible for *SendM2* to occur before or after *ReceiveM1*. This is represented by an example of the message sequencing in Scenario 2 in Fig. 12(b) where the sequencing between these two events does not matter. We refer to this lack of sequencing as *weak* sequencing. In this sample it assumed that the channel is a fifo channel, message *M1* is sent before sending message *M2* and also message *M1* is received before receiving message *M2*, but there is not any sequencing order between receiving *M1* and sending *M2*. *Strong* sequencing, on the other hand, would be represented by the message sequencing sample in Scenario 1 in Fig. 12(b), where it is important that *SendM2* would executed only after *ReceiveM1*.

While not being explicit about this, [7] implicitly assumes that there is no ordering constraint between *ReceiveM1* and *SendM2*. This means that Butler in [7] implicitly accepts weak sequencing. Jackson's JSD diagrams allow multiple levels of decomposition but since they are intended to represent sequential processes, they implicitly assume strong sequencing [13].

If we accept weak sequencing as the default interpretation (which is useful for many distributed systems), then we need additional notation to indicate further sequencing. For our purposes we found the use of explicit *guard* lines to be convenient. A guard line is an explicit line from one event to another, indicating that the target event must occur after the source event. An example is shown in Fig. 13. In this figure, according to our default there is weak sequencing between events, as a result *SendM2* can be occur after or before *ReceiveM1*. In the case

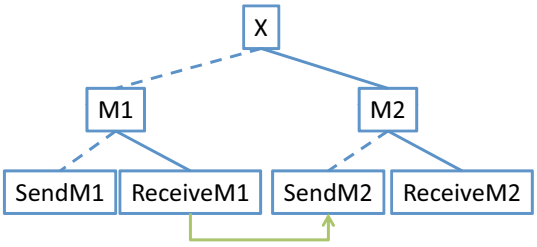

Fig. 13. Weak Sequencing Diagram with guard line

that according to requirements we would want to force them to be executed in a specific ordering, we insert a guard line between them. The inserted guard line means *ReceiveM1* should be executed before *SendM2*.

6.3 Weak Sequencing in the Media Channel Model

In our model of the media channel system the weak sequencing interpretation is used in further decomposition of the *close* event. As shown in Fig. 6, *receiveCloseRequest* should execute before *sendCloseAck*. This constraint is illustrated by a guard line in Fig. 14. It means that for sending the close acknowledge signal, a close request should be received before.

This guard line influences the Event-B model by adding a new guard to the *sendCloseAck* event requiring prior execution of the *receiveCloseRequest* event.

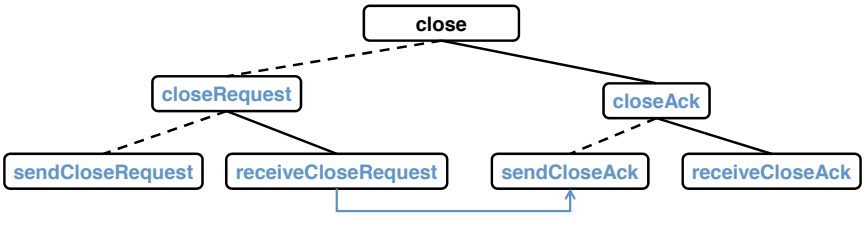

Fig. 14. Weak Sequencing Diagram in Decomposing Close Event

7 Conclusion and Directions for Future Work

An explicit representation of the sequencing of sub-events and refinement relationships, called atomicity decomposition has been used and assessed. The hierarchical diagrams introduce control structure in an incremental modeling of a case study. In abstraction media channel requirements are considered as three phases, establish, modify and close. In this paper we have shown that how each phase refine to detailed refinement using the benefits of atomicity decomposition technique in structuring requirements.

Building models of large and complex systems is not an easy task; the main reason is that it can result in very complex models and difficult proofs. Applying this technique to the multi media system partly shows that the technique can help overcome some of the complexity problems. Based on our experience in this case study, we believe that atomicity decomposition can be scaled to complex systems. The atomicity decomposition technique makes the standard refinement more systematic and visual. Sequential relations between levels of refinement during incremental modeling of a system are structured in atomicity decomposition diagrams.

It is interesting to compare our approach to the media channel system with the approach taken by Zave and Cheung [12]. Zave and Chueng present Promela models of the behaviour of each end of the protocol (sender and initiator respectively) and use the Spin model checker to verify that these models satisfy certain safety and liveness properties. In our approach with Event-B, we start with a more global view of the intension of the protocol and then use atomicity decomposition to arrive at models that have similar levels of detail to the Promela models since they include sending and receiving of messages by agents.

Sequential decomposition [7] appears to be a common pattern, but we illustrated two other patterns: the case splitting pattern and the guard line. Also we believe that the graphical technique provides representing ways of other reusable patterns. By modeling a wider range of systems on the future we anticipate the discovery of more patterns. Providing a structured refinement guideline can be considered as a future direction.

Atomicity decomposition provides a clear view of refinement steps, and can help in constructing models. At this stage building Event-B models corresponding to atomicity decomposition diagrams is done by hand, with some control variables and guarded events. An automatic model builder from atomicity decomposition diagrams will be developed in the future.

Acknowledgement

Partly supported by the EU research project ICT 214158 DEPLOY (Industrial deployment of system engineering methods providing high dependability and productivity) www.deploy-project.eu.

References

[1] Abrial, J.R.: Modeling in Event-B: System and Software Engineering. Cambridge University Press, Cambridge (2010)
[2] Butler, M.: Incremental Design of Distributed Systems with Event-B. In: Marktoberdorf Summer School 2008 Lecture Notes. IoS (November 2008)
[3] de Willem Roever, P., Engelhardt, K.: Data Refinement: Model-oriented Proof Theories and their Comparison Cambridge Tracts in Theoretical Computer Science, vol. 46. Cambridge University Press, Cambridge (1998)
[4] Rezazadeh, A., Butler, M., Evans, N.: Redevelopment of an Industrial Case Study Using Event-B and Rodin. In: BCS-FACS Christmas 2007 Meeting - Formal Method. In: Industry (2007)

[5] Abrial, J.-R., Butler, M., Hallerstede, S.: Rodin: An Open Toolset for Modelling and Reasoning in Event-B. International Journal on Software Tools for Technology Transfer, STTT (2010)
[6] Abrial, J.-R.: Refinement, Decomposition and Instantiation of Discrete Models. In: Abstract State Machines, pp. 17–40 (2005)
[7] Butler, M.: Decomposition Structures for Event-B. In: Leuschel, M., Wehrheim, H. (eds.) IFM 2009. LNCS, vol. 5423, Springer, Heidelberg (2009)
[8] Abrial, J.R.: The B-book: assigning programs to meanings. Cambridge University Press, New York (1996)
[9] Back, R.-J., Kurki-Suonio, R.: Distributed cooperation with action systems. ACM Trans. Program. Lang. Syst. 10(4), 513–554 (1988)
[10] Hallerstede, S.: Justifications for the Event-B Modelling Notation. In: Julliand, J., Kouchnarenko, O. (eds.) B 2007. LNCS, vol. 4355, pp. 49–63. Springer, Heidelberg (2006)
[11] Woodcock, J., Davies, J.: Using Z: Specification, Refinement, and Proof. Prentice-Hall, Englewood Cliffs (1996)
[12] Zave, P., Cheung, E.: Compositional Control of IP Media. IEEE Trans. Software Eng. 35(1), 46–66 (2009)
[13] Jackson, M.A.: System Development. Prentice-Hall, Englewood Cliffs (1983)