# Event-Based Modelling and Refinement of Distributed Monitoring and Controlling System (Extended Abstract)

Abdolbaghi Rezazadeh ar02r@ecs.soton.ac.uk
Michael Butler mjb@ecs.soton.ac.uk
Declarative Systems & Software Engineering Group
Department of Electronics and Computer Science
University of Southampton
Highfield, Southampton SO17 1BJ, United Kingdom

## Introduction

The importance of using formal methods in developing critical safety systems is now widely recognised [1, 2]. One of the well-known formal methods that supported with some tools is the B-method [3, 4, and 5]. B-method can support the whole life-cycle of software development. It has been applied to many research case studies and it has also been used in various industrial, critical applications like railway system, healthcare and etc. with high level of safety concern [6, 7].

Some effort has already been taking place in applying B-Method for system-level modelling [8] and there are varieties of suggested amendments in original B-Method available to make it more suitable for modelling of distributed and event-based systems [9, 10].

In this case study, we considering an event-based, system-level, modelling and refinement of a "Distributed Monitoring and Control System" for vehicle entering and leaving a controlled area using B. The system consists of different subsystems like the safety barriers and the remote-control. The main property of the undertaken system is the asynchronous nature of the communication between subsystems which is susceptible to delay, loss and error. This property can cause data to become inconsistent.

As usual we start with an informal presentation of the system followed by a very high-level formal specification of user requirements in single machine. By introducing stepwise refinements of preliminary formal specification we have tried to show how a single machine can be refined to a set of concrete machines that precisely define the operations of a whole distributed system.

## Informal presentation of the Monitoring and Control System

The main idea is to elaborate a distributed monitoring and security control system, which will be able to monitor and control the access of certain authorised vehicles to some hazardous areas according to a predefined security policy. The system consist of two main physical parts, remote-control and safety barriers. These two parts communicate with each other through external communication link. The barriers can control access to two different areas, which are called ***Controlled*** and ***Protected Areas***. A simple illustration of such areas is provided in fig (1). The Controlled Area surrounds the Protected Area. According to a predefined policy only during some special intervals of times a vehicle can enter the Controlled Area. In addition only authorised vehicle can enter the protected area based on the current security policy. The policy for accessing Controlled and Protected Area can be updated by an on-site official using the Remote-Control and the updated information will be send to related barriers.

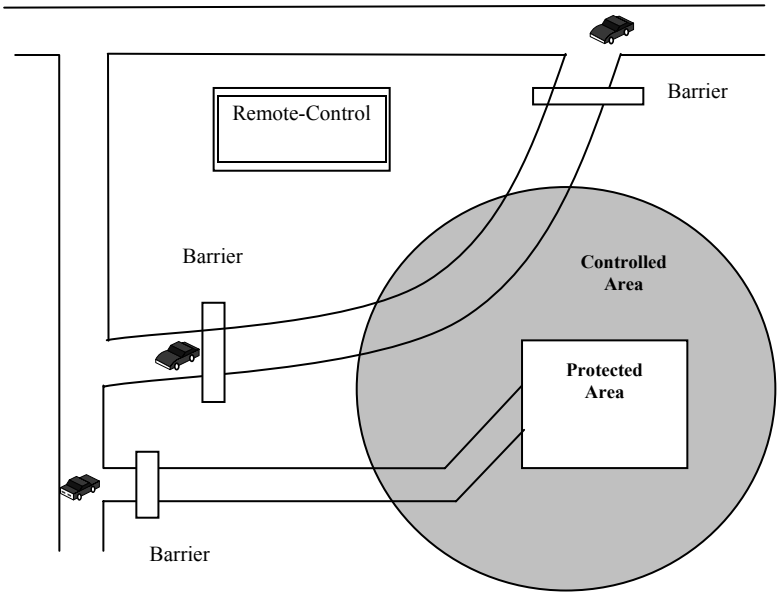


fig. (1)

Access to areas will be controlled by safety barriers which are able to communicate with the Remote-Control unit. It is clear that communication links are bidirectional and both parts are able of sending and receiving information. The communication link between barrier components and remote-control unit are at risk of delay, loss and error that must be taken in to account during system design. Implementing the security policy using special security tags on authorised vehicles has been considered. A set of valid tags can be introduced in the system and the remote-control is able to update the system with a new tag. Every tag is associated with particular access policy and this policy can be changed according to an update from the remote-control.

The remote-control can monitor the status of any barrier unit, change the status of controlled-area from blocked to unblocked or vice versa, update the set of valid tags, update the set of authorised vehicles, update the access policy associated with every valid tag and communicate to barriers unit with information related to above actions. The remote-control also can override the security policy in emergency cases to allow the system for entrance of an unauthorised vehicle to protected area.

The Barriers of protected-area can detect the presence of a vehicle, detect whether it contain a security tag and distinguish between different types of tag. Access is granted only to a single vehicle with an appropriate tag. Otherwise access is denied. A barrier can be opened for a single vehicle when the security policy permits access and will be closed immediately after that vehicle passed through. For vehicle exiting there is no need for tag checking. The barriers also communicate about vehicle passing information with remote-control.

The barriers in the controlled area perform much simpler tasks. They must allow a vehicle to pass through when the controlled-area is unblocked and otherwise access is denied. They also communicate with the remote-control.

**Event-B system specification**

The first task in our formal development is to create an abstract B-action system from informal presentation of case study in previous section. The above informal presentation of the system does not pretend to be complete or to have covered all the technical aspects. Indeed, it is merely the starting point for constructing the final concrete system. In the event-based discrete modelling a system is characterised by a finite list of variables that are modified by a finite list of events. An invariant establishes properties satisfied by variables and maintained by activation of

events reacting to the environment, when guards are true. Based on the undertaken approach now we are going to extract the main elements and building blocks of the first formal specification; they are state variables and related events.

First illustration of system is presented in fig (2). `BARRIER` is a type to identify the set of barriers units and `BARRIER_STATUS` to model the status of barriers. `CTRL_STATUS` represent two possible status of controlled-area and `VEHICLE, TAG` and `MSG` are types that have been used to identify vehicles, tags and messages respectively. The fact that every vehicle in system must be in some places and the type of access for a specific vehicle that approaching a barrier in protected or controlled areas demonstrated by `LOC` and `ACCESS` types. It can be perceived that the two types of barriers in protected and controlled area are presented with `PBARRIER` and `CBARRIER` as subset of barriers and `m1` to `m4` represent the format of different messages that can be exchange between barriers and remote-control. Before considering the ***VARIABLES*** **and** ***INVARIANT*** clause some clarifications about order of events occurrence in the system seems to be necessary.

**MACHINE** *AccessCtrl1*

**SETS** *BARRIER*;
*BARRIER_STATUS*={*open*,*close*,*failed*}; *CTRL_STATUS*= {*blocked*,*unblocked*};
*VEHICLE*; *TAG* ;*MSG*; *LOC*={*outside*,*controlled*,*protected*};
*ACCESS*={*permitted*,*denied*,*directly_obtained*,*withheld*}

**CONSTANTS**
*PBARRIER*, *CBARRIER*,
*m1*,*m2*,*m3*,*m4*

**PROPERTIES**
$PBARRIER \subseteq BARRIER \wedge CBARRIER \subseteq BARRIER \wedge$
$PBARRIER \cap CBARRIER = \varnothing \wedge PBARRIER \cup CBARRIER = BARRIER \wedge$
$m1: (TAG \times ACCESS) \rightarrowtail MSG \wedge m2: (VEHICLE \times TAG) \rightarrowtail MSG \wedge$
$m3: (CTRL\_STATUS) \rightarrowtail MSG \wedge m4: (VEHICLE \times LOC) \rightarrowtail MSG$

**VARIABLES**
*kmsgbuf*,*bmsgbuf*, *kctrl_state*,*kloc*,*ktag*,*kaut*,*kpolicy*,
*ctrl_state*,*loc*,*tag*,*aut*,*policy*

**INVARIANT**
$kmsgbuf \in \mathbf{seq}(MSG) \wedge bmsgbuf \in \mathbf{seq}(MSG) \wedge$
$kctrl\_state \in CTRL\_STATUS \wedge kloc \in VEHICLE \rightarrow LOC \wedge ktag \subseteq TAG \wedge$
$kaut \in VEHICLE \nrightarrow ktag \wedge kpolicy \in ktag \rightarrow ACCESS \wedge ctrl\_state \in CTRL\_STATUS \wedge$
$loc \in VEHICLE \rightarrow LOC \wedge tag \subseteq TAG \wedge aut \in VEHICLE \nrightarrow tag \wedge policy \in tag \rightarrow ACCESS$

fig (2)

Some basic events in system can be identified as `adding new tag, assigning a tag to a vehicle, changing current access policy` and `changing the status of controlled-area`. Regarding the asynchronous nature of the system these events can not happen in the remote-control and different barriers unit simultaneously. For example consider a simple scenario where the Remote-Control updates the system with information about a new access policy associated with a valid tag and meanwhile a vehicle with the same tag approaches a barrier which has not yet received the updated policy. In this situation the barrier will deal with the approaching vehicle according to old policy that it is no longer valid form view point of the Remote-Control. In other words, the variables in the Remote-Control and the barriers can be inconsistent in some time intervals due to delay and errors in communication links between these two parts. Therefore two different versions of variables have been considered for remote-control with a prefix of "`k`" and for barriers without this prefix. For example `ctrl-sate` represents the status of the controlled area as understood by barriers and `kctrl-sate` is state as understood by the Remote-Control. Accordingly for those events that can happen both in the Remote-Control

and the barriers we considered two different versions. These connected pairs of events are labelled with "*initialise*" and "*finalise*" prefixes.

The variable *kmsgbuf* is holding the messages that remote-control is sending for barriers and *bmsgbuf* is holding messages from barriers to remote-control. Both have the type of *sequence* to preserve the order of messages as a critical property of communication link. Considering the fact that authorised vehicles are a subset of all vehicles that holding a valid tag and each valid tag is mapped to a specific access type are reflected in invariant type definition of *kaut*, *kpolicy*, *aut* and *policy*.

**OPERATIONS**
**InitialiseAdd_tag**(*tt*,*aa*) =
**PRE** *tt*∈ *TAG* ∧ *aa*∈ *ACCESS* **THEN**
**SELECT** *tt*∈ *TAG* ∧ *tt*∉ *ktag* ∧ *aa*∈ *ACCESS* **THEN**
*ktag*:= *ktag* ∪ {*tt*} || *kpolicy*:= *kpolicy* ∪ {*tt*↦*aa*} || *kmsgbuf*:= *kmsgbuf*← *m1*(*tt*,*aa*)
**END**
**END**;
**FinaliseAdd_tag** =
**ANY** *tt*,*aa* **WHERE** *tt*∈ *TAG* ∧ *tt*∉ *tag* ∧ *aa*∈ *ACCESS* ∧ *kmsgbuf*≠ []∧ **first**(*kmsgbuf*)= *m1*(*tt*,*aa*) **THEN**
*tag*:= *tag* ∪{*tt*} || *policy*:= *policy* ∪ {*tt*↦*aa*}|| *kmsgbuf*:= **tail**(*kmsgbuf*)
**END**;
**InitialiseAssign_tag**(*vv*,*tt*) =
……..
**FinaliseAssign_tag** =
**…….**
**InitialiseChange_policy**(*tt*,*aa*) =
……
**FinaliseChange_policy** =
……..
**Enter_controlledArea**(*vv*) =
**PRE** *vv*∈ *VEHICLE* **THEN**
**SELECT** *ctrl_state*= *unblocked* ∧ *loc*(*vv*)= *outside* **THEN**
**loc**(*vv*):=*controlled* || *bmsgbuf*:= *bmsgbuf*← *m4*(*vv*,*controlled*)
**END**
**END**;
**Enter_protectedArea**(*vv*) =
**PRE** *vv*∈ *VEHICLE* **THEN**
**SELECT** *loc*(*vv*)= *outside* ∧ *policy*(*aut*(*vv*))= *permitted* **THEN**
**loc**(*vv*):=*protected* || *bmsgbuf*:= *bmsgbuf*← *m4*(*vv*,*protected*)
**END**
**END**;
**InitialiseEmer_Enter**(*bb*,*vv*) =
**PRE** *bb*∈ *PBARRIER* ∧ *vv*∈ *VEHICLE* **THEN**
**skip**
**END**;
**Exit_Area**(*vv*) =
**PRE** *vv*∈ *VEHICLE* **THEN**
**SELECT** *vv*∈ *VEHICLE* ∧ *loc*(*vv*)≠ *outside* **THEN**
**loc**(*vv*):=*outside* || *bmsgbuf*:= *bmsgbuf*← *m4*(*vv*,*outside*)
**END**
**END**;
……

fig (3)

In fig (3) some events of the system-level specification are illustrated. These events represent some observable behaviour of overall system but we have not put a clear boundary around individual subsystems. Furthermore we have not demonstrated the detailed operation of barriers.

**First Refinement of system**

In the first refinement both data and event refinements have been undertaken. Some detailed operations of the barriers in connection with the approaching vehicle and remote-control are introduced. For example events like *VehArv_CBarrier*, *VehArv_PBarrier*, *Detect_tag*, *Deny*, *Open*, *Close* **and** *Close_timeout* are internal operation of barriers that can be observed in that level. Introducing the barrier definition in the refinement process brought the feasibility of refining some operations of the first-level formal specification like *FinaliseAssign_tag*, *FinaliseChange_ctrlArea*, *Enter_controlledArea*, *Enter_protectedArea* **and** *Exit_Area*. The monitoring task is another aspect of the Remote-Control functionality that it is presented by *Request_barrierstate*, *Barrier_sendstate* **and** *Read_barrierstate* events.

**Decomposition to subsystem and further refinement**

Decomposition is a main approach to dealing with inherited complexity of distributed systems which is stated in [8] and [11]. From the previous refinement, decomposition of system to asynchronous subsystem is a straightforward task. The system as a whole comprises three subsystem named ***Remote-control***, ***Communication*** and ***Barrier***. Each subsystem can be represented by a single machine at that point which these machines are susceptible to further refinement. Some operation of communication mechanism is introduced in this level and more refinement can be envisaged for error handling, loss and security aspect like encryption.

**Conclusion**

We presented an event-based system-level modelling for an asynchronous distributed system with B. Dealing with complex properties of such systems can be a time consuming approach which needs sufficient training and skills in formal methods, system modelling and distributed systems. For better understanding and sufficient modelling maybe consideration of a variety of methodologies and tools could help developers. As further work more refinement of the barrier unit intending to model interactions between different physical parts of this subsystem could be considered. Communication subsystem can be subjected to more refinement for error-handling and security aspect modelling.

**References**

[1] A. Hall. Seven Myths of Formal Methods. IEEE Software, September 1990
[2] J. Bowen and M. Hinchey. Seven More Myths of Formal Methods. IEEE Software, July 1995
[3] J.-R. Abrial. The B book - Assigning Programs to Meanings. Cambridge University Press, 1996.
[4] B-Core. B Toolkit. www.b-core.com
[5] ClearSy. AtelierB. www.atelierb.societe.com
[6] E. Sekerinski and K. Sere (eds.). Program Development by Refinement - Case Studies Using the B Method. Springer-Verlag, 1998.
[7] P. Luigia et al. A Methodology For Integrating of Formal Methods in a Healthcare Case Study. TUCS Technical Report No 436 December 2001
[8] M. Butler. A System-Based Approach to Formal Development of Embedded Controllers for a Railway. Design Automation for Embedded Systems Vol 6. PP 355-366. July 2002
[9] JR Abrial and L. Mussat. Introducing Dynamic Constraints in B. In D. Bert editor, B'98:
[10] ClearSy . Event B Reference Manual -- June 2001
[11] M. Butler and M. Waldén. Distributed system development in B. Proceedings of the 1st Conference on the B Method, Nantes, France, pp 155-168, November 1996.