Formal development of a cross boundary route locking and releasing protocol

Paulius Stankaitis[1], Alexei Iliasov[1], Alexander Romanovsky[1] and Yamine Aït-Ameur[2]

[1] Newcastle University, Newcastle upon Tyne, United Kingdom
{p.stankaitis|alexei.iliasov|alexander.romanovsky}@ncl.ac.uk

[2] INPT-ENSEEIHT, 2 Rue Charles Camichel, Toulouse, France
yamine@enseeiht.fr

Abstract. The railway interlocking system is essential for ensuring safe operation of railway networks. Nowadays a computer based system is used to establish safe routes trains can pass through. In many cases a train is required to transition from one interlocking area to another and cross boundary route locking and releasing protocol must ensure a safe transition. In this paper we present a formal development of the cross boundary route locking and releasing protocol in Event-B. The developed distributed protocol not only ensures safety properties of the interlocking transition but also guarantees the deadlock-freedom of the railway system.

Keyword: Distributed Railway Interlocking · Event-B · Formal Verification · Stepwise Development · Safety Verification · System Liveness

1 Introduction

The railway interlocking system is essential for ensuring safe operation of railway networks. Nowadays a computer based system is used to establish safe routes trains can pass through. A railway intelocking system is capable of controlling only a portion of railway network therefore multiple interlocking systems have to be used to guarantee safety of the whole network. Often a train is required to transition from one interlocking area to another and cross boundary route locking and releasing protocol must ensure a safe transition. Because of distributed system complexity rigourous and systematic verification methods have to be used to guarantee correctness of such protocols.

Formal methods - a mathematical model driven methods provide a systematic approach for developing complex systems. They offer an approach to specify systems precisely via a mathematically defined syntax and semantics as well as formally validate them by using semi-automatic or automatic techniques. In spite of formal methods success in the railway domain considerably little has been done in considering distributed nature of the railway interlocking. In this

paper we present a formal development of the cross boundary route locking and releasing protocol in the Event-B [1] specification language.

The main protocol objective is to guarantee a safe rolling stock transition from one interlocking area to the following one. In addition to safety properties one should also guarantee liveness and deadlock-freedom of the system as well. The literature review (see below) showed that based on the engineering concept two main distributed railway models exist. The first concept of the interlocking system (described in the first paragraph) controls a number of railway agents and is more closer to real-world interlocking systems. Whereas the second is more fined grained interlocking system where each component has an autonomy to make a decision.

In the following section we describe Event-B specification language which was used to develop formal model and prove the properties of interest. In Section 2 we informally describe the cross boundary route locking and releasing protocol including key requirements and environment assumptions. In Sections **??** - **??** we present the formal model of the protocol and verification challenges. In the last section we discuss the modelling and outline concluding remarks.

Related Work. To authors knowledge the earliest attempt to formally analyse distributed railway solid-state interlocking systems was completed by Morley [8]. In this interesting work author developed a formal model of a protocol for a cross boundary route locking and releasing mechanism. By analysing temporal properties of the model he discovered that in certain scenarios safety properties can be violated. Few years later a paper by Cimatti et al. [3] presented an industrially driven formal methods study where authors formally modelled a communication protocol for safety-critical distributed systems including distributed railway interlocking systems. Their method used Statecharts diagrams to specify high level protocol properties and the OBJECTGEODE model checker for the protocol validation.

In other work a different concept of distributed railway control system was introduced by Haxthausen and Peleska [5]. Their presented engineering concept of the control system relied on a radio based communication and switch boxes - systems which can only control a single railway switch. Authors formally modelled the system with the RAISE [4] specification language which allowed to develop a formal model incrementally using a refinement process and prove refinement and safety properties with available justification tools. The timing properties of the design were considered in the extended work [7]. Similar ideas for distributed railway interlocking system were also presented in [2,6] where authors used Statecharts and Petri Nets to model and verify decentralised railway interlocking.

2 Distributed Resource Allocation Protocol Description

The objective of the protocol is to enable distributed atomic reservation of a collection of resources, for instance, two distinct agents A_0 and A_1 may require any resource collections r_1 and r_2 where $r_1, r_2 \subseteq R$. The protocol must guarantee that each agent gets all or nothing - partial request satisfaction is not permitted and ensure that every agent request will be eventually satisfied as long as certain degenerate situations are avoided.

A resource itself has an attributed memory where requests can be stored also a read pointer $\mathsf{rp}(r_k)$ and a promise pointer $\mathsf{pp}(r_k)$ - the largest promised index in the r_k request pool. The initial value $\mathsf{pp}(r_k)$ is unique for every request pool and is assigned statically. In our protocol concept a resource is only allowed to exchange messages with agents (and agents only with resources). Since we do not consider degenerate or mallicious situations messages cannot be altered or lost but requests can arrive at resources in any order.

r_0	r_1		r_2		r_3
0	0	${A_0}^*$	0	${A_1}^*$	0
1	1	${A_1}^*$	1	${A_0}^*$	1
⋮	⋮		⋮		⋮
n	n		n		n

Fig. 1: Distributed resource allocation blocking scenario. An asterisk symbol indicates that a slot has been promised but not locked for the agent.

Permitting request arrival delays can cause situations where different requests are blocking each other and causes the system to livelock. A blocking scenario is depicted in Figure 2 where agents A_0 and A_1 have both requested resources (r_1, r_2) and they were promised slots $(1, 2)$ and $(2, 1)$ in (r_1, r_2) respectively. Consequently if both agents would go ahead and lock these slots agents A_0 and A_1 would block each other and their requests could never be satisfied as partial request satisfaction is not permitted.

The principal mechanism of a solution we offer to prevent blocking situations and ensure progress is *distributed lane* - a virtual data structure which is only present at a conceptual level. To be more specific a distributed lane is a distributed data structure made of at first approximation a collection of resource request pools one per each resource where columns represent resources request pools and rows represent lane.

To lock (form a lane) a resource an agent has to go through a number of steps defined by the protocol described below.

	r_0	r_1	r_2	r_3
	0	1	2	3
	1	2	3	4
	2	3	4	5
l_0	3	4	5	6
l_1	4	5	6	
l_2	5	6		
l_3	6			

Fig. 2: An example virtual distributed lane data structure

Request. An agent A_i which intends to lock a set of resources $res \subseteq R$ generates a request to request pools associated with resources $\mathbf{r}$. Such requests are sent and received in no particular order and contain only agent name A_i. We define such message as $\mathsf{request}(\mathsf{A_i})$ and write $\mathsf{request}(\mathsf{A_i}) \rightarrow \mathsf{r_k}$ to state it is addressed to resource $r_k \in \mathbf{r}$.

Reply. Once a request pool r_k receives request $\mathsf{request}(\mathsf{A_i})$ it replies with a message $\mathsf{reply}(\mathsf{r_k}, \mathsf{pp}(\mathsf{r_k})) \rightarrow A_i$ and then increments $\mathsf{pp}(r_k)$.

ConfirmWR. After sending all $\mathsf{request}(\mathsf{A_i})$ messages an agent A_i awaits for all replies to arrive which carry values $\mathsf{pp}(r_k)$. Depending on these values following actions should be taken:

Write. If all reply values on reception are equal then A_i should write at index n to request pools $\mathsf{write}(\mathsf{A_i}, \mathsf{n}) \rightarrow \mathsf{r_k}$.

sRequest. If all values on reception are not equal then the agent must renegotiate a new index. This time an agent sends new (special) requests to a subset of resources. We define such message as $\mathsf{srequest}(\mathsf{A_i}, \mathsf{max}) \rightarrow \mathsf{r_k}$ where r_k is $\mathsf{r_k} \subset \mathbf{r}$ and must satisfy $\forall \mathsf{r} \cdot \mathsf{r} \in \mathsf{r_k} \Rightarrow \mathsf{reply}(\mathsf{r_k}) < \mathsf{max}(\mathsf{replies}(\mathsf{A_i}))$.

sReply. Once a request pool r_k receives a special request it replies with the following message $\mathsf{reply}(\mathsf{r_k}, \mathsf{max}) \rightarrow A_i$ where max the maximum value of $\mathsf{pp}(\mathsf{r_k})$ and received $\mathsf{srequest}(\mathsf{A_i})$.

pReady. A pre-ready message is sent by a resource to inform an agent that it is available for consumption and we define such message $\mathsf{pready}(\mathsf{r_k}) \rightarrow A_i$.

pReady (wr).

pReady (rl).

Lock. An agent waits for all pre-ready messages to arrive and once it receives them it sends a lock message to resources as follows $\mathsf{lock}(\mathsf{A_i}) \rightarrow r_k$.

Respond. A request pool r_k which receives a lock message will respond with a message $\mathsf{respond}(\mathsf{r_k}, \mathsf{response}) \rightarrow A_i$ where $\mathsf{response} \in \{\mathsf{confirm}, \mathsf{deny}\}$.

Decide. An agent waits for all respond messages to arrive and depending on these messages following actions will be taken.

Unlock. If one of the messages is a deny message, an agent $\mathsf{A_i}$ will send an unlock messages to all resources which replied with confirm message.

Consumption. If all messages were confirm messages, an agent $\mathsf{A_i}$ can proceed with resource consumption.

Release. An agent A_i will eventually release a resource by sending a message to to resource.

In addition to safety and deadlock-freedom one must ensure agent starvation freedom property in developing a distributed protocol. A protocol must ensure that agents will eventually be allocated resources which have been requested. A starvation could potentially occur with our protocol if multiple agents require similar resources and because of diagonal blocking agents would continuously need to renegotiate a new lane.

2.1 Refinement Plan

Acknowledgments. This work is supported by an iCASE studentship (EPSRC and Siemens Rail Automation). We are grateful to our colleagues from Siemens Rail Automation for invaluable feedback.

References

1. J.-R. Abrial. *Modeling in Event-B: System and Software Engineering.* Cambridge University Press, New York, NY, USA, 2013.
2. M. Banci, A. Fantechi, and S. Gnesi. The role of formal methods in developing a distributed railway interlocking system. In *Proceedings of the 5th Symposium on Formal Methods for Automation and Safety in Railway and Automotive Systems (FORMS/FORMAT 2004)*, pages 220–230, 2004.
3. A. Cimatti, P. L. Pieraccini, R. Sebastiani, P. Traverso, and A. Villafiorita. Formal specification and validation of a vital communication protocol. In *Proceedings of the Wold Congress on Formal Methods in the Development of Computing Systems-Volume II*, FM '99, pages 1584–1604. Springer-Verlag, 1999.
4. C. George, A.E. Haxthausen, S. Hughes, R. Milne, S. Prehn, and J.S. Pedersen. *The RAISE development method.* Prentice Hall Int., 1995.
5. A.E. Haxthausen and J. Peleska. Formal development and verification of a distributed railway control system. *IEEE Transactions on Software Engineering*, 26(8):687–701, 2000.
6. X. Hei, S. Takahashi, and N. Hideo. Toward developing a decentralized railway signalling system using petri nets. In *Proceedings of the IEEE Conference on Robotics, Automation and Mechatronics*, pages 851–855, 2008.
7. M.S. Madsen and M.M. Bæk. Modelling a distributed railway control system. Master's thesis, Technical University of Denmark, DTU, DK-2800 Kgs. Lyngby, Denmark, 2005.
8. M.J. Morley. *Safety assurance in interlocking design.* PhD thesis, 1996.