Transport4You1
Formal Methods Documentation

Version 1.0

Revision History

Date	Version	Description	Author
2011-01-16	1.0	Initial Draft	Gaurav Kushwaha

Doc. No.:

1. Introduction

1.1 Purpose of this document

Purpose of this document is to describe the usage of formal methods in Transport4you1 project, for the verification of some of the requirements.

1.2 Document organization

The document is organized as follows:

- Section 1, *Introduction,* describes contents of this guide, used documentation during developing process etc.
- Section 2, covers the reason for using Formal methods and the scope of it.
- Section 3 describes how Formal methods were used with the Agile software development.
- Section 4, 5 and 6 covers the mathematical definition of the timed automata, how we modeled the system, and what all verifications we performed on the model respectively.
- Section 7 contains references used for the preparation of this document.

1.3 Intended Audience

The intended audience is:
- Transport4you1 team
- Steering group
- Reviewer

1.4 Scope

This document covers the implementation of formal methods for the verification of some requirements based on the timed automata and its results

1.5 Definitions and acronyms

1.5.1 Definitions

Keyword	Definitions
Uppaal	Software for modeling

1.5.2 Acronyms and abbreviations

Acronym or abbreviation	Definitions
NTR	Nothing to Report. There is no information to a specific topic available or necessary.

2. Background

Transport4You is a project in Distributed Software Development(DSD), which is a joint course at Malardalen University and University of Zagreb. This project is a system that serves public transportation includes passenger routing, ticketing and transport unit control with timing constrains and safety specification. Usually real-time systems defined to be computer systems that operate correctly and within the expected and specified time limit. From this aspect our project considered partially a real-time system.

Since the investment for developing real-time systems is quite high and satisfaction for safety, security and specification mitigation is demanding there is a need for evaluating their basic design and function of a real-time system before committing ourselves into expensive and complex development process. One way of evaluating the early decisions, basic design and specification of real-time system before being involved in expensive development process is to model, simulate and verify it based on formal methods theories.

Among the possible modeling techniques such as finite state machines, process algebras, Petri-nets, temporal logic [1] symbolic model checking by the use of timed automata [2] shows to be promising first effort approach in modeling real-time systems because it provides a formal mechanism for manipulating the timing delay and the handling of the timing conditions. Based on "Timed Automata: Semantics, Algorithms and Tools"[3] and the work of Alur and Dill [4] several real-time systems have been studied as the car wash controller[5], Gear controller [6], railroad crossing [7], the formal model of Ada Ravenscar tasking [8], the formal verification of a Power Controller [9]. Other articles [10] and [11] proving the sufficiency of the timed automata in the modeling and verification of real-time systems.

Modeling and formal methods techniques are usually performed in early phases and since agile software development was followed in this project adaptation has been made to make simulation and verification possible not only from the beginning of the development.

3. Formal Method and Agile Software Development

Formal methods are particularly effective early in development at the requirements and specification levels [Wikipedia]. Variations of using formal methods theories are a way to verify and validate specifications to insure correctness of the software. In the other word is formal methods is a strict mathematical definition of the effect of the required operation and relatively is an expensive process.

In traditional software development methodologies big portion of time was spent in the beginning of the project for requirement gathering and specifications definition. These methodologies were help to apply formal methods theories. But in our project since we followed agile software development it was a challenge to combine our software development with formal methods. Variations of applying formal methods studied and modeling techniques was selected among the other techniques and more precisely timed automata. In our experience the nature of modeling was the key to make this combination between agile development and formal methods possible. Because we can start our model from a simple module and as long is the module is going into iterations the model also is going into iterations until we get a complete module confirms to a concrete model.

The development process was designed to have week long sprints for development and it was so short to have the same for formal methods, so we decided to define and combine two agile processes together. The first was the agile development process and the second agile formal methods development process with short first initial sprint and three times longer sprint than the development sprints for the next development period.

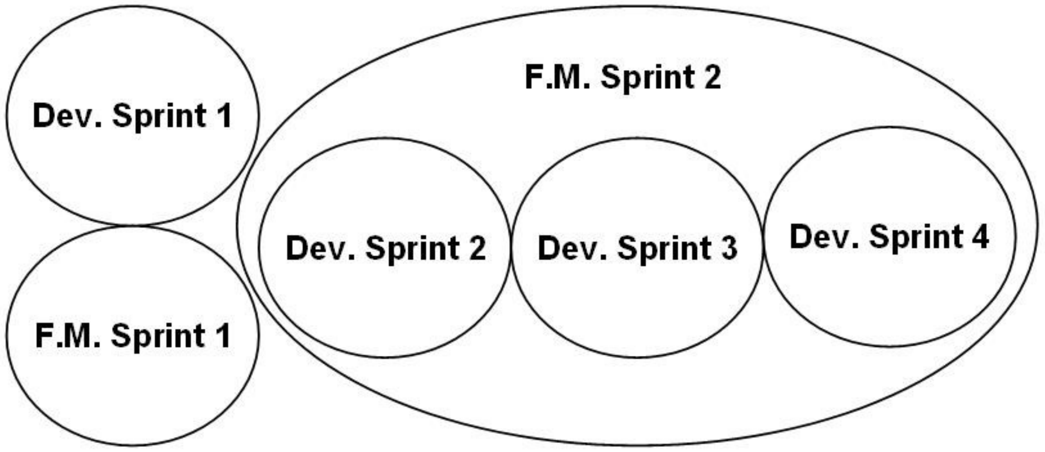


Figure 1

Our development sprint is one week long and after each sprint things getting clearer (in general) but the gathered requirement and information in each sprint might not enough to go for formal methods. Since applying changes to the model is expensive this way we are reducing the risk of change and we come up with a change that would be probably a major change and it is some thing has to be done.
In agile development we need to have the new requirement at the beginning of each sprint and the customer should not change the requirement during the sprint.Figure 1 illustrates that the development sprints are basted on the previous Formal methods Sprint

4. Timed Automata

A timed automaton is a finite state machine extended with clock variables [12]. The clock variables are constrains i.e. guards on edges to restrict the behavior of the system and act as enabling conditions.

4.1 Definition (Timed Automata)

A timed automaton is a tuple (L, lo, C, A, E, I) where L is a non-empty finite set of locations, $l0 \in L$ is the initial location, C is the set of clocks, B(C) is the set of conjunctions over simple conditions of the form x#c or x-y # c, where $x, y \in C$, $c \in N$ and $\# \in \{<, \leq, =, \geq, >\}$, A is a set of actions, co-actions and the internal τ -action, $E \subseteq$ L x A x B (C) x 2C x L is a set of edges between locations with an action, a guard and a set of clocks to be reset, and I : L $\rightarrow$ B (C) assigns invariants to locations.

4.2 Definition (Semantics of TA)

Let (L, lo, C, A, E, I) be a timed automaton. The semantics are defined as a labeled transition system (S, so, $\rightarrow$), where $S \subseteq L \times RC$ is the set of states, so = (lo, uo) is the initial state, and $\subseteq S \times \{ R \geq o \cup A\} \times S$ is the transition relation such that:

– (l, u) $\xrightarrow{d}$ (l, u + d) if $\forall$ d' : 0 $\leq$ d' $\leq$ d $\Rightarrow$ u + d' $\in$ I(l), and

– (l, u) $\xrightarrow{a}$ (l', u') if there exists e = (l, a, g, r, l') $\in$ Es.t. u $\in$ g,

u' = [r $\rightarrow$ 0]u, and u' $\in$ I(l),

where for d $\in$ R $\geq$0, u + d maps each clock x in C to the value u(x) + d, and [r $\rightarrow$ 0]u denotes the clock valuation which maps each clock in r to 0 and agrees with u over C\r [3].

A model of timed automata may consist of a few related timed automata, called network of timed automata, and have common set of clocks and actions. The edges of timed automata model have three optional labels as following:

1. Guard: It is a condition on the edge and must be satisfied for the edge to be taken. The condition usually is an integer clock type value. Figure 2 shows the edge between A0 and A1 can be taken only if the two required conditions, clock x is greater than or equals 2 and i is equal to 3 satisfied.
2. Synchronization action: is the action performed when the edge is taken and the taken edge is synchronizes with another edge in another related timed automata. Figure 2 shows a network model of timed automata consist of two simple process modeled with timed automata. In timed automata the way of communication between models is channel and channels are assigned to edges and the edges are synchronized. When an edge is taken, another action in another model is done synchronal. In figure 2 the two model communicate via channel a. The edge between A0 and A1 must be taken showed by a! when the edge between B0 and B1 is required to be taken, showed by a?.
3. assignments: is clock assignment or clock reset when the edge is taken. Figure 2 shows assignment of clock x is reset to 0 and integer variable i is increased by four when the transition between A0 and A1 is taken.
4. Invariant: is a constraint on the location which means rather than edge. Figure 2 illustrates two invariants on A0 and B0. The constraint on A0 means this location should be left in less than or equal 10 time unit.

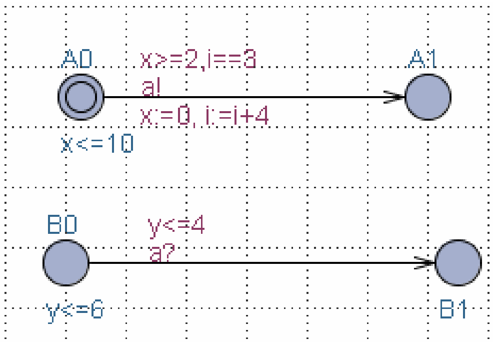


Figure 2

The variables model the logical clocks in the system. The clocks initialized with zero at the system beginning, and then increase synchronously at the same rate [3].

Timed automata design is usually done in the early phases when decision regarding the system structure and behavior must be taken. In our experience we are following timed automata design in iteration to adapt to our software development methodology. Timed automata model provide a graphical and a mathematical system view and enabling early verification and validation.

5. Modeling

The model is implemented in Uppaal. Uppaal is model checker for network of timed automata. It is used as a tool for modeling, simulating and verification of real-time system.

5.1 Transport unit module modeling

The transport unit is heart of the project and it is organizing the boarding activities. The activities are controlling transport unit location, doors, cell phone detection, preparing list of on board passengers and ticketing. Assumption has been made regarding the main application that the door sensors and GPS device are collaborating with the main application and actually the sender for these devices became the main application. For example the sender for channel BusInStation is the main station as result of the GPS device.

5.2 Transport unit timed automata modeling and Model Simulation

UPPAAL is used to simulate the model. The future behavior of a timed automaton depends on its present state and the value of all its clocks and data variables.
The model is considering a bus with many stations and one passenger. Passengers could come with registered cell phone to board the cell is detected when the bus's doors are closed and billed for 60 minutes valid ticket. Then the passenger is notified for the bill with SMS.

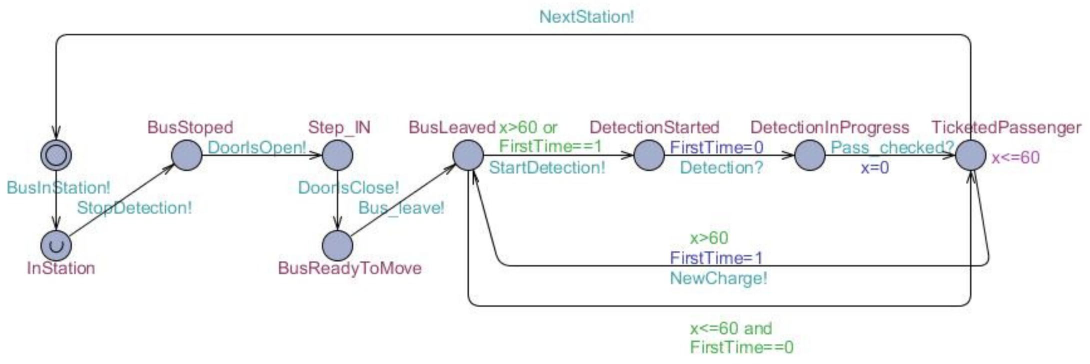


Figure 3

The transport unit consists of 4 models in addition to the main model. The modules are Door, CellCheker, Bus, BackEnd and main. Every model has collection of behaviors modeled in timed automata and synchronized with other models throw channels. The model assumed to start by arrival of the bus to station then the cell detection is stopped before the bus stopping in the station by using the broadcast channel StopDetection. The doors will be open and the system goes to state Step IN if exist a passenger will come into the bus. Then the model is going to state BusReadyToMove and then BusLeaved throw channels DoorIsClose and BusLeave. When the bus is in the way the detection is started and the model is going to state DetectionStarted throw the channel StartDetection. In two cases this edge could be taken, either it is the first time this passenger is detected or the passenger's ticket is expired. After first detection a flag of first time detection is set to zero and after the checking and ticket issuing timing counter is set. When the passenger is in state TicketedPassenger that means the passenger holds a valid ticket for 60 minutes. The passenger can travel between several stations but the time if exceed the 60 minutes then will be charged for a new ticket throw channel NewCharge.

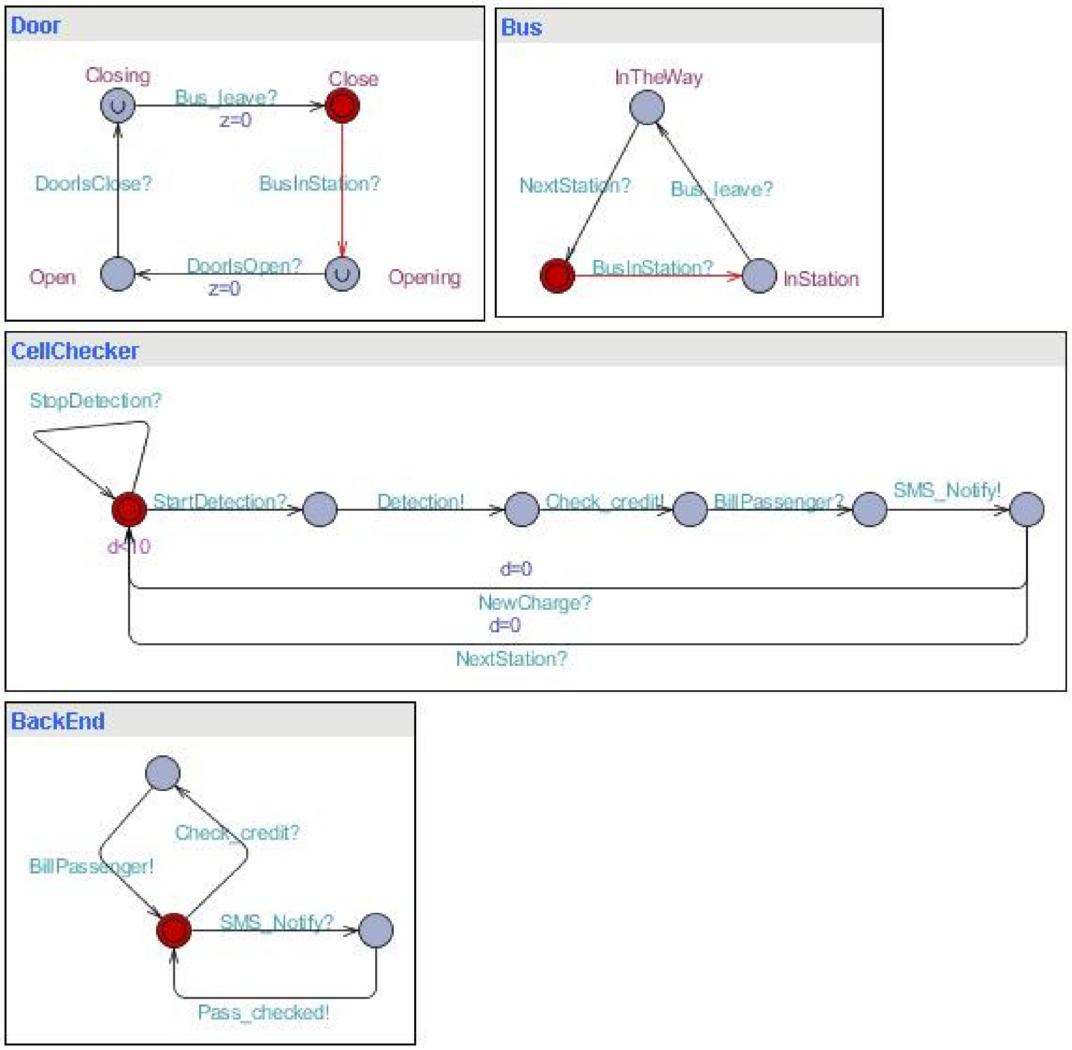


Figure 4

6. Verification

UPPAAL is able to check for reach ability properties, especially if a combination of locations and constraints on clock and data variables is reachable from an initial state [13]. Properties of the following forms might be verified using UPPAAL verifier:

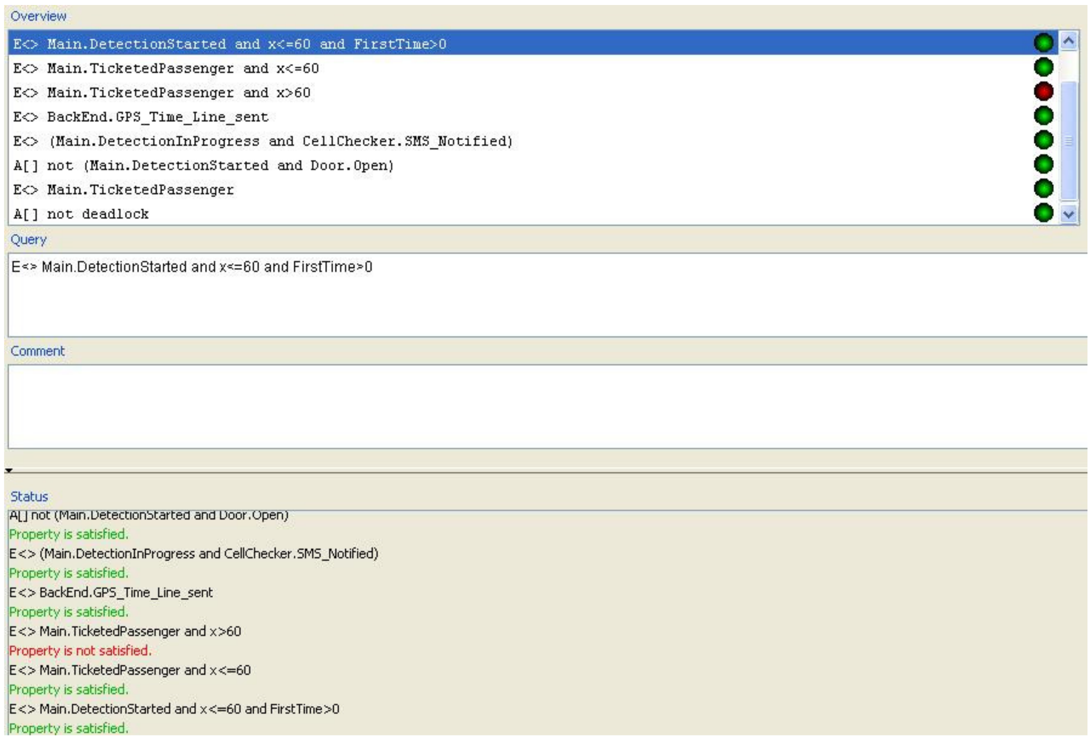
- E<> S: to be satisfied a path must exist where S eventually holds
- A [] S: to be satisfied S must be satisfied for all reachable states
- E [] S: to be satisfied a path must exist where S always holds
- A<> S: to be satisfied S must eventually hold for all paths

Using the above we can verify the following properties of our model:

1. Test cases TMA 1, 2, 3 and TUA 1, 2 from acceptance test plan verified
2. No deadlock in the system. In UPPAAL this requirement is symbolized as: A[] not deadlock
3. Passenger is ticketed after ticketing issues is handled. E<> Main.TicketedPassenger

4. The detection started when the doors are closed. A[] not (Main.DetectionStarted and Door.Open)
5. Notifications are sent out for all cases of ticket handling. E<> (Main.DetectionInProgress and CellChecker.SMS_Notified)
6. The GPS location, boarding time and line number is sent after detection and when checking. E<> BackEnd.GPS_Time_Line_sent
7. Ticket is valid for 60 minutes only and this test gives a negative result. E<> Main.TicketedPassenger and x>60
8. The ticket is valid for less than or equal 60 minutes and the passenger can stay in the bus for that time. E<> Main.TicketedPassenger and x<=60
9. The passenger is ticketed for two reasons first when the passenger enters the transport unit for the first time and the second when the trip time exceed 60 minutes. E<> Main.DetectionStarted and x<=60 and FirstTime>0

UPPAAL's result are indicated in the figure 5.



```
E<> Main.DetectionStarted and x<=60 and FirstTime>0        ●
E<> Main.TicketedPassenger and x<=60                       ●
E<> Main.TicketedPassenger and x>60                        ●
E<> BackEnd.GPS_Time_Line_sent                             ●
E<> (Main.DetectionInProgress and CellChecker.SMS_Notified) ●
A[] not (Main.DetectionStarted and Door.Open)              ●
E<> Main.TicketedPassenger                                 ●
A[] not deadlock                                           ●
```

```
E<> Main.DetectionStarted and x<=60 and FirstTime>0
```

```
A[] not (Main.DetectionStarted and Door.Open)
Property is satisfied.
E<> (Main.DetectionInProgress and CellChecker.SMS_Notified)
Property is satisfied.
E<> BackEnd.GPS_Time_Line_sent
Property is satisfied.
E<> Main.TicketedPassenger and x>60
Property is not satisfied.
E<> Main.TicketedPassenger and x<=60
Property is satisfied.
E<> Main.DetectionStarted and x<=60 and FirstTime>0
Property is satisfied.
```

Figure 5

7. References

[1] Branicky, V., Borkar, S., Mitter, S.K. (1994) A unified framework for hybrid
control, Proceedings of the. eerd Conferance. Decision and Control, Lake Buena
Vista 4228-4234
[2] Macmillan, K. (1993) Symbolic Model Checking, Kluwer Academic
[3] J. Bengtsson and W.Yi. Timed Automata: Semantics, Algorithms and Tools
[4] Alur and David L. Dill. A theory of timed automata. *Journal of Theoretical
Computer Science*, 126(2):183–235, 1994.
[5] Aneta Vulgarakis, Aida Caušević. Applying REMES behavioral modeling to PLC systems
[6] Magnus Lindahl, Paul Pettersson, Wang Yi. Formal Design and Analysis of a Gear Controller: an Industrial
Case Study using UPPAAL
[7] P.R D'Argenio and Ed Brinksma. A calculus for timed automata. June 1996
[8] K.Lundqvist and L.Asplund. A formal Model of the Ada Ravenscar Tasking
Profile; Delay Until. 1999
[9] K.Havelund, K.G.Larsen and A.Skou. Formal Verification of a Power Controller
Using the Real – time Model Checker UPPAAl.
[10] H.Bowman, G.Faconti,J-P Katoen, D.Latella and M.Massink. Automatic
Verification of a Lip Synchronization Algorithm using UPPAAL
[11] T.A. Henzinger, X.Nicollin,J.Sifakis and S.Yovine. Symbolic model checking for
real-time systems.*Information and Computation*, 111:193-244,1994
[12] G.Behrmann, A.David, K.G.Larsen. A Tutorial on Uppaal, 2004
[13] K.G.Larsen, P.Petterson and W.Yi. UPPAAL in a Nutshell, Int. Journal on Software Tools for technology
Transfer, Springer-Verlag,vol 1,number 1-2, pp134-152,Oct 1997