# ModelicaML modeling analysis of MCPS-based high-speed railroad safety monitoring system

Yuan Li[1, a],Can Qu[*1, b] ,Xingye Ruan[1, c]

[1]School of City College of Huizhou, Huizhou 516001, China.

[a]158481886@qq.com, [b]505609551@qq.com, [c]348928205@qq.com

**Abstract.** Mobile Cyber-Physical System(MCPS) focuses on modeling of the physical world, dynamic continuity, real-time, spatial, security, and real-time predictable communication problems. Based on the example of high-speed railway system, this article mainly analyzes and studies the railway safety monitoring subsystem. Firstly, a brief analysis of the railway safety monitoring system and architecture is made, and the database connection pool and space-time analysis model are used as the software modeling system according to the requirements; then, SysML and Modelica are used to model the study, based on which a reconfigured modeling language called ModelicaML is proposed to model and simulate the modern engineering; lastly, the ModelicaML is used to physically model the safety detection system of high-speed railroad system, and finally, it is simulated and verified.

**Keywords:** MCPS ; ModelicaML modeling ; Simulation verification.

## 1. MCPS Review

Cyber-Physical System (CPS) is a highly integrated system of computation, communication and control technologies, which realizes distributed sensing of external environment, reliable data transmission, intelligent information processing and real-time control of physical processes through feedback mechanisms by embedding sensing, communication and computing capabilities in physical devices.

Relatively speaking, Mobile Cyber-Physical System (MCPS) is a more specific CPS base on information fusion, with the help of the current advanced cloud computing and big data, and has been applied to aerospace, aviation, automotive, and energy wide industries with good results.

## 2. Modeling Language

### 2.1 SysML

SysML Model OverviewThe System Modeling Language (SysML)[1] is a graphical modeling language for systems engineering applications.SysML is built on top of UML and supports system and inter-system specification, analysis, design, verification, and validation. In the field of systems engineering, the main goal of SysML is to unify and replace different document-centric approaches with a single system modeling language. Such a single model-centric approach improves communication, assists and manages the design of complex systems, and enables early verification of systems.The SysML diagram taxonmy[2] is shown in Figure 1 below.

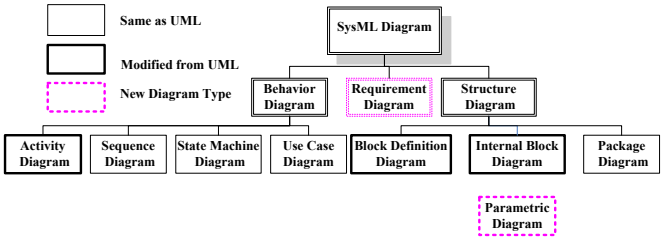


Figure1 SysML diagram taxonmy

### 2.2 Modelica Overview

Modelica is a popular language for modeling multi-domain physical systems. It is an open, object-oriented, equation-based language that can easily model complex physical systems across different domains to meet multi-domain needs, such as electromechanical models (mechanical, electronic, hydraulic, and control subsystems in robotics, automotive, and aerospace applications). It has high model reusability, easy and convenient modeling, and model closeness to the actual physical system.

Modelica models are composed of three elements: variables, equations and algorithms, and nested classes. The variables represent the properties of the model, usually representing a physical quantity; the equations and algorithms describe the behavior of the model, expressing the constraint relationships between variables; and the nested classes are the basic structural elements of the Modelica language and are the basic units that make up the Modelica model.

### 2.3 ModelicaML-based physical modeling

Existing SysML models are usually limited to system software modeling or event-based discrete-time simulation, and the use of SysML models in combination with Modelica models can improve the expressiveness and accuracy of modeling, and better simulate hardware and software simulations to break through the above limitations.Therefore, this article proposes a refactoring modeling language called ModelicaML, which reuses the concept of SysML, the main purpose of which is to implement an efficient way to create, read, understand, and maintain Modelica models, and generate graphical models into executable Modelica codes.

Based on SysML and Modelica, modeling design using ModelicaML focuses on the graphical description of Modelica code, and it is a ModelicaML framework resulting from the extension of the UML metamodel[41] . The modeling process has three main steps: (1) modeling the system with ModelicaML according to the requirements; (2) generating Modelica code based on the modeling graphics from the previous step; (3) simulating the code using Modelica simulation tools; the specific steps are shown in Figure 2 below.

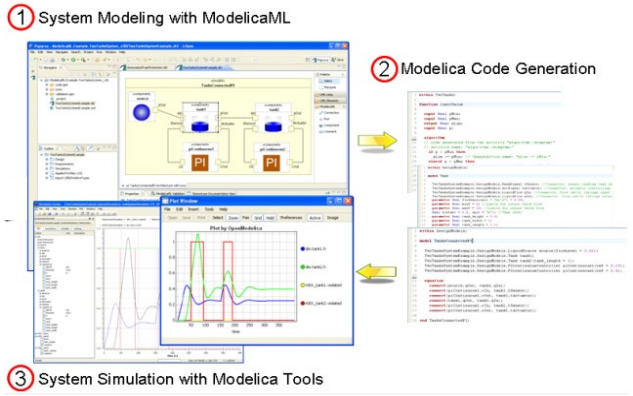


Figure2 ModelicaML modeling three main steps

## 3. The physical model analysis of high-speed railroad safety monitoring system based on ModelicaML

### 3.1 Physical model of state transition

Enter a set of data that matches the requirements analysis to be validated by simulation and create the selection standard based on the type of requirements as well as the system design model. The recommended steps for selecting requirements are as follows.

Read the requirements and determine if this model can be evaluated using simulation.

Confirm if this requirement is complete and if an accurate and testable design model can be used.

If yes: mark this requirement as selected

If no: skip this requirement

First we analyze the HSR state transition system for the train. For clarity of presentation and modeling needs, we use a tabular format to define and split the requirements, and the requirements are described in Table 1.

Table1 HSR system state transition requirements description

| Requirements ID | Requirements description |
|---|---|
| 01 | HSR state transition is mainly a protection system consisting of a central controller and five subsystems, namely sensors, speed detectors, braking system, alarm system and reset system |
| 02 | Sensors are attached to the side of the train to receive and detect signals as they approach the track and send the results to the train control room |
| 03 | In the control room, the controller calculates the last signal value of the train |
| 04 | There are three sensors that generate values in the range of 1-3, each representing the current status of the train |
| 05 | If the calculation results in a normal operating state and the signaller displays a value (0,1),the train is operating normally |
| 06-1 | If the signaller displays the result (1,2),The warning signal in the control room is activated |
| 06-2 | If the speed of the train does not reach a descent speed of 0.5 m/s after the warning signal is activated, the braking system will be automatically activated after 2s |
| 07 | If the display shows a result of (2,3), it is a danger signal and the system will activate the braking system and alarm within 0.5s |

The HSR signal switching system in our design model here has three tracking side signals: Proceed, Caution and Danger. When the HSR system receives a warning signal, it monitors the driver's response within a specified time frame to ensure that the train is moving normally. If the driver fails to reduce the train's speed, the HSR system receives a warning signal or a danger signal and activates the train's braking system. Figure 5-18 shows the top-level architecture of the train system consisting of the driver, as well as the internal structure of the HMI system, the control system, the engine and the braking system. Each component behavior is modeled in ModelicaML, as shown in Figure3 below.

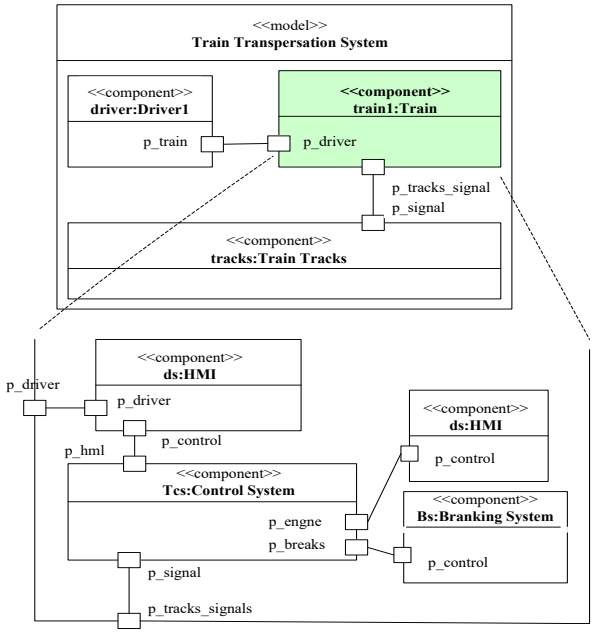


Figure3 Train transportation system and train architecture in ModelicaML

For the above requirements description, we analyzed and then physically modeled the HSR state transition, which is mainly composed of a central processor and five subsystems, which are sensors, speed detectors, braking system, alarm system and reset system. Among these five subsystems, we mainly analyzed the state model displayed by the sensor and the control subsystem modeling in detail respectively, using the Eclipse-based plug-in Papyrus as the modeling tool.

State transformation mainly refers to the process of changing the status of the train due to the speed of the train or the influence of external conditions and environment while the train is moving on the

main line. If the status changes, the driver needs to take corresponding measures according to the urgency of the situation or not. If no action is taken within the specified time, the HSR system will automatically process the train's forward speed at different times according to the status mode to ensure the safety of the train. Here we mainly divide the status mode into four levels, namely stationary, normal operation, warning and danger. In different status, different processing operations are required, and the specific status descriptions are shown in Table 2.

Table2 description of train status

| Status Mode | Status Code | Status Description |
|---|---|---|
| Static | State-0 | The train is at a stand still in the siding |
| Proceed | State-1 | The train is in normal operation with the speed within 250km/h, and the train status signal will be displayed as (0,1) |
| Caution | State-2 | When the train speed exceeds 250km/h and is below 300km/h, the alarm will issue a warning and the train status signal will be displayed as (1,2) |
| Danger | State-3 | The train speed is over 300km/h, and the train status will be displayed as (2,3) |

From the above requirement description we first model the state of the system, for brevity we use state codes to represent the different states, then the different state transition processes are shown in Figure 4 below.

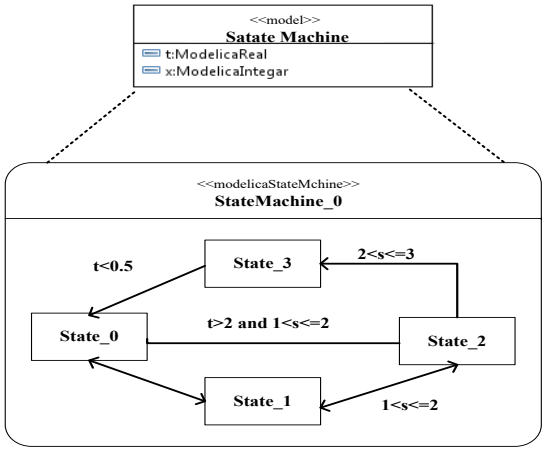


Figure 4 Train signal state transition model

A dynamic continuous model is simulated in the train state model using ModelicaML, and the code is shown in Figure 5.

```
model StateMachine
  Boolean State_0;
  Boolean State_1;
  Boolean State_2;
  Boolean State_3;
  Integer signal;
  Real time1;
equation
  der(time1) = time;
algorithm
  when initial() then
    State_0 := true;
  end when;
  if pre(State_0) and signal < 1 then
    State_0 := false;
    signal := 1;
    State_1 := true;
  elseif pre(State_1) and time1 > 1 and signal <= 2 then
    State_1 := false;
    signal := 2;
    State_2 := true;
  elseif pre(State_2) and time1 > 2 and signal <= 3 then
    State_2 := false;
    signal := 3;
    State_3 := true;
  elseif pre(State_3) and time1 <= 0.5 then
    State_3 := false;
    signal := 0;
    State_0 := true;
  elseif pre(State_2) and time1 > 2 then
    State_3 := false; signal := 0;
    State_0 := true;
  end if;
end StateMachine;
```

Figure 5 Modelica code for train state model

The simulation model of the state-transition system is shown below: Figure 6 shows the simulation graph of the variation of the signal volume over time, and Figure 7 shows the variation of the signal volume over time.

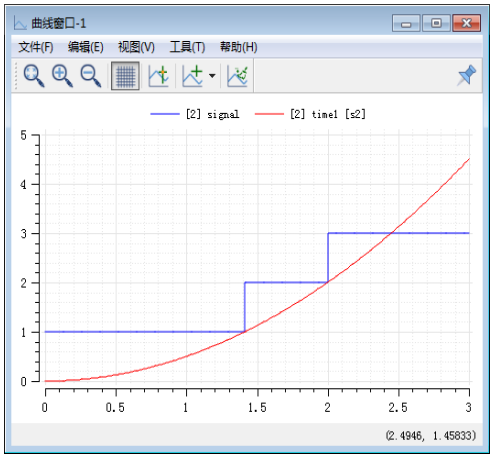


Figure 6 state of the signal changes over time

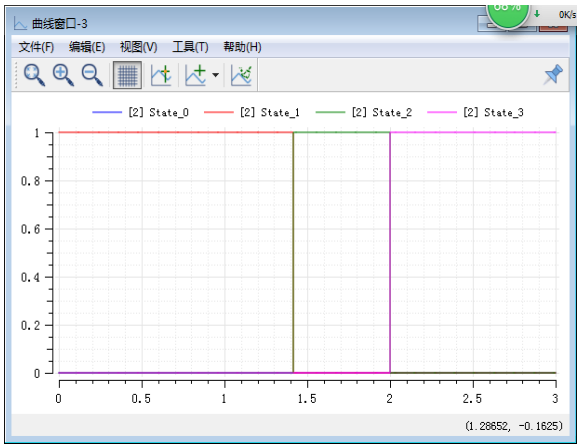


Figure 7 Train status changes

### 3.2 Physical modeling of control subsystems

The selected subset of requirements will be transferred to the tool in the modeling and used in the subsequent steps. The second step is to have each requirement represented formally so that it can be accurately evaluated during automated simulations. For requirement definition 06-1: If at any time the controller calculates a "caution" signal, it will enable the 'In-cab alarm system' within 0.5 seconds. Based on this statement, we can confirm the measurable attributes contained in the requirement statement, i.e., receipt of a warning signal, activation of the alarm, and time frame constants. The Formalized requirement properties ModelicaML are shown in Figure 8.

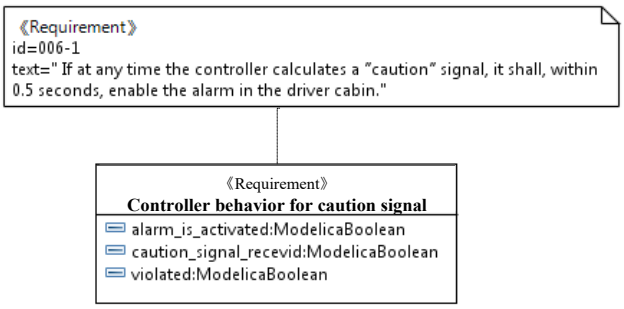


Figure 8 Formalized requirement properties ModelicaML

To confirm if the requirement is satisfied, we make the following assumption: this requirement will be implemented as long as the system evaluates that the monitor does not violate the system requirements. Now we define a violation relationship. This example uses a state machine to specify when a system requirement is violated. Here we have done the following analyses based on the actual situation: what is the maximum time required for the driver to react when the alarm is activated; what is the maximum expected time when the controller cannot be activated instantaneously; and what is the minimum speed reduction of the train by the braking system in HSR.

After formalizing the requirements and selecting the correct design model for validation, it is also necessary to create test models, define test cases and connect the requirement attributes to the values in the design model. This is to ensure the correct connection between the requirement attributes, i.e. the correct design model attribute values. In this case, the requirement attribute of the received warning signal is true when the controller attribute train signal state is consistent with the defined warning signal, and we define the received warning signal attribute as follows:

"caution_signal_received =design_model.train1.pc1.tcs.controller.tracks_signals_status== 1;"

In addition, we also need to create the following properties and a use case model for each property.

whether the test passed or failed:

"test_passed := evaluated and not violated; "

whether all required test cases were evaluated:

"evaluated := if req1.evaluated and ... and req N.evaluated then true ...; "

if a requirement violation occurred:

"violated := when {req1.violated,... ,req N.violated} then true ...; "

Through the analysis of the above situation, the train speed control model is created as Figure 9.

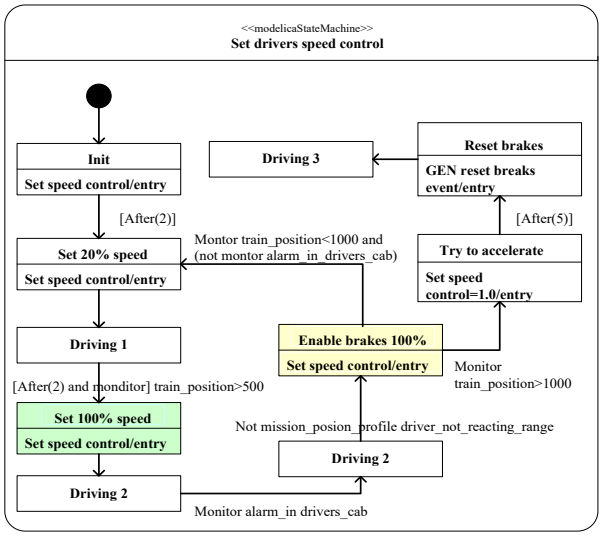


Figure 9 train speed control model

After creating the test model by analyzing the above scenario, we can run the simulation and observe the results as Figure10.

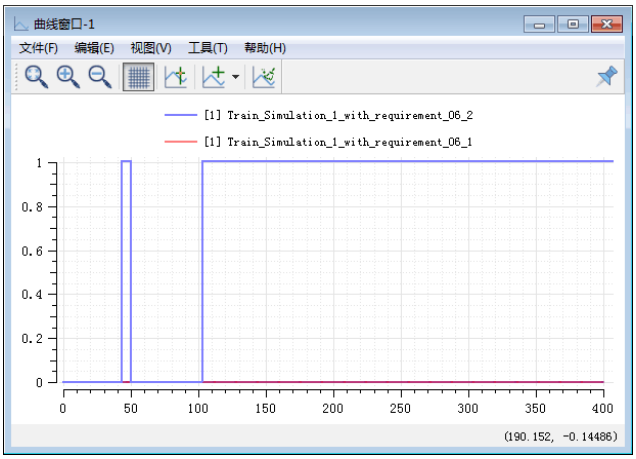


Figure 10: Test execution and requirement violations observation

## 4. Summary

Based on an example of a high-speed railroad system, this article focuses on the analysis of the railroad safety monitoring subsystem, and also models the state transition of the monitoring system and the control subsystem respectively, finally Through the simulation data, its scientificity and theoretical feasibility are verified.

## References

[1] Holt J, Perry S. SysML for Systems Engineering[M]. Institution of Engineering and Technology, 2008.

[2] Weilkiens T. Systems Engineering with SysML/UML: Modeling, Analysis,Design[M]. Morgan Kaufmann OMG Press/Elsevier, 2007.

[3] Iqbal S, Allen G. Designing Aspects with AODL[J]. International Journal o f Software Engineering, 2011, 4(2).

[4] Schamai W, Fritzson P, Paredis C J J, et al. ModelicaML Value Bindings for Automated Model Composition[C]// Symposium on Theory of Modeling & Simulation Orlando. 2012.

[5] Hakam I. ModelicaML Graphical Modeling Environment Based on Eclipse MDT Papyrus[J]. 2011