UDC 004.415.53:004.9:793
Original Scientific Paper

# Computer Games as Virtual Environments for Safety-Critical Software Validation

**Štefan Korečko** *stefan.korecko@tuke.sk*
*Faculty of Electrical Engineering and Informatics*
*Technical University of Košice, Košice, Slovakia*

**Branislav Sobota** *branislav.sobota@tuke.sk*
*Faculty of Electrical Engineering and Informatics*
*Technical University of Košice, Košice, Slovakia*

**Abstract**

Computer games became an inseparable part of everyday life in modern society and the time people spend playing them every day is increasing. This trend caused a noticeable research activity focused on utilizing the time spent in a meaningful way, for example to help solving scientific problems or tasks related to computer systems development. In this paper we present one contribution to this activity, a software system consisting of a modified version of the Open Rails train simulator and an application called TS2JavaConn, which allows to use separately developed software controllers with the simulator. The system is intended for validation of controllers developed by formal methods. The paper describes the overall architecture of the system and operation of its components. It also compares the system with other approaches to purposeful utilization of computer games, specifies suitable formal methods and illustrates its intended use on an example.

**Keywords:** formal methods, computer games, games with purpose, validation, verification

## 1. Introduction

For millions of people in developed countries computer games have become an integral part of everyday life. According to the recent statistics [1] the daily time spent playing computer games per capita in USA will reach an half of hour in the nearby future. There have been many disputes about positive and negative impacts of these games on society but it is an indisputable fact that the time and effort spent in developing and playing the games is increasing. And it is only natural that scholars came with an idea to use the effort "wasted" during gameplay for pure entertainment in some meaningful way. This resulted in an emergence of *serious games*, which can be defined as "any piece of software that merges a non-entertaining purpose (serious) with a video game structure (game)" [2]. While this definition is broad, the term is mostly used for games with educational goals. A

special group of serious games are so-called *games with purpose* [3]. They look like ordinary games but the gameplay is designed in such a way that the players are (unconsciously) helping to solve computational problems by playing them.

Safety-critical systems (SCS) are systems, which failure could lead to unacceptable consequences [4] such as a loss of human lives or of expensive equipment. SCS are usually controlled by computers and it is up to these discrete controllers to ensure that no failure occurs. To guarantee this, formal methods (FM) for specification and verification of discrete systems should be used during their development. A typical FM for hardware or software development provides a language with unambiguously defined syntax and semantics to write specifications of systems and their safety-critical properties and a mathematical apparatus to analyse, verify or refine these specifications. The prominent formal verification techniques are model checking and theorem proving and they provide one significant advantage over testing: after the system is verified we can be sure that the properties hold in every state of the system. However, no formal method can verify the correctness of the properties themselves as these are "distilled" from informal requirements. The process of checking a formal specification of a system against the informal requirements is called validation and some of the contemporary FM tools support it through so-called animation, which is a (symbolic) execution of the specification. But these tools offer only basic execution and do not take features of the environment where the specified system will operate into account.

The question is how to provide virtual environments realistic enough to allow serious reasoning about the requirements and corresponding formalized safety properties. Here we propose to embed an executable prototype of the formally specified software controller into a computer game, which serves as a virtual representation of the device and environment where the controller will operate. The benefit of using computer games in this way is twofold. First, we can utilize already existing games and assets together with the contemporary game engines to create the environments relatively fast. Second, such a game can be used as game with purpose: its users will interact with the controller (i.e. any entity representing it in the game) during gameplay and generate valuable data for analysing the behaviour of the controller. Two implementations of the proposal have been considered and experimented with: an extension of an already existing complex game by an interface for formally developed controllers and a development of a separate game, built around a specific controller. In this paper we deal with the first one, which allows using verified safety-critical software controllers (*SCSC*) with *Open Rails*[1] (*OR*) 3D train simulator. We call the implementation *OR/TS2JC* as it consists of two components, a modified version of OR and an application called *TS2JavaConn* (*TS2JC*), which provides communication between OR and SCSC. In OR/TS2JC the SCSC manipulate signals, switches and trains according to events that occur in OR during gameplay (simulation). Trains can also be controlled manually, by players. The implementation is a development of a previous one [5], which utilized a much

[1] http://openrails.org/

simpler 2D simulation game called Train Director and has been used as a virtual laboratory environment in FM teaching. The replacement of Train Director by Open Rails was necessary in order to provide a truthful representation of the environment where SCSC should operate as OR has a realistic model of train operation. In addition, because OR is a successor of the Microsoft Train Simulator (MSTS), thousands of routes, faithfully modelled for MSTS after the real ones from all over the world, can be used with OR/TS2JC. This allows to develop, validate and evaluate controller prototypes for routes where automatization of railway traffic control didn't even started.

In this paper we present the current state of OR/TS2JC development and the rest of it is organized as follows. Section 2 compares our work to similar ones of others and section 3 specifies FM suitable for our approach and explains the problem of the verification and validation on a small example. Section 4 introduces a general scheme of the connection between a SCSC and a game, used in both implementations, and section 5 deals with OR/TS2JC. The paper concludes with a summary of achieved results and plans for future research and development.

## 2. Related Work

The idea of using an already existing non-serious game for research purposes is not new. One of the most prominent examples is the real-time strategy game Starcraft, which has been turned into a virtual laboratory for numerous works in artificial intelligence [6], such as a design of a new pathfinding algorithm for combat units [7] or a genetic programming-based automatic generation of high-level strategies [8]. According to [6], most of these works are implemented via the Brood War Application Programming Interface (BWAPI) [9], which allows replacing a human player with a computer program (bot). Our work is most related to the BWAPI itself. The similarity is in the primary purpose – to provide an interface through which a separately developed program is able to control the gameplay. However, it differs in the computer game chosen and the focus on the support of SCSC, compiled from source code generated by contemporary FM tools. In addition, the OR/TS2JC interface allows to control not only elements operated by human players (i.e. trains), but also elements normally managed by OR itself (i.e. signals and switches).

Formal methods and games are combined in the Crowd Sourced Formal Verification (CSFV) program of the U.S. defence agency DARPA. The CSFV web site[2] provides several games with purpose [10] where the task of human-assisted formal verification, which requires highly-trained professionals, is transformed into a gameplay accessible to much greater audience of casual players. For example, Xylem [11] is a game, where players help to find program loop invariants by observing and describing behaviour of plants. The similarity between CSFV and our approach lies in the utilization of the players' effort for FM-related tasks. It is even targeted to the same area of FM application, the development of reliable, verified, software. On the other hand, OR/TS2JC uses a modification of an already existing

[2] http://www.verigames.com/

game while CSFV games are newly designed with the verification purpose in mind. In fact, CSFV is complementary to our work: The CSFV games can be used to verify SCSC, which are subsequently evaluated and validated using OR/TS2JC.

Our implementation with OR as well as the previous one [5] with Train Director have been also inspired by existing solutions for animation and visualization of formally specified systems. Two such tools exist for Event-B [12], a formal method similar to B-Method [13], which is our primary FM of choice. The first tool is Brama[3], where custom visualizations can be made by connecting a specification in Event-B with Adobe Flash animation. The second one is B-Motion Studio [14], with animations composed from pictures. These solutions are different from OR/TS2JC as the visualizations have to be created from scratch but without a limited domain (railways), they connect with formally developed software on the specification and not implementation level and are limited to one formal method. The same can be said about the APEX framework [15], which connects a multi-user 3D application server for creation and running of 3D virtual environments called OpenSimulator[4], with CPN Tools[5] an editor, simulator and analyser for Coloured Petri nets (CPN) formal method. The APEX is similar to our work in utilizing an existing tool (OpenSimulator) to create virtual environments with entities managed by formally developed controllers (by CPN in APEX). Another similarity is in the utilization of the TCP protocol for the communication between the simulator and the controller.

## 3. Formal Methods: Suitability, Verification and Validation

OR/TS2JC has been developed as a virtual environment for evaluation and validation of railway SCSC and as such is suitable for software development formal methods fulfilling the following criteria:

1. *Formal verification support.* The language of the method has to support safety properties specification and there should be a method and a software tool for formal verification of the system specification against these properties.
2. *Verified development process.* All effort put into writing the formal specification of the system and its properties can be wasted if there is no way to verify that the same or similar properties will hold in an implementation of the system. Therefore the method should be able to generate executable specifications automatically or provide means to verify that the properties still hold in a manually written implementation of the system.
3. *Translation to Java.* OR/TS2JC accepts SCSC written in the Java general purpose programming language, so the tools for the formal method have to include a compiler to this language. Java has been chosen because it is

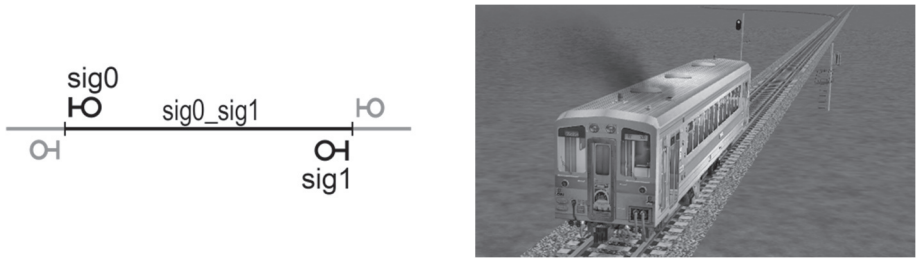
[3] http://www.brama.fr/

[4] http://opensimulator.org/

[5] http://cpntools.org/

supported by several formal methods that fulfil the first two criteria and because of its popularity with students and developers.

Only the third criterion is mandatory, but without the first two our effort will not be meaningful: Why to spend a lot of time by preparing an environment such as OR/TS2JC for validation and evaluation when there are no means to verify the validated properties in both formal and executable specification of the system? Fortunately, several FM fulfil the criteria, such as B-Method [13], Event-B [12], Perfect [16] and VDM [17]. These FM are frequently used in practice, with B-Method and VDM primarily in the railway domain.




a) b)

Figure 1. Schema of a track segment guarded by two signals (a) and a train leaving a similar segment in Open Rails simulator (b).

Before going to OR/TS2JC and the principles behind it, let us illustrate SCSC development process on an example. The example uses *B-Method* (*B*) and presents a part of a controller of a very simple railway track segment, containing one track section (sig0_sig1) and two signals (sig0, sig1), which guard the access to it (Fig.1 a). The development of its controller in B starts with writing an abstract formal specification in B-language, the specification language of B. The specification consists of one or more components, called *B-machines*. For our controller we need only one machine, TContrl. Part of the specification of TContrl in B-language looks as follows:

```
MACHINE TContrl
SETS SIGNAL={green, red}; SECTION={free,occup}

CONCRETE_VARIABLES sig0, sig1, sig0_sig1

INVARIANT sig0:SIGNAL & sig1:SIGNAL & sig0_sig1:SECTION &
            (sig0=red or sig1=red) & ((sig0=red   & sig1=red) or sig0_sig1=free)

INITIALISATION sig0:=red || sig1:=red || sig0_sig1:= free

OPERATIONS
  ok<--reqGreen_sig0 =
    PRE sig0=red & sig1=red & sig0_sig1= free THEN sig1:=green || ok := TRUE END;
  …
END
```

The clause SETS of TContrl defines two new types, the enumerated sets SIGNAL and SECTION. Three state variables of the machine, representing the signals and section of the track segment, are listed in the CONCRETE_VARIABLES clause. The INVARIANT clause defines properties of the variables; its first line types the variables and the second line contains a *safety property*, which can be informally written as

"*only one of the two signals sig0 and sig1 can be green and both of them are red if sig0_sig1 is occupied*".

INITIALISATION contains an operation that sets the initial state of the machine, here both signals to red and the section to free. The operator ":=" is assignment and "||" is parallel composition (i.e. S1||S2... means do S1 and S2 at once). Finally, we have operations, which can be called from other components. The controller has several operations and the one shown here, reqGreen_sig0, is called when a train approaches sig0. The operation has no input parameter and one output parameter, named ok. It contains a command that cannot be found in usual programming languages, the PRE P THEN S END, where P is a condition and S is a command. It means "if P holds it is safe to execute S, otherwise anything can happen". The *verification* of TContrl is a formal proof of a set of theorems saying that the initialisation operation establishes the invariant properties and every operation maintains them. In simple cases the proof is discharged automatically by corresponding tools (e.g. Atelier B[6]), in complicated ones it has to be performed with a human assistance.

After the proof we *refine* (i.e. concretize) the abstract specification into an implementable one. This can be a complex process with several steps and significant modification of used algorithms and data representation. In our case it is sufficient to replace "||" by ";" (sequential composition) and get rid of the non-implementable PRE command. For example, the refined reqGreen_sig0 will look like

```
ok<--reqGreen_sig0 =
  IF sig0=red & sig1=red & sig0_sig1= free THEN sig1:=green ; ok := TRUE
  ELSE ok := FALSE END;
```

Correctness of the refinement steps is, again, verified by formal proofs. Finally, we can generate the Java code of the controller using BKPI compiler [18]. After this the method generated from reqGreen_sig0 will be:

```
public Boolean reqGreen_sig0() {
   Boolean ok;
   if (((sig0 == SIGNAL.red && sig1 == SIGNAL.red) && e0_sig0 == SECTION.free)) {
      sig0 = SIGNAL.green; ok = true; } else { ok = false; }
   return ok;
}
```

[6] http://www.atelierb.eu/en/

The code we got is correct but only with respect to the safety properties we specified. But the properties themselves can be incorrect. For example, the section sig0_sig1 can be so short that a train already inside one of the adjacent sections will be unable to stop before sig0 or sig1. And this is exactly where OR/TS2JC can help to validate the properties: the controller is connected to a corresponding reasonably realistic representation of the track segment in OR (Fig.1 b) and adequacy of the properties is evaluated during gameplay.

## 4. Game and Controller as a Hybrid System

Both implementations of the embedding of an executable SCSC prototype into a computer game implement the same architectural principle, derived from the operation of a hybrid system. By the hybrid system we mean a system defined according to [19], i.e. a system that combines time-driven and event-driven subsystems. These two kinds of systems differ in the state transition mechanism [19]: the state of a time-driven system changes as time changes while the state of an event-driven one changes at certain points in time only, i.e. when some event occurs.

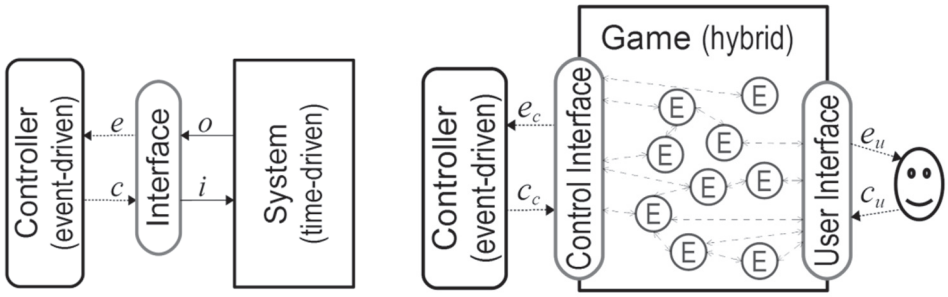



Figure 2. Hybrid system (a) and a general schema of a controller and computer game composition (b).

In particular, we consider a configuration where an event-driven controller is supervising a time-driven system (Fig. 2 a) and the controller has a form of a computer system, running a SCSC. The controller reacts to events (*e* in Fig. 2 a) observed in the time-driven system by issuing commands (*c*), which affect the system. Of course, there needs to be an interface, which creates *e* on the basis of output signals (*o*) and transforms *c* to input signals (*i*) of the system. For example, the output signals can be about the distance between a train and a railway signal and when the distance decreases below a certain value, the interface generates an event "the train requests green" to which the controller reacts by running a procedure (e.g. reqGreen_sig0 from section 3), which evaluates the situation and issues a command. Then the interface transforms the command to an input signal, which switches the railway signal.

In our work the real system is replaced with a computer game (Fig. 2 b). In general, a contemporary computer game is a hybrid system as some of its variables

change with time (e.g. the ones computed by a physics engine) and others when an event (e.g. an input from a player) occurs. We can see the game as a composition of entities ($E$), some of them time-driven, some event-driven and some hybrid, which communicate via corresponding internal interfaces (dashed lines in Fig. 2 b). In addition to the user interface, which communicates events ($e_u$) to the player as audio and video and processes his commands ($c_u$), the game will need a *control interface* to interact with the controller. Both interfaces can share the same set of events and commands, as in the case of BWAPI where the controller replaces a human player. On the other hand, the sets can be disjoint, e.g. when a controller is supervising a non-playable character. In OR/TS2JC some of the events and commands related to trains are shared by both interfaces while others belong to only one of them.

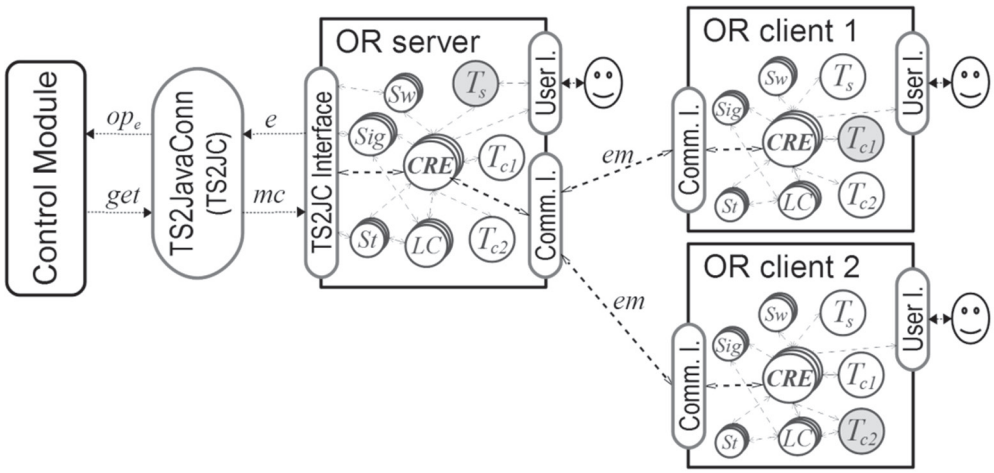



Figure 3. OR/TS2JC architecture.

## 5. OR/TS2JC

The OR/TS2JC (Fig. 3) differs from the general schema (Fig.2 b) in two important aspects. First, there is a separate application, *TS2JavaConn* (*TS2JC*), serving as an additional interface between the modified Open Rails (OR) and SCSC, which we call *control module*. *TS2JC* makes it easier to adapt our work to different railway simulators and is in more detail described in section 5.2. Second, OR itself can be run in a multiplayer mode where each instance is responsible for one train and only the server instance connects to TS2JC. The multiplayer allowed us to implement three different ways of train control. We return to them later, in section 5.3.

The *control modules* (*CM*) are Java applications possibly, but not necessarily, generated by FM tools. They contain two kinds of methods:

1. *Event methods*, called by *TS2JC* when a corresponding event occurs in OR. It is expected that they change the state (i.e. the values of variables) of the module. The reqGreen_sig0 from section 3 is an event method.

2. *Getters.* They return current state of signals, switches and trains, computed by the event methods. Getters should not change the state of the module.

It is up to the developer how the variables of CM will capture the state of the controlled railway scenario and what exactly the methods will do and return.

As in the general schema, the modified OR can be seen as a composition of different entities. Here we distinguish entities representing railway signals (*Sig* in Fig. 3), switches (*Sw*), stations (*St*), trains ($T_S$, $T_{c1}$, $T_{c2}$), locomotive controllers (*LC*) and other entities (*CRE*), including the audio-visual output renderer and the part responsible for the communication between OR instances. The most important modifications of OR have been the creation of the TS2JC interface, changes in the game logic in order to cooperate with the interface, extension of the interface for client-server communication (Comm l. in Fig. 3) to incorporate TS2JC messages and changes in the OR graphical user interface to be able to set new preferences and see names of entities in corresponding views.

To use OR/TS2JC we first need to start one or more instances of the modified OR with exactly one of them in the server mode. In each instance we set the same route and activity and choose a user name. The user names have to be the same as train names in the corresponding CM. Then we start TS2JC and connect to OR. This causes TS2JC to request route data from OR. After receiving the data TS2JC displays their names in its GUI. Then we load a CM in TS2JC and TS2JC checks whether it is a correct module for the route. If yes, the route and trains state in OR is set according to the CM and we can start the simulation. After this the user(s) can operate manually operated trains or act as validators by observing the behaviour of the controller.

### 5.1. Event Processing

During the simulation most of the communication between OR and CM is initiated by events caused by trains in OR. In general, the communication proceeds as follows:

1. An event occurs in an OR instance.
2. The instance composes a message consisting of the event name and data (parameters). If the instance is an OR server then the message (*e* in Fig. 3) is sent directly to TS2JC. If it is a client instance, the message is first sent to the server instance as an extended multiplayer message (*em*) and then to TS2JC as *e*.
3. TS2JC receives *e* and translates it into a call ($op_e$ in Fig. 3) of the corresponding method in the CM.
4. TS2JC executes $op_e$. This usually changes values of the CM variables.
5. TS2JC calls getters of the CM, stores returned values (*get* in Fig. 3) and compares them with the values $get_p$ stored after the previous event.
6. TS2JC composes so-called *multicommand message* (*mc* in Fig. 3), consisting of those values of *get* that are different from $get_p$ and sends *mc* to the OR server. The server receives *mc* and updates all affected signals,

switches and trains. Connected client instances are updated, too, via *em* messages.

| Event | Data | Cause |
|---|---|---|
| *speedChanged* | *t, s* | Speed of the train *t* changed by an amount set in preferences of the modified OR. The resulting speed value is *s*. |
| *redSignalApproach* | *i, f, t* | The distance between the train *t* with the destination *f* and the signal *i* decreased to predefined distance *rs*. |
| *requestGreen* | *i, f, t* | The distance between the train *t* with the destination *f* and the signal *i* decreased to predefined distance *rg*. |
| *signalStateChanged* | *i, f, t* | The signal *i* before the train *t* with the destination *f* has been switched to green. The train *t* is before *i* if it is as close or closer to *i* as predefined distance *sc*. |
| *sectionEnter* | *t, s* | The train *t* entered the section *s*. |
| *getSpeed* | *t, s* | The train *t* had entered the section *s* and asked for the speed required for *s*. |
| *sectionLeave* | *t, s* | The train *t* left the section *s*. |
| *stationApproach* | *t, a, f, s* | The distance between *t* with *f* and the station *a*, which occupies the section *s*, decreased to predefined distance *sa*. |
| *requestDepartureStation* | *t, a, f, s* | The train *t* with the dest. *f* asked for permission to leave the station *a* occupying the section *s*. |
| *canLeaveStation* | *t, a, f, s* | The permission for *t* with *f* to leave *a* occupying *s* has been granted. |

Table 1. Selection of events triggered during simulation

The process repeats for every event. The most important events are listed in Table 1. The data sent with event messages are identifiers of involved entities: trains (*t*), signals (*i*), sections (*s*) and stations (*a*, *f*). The sections are defined by surrounding switches and signals, so we have sections between two signals, two switches or a signal and a switch. The events *redSignalApproach* and *requestGreen* are similar; they differ only in the train distance from the signal. The *requestGreen* can be seen as a failsafe and its predefined distance *rg* should be shorter than *rs*. The processing of *getSpeed* differs from other events as it doesn't involve the calling of all the getters from CM. Only a method associated with the event is called and the speed required for *s* is expected as its output parameter. There are also other

important events, for example *ready*, which occurs when an OR instance with a train *t* became available and *getDistance* to get the distances *rs*, *rg*, *sc* and *sa* from CM.

### 5.2. TS2JavaConn and Customizable Control Interface

*TS2JavaConn* (*TS2JC*) is a separate Java application responsible for three important tasks:

1. *Communication between OR and CM*. TS2JC translates event messages from OR to CM method calls and values returned by CM getters to commands for OR.
2. *Control process monitoring.* The Overview tab of the TS2JC user interface (Fig. 4 a) lists state of all important elements in the actual route and activity, loaded in OR. It also contains a logger with a detailed history of events, method calls and commands. Most of the event and data names shown in the logger are the same as in Table 1.
3. *Control module template generation*. TS2JC allows to generate a template of a control module with variables and method (operation) headers corresponding to the route loaded in OR. This feature is currently available for Java and specification languages of B-Method and Perfect.

As we mentioned earlier, there are several FM with code generators to Java. To make it possible to use the CM code generated by them without any or with minimal modifications, TS2JC allows to customize the way how CM and OR communicate. The customization is implemented by means of a *properties file,* associated with each CM. The properties file defines how the event methods are called, which event data are used, whether the data are a part of method names or input parameters and what is the meaning of values the getters return. For example, an event message

*requestGreen* with parameters *i=sig0, f=sta3* and *t=train0*

can be translated to a method call, which uses only the first parameter as a part of its name

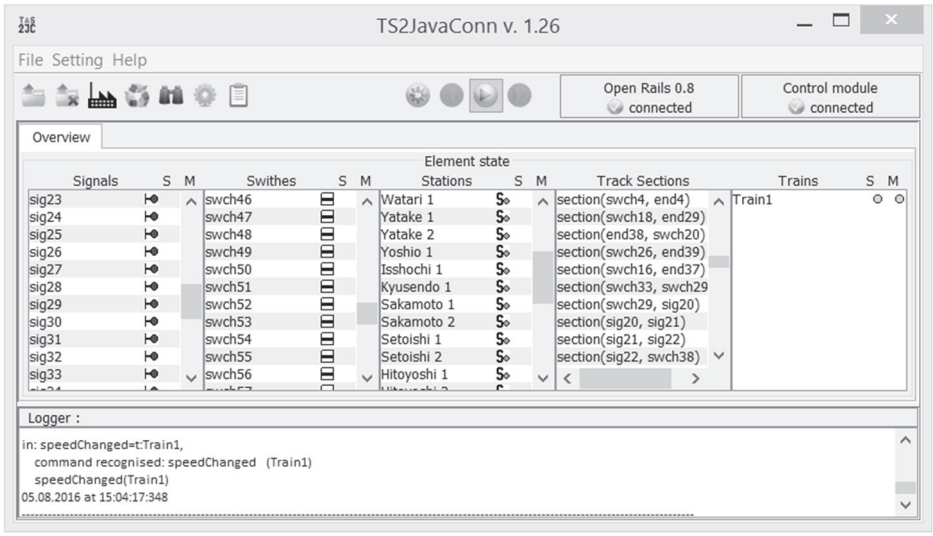
*requestGreen_sig0()*

or uses first two event parameters as method parameters

*requestGreen(sig0, sta3)*

or combines both ways

*requestGreen_sig0(sta3)*.




a)


b)

Figure 4. TS2JavaConn (a) and Open Rails (b) during a simulation of Hisatsu Line route.

### 5.3. Multiplayer and Three Ways of Train Control

Open Rails is a train simulator, so one OR instance handles one train operated by a player. Yet, for faithful representation of real-life situations we need routes populated by more than one train. To accomplish this, two approaches were considered. First, we tried to use automatic (AI) trains, which can be added to routes and operated according to a schedule, defined for corresponding activity. However, required modifications of OR proved to be too complex and the achieved result has been only partially satisfying (i.e. a route and activity with several AI trains and only one human operated train) Therefore, it was decided to utilize the multiplayer mode of OR. The already existing protocol for communication between OR server and client instances has been extended to be able to carry the multicommand messages,

so only the server instance needs to communicate with TS2JC. To provide maximal flexibility, three different ways of train control have been implemented and each route and activity, controlled by one CM, can contain trains of all three corresponding types. Namely, the train types are:

1. *Manual*. A train controlled by a player via the OR user interface. The availability of such trains is very important for the validation and evaluation process as they allow to test CM against unpredictable behaviour of human operators (i.e. a driver ignoring signals set by CM).
2. *CM-controlled*. Train operated by CM. The control is realised via CM responses to the *getSpeed* event, which define train speeds for individual sections. How the train reaches the speed is implemented on the OR side.
3. *Bot*. Train operated by locomotive controllers (*LC* in Fig. 3), added to OR. These trains always obey signals and defined schedule and are useable when there is a need to populate the route with trains without any additional investment in their control.

All three types are included in Fig. 3: a manual train $T_S$ on the OR server, a CM-controlled train $T_{c1}$ on the OR client 1 and a bot train $T_{c2}$ on client 2.

## 6. Conclusion

The development of the OR/TS2JC environment, where validation and evaluation of railway controllers, developed by contemporary FM, can be realized in a form of gameplay, reached an important milestone; it is ready to be used with existing railway routes and activities, created originally for MSTS. Great amount of time and effort has been invested not only to the design and implementation of the communication interface, JS2JC and modifications of OR, but also to the fixing of several issues that emerged during the testing with MSTS routes, such as unambiguous names of track sections. In the future the development of OR/TS2JC will focus on better support of validation tasks, such as more detailed logs, development of concrete control modules using FM mentioned in section 3 and other formalisms, e.g. those based on the category theory [20], [21], and an extension of the communication between OR and CM, especially in the area of train control. We also intent to use the Java interface [22] of the ProB [23] animator and model checker for B-Method, Event-B and several other FM to be able to connect with SCSC on the formal specification and not implementation level. Regarding B-Method, the CM can also benefit from a combination of B-Method and Petri nets, pioneered in [24], [25] and further extended to other Petri net dialects and railway domain in works such as [26] or [27].

It should be also noted that while OR/TS2JC has been developed with FM in mind, the controllers don't need to utilize FM at all. The only condition is that they are written in Java and provide interface acceptable by OR/TS2JC. OR/TS2JC is available from *http://hron.fei.tuke.sk/~korecko/FMInGamesExp/*.

## Acknowledgements

The paper is a result of the Project implementation: University Science Park TECHNICOM for Innovation Applications Supported by Knowledge Technology, ITMS: 26220220182, supported by the Research & Development Operational Programme funded by the ERDF. The authors would like to thank all the students who participated on the development of OR/TS2JC.

## References

[1] Statista. (2016). Daily time spent playing video games per capita in the United States in 2008, 2013 and 2018, [Online]. Available: http://www.statista.com/statistics/186960/time-spent-with-videogames-in-the-us-since-2002/.

[2] D. Djaouti et al., "Classifying Serious Games: the G/P/S model," in *Handbook of Research on Improving Learning and Motivation through Educational Games: Multidisciplinary Approaches*, P. Felicia, Ed., IGI Global, 2011, pp. 118-136.

[3] L. von Ahn, "Games with a Purpose," *Computer*, vol. 39, no. 6, pp. 92–94, June, 2006.

[4] J. C. Knight, "Safety critical systems: challenges and directions," in *24rd Int. Conf. on Software Engineering (ICSE 2002)*, Orlando, FL, USA, 2002, pp. 547-550.

[5] Š. Korečko and J. Sorád, "Using simulation games in teaching formal methods for software development," in *Innovative Teaching Strategies and New Learning Paradigms in Computer Programming, R. Queirós, Ed.,* IGI Global, 2015, pp. 106–130.

[6] Ontanon, S. et al., "A Survey of Real-Time Strategy Game AI Research and Competition in StarCraft," *IEEE Trans. on Computational Intell. and AI in Games*, vol. 5, no. 4, pp.1-19, 2013.

[7] J. Hagelbäck, "Potential-field based navigation in StarCraft," in *2012 IEEE Conf. on Computational Intell. and Games (CIG)*, Granada, 2012, pp. 388-393.

[8] P. Garćıa-Sánchez, A. Tonda, A. M. Mora, G. Squillero and J. J. Merelo, "Towards automatic StarCraft strategy generation using genetic programming," in *2015 IEEE Conf. on Computational Intell. and Games (CIG)*, Tainan, 2015, pp. 284-291.

[9] Heinermann, A. (2017, April 19) *BWAPI—An API for interacting with with Star-Craft: Broodwar* [online] Available: http://bwapi.github.io/

[10] D. Dean et al., "Lessons learned in game development for crowdsourced software formal verification," in *2015 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE'15)*, 2015.

[11] H. Logas et al., "Software Verification Games: Designing Xylem, The Code of Plants," in *Proc. 9th Int. Conf. Foundations of Digital Games (FDG 2014)*, Ft. Lauderdale, FL, USA, 2014.

[12] J. R. Abrial, Modeling in Event-B: System and Software Engineering, Cambridge University Press, 2010.

[13] J. R. Abrial, *The B-Book: Assigning Programs to Meanings*, Cambridge University Press, 1996.

[14] L. Ladenberger et al., "Visualising Event-B Models with B-Motion Studio," in *Proc. of Formal Methods for Industrial Critical Systems*, LNCS vol.5825, Springer, 2009, pp. 202-204.

[15] J. L. Silva et al., "The APEX Framework: Prototyping of Ubiquitous Environments Based on Petri Nets," in *Human-Centred Software Engineering*, LNCS, vol. 6409, Springer, 2010, pp. 6-21.

[16] D. Crocker, "Safe object-oriented software: the verified design-by-contract paradigm," in *Practical Elements of Safety*, Springer, 2004, pp. 19-41.

[17] J. Fitzgerald et al., Validated Designs for Object–oriented Systems, Springer, 2005.

[18] Š. Korečko and M. Dancák, "Some Aspects of BKPI B-language Compiler Design," *Egyptian Comput. Sci. J.*, vol. 35, no.3, pp.33–43, 2011.

[19] Ch. G. Cassandras and S. Lafortune, Introduction to Discrete Event Systems, second edition, Springer, 2008.

[20] W. Steingartner and V. Novitzká, "A new approach to semantics of procedures in categorical terms," in *13th IEEE Int. Scientific Conf. on Informatics*, Poprad, 2015, pp. 252-257.

[21] W. Steingartner et al., "An Analysis of some Aspects of Component-Based Programming for Selecting Appropriate Categorical Structures as Their Models," *Acta Electrotechnica et Informatica*, vol. 17, no. 2, pp. 3-10, 2017

[22] J. Bendisposto and J. Clark (2014): *ProB 2.0 Developer Handbook*, [online]Available: https://www3.hhu.de/stups/prob/index.php/ProB_Java_API_Developer_Manual

[23] M. Leuschel and M. Butler, "Prob: an automated analysis toolset for the B method," *Int. J. Softw. Tools Technol. Transf.*, vol. 10, no. 2, pp. 185-203, March 2008.

[24] Š. Korečko and Š. Hudák, "Implementing Petri nets via B-method", in: *Proc. Of Electron. Comput. and Informatics (ECI' 2004)*, Herľany, Slovakia, 2004, pp. 103-110.

[25] Š. Korečko et al., "Petri net - like treatment of B-machine behaviour," *J. of Inform., Control and Management Syst*. vol. 5, no. 2, pp. 213-223, 2007.

[26] P. Sun et. al., "A joint development of coloured petri nets and the B method in critical systems," J. of Universal Comput. Sci., vol. 21, no. 12, pp. 1654-1683, 2015.

[27] Z. Boudi et al. "A CPN/B method transformation framework for railway safety rules formal validation," *European Transport Research Review*, vol. 9, no. 2, art. no. 13, 2017.