Using Simulation and 3D Graphics Software to Visualize Formally Developed Control Systems

ŠTEFAN KOREČKO, BRANISLAV SOBOTA, CSABA SZABÓ
Department of Computers and Informatics
FEEI, Technical University of Košice
Letná 9, 042 00 Košice
SLOVAK REPUBLIC
stefan.korecko@tuke.sk, csaba.szabo@tuke.sk, branislav.sobota@tuke.sk, http://dci.fei.tuke.sk

Abstract: This paper proposes an approach connecting an automated control system, developed using formal methods, with an emulation of an environment where it is intended to operate. The motivation is to improve formal methods teaching by providing virtual representation of environments where the formally developed systems are or should be used. The emphasis is on the visualisation of the implementation of the control system and use of existing tools for visualisations and existing models for virtual environments and devices. The paper also describes a concrete application of this approach: a connection of a track layout, simulated by the Train Director software, with a control system, developed using the B-Method. This includes utilisation of various tools developed at the home institution of the authors: translator from the language of B-Method to Java, an application to interconnect the control system with the Train Director and a customized version of the Train Director.

Key-Words: B-Method, Event-B, Perfect Developer, visualization, simulation, virtual reality, control systems

1 Introduction
Formal methods, such as B-Method [1], Event-B [2] or Perfect Developer [3], are used for a verified design and development of software systems. They provide a way to write a formal specification of a system, including desired properties, prove that it really has the properties and refine the specification to an implementable one, preserving the properties. Maybe a use of these methods cannot guarantee a total correctness of the system, but it can at least ensure that the system will never make a decision that can harm human lives or cause an extensive damage.

Despite their indisputable benefits the formal methods are not used as often as desired, even in their typical area of use: a design and development of safety-critical systems making automated decisions, such as automated control systems of transportation vehicles, military or medical systems.

One of the reasons of this situation is that software engineering students cannot sufficiently imagine a typical environment where it is appropriate to use formal methods, because they simply have no experience with such environments. Hence there is a risk that they will have no real motivation to use the methods.

In this paper we explore the ways of overcoming this issue by substituting the environment by its emulation and connecting the emulation with formally developed control system. The emulation will represent the environment and devices to be controlled and will be provided by appropriate simulation software, preferably with a possibility of 3D visualization and utilization of virtual reality technologies. The use of general-purpose or gaming 3D engines is also possible. We also present a concrete application of this approach: a connection of a scenario, simulated by the Train Director software, with a control system, developed using the B-Method. The scenario consists of track layout and train schedule and the control system serves as a centralized traffic control. This application involves a use of several tools developed at the home institution of the authors: translator from the language of B-Method to Java and C# programming languages, application to interconnect the control system in Java with the Train Director and a customized version of the Train Director.

Our effort concentrates on the three formal methods mentioned above because of their successful industrial use (especially B-Method, for subway control systems), broad academic and industrial community (Event-B, via Rodin and Deploy projects) and application of object-oriented approach and suitability for teaching (Perfect Developer, as reported in [4]). There is also a good
tool support for these methods: B-Method has an industrial tool called Atelier B for writing, refining and proving specifications and generating code, Event-B can use Atelier B or Eclipse-based Rodin platform and Perfect Developer is also a name of a tool with the similar functionality as Atelier B.

2 Related Work
An idea to visualize formal specification of a system by means of computer graphics, including 3D, is not new. It has been proposed and used many times, primarily to make formal specifications more understandable for domain experts, validating the specifications against requirements.

Brama [11] serves as an animator (i.e. a tool that can “run” a formal specification) and visualisation tool for Event-B. It is available as a plug-in for Rodin and allows making a custom visualisation by connecting the specification with Adobe Flash animation. The connection is defined by gluing code written in ActionScript.

Another visualisation solutions for Event-B (and B-Method) are provided by a team that developed an animator and model checker for these methods, called ProB [7]. The team first introduced a solution similar to Brama, which used Java for the gluing code. However, they had concluded that in many cases it is inconvenient to prepare the Flash animation and write the gluing code and introduced a simpler and easy to use solution [8] where animations are composed from pictures and the gluing code is written in the language of B-Method. Finally, this evolved to a more sophisticated B-Motion Studio [6], a plug-in for Rodin, which allows to create rich animations using the approach of [8].

Similar solutions have been developed for other formal methods as well. A notable one is the Geist3D integrated development tool [10] for automated manufacturing systems, which allows to combine realistic prototypes of machinery, prepared using virtual reality technology, with a control system written in a modelling language based on timed high-level Petri nets.

3 Connecting Control System with Virtual Environment
The approach presented here is in many ways similar to the previous ones: As in the case of virtually all the mentioned solutions we would like to connect a formally developed system with a graphical representation of devices and an environment where it is used. As in the case Geist3D we would like to use 3D graphics and state-of-the-art virtual reality technology. As in the case of the ProB-based visualisation, our goal is to teach formal methods better. On the other hand, it differs in (at least one of) the following respects:

- The emphasis is on the use of existing tools for visualisations and, if possible, of existing models for virtual environments and devices.
- The virtual environment is created primarily for the implementation and not for the formal specification of the developed system.

The benefit of using existing solutions is twofold. First, we save a lot of work that would be required to implement a simulation or visualisation tool and, if lucky, also to prepare corresponding models. Second, we can propagate formal methods in the community that already uses the tool by providing case studies and tutorials directly for the environment they already know.

One can say that the focus on the implementation instead of specification is unfounded: After generating an executable code the visualization is no more needed, because the application can be simply run. However, this is not entirely true. The formal methods usually come with very limited application program interface (API), for example in B-Method one can use string variables only as input parameters of operations. Only the Perfect Developer provides quite nice API, but even this lacks a support for graphical user interfaces. Finally we get an application with limited functionality that can be run in text mode only. And this is hardly satisfying for someone (a student) spending a lot of time developing the application. We cannot blame formal methods for this, because they are usually not intended to write complete applications and, as stated earlier, their typical area of use is rather special.

To teach formal methods in more satisfying and motivating way we propose to use virtual environments, instead of the real ones that are not normally accessible on “academic ground”. The overall scheme is shown in Fig. 1. It involves several pieces of software, including existing ones (white boxes) and new or modified ones (grey boxes). The existing tools are development environments for formal methods (FmTool), such as Atelier B or Perfect Developer, animators, such as ProB and visualisation and simulation tools (VisSim) for running the virtual environments. The VisSim component can be, for example, an appropriate
A simulator such as Train Director, a general purpose 3D graphics engine such as OpenSceneGraph or a gaming engine like Unity 3.

Whether the code generator (CodeGen) will be a new or existing one depends on the FmTool and VisSim components chosen. With available generators, including our own BKPI generator for B-Method, we can translate a system implemented using B-Method or Perfect Developer to ADA, C/C++, C# or Java. However, if we require a code in a scripting language of some graphics engine, we need to extend the generators to support it.

Probably the most crucial component is Conn, which provides cooperation between the developed control system and virtual environment. In general this gluing component can have a form of an executable application or a programming library (API) written in a language supported by given VisSim tool for scripting. The Conn implemented as an application is suitable for VisSim tools that don’t support scripting or when we don’t want to rely on the supported scripting languages. However, in the case of 3D engines the form of API is preferred (provided that the CodeGen can produce code in the scripting language of the engine), because a communication with external application via Conn as a separate tool can lead to an unacceptable decrease in frame rate. This is especially critical for a use of stereoscopic visualisation, which requires double than normal frame rates.

Of course, a practical application of this approach to formal methods teaching means an additional work for teachers involved. The work is in the preparation of assignments for students, where we not only need to prepare precisely formulated assignments but also to design the virtual environments where results, formally developed applications, will run. In addition the environments should be functional to allow students to explore them when designing their applications. On the other hand it gives us a chance to prepare interesting ones, for example a development of a yard control system for a workable track layout. Utilisation of game engines and existing moddable games can lead to even more motivating assignments such as “Given a set of movement sensors program the automated defence turret to not let the enemy cross the camp border.” or “Develop a control application for a bot in the multiplayer mode of game XY”.

Because the connection between the system developed and the virtual environment in on the implementation level, the approach is not limited to the area of formal methods. We can use an expert system or a conventionally developed control application instead.

### 4 The Train Director Case

In this section we present a proof-of-concept application of the approach proposed – an integration of control application developed using B-Method with simulation of corresponding scenario in Train Director. We used Atelier B as the FmTool and ProB as the animator.

The CodeGen component is BKPI code generator, developed at the home institution of the authors. The version used here is an enhancement of the prototype [5] and allows to translate the implementable specification of B-Method to Java or C#. Its translation strategy is based on that of jBTools [9], but differs in various aspects, such as machines import mechanism and handling of output parameters. It also utilizes high-level features of modern programming languages, such as collections and generics. To help to preserve an internal consistency of the generated code within a bigger software development project the generator offers several modification policies. The development of BKPI was motivated by the fact that the latest version of jBTools is from 2003 and the only other freely available compiler for B-Method is ComenC
translator, shipped with Atelier B, which translates only to the C language.

The VisSim tool used is Train Director (TD) [12]. In TD one can design a scenario consisting of a track layout (as in Fig.2) and trains schedule. During the simulation trains move according to the schedule and signals and switches are operated partly automatically and partly by user. It is also possible to write scripts reacting to events in the scenario. The newest version of TD provides a server interface for external control. To fit our purposes TD was moderately modified. The modifications included disabling of internal control mechanisms, minor changes in user interface and correction of bugs found. In this customized TD the track layout is controlled solely by an external application. TD then serves as editor for track layouts and schedules and simulator of train movement according to the schedule selected.

Communication between an external control application and TD is provided by our connector tool named Td2JavaConn, which uses TD server interface. A special configuration file is used to map setting of signals, switches and entry points in TD to variables in the control application (CA) and events in TD to methods in CA. Various types of events can occur in TD, the most important are:

- request green – occurs when train stops before a red signal or someone tries to change a signal to green manually,
- enter section – when train enters a track section and
- leave section – when train leaves a track section.

The simulation itself proceeds as follows: First Td2JavaConn checks whether the control application contains all the methods needed to control the track layout. This is done using the Java reflection API. After successful control Td2JavaConn initialises the control application, reads values of its variables and sets up signals, switches and entry points according to them. Entry points are places where a new train can enter the system and they behave as signals. Then the simulation runs until some event occurs. After the event occurs, the corresponding method in CA is executed by Td2JavaConn. After executing the connector reads values of CA variables and adjusts settings of signals and switches in TD with respect to them. Then the simulation continues until the next event occurs or until terminated by user.

To give an impression of how a formal specification can look like, we present a piece of the specification of the control system for the track layout from Fig.2. This consists of a single B-Method specification component, a B-machine SimpleTrackCntrl. The B-machine has five parts: SETS defines new types, VARIABLES lists state variables, INVARIANT defines types and other properties of the state variables, INITIALISATION establishes an initial state of the B-machine and OPERATIONS lists operations that modify state (variables) of the machine:

```bmethod
MACHINE SimpleTrackCntrl

SETS SIGNAL={green, red};
SWITCH={switched, none} 
VARIABLES entry0, entry1, sig0, ..., sig7, swch0, swch1, entry0_sig1, sig0_swch0, swch0_sig2, ..., sig6_entry1
INVARIANT /*typing*/
  sig0:SIGNAL &... & sig7:SIGNAL &
  swch0:SWITCH & swch1:SWITCH &
  entry0:SIGNL & entry1:SIGNL &
  entry0_sig1:0..1 &...& sig6_entry1:0..1 &
  ... /*exclusivity of green for sig4,5,7*/
  (sig7=green=>(sig4=red & sig5=red)) &
  (sig4=green=>(sig5=red & sig7=red)) &
  (sig5=green=>(sig4=red & sig7=red)) &
  ...
  /*condition for green on sig4*/
  (sig4=green=>
    (sig5=red & sig7=red & swch1_sig7=0 &
     sig4_swch1=0 & sig5_swch1=0 &
     swch1=switched)) &
  ...
INITIALISATION 
  Sig 0:=red ... || sig7:=red ||
  swch0:=none || swch1:=none ||
  entry0:=red || entry1:=red ||
  entry0_sig1:=0 ||...|| sig6_entry1:=0 
OPERATIONS
  ...
  ok<--reqGreen_sig4 = IF
```
sig4=red & sig5=red & sig7=red &
sig4_swch1=0 & sig5_swch1=0 &
swch1_sig7=0
THEN
  swch1:=switched || sig4:=green
  || ok := TRUE
ELSE
  ok := FALSE
END;
...
ok<--enterIW_sig4_swch1 =
PRE
  sig4=green
THEN
  sig4_swch1:=1 || swch1_sig7:=1 ||
  sig4:=red
  || ok := TRUE
END;
...
END

In the machine we have two sets – one for states of signals and one for states of switches. The variables represent entry points (i.e. entry0), signals (sig0), switches (swch0) and track sections (entry0_sig1).

In the INVARIANT the typing of variables is followed by safety conditions. The section variables can have value 0 (free) or 1 (occupied). From numerous safety conditions we selected those concerning sig4. The first three conjuncts state that only one of sig4, sig5 and sig7, can be green and the last one defines what must be satisfied when sig4 is green: surrounding signals must be red, swch1 must be switched (i.e. up) and sections around swch1 must be free. The initialisation operation sets all signals to red, switches to not switched and sections to free. The “||” operator is parallel composition, “C1||C2” means “do C1 and C2”.

From the OPERATIONS part we again chose some of those concerning sig4. The reqGreen_sig4 is called when the request green event occurs for sig4 and sets sig4 to green if the corresponding conditions are satisfied. The IF THEN ELSE END command is classical conditional statement. The operation enterIW_sig4_swch1 is called when a train enters section sig4_swch1 from left. Here PRE command is used instead of IF. The “PRE P THEN C END” means “do C if P holds. If P doesn’t hold do anything.” This corresponds to the real situation: we have a full control over the signal, but we cannot prevent a train to enter a section.

The machine SimpleTrackCntrl has been subsequently refined to an implementation component and the implementation translated to Java using BKPI code generator. In this case both the implementation and the Java code look very similar to the machine, so we will not show them here.

5 Conclusion
In the paper we proposed an approach connecting a formally developed control system with an emulation of an environment where it is intended to operate. Our emphasis is on the visualisation of the implementation of the control system and use of existing tools for visualisations and existing models for virtual environments and devices.

The application of the approach to Train Director has both of these properties: we used a sophisticated simulation tool without the need to implement it and we can also use existing scenarios virtually without any modification. The control system was used in the form of Java application, generated from the implementation components of B-Method.

While the primarily motivation of our work is to improve formal methods teaching by putting formally developed systems into a proper “virtual” environments, it can be also used to propagate utilization of formal methods in practice, because the formally developed software will be connected with tools the practitioners or enthusiasts from given area already know and use. This way they can learn about formal methods and their benefits for safety of software systems.

Our future work in this area will aim at the development of more case studies for Train Director and B-Method. We also intend to explore possibilities of 3D engines usage. Here the Unity 3 gaming engine seems to be very promising, because of its state-of-the-art graphics, possibility of utilisation of virtual reality technologies and stereographic visualisation and a support of C# as a scripting language. It will also allow us to connect something the students like very much (games) with something they usually don’t like (well, yes, formal methods).

Acknowledgment
This work has been supported by VEGA grant project No. 1/0646/09: “Tasks solution for large graphical data processing in the environment of parallel, distributed and network computer systems”. The authors also wish to thank Ivan Chalupka and Ján Sorád for their work on the design and implementation of the BKPI Code generator and Td2JavaConn application.
References: