A Generic Flash-based Animation Engine for ProB [⋆]

Jens Bendisposto and Michael Leuschel

Heinrich-Heine Universität Düsseldorf
{bendisposto,leuschel}@cs.uni-duesseldorf.de

Abstract. Writing a formal specification for real-life, industrial problems is a difficult and error prone task, even for experts in formal methods. In the process of specifying a formal model for later refinement and implementation it is crucial to get approval and feedback from domain experts to avoid the costs of changing a specification at a late point of the development. But understanding formal models written in a specification language like B requires mathematical knowledge a domain expert might not have. In this paper we present a new tool to visualize B Machines using the ProB animator and Macromedia Flash. Our tool offers an easy way for specifiers to build a domain specific visualization that can be used by domain experts to check whether a B specification corresponds to their expectations.
Keywords: B-Method, Tool Support, Animation.

1 Motivation

In [1] A. Hunt and D. Thomas describe a shortcoming on formal methods:

> *Most formal methods capture requirements using a combination of diagrams and some supporting words. These pictures represent the designers' understanding of the requirements. However in many cases these diagrams are meaningless to the end users, so the designers have to interpret them. Therefore there is no real formal checking of the requirements by the actual user of the system - everything is based on the designers' explanations, just as in old-fashioned written requirements. We see some benefit in capturing requirements this way, but we prefer, where possible, to show the user a prototype and let him play with it.*

In previous work [2] we presented the Prolog based ProB animator and model checker for the B Method,

which addresses the problems mentioned by Hunt and Thomas. ProB can help a specifier gain confidence that the model that is being specified, refined and implemented, does meet the domain requirements. This is achieved by the animation component of ProB, that allows to check the presence of desired functionality and to inspect the behaviour of a specification.

[⋆] This research is being carried out as part of the EU funded research projects: IST 511599 RODIN (Rigorous Open Development Environment for Complex Systems).

For a domain expert with little knowledge about the mathematical notation of B, however, it might still be too difficult to understand the meaning of a specific B state; in other words, understanding a model still relies on the designers' explanations. We believe that a broad industrial acceptance of formal methods needs tools that can mediate between domain experts and formal method experts.

In this work we present a generic Flash-based animation engine as a plug-in for ProB which allows to easily develop visualizations for a given specification. Our tool supports state-based animations, using simple pictures to represent a specific state of a B specification, and transition-based animations consisting of picture sequences. To avoid the creation of many different animations the tool supports composing visualizations from individual subcomponents.

2 Flash Animation Server

The Flash animation server is a plug-in for the Eclipse version of ProB that offers support for rapid creation of domain specific visualizations. Such an animation can be seen as a prototype for the software as mentioned by Hunt and Thomas. A domain expert can get a feeling what a B operation does and he can check whether his expectations are met, without having to know the mathematical notation or relying on the specifiers' explanations.

Each state of a B machine can be represented by a set of graphical objects such as text labels or pictures. In addition it is possible to attach a movie to a state changing operation. We use Macromedia Flash which is the de facto industry standard for web animations. It is available on many platforms and, in contrast to dynamic HTML, a Flash movie looks the same in different browsers. Also it comes with many features and tools that help create professional animations.

Obviously one has to define the mapping between a state and its graphical representation. This *gluing code* could be written using the Flash built-in programming language ActionScript. Unfortunately, ActionScript is very limited and error prone, therefore we developed an animation framework which frees the user from having to use ActionScript. Our animation framework comprises a generic Flash movie on the client side, i.e., it is not necessary to create different Flash movies for different machines. The only thing one has to provide for each client is the generic movie together with the required pictures. However, it is also possible to use ActionScript, if desired, but for most applications the generic movie is sufficient.

For the server side a piece of gluing code is needed, that defines the mapping between a state (or two states plus an operation) and a graphical representation. This gluing code can be written in Java. In addition to the generic movie, we have developed a set of Java objects that can be used inside the gluing code as an abstraction for the Flash objects. These objects live inside a container named *canvas*. For example, if we want to create a new image named "image1" in the upper left corner and load the file "old.jpg", we call *can-*

vas.createNewFlashMovie("image1","old.jpg",0,0) if we want to replace that image with a new one called "new.jpg", we could use *canvas.get("image1").setUrl("new.jpg")*. The gluing code has also access to the machine's current state, the last operation executed and the machine's state before executing this operation using a Java object named *machine*. This gives the opportunity to write more sophisticated gluing code.

When any operation is being executed the Flash animation server will be notified by ProB. The animation server then calls the statechange method of the gluing code for the particular machine. The gluing code will typically read information from the machine object, do some updates on the canvas and finally call the method *canvas.commitChanges()*. Our animation server then calculates a XML message from the changed canvas and broadcasts it to all connected clients whose generic movie will display the new representation.

3 Example of an Application

We applied our generic solution to several nontrivial B specifications. The waterlock example (a detail view of an animation is shown in figure 1) is inspired by a case study from [3]; the model describes a system of waterlocks that can be operated separately. The artwork for the example has been rendered using Bryce[1]. Setting up the scene in Bryce took about two days; excluding the time to render the scene and the animations. Writing the gluing code took less than one hour. This shows that the effort to create an animation is mainly determined by the artwork. There is another example, downloadable on the tool's website, that has been developed during a workshop within three hours including writing the gluing code and creating the artwork. This example shows that our plug-in can be used for rapid visualization development.

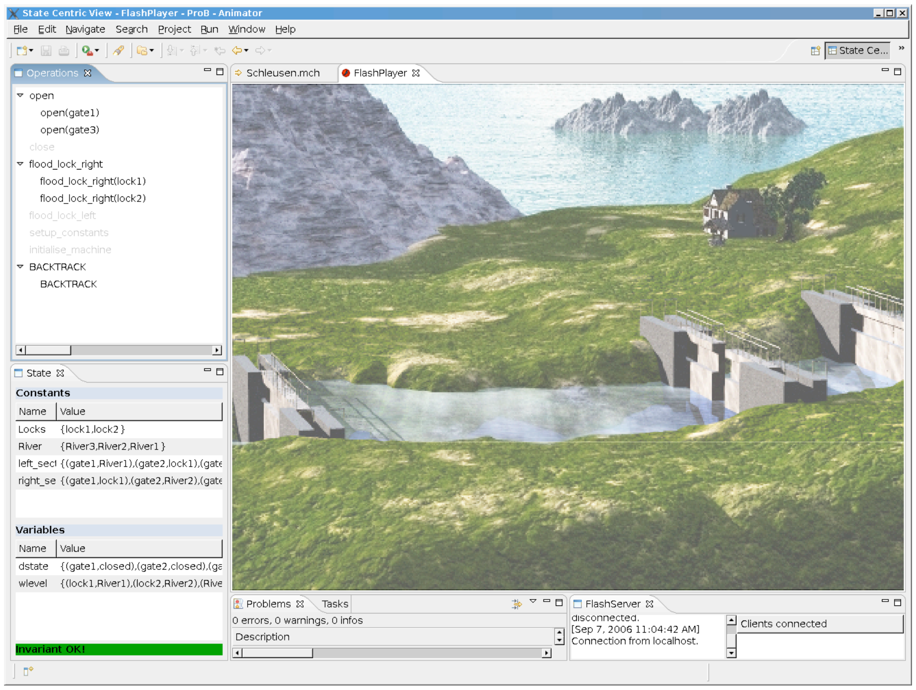
4 Related and Future work

The company ClearSy is currently developing a commercial visualization tool for B specifications, also based on Macromedia Flash technology called Brama. Brama will be available as a plug-in to the RODIN platform[2] and it also uses a B animator. In contrast to our generic solution, Brama does require programming in Flash since it provides a library for Action Script instead of an abstraction layer.

Several items can be pointed out for the most pressing future work:

1. Writing the gluing code is still a relatively cumbersome task, we are developing a graphical interface to setup the animation and an automatic code generator to generate the code.

[1] http://bryce.daz3d.com
[2] http://rodin-b-sharp.sourceforge.net/


State Centric View - FlashPlayer - ProB - Animator
File Edit Navigate Search Project Run Window Help
State Ce...
Operations
Schleusen.mch
FlashPlayer
State
Problems
Tasks
FlashServer

```
open
    open(gate1)
    open(gate3)
close
flood_lock_right
    flood_lock_right(lock1)
    flood_lock_right(lock2)
flood_lock_left
setup_constants
initialise_machine
BACKTRACK
    BACKTRACK
```

Constants

Name	Value
Locks	{lock1,lock2}
River	{River3,River2,River1}
left_sect	{(gate1,River1),(gate2,lock1),(gate
right_se	{(gate1,lock1),(gate2,River2),(gate

Variables

Name	Value
dstate	{(gate1,closed),(gate2,closed),(ga
wlevel	{(lock1,River1),(lock2,River2),(Rive

Invariant OK!

0 errors, 0 warnings, 0 infos

Description

```
disconnected.
[Sep 7, 2006 11:04:42 AM]
Connection from localhost.
```

Clients connected

Fig. 1. Visualization of a waterlock system - animation stills

2. We will extend the abstraction layer for Flash components to enable two-way-communication. Therefore we will support Flash Buttons, this will help to generate prototype user interface from B machines.

In summary, we have presented a generic animation framework to visualize B specifications using Flash technology and ProB. We hope that this new tool will help make formal methods more appealing in an industrial setting, notably by allowing domain experts to understand formal specifications. Our tool is available on `http://www.stups.uni-duesseldorf.de/ProB/eclipse`.

References

1. Andrew Hunt and David Thomas. *The Pragmatic Programmer - From Journeyman to Master*. Addison-Wesley, 2005.
2. Michael Leuschel and Michael Butler. ProB: A model checker for B. In Keijiro Araki, Stefania Gnesi, and Dino Mandrioli, editors, *FME 2003: Formal Methods*, LNCS 2805, pages 855–874. Springer-Verlag, 2003.
3. Bram De Wachter. *dSL, a Language and Environment for the Design of Distributed Industrial Controllers*. Dissertation, Université Libre de Bruxelles, December 2005.