High-Level versus Low-Level Specifications: Comparing B with Promela and ProB with Spin

Mireille Samia, Harald Wiegard, Jens Bendisposto, Michael Leuschel

Institut für Informatik, Universität Düsseldorf
Universitätsstr. 1, D-40225 Düsseldorf
{samia|bendisposto|leuschel}@cs.uni-duesseldorf.de
harald.wiegard@uni-duesseldorf.de

Abstract. During previous teaching and research experience, we have accumulated anecdotal evidence that using a high-level formalism such as B can be much more productive than using a low-level formalism such as Promela. Furthermore, quite surprisingly, it turned out that the use of a high-level model checker such as ProB was much more effective in practice than using a very efficient model checker such as Spin on the corresponding low-level model.
In this paper, we try to put this anecdotal evidence on a more firm empirical footing, by systematically comparing the development and validation of B models with the development and validation corresponding Promela models. These experiments have confirmed our previous experience, and show the merits of using a high-level specification language such as B, both in a teaching and in a research environment.

Key words: Model Checking, ProB, Spin, B-Method, Promela [1]

1 Introduction

In the past, we have accumulated a variety of anecdotal evidence about using high-level and low-level formalisms for formal modelling, both in the context of teaching and in the context of academic or industrial applications.

For example, while teaching a formal methods course at the University of Southampton[2], on multiple occasions, the students were able to develop a B model of a particular task and validate it using ProB, whereas attempts using Promela and the model checker Spin turned out to be fruitless. A similar situation arose while teaching a course at the University of Düsseldorf[3]. On the one hand, it is not surprising that the use of a high-level formalism makes the model development easier. However, one would expect that using an extremely well tuned model checker such as Spin would result in better validation.

In this paper, we present an empirical case study, which tries to provide a more empirical foundation for these anecdotal evidences:

[1] This research is being carried out as part of the DFG funded research project GEPAVAS.
[2] CM401: Formal Design of Systems, taught in 2003 and 2004 at master's level
[3] Softwaretechnik III, taught once, also at Master's level

– How much time does it take to develop a B model compared to an equivalent low-level Promela model?
– How big is the B model compared to the Promela model?
– What is the performance of the model checkers ProB and Spin on these models?

The methodology we adopted was the following:

– a student developed B and Promela models for a variety of tasks, ranging from protocols to puzzles. The goal was to cover a wide variety of scenarios,
– the student carefully measures his time usage. In order to arrive at a more just result, the student alternated between first developing the B model and first developing the Promela model.
– The student also carefully measured the model checking performance, and whether the tools were successful.

The student had a background in both B and Promela, and has had experience both with ProB and with Spin.

2 Background

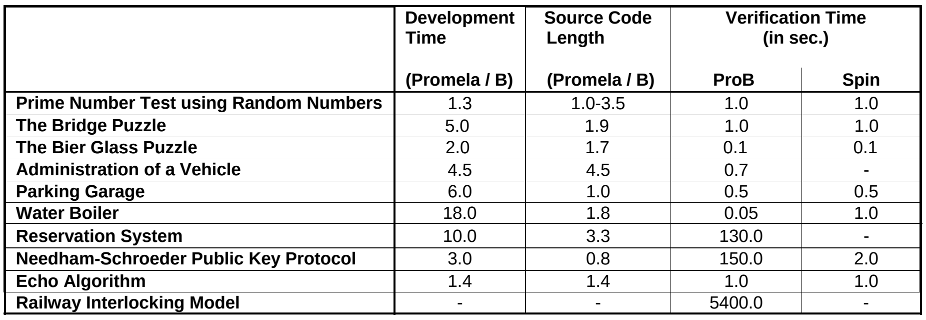
The B-Method, originally developed by J.R. Abrial [1], is based on the notion of abstract machines and refinement. In B, a system is seen as a set of states and operations, which modify the state. In order to prove consistency of the system, one has to prove, that each operation preserves the model's invariant. B uses the same notation for all the stages of the development of a system, which goes on for successive refinement steps. There are several tools to support developing models with B, one of them is the model checker ProB [5]. Promela[4] is a formalism for the verification of models [3], in particular protocols. It is based on concurrent communicating processes, that communicate via channels. The message transmission can be immediate or with delay, i.e. a channel can have a capacity of elements stored inside. The syntax and datatypes of the Promela language are influenced by the high-level language C. Promela is supported by the model checker Spin that is used and verifies the correctness of a model. It offers the possibility to verify the validity of LTL formulas, which are automatically converted into *never claims*.

3 Case Studies of Different Models

Figure 1 contains a summary of all the case studies that were conducted. The column entitled "development time" describes how much more time consuming the development of the Promela model was compared to the B model. Similarly, the column entitled "source code length" describes how much longer the Promela models were compared to the B models (counting the number of tokens). Finally,

[4] Process or Protocol Meta Language

the verification time using ProB for B and Spin for Promela are given. One can see that in one instance (Needham-Schröder protocol) Spin was much more efficient than ProB. Overall, however, the performance is comparable and despite various attempts, the student was unable to verify the Promela versions of the vehicle administration and the reservation system using Spin. The Promela version of the "railway interlocking model" could not be completed in time.



	Development Time	Source Code Length	Verification Time (in sec.)	
	(Promela / B)	(Promela / B)	ProB	Spin
Prime Number Test using Random Numbers	1.3	1.0-3.5	1.0	1.0
The Bridge Puzzle	5.0	1.9	1.0	1.0
The Bier Glass Puzzle	2.0	1.7	0.1	0.1
Administration of a Vehicle	4.5	4.5	0.7	-
Parking Garage	6.0	1.0	0.5	0.5
Water Boiler	18.0	1.8	0.05	1.0
Reservation System	10.0	3.3	130.0	-
Needham-Schroeder Public Key Protocol	3.0	0.8	150.0	2.0
Echo Algorithm	1.4	1.4	1.0	1.0
Railway Interlocking Model	-	-	5400.0	-

Fig. 1. The results of the comparison of ProB with Spin.

Below we provide more details about the individual experiments, except for the "prime number test" and the "vehicle administration" examples, which are two very simple specifications and which we do not elaborate on further.[5] [6]

3.1 The Bridge Puzzle

Description of the Model In this puzzle, several persons are supposed to cross a narrow bridge within a specified time. Each person moves at a different speed, and at most two persons are allowed to cross simultaneously. To cross safely they have to carry a torch, and the group has only one.

Source Code Size In the developed Promela model, a choice must be done for every person in a nondeterministic if-statement, whether or not he/she crosses the bridge. It is also necessary to query individually the time required to cross the bridge. The result is, that the size of the Promela model depends on the

[5] The vehicle administration problem did uncover one curious behaviour of Spin though, where the use of a variable labeled *auto* lead to inexplicable errors when model checking (but not simulation).

[6] The models can be found in the student's bachelor thesis in [8].

number of persons. That means, that the model becomes inflexible, which is based on the predefined number of people. [7]

On the other hand, B allows the definition of sets. In the model, we define a set that consists of an arbitrary number of persons. Adding another person is as simple as adding it to the set and putting a tuple into a function that links persons and the time they need to cross the bridge. The rest of the model remains the same no matter how many elements the set contains. This results in a very compact and flexible code.

The code of the Promela model, which is an example with four persons, is about twice the size of the B model's code. The more persons a model has, the greater is the gap between the size of the two models. The reason is that for each additional person, the Promela model requires more extra code than the B model.

Verification Time To solve the puzzle, we need to find a trace of operations that leads into a state, where the maximum time has not yet elapsed and all persons crossed the bridge safely. ProB can automatically search for such a condition. For a successful search for the defined goal in Figure 2, ProB needs significantly less than one second. In Figure 3, History lists the steps of the successful search for the bridge puzzle.

```
GOAL == time =< maximum_time & ran(person_pos)={dest}
```

Fig. 2. The GOAL-Definition in ProB.

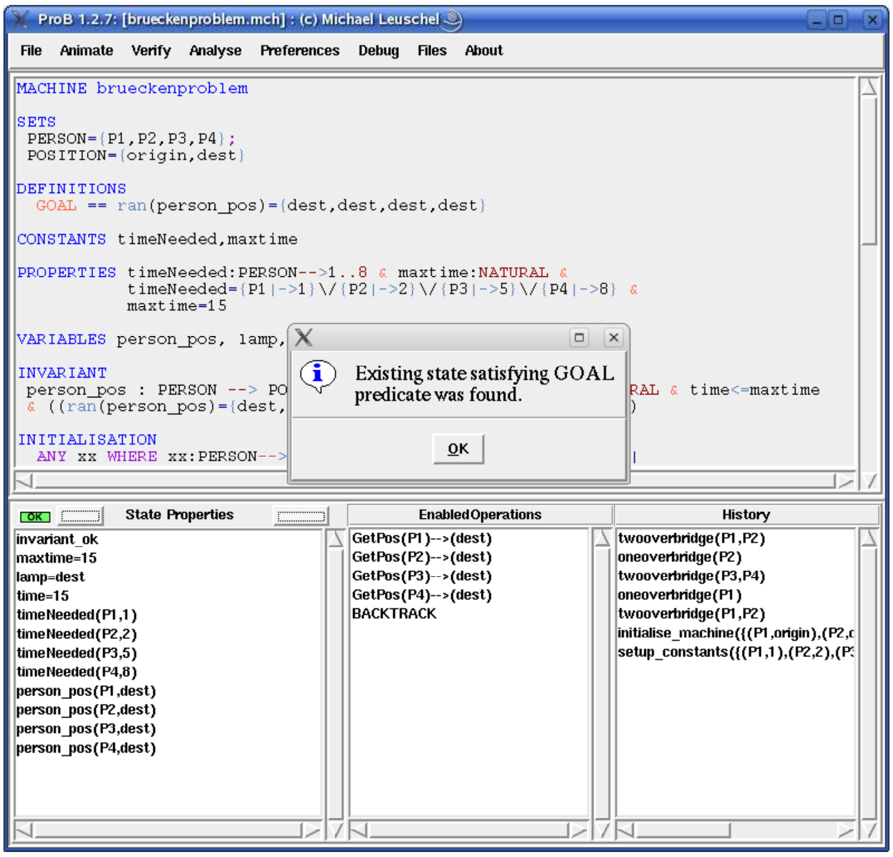
Finding the solution with Spin also takes about one second. In order to find it, the user has to encode the goal into a neverclaim that is less intuitive than plain predicate logic.

Development Time The development time of the Promela model was five times longer compared to the development time of the B model.

3.2 The Beer Glass Puzzle

Description of the Model The model treats — similar to the brigde puzzle — the search for a solution for a given task. In this model, an amount of beer, such as 400 ml, should be measured using a 500 ml large glass and a 300 ml small

[7] An experience Promela user may be able to reduce the size of the model by modelling each person as an individual process. This, however, also makes the model development more tricky.


ProB 1.2.7: [brueckenproblem.mch] : (c) Michael Leuschel
File Animate Verify Analyse Preferences Debug Files About
OK
State Properties
EnabledOperations
History

```
MACHINE brueckenproblem

SETS
 PERSON={P1,P2,P3,P4};
 POSITION={origin,dest}

DEFINITIONS
  GOAL == ran(person_pos)={dest,dest,dest,dest}

CONSTANTS timeNeeded,maxtime

PROPERTIES timeNeeded:PERSON-->1..8 & maxtime:NATURAL &
           timeNeeded={P1|->1}\/{P2|->2}\/{P3|->5}\/{P4|->8} &
           maxtime=15

VARIABLES person_pos, lamp,

INVARIANT
 person_pos : PERSON --> PO                                  RAL & time<=maxtime
 & ((ran(person_pos)={dest,                                  )

INITIALISATION
  ANY xx WHERE xx:PERSON-->                                  |
```

Existing state satisfying GOAL predicate was found.

OK

```
invariant_ok
maxtime=15
lamp=dest
time=15
timeNeeded(P1,1)
timeNeeded(P2,2)
timeNeeded(P3,5)
timeNeeded(P4,8)
person_pos(P1,dest)
person_pos(P2,dest)
person_pos(P3,dest)
person_pos(P4,dest)
```

```
GetPos(P1)-->(dest)
GetPos(P2)-->(dest)
GetPos(P3)-->(dest)
GetPos(P4)-->(dest)
BACKTRACK
```

```
twooverbridge(P1,P2)
oneoverbridge(P2)
twooverbridge(P3,P4)
oneoverbridge(P1)
twooverbridge(P1,P2)
initialise_machine({(P1,origin),(P2,c
setup_constants({(P1,1),(P2,2),(P3
```

Fig. 3. ProB found a solution for the bridge puzzle.

glass. Both glasses can be only entirely filled or fully emptied. Moreover, the content of a glass can be decanted in the other glass till this glass is filled. Only a maximum total amount of liquid can be used, which is defined by the constant *limit*.

Source Code Length Similar to the bridge puzzle, the advantage of the high-level mathematical B notation is also shown here. In the Promela model, many `if`-statements are needed to select all possible cases. For instance, filling the big glass with the content of the small one comes with a case distinction. It is necessary to handle the two possible cases "content of the small glass fits completely in the large glass" and "content of the small glass does not fit completely in the large glass", e.g., if the big glass already contains beer (cf. Figure 4).

However, in B, the minimum function can be used resulting in a much more compact code (cf. Figure 5). The Promela code of the beer glass puzzle is 1.7 times longer than the B Code.

```
if
:: atomic{((big+small)<=maxBig)
          -> big=big+small;   small=0
         }
:: atomic{else
          -> small=small-(maxBig-big);
             big=maxBig
         }
fi
```

Fig. 4. In Promela, the Else case should be treated separately.

```
big  :=min({big+small,maxBig}) ||
small:=small-(min({big+small,maxBig})-big)
```

Fig. 5. The B Code is compact due to the use of the minimum function.

Verification Time The search for a solution with ProB is also easier than with Spin. Using predicate logic, it is easy to specify a goal, which ProB should find. We used the predicate $(big = solution \lor small = solution) \land used \leq limit$ as the goal. The goal is achieved, when a state is found, where the desired quantity of liquid is either in the small or in the large glass, and the selected limit is exceeded. ProB is able to find a solution in less than one second. In Spin, no goal state can be defined. Instead, we need to use a NeverClaim. It can be automatically generated from the LTL Formula $G \neg foundSolution$. The NeverClaim is a representation of the negation of the LTL formula. If a solution exists, the LTL-formula is violated. Spin recognizes this case. Using a “guide simulation”, it can display the executed steps which lead to the violation of the formula. Spin can also find the solution in less than a second.

Development Time Although the model was first created in B and then in Promela, the creation of the Promela model has taken twice the time compared to the B model.

3.3 An Example of a Parking Garage

Description of the Model The example of a parking garage is a simple model: as long as the parking garage is not full, a car can enter, and then can go out.

Source Code Length In Promela and also in B, the model's code is quite compact. In both cases, the variable *usedSpace* is incremented or is decremented by the actual number of the cars inside the parking garage.

Whenever the Promela model uses several independent processes (or a process with multiple instances), the query of the actual value of the variables and their changes must be included in an *atomic*. Otherwise, the variable can accept an illicit value, whenever, for instance, between querying whether there is a car park in the parking garage and the execution of `usedSpace++`, another process increases the variable to its maximum allowed value. With a proper NeverClaims, SPIN finds the source of errors - with one exception.

If the type of *usedSpace* is byte and the value of the variables was previously zero, then `usedSpace--` sets the variable's value to 255. Initially, the student's model used byte, meaning that the Promela model did not correspond to the desired specification and that certain errors were not detected by SPIN (e.g., overflowing as increasing 255 by 1 gives 0 or underflowing as decreasing 0 gives 255). This problem does not occur in B and PROB. If a variable, defined as $NATURAL$ and having the value zero, is decreased by one, then the variable value is -1. Then, an invariant's violation occurs.

Verification Time Taking into account these restrictions, the models with SPIN and PROB can be verified in a second.

Development Time The time spent on the B model is about fifteen minutes, and on the Promela models 90 minutes.

3.4 The Water Boiler

Description of the Model We specified a simple water boiler. It may only be turned on, if the lid is closed and water is filled. The lid may only be opened, whenever the water boiler is turned off. When the lid is open, water can be filled or distributed.

Source Code Length For each action of the water boiler model, the B model offers an operation, which can be executable depending on the preconditions. To ensure that no illicit condition occurs, the model has several invariants. The Promela source code is around the factor 1.8 longer than the B code.

Verification Time In the modelling phase, due to the possibility of interactive simulations, the student found it slightly easier to be convinced of the desired functionality of the B model. For the verification, PROB requires 0.05 seconds.

The Promela-model consists of an active process "User", which can execute non-deterministically one of the possible actions. To ensure that the corresponding preconditions remain unchanged before the end of their respective action, every action was wrapped in an `atomic` statement. This, however, meant that the Promela model was no longer verifiable using SPIN. The reason is that, the actions “fill water” and “distribute water” both have a `do`-loop, whose termination is not guaranteed. It is possible to replace the `do`-loop by an `if`-construct with an entry for every potential value of *ii*. This requires more work in programming, but allows both a proper verification as well as a realistic simulation. The verification of the model with SPIN is then possible within about a second.

Development Time Because of this difficulty, the modelling in Promela took about 18 hours. The creation of the model with PROB was possible within one hour.

3.5 The Echo Algorithm

Description of the Model The Echo algorithm [2] is designed to find the shortest paths in a network topology. A start node sends an explore-message to all neighbors. Each node is marked with red, when it receives an explore-message for the first time. Moreover, it memorizes the nodes, from which it received the message, as a shortest path to the initialization node. It also sends, in turn, explore-messages to its other neighbors. Whenever the node receives either an explore-message or an echo-message from all its neighbors, to which it sent one of such messages, the node will be marked green and sends an echo-message to the nodes, from which it had first received an explore-message. When all nodes are marked green, the cycle is finished.

In the B model, every type of message type has one corresponding operation. The operations are active, as soon as a node sends the appropriate message. The execution of the operation reflects the receipt of the message by the node’s recipient. By the non-deterministic order, the selection of the active operations is assured that any various long message runtime in the channel of the model is taken into account. It is possible that many messages are simultaneously on the channel. The edges are depicted as functions between the nodes. With the proper invariants, it is easy to verify if a protocol ensures that all nodes are marked green, when no message is in the channel. Furthermore, all shortest paths must be known as soon as all nodes are marked green.

In addition, with PROB, it is also possible to verify the accuracy of an LTL-formula. This example presents the LTL-formula `"GF{ran(state)={GREEN}}"`, i.e. the verification, to check whether there always exists a final state, in which all nodes are marked green. The verification of the model with PROB is possible within one second.

The Promela model offers a simple modelling of the message channels. As in the B model, many messages can pass simultaneously in the channel. The modelling of the message communications between the nodes is possible. For the Echo Algorithm, the simulation of SPIN provides a graphics of the protocol’s execution (cf. Figure 6).

A clear disadvantage of the Promela model is the cumbersome modelling of the edges between the nodes. Each node has an array, in which its neighbors’ nodes’ are defined. This is much more cumbersome and less compact than the presentation of the edges as a function between the nodes as in the B model.

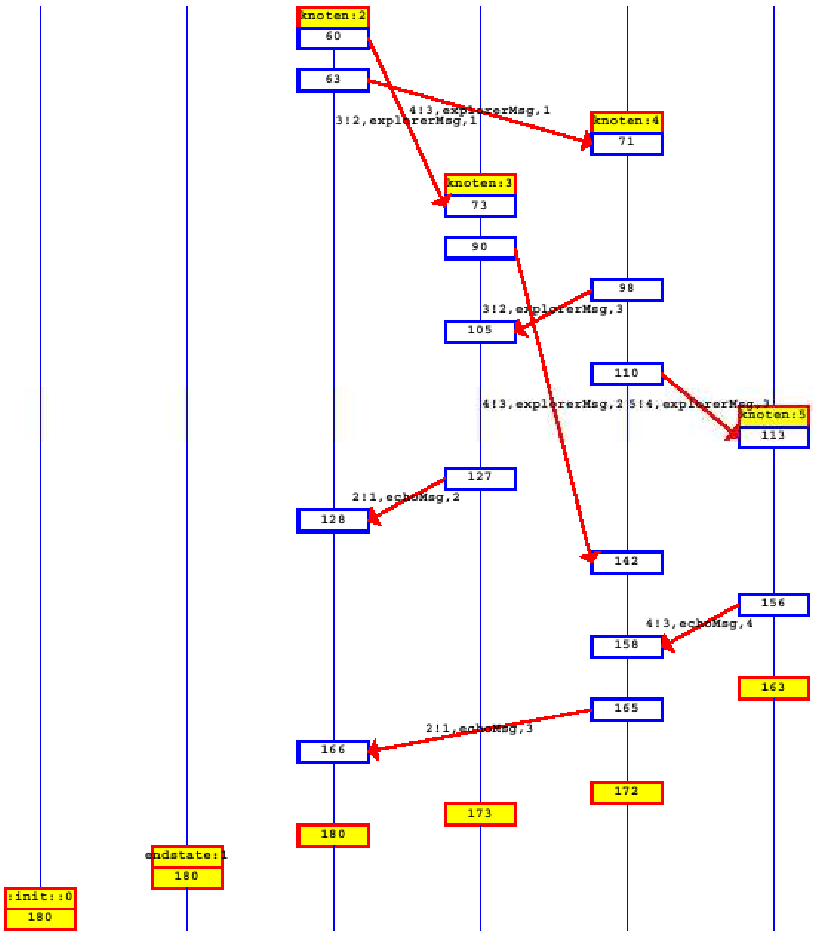


Fig. 6. The Graphical Output of the Spin-Simulation.

Despite the simple simulation of the message channels, the Promela-model is by a factor 1.4 longer than that of the B model. The same is true for the time required to create the model. With Spin, the verification is possible within about a second.

3.6 A Reservation System

Description of the Model This model specifies a reservation system for trains, which makes it possible to book, in different wagons, different kinds of seats, such as first class, second class, and so on.

The Promela model consists of a self-defined type of an array called *train*, which contains different wagons. During the initialization, Spin lacks the opportunity to assign the places a category. Hence, it requires much more time to create the model.

During the creation of the Promela model, it is often useful to execute a simulation with SPIN in order to verify if the result is as expected. Due to the use of arrays, in particular nested arrays, the simulation with SPIN is extremely slow. In this model, the SPIN simulation requires, for every step of the test, about 0.9 seconds. This is a very considerable testing time, since just one seat reservation thus requires about five minutes.

For each of the reservation server, customers and vendors, there exists a process in the Promela model. The communication between these processes is done through the message channels. This is especially important for the communication between sellers and the reservation system. Another process exists, called Watchdog process, which checks if every place is marked as reserved (cf. Figure 7). The statement `assert(false)` does not guarantee the detection of an error, whenever a condition is not satisfied.

Due to high memory assignment and the considerable search needed, no available verification method of SPIN could verify the reservation system model.

```
active[1] proctype watchdog()
{
  byte ii=0;
  do
  :: (ii<(MAXRESERVATIONS-1)) -> ii++
  :: (ii>=(MAXRESERVATIONS-1)) -> ii=0
  :: (reservations[ii].dest>0)
     -> if
        :: (train[reservations[ii].train]
             .waggon[reservations[ii].waggon]
             .compartment[reservations[ii].comp]
             .place[reservations[ii].place].status==belegt)
        :: else -> assert(false)
        fi
  od;
}
```

Fig. 7. A Watchdog process for the reservation system.

On the other hand, in the B model, each place is assigned a category. No nested arrays and loop traversals are needed. PROB verifies the model within 130 seconds. The verification time depends on the size of the set *seat*. The more seats exist, the longer the verification takes.

The Promela model is longer by a factor of 3.3. While the B model was developed and verified with ProB within a day, the Promela model took ten days to develop.

3.7 Needham-Schroeder Public Key Protocol

Description of the Model The Needham-Schroeder public key protocol is an authentication protocol for creating a secure connection over a public network [7]. The model consists of a network with the two normal users called Alice and Bob, an attacker named Eve and the keyserver. The first version of this protocol, developed in 1978, contains an error which was found in 1995 [6]. The error is that Eve is allowed to communicate with Bob, where Bob thinks to communicate with Alice. This error is found by ProB as well as Spin. In the corrected version of the protocol [6], ProB and Spin find no error.

In the Promela model, a separate process is created for every network user and the keyserver. The communication between the processes is easily described, since Promela supports the modelling of message channels.

The following LTL formula was used to check whether the protocol can successfully run to completion:
`<>((okAlice && okBob && aliceTalksToBob && bobTalksToAlice))`. In order to validate the protocol, the LTL formula in Figure 8 has been checked.

```
[]((okAlice && okBob) -> (aliceTalksToBob <-> bobTalksToAlice))
```

Fig. 8. LTL-Formula violated by Needham-Schroeder-Protocol

A verification of the model with Spin was impossible, because high memory was needed. A simpler version of the model, which reduces the messages sent by Eve, was created. Spin verifies if the previous mentioned bug exists, within about two seconds.

In the B model, the messages sent over the network are stored in global variables. The actions executed by Alice, Bob and Eve are created as single operations. The B model differs significantly from the Promela process-oriented model. Instead of LTL formulas, the B model used invariants (ASSERT_LTL would have been also possible). Because the verification of the first model takes several hours, similar to the Promela model, the new version of the B model was simplified. ProB finds the well-known bugs of the Needham-Schroeder public key protocol within about 150 seconds.

The Needham-Schroeder protocol is the only model, where the B source code is longer than the Promela source code (factor 1.3). However, the creation of the Promela model is about three times longer than the B model.

3.8 A Railway Interlocking System

Description of the Model In this model, signals (such as slow or green) and switches for the block sections are possible. No two trains can be on the same block section. No two crossed block sections can be entered at the same time.

The B model, the signals and the switches are assigned to functions. Variables include, for instance, the current status of the signals and the position of the train.

The model fulfills the requirements of a railway system, since it is possible to enter and move trains in the system. Moreover, invariants are formulated as conditions to meet the correctness of the system. Within approximately 1.5 hours, ProB can verify a system with an initial example of two trains, seven block sections, two switches, a crossing rail and eight signals. This complex model was created within three weeks. The model is flexible, since within a few minutes further block sections or trains can be added by changing the respective initialization of the function variable.

In the Promela model, functions and sets cannot be used. The model consists of individual processes for signals, switches, trains and the "Supervisor" in the interlocking system. The communication is via message channels.

The dependencies between signals, switches and block sections are stored in arrays, which makes the Spin simulation extremely slow. For the execution of just one single command, Spin needs about a second. Hence, the simulation of a small part of the model requires several minutes.

Despite the longer work, it was impossible to create a fully functional Promela interlocking system. The Promela model contains more code than the B model.

For the railway interlocking system, the B method is most appropriate than the Promela model.

4 Conclusion and Future Work

In summary, we have studied the elaboration of B-models for ProB and Promela models for Spin on ten different problems. With one exception (the Needham-Schroeder protocol), all B-models are more compact than the corresponding Promela models. On average, the Promela models were longer by a factor of 1.85 (counting the number of symbols) and took roughly a factor of 2-3 more time to develop than the B models. No model took less time in Promela, while some models took up to 18 times more time in Promela. One Promela model could even not be fully completed within the timeframe of this study.

More surprisingly, the study also found that ProB and Spin were comparable in practical model checking performance as measured by the user. This is despite the fact that ProB works on a much higher-level input language analysed via a Prolog interpreter, and thus being much slower (when looking purely at the number of individual states that can be stored and processed) than Spin which is compiling to C.

This study confirms our own anecdotal evidence while teaching B and Promela, as well as developing our own B and Promela models within research projects.

We believe that this study shows the merits of using a high-level specification language such as B, both in a teaching and in a research setting. Note that, based on the findings in this study, we have tried to provide some explanations for the model checking performance in this study in [4].

As far as teaching formal modelling, we believe that it is both easier for a student to write models in a high-level specification language and easier to make use of tool support to validate those models. We also believe that it maybe appropriate to give students a first introduction to model checking using a high-level language such as B, also because the high-level tools are more forgiving and more helpful in locating errors.

References

1. Jean-Raymond Abrial. *The B Book: Assigning Programs to Meanings.* Cambridge University Press, 1996.
2. Ernest J.H. Chang. Echo algorithms: Depth parallel operations on general graphs. *IEEE Transactions on Software Engineering*, 8(4), 1982.
3. Gerard J. Holzmann. *The Spin model checker: primer and reference manual.* Addison-Wesley, Boston, 2004.
4. Michael Leuschel. The high road to formal validation. In *ABZ*, pages 4–23, 2008.
5. Michael Leuschel and Michael J. Butler. ProB: an automated analysis toolset for the B method. *STTT*, 10(2):185–203, 2008.
6. Gavin Lowe. An attack on the Needham-Schroeder public key authentication protocol. *Information Processing Letters*, 56(3):131–136, November 1995.
7. Roger Needham and Michael Schroeder. Using encryption for authentication in large networks of computers. 21(12):993–999, Dezember 1978.
8. Harald Wiegard. Vergleich des model checkers ProB mit Spin. 2008. Bachelor's Thesis, Institut für Informatik, Universität Düsseldorf.