ToLERo: TorX-Tested Lego Robots

Arjan Snippe
University of Twente
a.snippe@student.utwente.nl

ABSTRACT

Model-based software testing is done by generating test-cases from a model that represents the workings of a System Under Test (SUT). Then the tool runs those tests and compares the results with the actual SUT. One of the issues here is the synchronization of the communications between the tool and the SUT. In the testing tool TorX this is the task of the Adaptor. For this paper we've developed an Adaptor for a simple Lego Mindstorms robot. The goal behind this is to obtain a generic Adaptor from this specific Adaptor.

Keywords

Model-based Testing, Adaptor, TorX, Reactive Systems

1. INTRODUCTION

Testing is an integral part of the Software Engineering process, finding out whether or not a system works according to the specifications. There are several ways to check if a system meets the requirements, black-box and white-box testing will be discussed. With black-box testing we check to see if the system works according to the specification by looking at its inputs and outputs. Here we are not concerned with the inner workings and structure. In white-box testing on the other hand, we do take the inner structure into account.

One way to perform a black-box test is to use model-based testing. With this method we describe the behavior of the SUT using a model. Then we apply a tool such as TorX to generate test cases bases upon that model. Using TorX we verify if the SUT responds according to the expectations specified in the model. The testing tool keeps a record of the current state the SUT is in and reports any errors it finds.

In this paper we want to look at one specific part of TorX: the Adaptor. The task of the Adaptor is to provide the communications between the SUT and TorX; this is done by correctly translating the messages into something the SUT can understand, and then passing them on to the SUT. The main problem with the Adaptor at the moment is that each time one needs an Adaptor, one needs to write a new one. This can be a time-consuming effort.

13th Twente Student Conference on IT, June 21st, 2010, Enschede, The Netherlands.

We propose to first write a specific Adaptor for a Lego robot we also want to build. The goal for this Adaptor is not only to correctly facilitate the testing of our Lego robot, but also to serve as a basis for a more generic Adaptor.

Then we will separate the parts of the specific Adaptor that are actually specific for this Adaptor and the parts that can be reused for other Lego robots. The end product will then be an Adaptor that can easily be modified to support different Lego robots.

2. TORX

The model based testing tool TorX was developed to test systems with respect to model based specifications. TorX is based on the *ioco* test-theory [1]. This means that a SUT is correct if and only if the outputs it gives are correct according to the model that specifies the SUT. An important output that should not be forgotten here is quiescence, which is the absence of any output.

TorX has been used to test several different systems in both academic and industrial environments [6]. The goal in developing TorX was to have a flexible and open system. As specified in Figure 1, TorX has 4 main components: the Explorer and the Primer who translate and prepare the model for testing, the Adaptor that provides the communications between the SUT and the rest of TorX, and the Driver which is the main component that makes most of the testing decisions.[6]

Figure 1: The basic architecture of TorX

2.1 The Adaptor

The tasks the Adaptor has to fulfill can be split into two basic tasks. First of all the Adaptor needs to translate all the messages used in the model into stimuli for the SUT, it also needs to translate the outputs generated by the SUT into observations the driver can use to verify the behavior of the SUT.

The second task the Adaptor needs to fulfill is to send the translated messages form the driver to the SUT, or vice versa. It is important that all the messages are sent as fast as possible, and that none of the messages are lost during the process, otherwise the Adaptor might generate errors that do not actually exist, violating the "no false positives" principle.

3. PROBLEM STATEMENT

Although the job of the Adaptor is important, there is no systematic method of obtaining one. This means that for each SUT one would need to develop a new Adaptor to be able to communicate with it. The goal of this paper is to find a more generic Adaptor and thereby making it easier to support communication with a new SUT in the future.

3.1 Research Questions

The main problem we want to tackle is the lack of a generic Adaptor which one can easily use for a SUT. We will begin developing a specific Adaptor for a Lego robot. Therefore the first question we want to answer is:

> How is it possible to write a specific Adaptor for a Lego robot?

We will use this as a proof of concept. From there we can try to obtain a more generic Adaptor for Lego Mindstorms, this leads us to the more significant question:

> Which parts of the specific Adaptor can be generalized for all Lego Robots?

Such an Adaptor would of course need to be able to handle every input and output from any Lego Mindstorms motor and sensor. It needs to be robust and adaptable to be able to handle any Lego Robot.

Because of the fact that Lego Mindstorms isn't the only desired application for this Adaptor, it would be useful to have a more formal procedure for obtaining Adaptors for any combination of SUT. Therefore we also want to answer the following question:

> To what extent is the Lego Adaptor applicable to other types of reactive systems other than Lego systems?

4. REQUIREMENTS

As described above, the Adaptor has two main tasks. The Adaptor we want to build needs to handle those jobs correctly.

The first task for the Adaptor is to translate the messages from the driver into stimuli for the robot, and to translate the output from the robot into observations. Because we are going to test a Lego robot, this means we need to translate the stimuli into actions performed by one or more Lego Mindstorms motors. The observations will be a bit more difficult. The sensors provided by Lego Mindstorms do not always generate the expected results. Because we are performing a black-box test we will not be using the motors or sensors used by the SUT itself. Therefore we will be using a second NXT microcomputer.

The second task for the Adaptor is to correctly pass all the messages from and to the SUT. This is important because the Driver needs to know if the sequence in which the messages arrive at the Adaptor are as good as possible. The problem here occurs when both the stimulus and the observation are sent at about the same time. In this case TorX might think the system is in a different state then the actual state. This might lead to errors that don't exist.

The synchronization problem occurs because the communication between the Adaptor and the Driver needs to be synchronized, while the communication between the Adaptor and the SUT can't be synchronized.

Another problem that might occur is a buffer-overflow. Sometimes the SUT sends out a burst of observations at the same time, first of all, none of these observations may get lost at any point, all of these must be sent to the Driver as soon as possible.

The generic Adaptor will also need to fulfill these requirements. But it will also need to be as generic as possible, so it can be used for other SUTs as well.

5. EXISTING SOLUTIONS

TorX currently has a working Adaptor; this Adaptor works by passing the messages given in the model to the standard-in and standard-out of the computer or by using a TCP socket. The current Adaptor is capable of correctly buffering the messages from the system to be processed later by the driver or the SUT. Most of this code is still useful and we propose to use this Adaptor as a basis for the new generic Adaptor.

Next to TorX, several other model-based testing tools were developed [2], amongst these at least TGV [4] and Gotcha [3] have been successfully applied in industrial environments [5]. Unfortunately most papers concerning these tools don't elaborate on the communication problems we want to research; instead they choose to elaborate on the mathematics behind the tools [4] or the differences between the tools [2]. The paper on TGV does discuss on the communication, and does describe the interface in a similar manner as TorX [4]; however it doesn't elaborate on any communication problems.

6. IMPLEMENTATION

First of all, we needed an SUT we could test. For our SUT, we decided to use a ball-sorter. We will provide a number of black and/or white marbles one at a time to a cross shaped sorter, once the marble lands in the sorter, a sensor will decide if t is either a black or a white marble. According to the input provided by the sensor, the sorter will either turn clockwise for a white marble, or counter-clockwise for a black marble. After all marbles have been sorted, one would end up with lines of the same color. Figure 2 gives a basic idea of the sorter itself

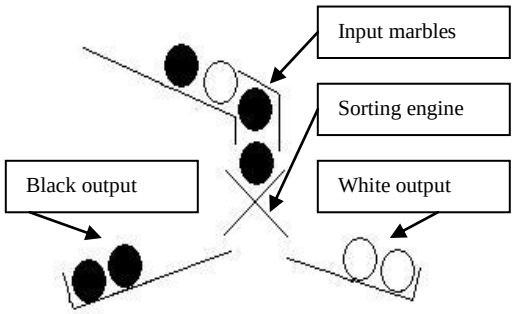

Figure 2: The basic structure for the robot to be used as the SUT

After we built the sorter, we added a few extra motors and sensors to the robot, these motors and sensors would be used by a second NXT microcomputer. These additions will allow TorX to provide either a black or a white marble to the sorter, and also later on see if the marble did get sorted correctly. Here it doesn't matter what

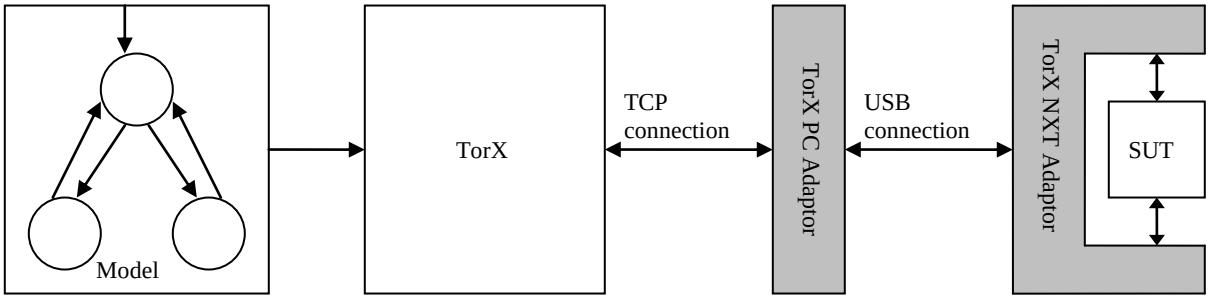

Figure 3: Overview of the final product and its relation to other systems

the color is of the sorted marble because a black marble in the white output bin will always generate an error.

Figure 3 provides the overview of the Adaptor and its relation to TorX. On the right side of the figure we see the SUT and built around it, the TorX NXT Adaptor. These are the two parts we've discussed above.

Also important for testing a system is the model that describes the behavior of the system. The model for our sorter provided in figure 4. In the starting state it can decide to either feed a white marble to the sorter or a black marble, after that it will wait for the response from the system.

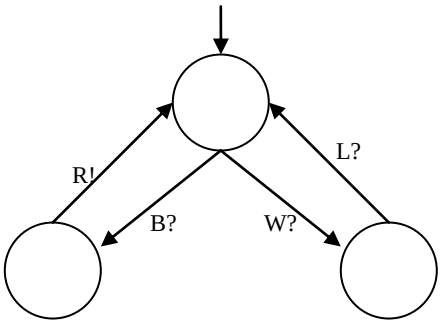

Figure 4: Model of the Sorter

The last two pieces needed to test if the SUT works are the two pieces of the Adaptor. We need to divide the system into two separate pieces because there is no direct communication possible between TorX and a Lego Mindstorms microcomputer.

The first piece will be the TorX PC Adaptor; its job is to provide the communications between TorX and the NXT microcomputer. Communications between TorX and the PC Adaptor will be done using a TCP Socket. Communications between the PC Adaptor and the NXT Microcomputer go through a USB connection.

The second part is the TorX NXT Adaptor; this part will translate the messages from TorX into actions, and will report any observations back to TorX.

All the different parts within the NXT Adaptor and the PC Adaptor have also been displayed in Figure 5.

6.1 TorX PC Adaptor

The TorX PC Adaptor is subdivided into two parts, the Stimuli Handler and the Observation handler.

First of all, the PC Adaptor will set up both the connections with the NXT micro computer and with TorX. After that it will start two different threads, one for the observation handler, and one for the stimuli handler. These set-up tasks are completely generic, and can be re-used for other Adaptors.

The job of the Stimuli Handler is to translate the messages from TorX into integers and send them to the NXT microcomputer. The job of the Observation Handler is the same, just vice versa. For other Adaptors, these parts will need new translation rules, other than that, these parts can be reused as well.

When modifying this part of the Adaptor, please note that the end-of-line characters after the return messages are a necessity, otherwise TorX will not recognize these messages as the same as the ones in the model.

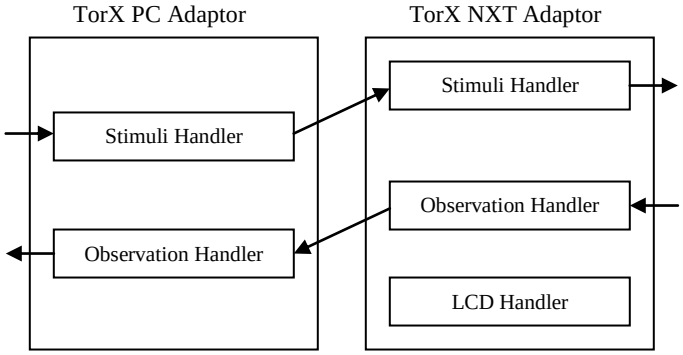

Figure 5: Message flow between the Adaptor parts

6.2 TorX NXT Adaptor

The NXT Adaptor is subdivided into three different parts; similar to the PC Adaptor we have both a Stimuli Handler and an Observation Hander. There is however a new part added to this system: The LCD handler.

Similar to the PC Adaptor, the NXT Adaptor begins with setting up the connection between the PC Adaptor and the NXT Adaptor and then setting up the three threads. This part of the Adaptor is generic as well.

The Stimuli Handler then proceeds to translate any messages from TorX into actions performed by the Adaptor. The translation part of the Stimuli Handler will need to be modified for other Adaptors. The actual functions for controlling the Lego motors will need to be rewritten for other Adaptors.

The Observation Handler translates any input from the two light sensors connected to it into observations for TorX. The

functionality for this part will need to be rewritten as well for any other Adaptor.

The LCD handler displays some useful numbers on the screen, this mostly for de-bugging. It can probably be left out in other Adaptors, should a Software Engineer decide he doesn't need it. Because the de-bugging data is very specific to this system, this part is also not very re-useable for other Adaptors.

7. GENERIC ADAPTOR

Due to time constraints we haven't been able to set up any kind of interface for a generic Adaptor, however the structure used for the generic Adaptor, as described above, can be applied to many more Lego Mindstorms Robots. All one needs to change are the messages the PC Adaptor expects from both sides and correctly translate them into new messages for the next application.

The same goes for the NXT Adaptor. One will need to modify the expected messages it receives and change them into their own desired actions. One will also need to set up their own sensors to check the outputs.

One of the things we wanted to do is write a specific class for handling the light sensors in the way this system is using it. In that case the re-use of this specific part could very well be implemented with a single function call in the Adaptor.

8. CONCLUSIONS

To answer our first research question we've built a specific Adaptor for a specific Lego Robot, the biggest issue with writing this specific Adaptor was to setting up the communications between TorX and the PC Adaptor and between the PC Adaptor and the NXT Adaptor.

As for writing a generic for Lego Mindstorms robots, even though we didn't have the time to write one ourselves, a lot of parts written for the specific Adaptor can be reused for the any other Lego Mindstorms Adaptor.

As for our last question on a completely generic Adaptor; the current Adaptor TorX is using is well capable of communicating with different kinds of systems. Other than the ability to communicate using TCP Sockets, it can also communicate more directly using the standard-in and standard-out from the command-line. The main issue then remains the fact that the messages need to match the messages TorX is expecting perfectly.

9. REFERENCES

[1] Belinfante, A. JTorX: a Tool for On-line Model-Driven Test Derivation and Excecution, In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), (2010) pp 266-270

[2] Belinfante, A.F.E. and Frantzen, L. and Schallhart, C. (2005) *Tools for Test Case Generation.* In: Model-Based Testing of Reactive Systems: Advanced Lectures. Lecture Notes in Computer Science 3472. Springer Verlag, pp. 391-438

[3] Benjamin, M. Geist, D. Hartman, G. Mas, Smeets, R. and Wolfsthal, Y. (1999) *A study in coverage-driven test generation.* In: Proceedings of the 36th ACM/IEEE Conference on Design Automation (DAC'99). pp. 970-975

[4] Jard, C. and Jeron, T. (2004) *TGV: theory, principles and algorithms, a tool for the automatic synthesis of conformance test cases for non-deterministic reactive systems*. In: Software Tools for Technology Transfer (STTT). pp. 297-315

[5] Jeron, T (2009) *Symbolic Model-based Test Selection* In: Electronic Notes in Theoretical Computer Science 240 (2009). pp. 167-184

[6] Tretmans, G.J. and Brinksma, H. (2003) *TorX: Automated Model-Based Testing.* In: First European Conference on Model-Driven Software Engineering, December 11-12, 2003, Nuremberg, Germany. pp. 31-43