Source Code Test by Simulation for Mobile Autonomous Systems

Thomas-Peter Czudnochowski, Birgit Koch, Jochen Wittmann
czudnochowski@gmx.de, [koch|wittmann]@informatik.uni-hamburg.de
Universität Hamburg, Fachbereich Informatik, Arbeitsbereich Technische Informatiksysteme
Vogt-Kölln-Straße 30, 22527 Hamburg

KEYWORDS
code test, simulation, mobile autonomous systems, mindstorms, LejoSim

ABSTRACT

The students of computer science have the possibility in their studies, to gain virtual experience in developing of embedded systems in JAVA. They can go along with all the phases, from the analysis of the program up to the use of the system. Here the practice admits that the extensive testing of the implementations and the abolition of errors constraints the quality of the results. This contribution introduces a method of resolution called LejoSim, which enables the students to test their software in a comfortable and economic way and allows them to focus on the actual problem/exercise.

LejoSim makes an API for robot programming available to the students to allow them to embed the robot program automatically into a simulation environment and to test and debug it within the development environment.

1 Test and Debug Methods in Programming of LEGO Mindstorms robots

The students develop the robot software using an iterative and incremental development process. The implementation and test activities alternate. To test software in an iteration loop, the software gets transmitted to the robot, the robot gets started and the behaviour of the robot gets observed. The system presents itself as a **Black-Box-System** to the developer while being tested when running in a physical environment. The software can only be evaluated according to the visually observable behaviour of the robot.

During the tests, no information can be gained about the actual internal state of the system or the program flow during the program execution. When the observed behaviour of the robot differs from the expected behaviour, there is at first only the static code-analysis available to the developer. In case, the developer is not able to find the source of the misconduct by means of the static code-analysis, he is forced to refine the test with code-preparations and repeat it until the source of the misconduct is classified as found.

The code preparations serve the containment of the potential source of defect. They represent Output-Commands which allow to bring out the value of a program variable to a small display of the robot or to send an acoustic signal for the passing of a program branch. To pinpoint the source of the defect in the program code with these appliances, the student must plan each test interaction very exactly with the appropriate code-preparations.

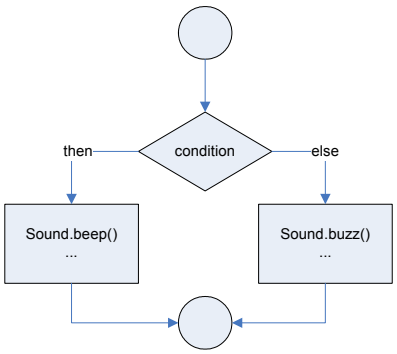

Figure 1: An Acoustic Outgo for the Proceeding of a Program Path

Each test passage is associated with a time-consuming transmission of the program to the robot. The previous preparations must be removed again for every transmission. Provided that the developer was successful in

finding a program error, all the previous code preparations must be removed completely. To assure that the code preparations do not bear any side-effects, a final test must be carried out.

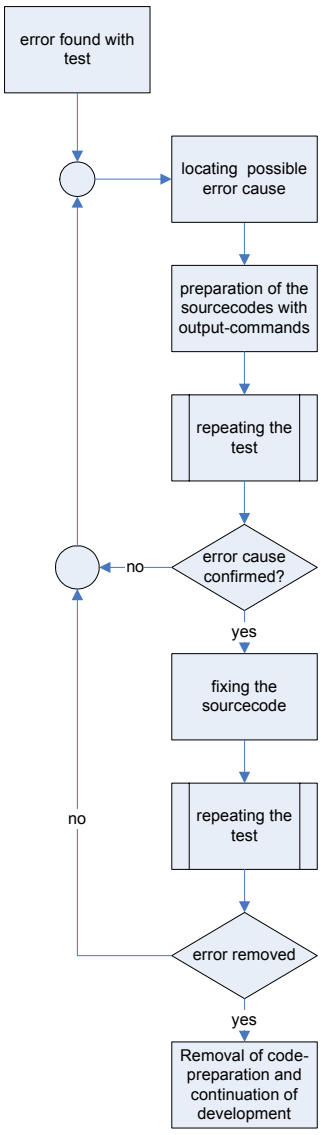

Figure 2: Debug Process with Code Preparations

2 Layer Model of the Robot Application

During the apprenticeship, the LEGO Mindstorms robots are driven with the virtual JAVA-machine leJOS. LeJOS makes a class library available, which the robot program can use to address the hardware components of the robot. These are above all the classes Motor.java for the control of the motors and Sensor.java, which allows to readout the actual values of the attached sensors. Further classes are LCD.java for the display of the integers to the display of the robot and Sound.java for the outgo of acoustic signals. These classes use the class ROM.java to accomplish the native calls.

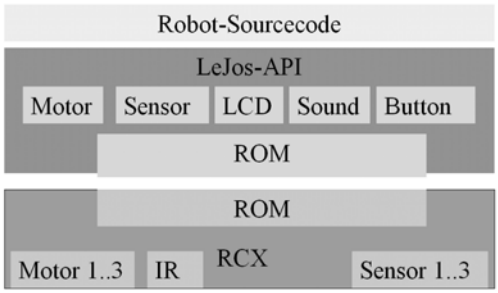

Figure 3: Layer Model of the leJOS Robot Application

3 Demands on the Method of Resolution

Section 1 points up the fussiness of the test and debug process of the LEGO Mindstorms robot programming. The disadvantages of the method are in general:

- The system presents itself in runtime as a Black-Box-System.
- Prevalent debug methods are not available. To pinpoint the sources of defect, improvisation with the code preparations is necessary. The tested code is thus not identical with the code which is to be applied when running the robot.
- The test- and debug operations are very time-consuming.

The demands on the method of resolution arise from inversion of these disadvantages:

- The system should be tested and debugged as a White-Box-System.
- Prevalent debug methods should be available. There should be no need for code preparations.
- The test- and debug operations should not be time-consuming.

In addition, the method of resolution should not be a new burden for the students. Thus there are also the following demands:

- A smooth integration of the method into the development process and development environment should be possible.
- For the students, the use of the method should be simple and intuitive with no need for additional introduction.
- There should be no need for a special regard for the method during the implementation. Also, there should not be differentiation between the tested code and the code which is to be run with the robot.

4 The Method of Resolution LejoSim

The method of resolution LejoSim is an extension of the developing environment **Eclipse** and was developed according to the demands specified in section 2. It supplies a Plug-In for the development environment, which allows starting the robot programs without any modifications as a desktop application. Thereby, it escapes the time-consuming transmissions of the source code to the robots during the test- and debug-processes while all the test- and debug-tools of the development environment still can be used without any constraints.

For the accomplishment of the function tests, LejoSim supplies a GUI, which makes it possible to observe the state of the robot hardware components while running the program, such as engines or sensors. To test the reaction of the robot program concerning the sensor values, you can put in values for the sensor.

For the accomplishment of the **acceptance test** and the **system tests**, LejoSim supplies a simple **simulator**, which allows testing and evaluation of the interaction between the robot and the physical environment. For this purpose, there is a simple robot editor for the adaptation of the simulation to the robot design. It is possible to set up the sensor connection, sensor type, engine connection and kinetics of the robot for the simulation. The situation, which should be tested in a modelled environment, can be simply made by Drag&Drop, while placing the robot and other objects such as a ball with a mouse into the environment model.

5 Realisation of LejoSim

The reasons for the disadvantages of the functioning as described in section 1 are to be found substantially in the leJOS´s API dependence on the hardware. The leJOS API is placed as a middle layer between the robot program and the robot hardware.

LejoSim removes the hardware dependence of the robot program by supplying a LejoSim API containing alternative classes to all original leJOS classes which contain ROM-calls. The LejoSim classes have an identical signature to the original leJOS classes but they don't contain any ROM-calls. With these LejoSim classes, the hardware-depending classes of the middle layer leJOS are replaced. This means that the hardware can be dropped. Now it is possible to start the robot program within the development environment as a desktop-application and to test it and to use all debug-features of the IDE.

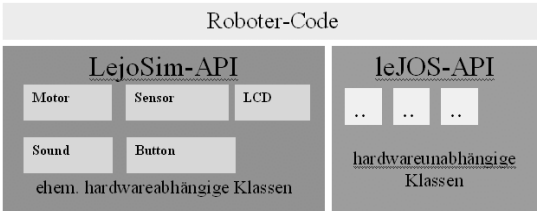

Figure 4: Layer Model of a LejoSim leJOS Application

The next main feature is to make functional tests possible to the programmer. The programmer is enabled to put in some certain sensor values while running the program and to watch the robots reaction by observing the motor states. For this purpose LejoSim offers a GUI containing input panels for the sensor values and view panels for displaying the hardware model states. The LejoSim architecture is similar to the common MVC-pattern. The classes of the LejoSim API are used as models and corresponding SIM-classes of LejoSim are used as related controller-classes.

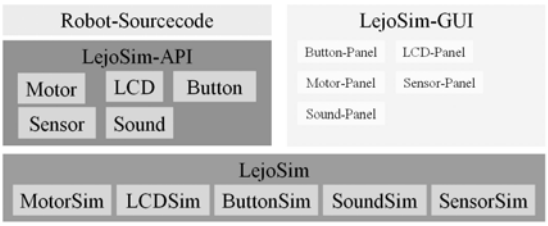

Figure 5: Integration of a GUI for Function Tests

Furthermore, the LejoSim layer acts as a **communication layer** for the robot application as the main-thread and the concurrent simulation-thread of the simulator.

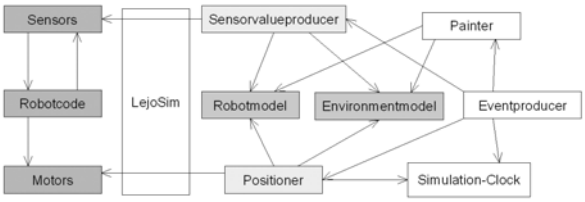

Figure 6: Communication between LejoSim Simulator and Robot Application

The simulator is able to read the motor states by using the controller-classes of the LejoSim layer and also, to use them to calculate the resulting model states of the simulation models. The new sensor values of the simulated models can now be calculated and sent to the LejoSim API sensor models by using the LejoSim layer. Finally the robot application can use these values by reading them out of the API's sensor models.

6 Conclusion

With LejoSim there is a tool, students can use in their projects during their studies for testing and debugging the robot programs in an uncomplicated and economical way. They can not only get first programming experiences with using JAVA and project experiences in developing complex embedded systems by using LEGO robotic kits. They also learn how to use simulation systems as a supporting tool for the development of complex embedded systems. On the one hand, they learn about the economical and technical advantage of using simulation systems for testing purposes in a complex project. On the other hand, they might feel the restraints of the usability of testing with a simulation system. Due to the need of running the developed system with real robots in a physical environment at the end of the project, they can see that the quality of function tests and system tests will strongly depend on the quality of the simulation's model design.

7 References

- *Czudnochowski, T.*: „*Dynamisches Testen von Quellcode im Bereich mobiler autonomer Systeme*“, Diplomarbeit, Universität Hamburg, 2005
- *Ferrari, G.*: „Programming LEGO Mindstorms with JAVA“, Syngress Rockland MA, 2002
- *Koch, B.*: „*Einsatz von Robotikbaukästen in der universitären Informatikausbildung am Fallbeispiel 'Hamburger Robocup: Mobile autonome Roboter spielen Fussball'“,* Diplomarbeit, Universität Hamburg, 2004.