Fujaba hits the Wall(-e)

Fujaba hits the Wall(-e)

Pieter Van Gorp*
Eindhoven University of Technology,
School of Industrial Engineering
De Lismortel 2
5600MB, Eindhoven, The Netherlands
p.m.e.v.gorp@tue.nl

Ruben Jubeh,
Bernhard Grusie
Kassel University,
Software Engineering Group
Wilhelmshöher Allee 73
34121 Kassel, Germany
ruben@cs.uni-kassel.de,
bernhard.grusie@gmx.de

Anne Keller†
University of Antwerp,
Dept. of Mathematics and Computer Science
Middelheimlaan 1
2020, Antwerpen, Belgium
anne.keller@ua.ac.be

ABSTRACT

With the ever increasing pervasiveness of software in every day's life, it is quite easy to explain children the importance of software development. Especially when using gadgets such as LEGO robots, one can fascinate young pupils. It is much harder though to find a fair link to the actual educational and research programs from a particular university without blowing the audience away with details of a particular Java framework. This paper illustrates how one can use Fujaba to involve children from 8 to 18 years old in realistic requirements elicitation workshops. The children implicitly get in touch with the object-oriented paradigm by playing in the real world the communication between objects in a robot's computer. Fujaba's visual object browser provides a convincing means to illustrate that the game adequately represents the robot's internals.

1. INTRODUCTION

Fujaba has been used for educational robotics programming since 2002 [2, 3]. The Fujaba NXT framework provides the technical infrastructure for interacting with the *NXT* version of the LEGO Mindstorms hardware [4]. So far, there has been more focus on the fine-tuning and debugging of this framework than on the elaborate use thereof. This paper describes the current state of the NXT framework and provides an overview of the quickly growing set of educational projects in which it has been used so far.

This work is based on a collaboration between the developers of the framework (from the University of Kassel) and external "users" thereof (from the University of Antwerp and the University of Eindhoven). From this collaboration, one can conclude that:

- the framework supports smooth migrations to new implementation layers (e.g., to move to a non-polling sensor implementation),
- even when both teams use completely different robot hardware designs, one can share valuable software artifacts.

This paper also specifically points to the limitations of the NXT framework that was presented on the Fujaba Days in 2008 and indicates which problems were overcome and which tough issues remain unsolved so far. As a major novelty, this paper shows how to combine story driven modeling (SDM [1, 11]) with Statechart modeling for optimizing model readability. Obviously, such readability is an essential property of models that are used in an educational context. The remainder of this text is structured as follows: Section 2 briefly introduces the Fujaba NXT framework that is used throughout this paper. Section 3 presents an overview of the educational projects that use this framework. Section 4 describes which lessons have been learned from developing the NXT examples for these projects. Finally, Sections 5 and 6 close this paper by summarizing, concluding and pointing to future work.

2. THE FUJABA NXT FRAMEWORK

This section briefly revisits those concepts from [4] that are essential for understanding the core of this paper. More specifically, a first subsection describes the framework's architecture while a second subsequent explains tools for executing a Fujaba NXT application.

2.1 Architecture

The Fujaba NXT framework adds the ability to model complete applications for LEGO Mindstorms in Fujaba using model driven development with a graphical programming environment. It relies on the LeJOS [10] framework, a Java firmware and API for LEGO Mindstorms. The basic approach is to remote control a NXT with a program running on a host PC. The main advantage of remote controlling the NXT is that it can be easily debugged with a Standard Java Debugger. Debugging a remote program running directly on the NXT is not possible yet. The Fujaba NXT framework consists of four layers, where each layer should communicate only with its adjacent layers:

*Thanks to Ronny Mans and Jana Samalikova for supporting the TU/e workshop mentioned in this paper. Thanks to Nena Van Deun for supporting the tryout session for that workshop.

†This work was partially funded by (i) the Interuniversity Attraction Poles Programme - Belgian State – Belgian Science Policy, project *MoVES*, and (II) the Research Foundation – Flanders (FWO) project G.0422.05. Additionally, the authors wish to thank Nick Baetens and Glenn van Loon for both their conceptual as well as practical work on the copy robot.

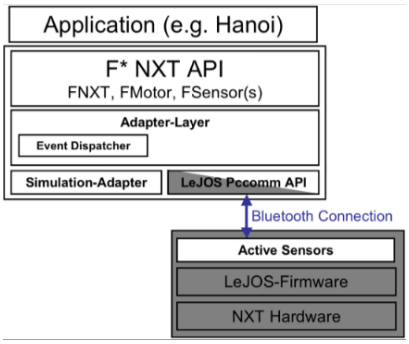

Figure 1: Overview of the architecture

1. The Application layer: A Fujaba model for the concrete application and robot, most likely containing concrete classes of the problem domain. This can be a Robot (sub)class composing objects of the next layer (e.g. certain sensor types) to reflect the used hardware, and environment and problem describing classes.

2. Library API Layer: this is also a Fujaba model, containing pure object-oriented wrapper classes equivalent to the LeJOS core NXT classes like (F)NXT, (F)Motor and (F)Sensor. Furthermore, it consists of some abstractions and basic algorithms. As being modeled itself in Fujaba, these classes conform to the standard Fujaba access style, so they can be easily used in the application layer. Furthermore, Application and Library layer objects can easily be visualized and manipulated using eDOBS: one can interactively invoke methods to control the robot manually.

3. The Adapter Layer: Hand-coded classes which can be exchanged at runtime, either adapting the wrappers to the LeJOS Pccomm API or just simulating the robot hardware to run an application without actual hardware. The simulation adapters still don't adapt to (3D) Mindstorms simulation environments like [9] yet, but allow to run applications as JUnit test without the actual hardware. The LeJOS adapters also deal with the bluetooth connection to the NXT and synchronize and serialize all NXT commands and requests. A benefit of that approach is that upgrading to newer LeJOS versions just requires this layer to be updated, upper layers remain untouched. This is also the layer where the ActiveSensors plug in, as described in Section 4.2.

4. The Foundation layer - all LeJOS classes, the LeJOS firmware with a TinyJava virtual machine (VM) on the NXT control unit (often called the NXT brick) and since 2009 also the ActiveSensors component running on the NXT.

Layers 2-4 form the whole framework. The Library already contains some Layer-1 classes like Fork and MobileRobot abstraction, but usually these need to be extended.

2.2 Running and Debugging NXT Applications

Using the eclipse Dynamic Object Browsing System (eDOBS), one can (i) inspect the java heap space and watch a visualization of objects, their attributation and links and (ii) interactively change attributes (or links) and invoke methods. This is very useful in the NXT context, as one can control the robot interactively by invoking motor methods manually within the context of the running program. Figure 2 amongst other connected objects, the fork object and its attributes.

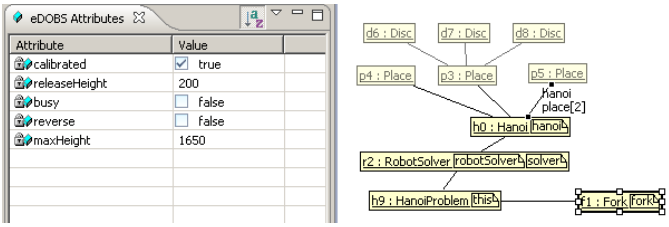

Figure 2: Using eDOBS to control NXT instances.

3. EDUCATIONAL PROJECTS

This section presents the educational projects in which the NXT library was applied. Each project was developed with a different target audience in mind. Therefore, for each project the strengths, weaknesses, opportunities and inherent limitations are evaluated.

3.1 Towers of Hanoi: Fujaba Robotics Classic

3.1.1 Case Description

The Towers of Hanoi example was first elaborated by Diethelm et al. [2, 3]. The 2002 solution relies on very limited LEGO hardware of the so-called *RCX* (Robotic Command Explorer) type. That RCX hardware has (among other problems) rather unreliable infrared PC communication. Since 2008, the NXT library relies on more modern hardware (e.g., bluetooth instead of infrared communication). In [4], the second author of this paper describes how this overcomes much of the problems from 2002. A remaining weak point of the 2008 solution is that it makes the robot drive blind just with the help of a single touch sensor. Especially the 90-degree turn to drive to another place is unreliable. In the 2009 version, the robot uses straight black lines leading from the disc places to find each place. A single light sensor is used to detect that the robot crosses the black line while driving between the places. After turning almost 90 degrees, a black/white-edge line follower algorithm to follow the line to the disc place. Figure 3 shows the robot with a disc picked up, following the black line on the ground.

Mind the so-called "continuous track" wheels, whose design is primarily known from caterpillar tanks. As discussed in [4], this wheel design provides a precise steering mechanism and the capability of turning on the spot, and is capable of carrying high loads. The hardware design has been improved in 2009. For example, the robot uses now a very sophisticated fork construction operated by a nylon wire, which allows precise fork movements and doesn't fail accidentally in comparison to the 2008 chain actuated fork.

3.1.2 Models

While the actual application model (called HanoiRobot) is quite small (5 story diagrams, 15 story patterns), its story patterns contain quite a number of collaboration statements

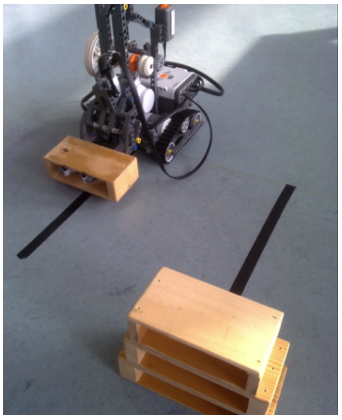

Figure 3: UniKassel Forklift robot in action

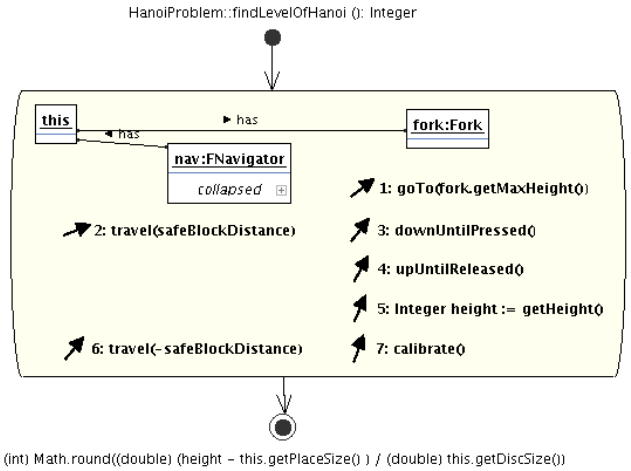

Figure 4: Story diagram for measuring the size of a Hanoi tower.

and rarely any graph transformations. As an example of the consequences on model readability, Figure 4 shows the method responsible for measuring the initial disc tower height. In general, the exclusive use of collaboration statements is a rough indicator of bad modeling style in Fujaba. However, it is still under evaluation whether this is the case here and how the modeling style could be improved. On the one hand, the use of collaboration statements seems reasonable since one is modeling a simple control command sequence. On the other hand, one could emphasize more the timing effect of statement 3 and 4 by translating this monolithic story pattern in a statechart.

3.1.3 Analysis

The main strengths of the Towers of Hanoi example are that (i) it has been implemented several times already (which facilitates a comparison between the different approaches), and (ii) it involves challenging tasks in the real world (picking up a block, delivering it at a variable height, moving from tower to tower, ...). An important weakness is that it requires an understanding of recursion. It turns out that very little children come to the intended solution spontaneously. As a potential opportunity, one could indicate that there is also a non-recursive algorithm for solving the Hanoi challenge. Since that algorithm is very complex too, the authors do not consider it a viable alternative. Instead, the Hanoi example will primarily be used as an internal test-case: from that perspective it presents the opportunity to use new sensor types (compass sensor for accurate 90-degree turns, ultrasonic, ...). Reusable functionality will obviously be moved to the appliation-independent library layer.

3.2 Forklift in Factory

This example has been designed in the context of a promotion event at the University of Antwerp in 2008. During one week, the university welcomed secondary school classes in order to motivate them for higher education in exact sciences. The local Software Engineering groups working with Fujaba aimed to seize this opportunity for making students between the age of 14 and 18 excited about model-driven software development.

3.2.1 Case Description

The goal of the role playing game is to derive behavioral models for an autonomous forklift robot. This robot needs to pick up all goods from a bill of materials and deliver them to an output line. More specifically, the robot needs to pick up four wheels, an engine, and a bodywork kit to enable the assembly of a car. In practice, students should mimic the different pieces of a robot (its navigator, its wheels, its sensors, ...) and walk through a classroom. Tables are arranged in rows that mimic the different shelves in the factory. At the end of each row, they can pick up an item. The robot is initialized next to the first row and should return there for item delivery.

At first, the instructor explains the factory layout: the path between all rows and inside each row is assumed to be marked with colored lines. Moreover, each crossing point is marked with an additional dot on the working floor. The instructor clarifies that this approach is needed to make the role playing game independent of the concrete number of shelves, and the distance between them.

Secondly, the instructor explains two example robots (the continuous-track design, as shown on Figure 3 and a trike design, as shown on the bottom left of Figure 5. Students should agree that although these two robot designs require a different driving controller, the two robots should in principle be able to move through the factory and pick up goods using the same algorithm.

In a third step, the instructor opens a question and answer session to investigate different strategies for following a line marker on a factory floor. Students get in touch with cost considerations (one sensor designs versus two sensor designs) and design freedom (there are multiple valid solutions in any engineering project).

In a fourth step, students derive what different participants are relevant for solving the role playing game. After some discussion, one agrees that it is reasonable to use a Robot instance, a Navigator instance, two Motor instances for driving, one motor instance for affecting the forklift, and to sensors (for following the line and for detecting crossings).

Figure 5: Pictures from the Science Week event in Antwerp, 2008.

Fifth, the instructor starts the NXT robot and accesses the corresponding objects via eDOBS. Students are made aware of the graph structure: objects are represented as nodes and can only talk to those objects that are directly connected via a link. Additionally, the instructor asks what responsibilities belong to which object and validates whether this corresponds with the information from eDOBS. At this point, the instructor executes some methods to show that the visual representation is not just a pretty picture but is actively connected to the real-world robot.

Sixth, students are divided in groups. Each group relates to one object and all group members get a T-shirt with the same color. Each group has one active member who will perform the actual role playing whereas the others can help him/her to figure out what to do. All active members are connected with cords to form a structure that corresponds with the eDOBS diagram shown before. During the role playing game, students are made aware once more that they cannot send messages to an arbitrary object. By referring back to the methods on the eDOBS diagram, students iteratively learn to be precise in naming the correct method names and using proper arguments.

Finally, the instructor shows the model-driven implementation of some methods.

3.2.2 Models

Figure 6 models how one can keep track of the robot's location: it is assumed that the robot has an internal representation of the factory layout. This enables the robot to maintain a pointer to the shelf where it is currently located at. Figure 7 shows in more detail how item location and stock levels are represented. The diagram also shows the concept of a "JobDescriptor", which represents both a bill of material and a delivery location: the ordered Descriptor2Product association indicates what items need to be picked up whereas the Descriptor2Target association indicates where the items need to be dropped.

Finally, Figure 8 shows an eDOBS instance graph for these data structures. In fact, this instance graph has been used

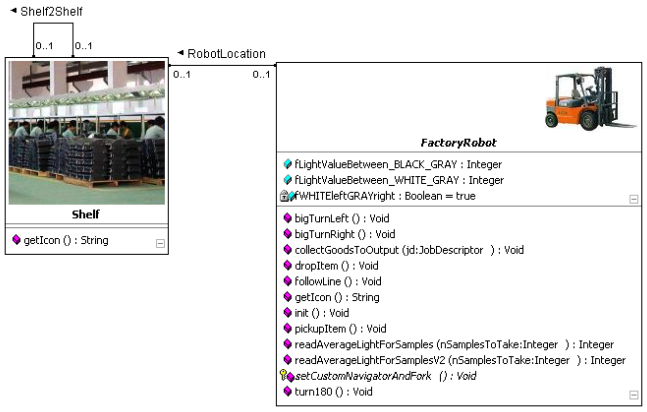

Figure 6: Class diagram for factory representation.

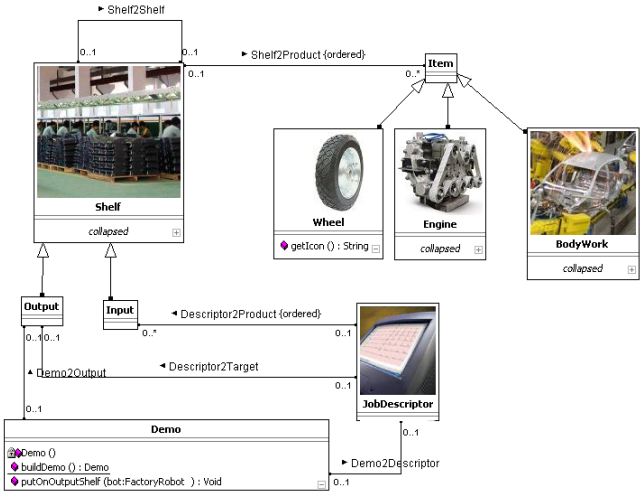

Figure 7: Class diagram item representation.

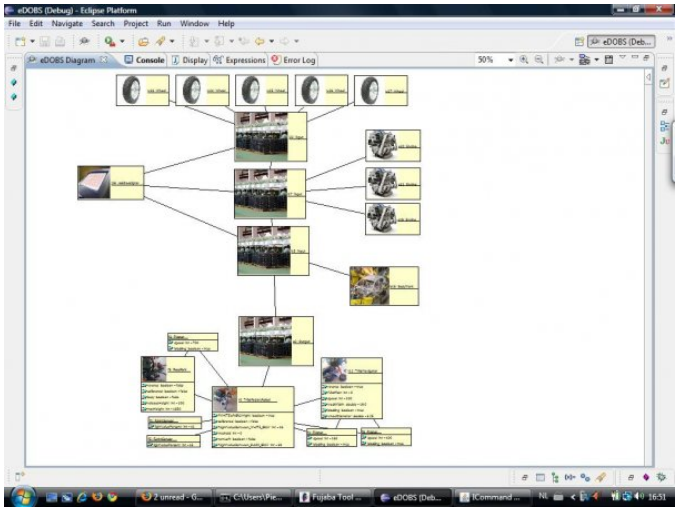

Figure 8: eDOBS snapshot: example object graph.

for the role playing in the 2008 science week event. The graph represents a situation where five wheels, three engines and one piece of bodywork are in stock. The robot is located on the shelf to which items need to be delivered. The job descriptor is displayed on the left of the diagram. It points to all goods that are needed to assemble a car. When handing over this job descriptor to the robot, it should visit shelves iteratively and inspect whether the item on the top of the bill of material is located on a particular shelf. If so, the robot should turn and collect the item. Otherwise, it should move on to the next shelf. If there is no such shelf, the robot can (for example) signal that an item is missing on the shelfs.

3.2.3 Analysis

The authors observe the following strengths in this example:

- most students are aware that several factory workfloors are full of lines and therefore their game has some industrial relevance,
- the game design is flexible and can be adapted easily for half hour sessions as well as sessions beyond one hour,
- the game can easily be completed, which leaves the student with a satisfied impression,
- the complication of having multiple robot designs provides a nice basis for discussing inheritance and delegation mechanisms in a manner that is surprizingly understandable,
- the application domain can be visualized nicely in eDOBS by using some pictures of shelves and other factory elements,
- the example involves quite some graph rewriting, which makes it a nice application for Story Driven Modeling (since quite some Story Patterns have create and destroy markers). It turns out that the students easily understand the semantics of such operations.
- the story diagram implementation turned out to be sufficiently understandable and at least much more recognizable than the underlying program code,
- the T-shirt based approach turns out to be succesful for scaling up role playing games to groups of about 20 students. Students also appreciate that they are backed up by peers when they don't understand what to do.

However, the game has some weaknesses:

- For children under the age of 16, this game may be boring, since they may not yet be interested in industrial relevance,
- In 2008, the Fujaba solution relied on story diagrams exclusively. These story diagrams were still too technical,
- The trike design from 2008 was built in a rather ad-hoc manner. It turned out (too late) that in the real world, the specific trike design makes precise steering very hard. More specifically, turning 90 degrees on the spot requires several iterations of steering forward and backward. Although this is interesting in general, it is irrelevant for this specific application.

Given the new – *Statechart-based* – modeling approach, there is an obvious opportunity for making the underlying Fujaba models even more understandable. Finally, the authors are aware that the T-shirt based approach also has inherent limitations: first of all, we expect a practical upper bound of about 25 in the number of participating students; secondly, the instructor (or an assistant) needs to keep friendly control of those students that are not actively role-playing. Especially when trying to correct minor role playing errors, these students may start loose focus.

3.3 A Copy Robot

The copy robot example was developed by students of the University of Antwerp in the course of a student project following the science week held at the University of Antwerp. The goal was to produce a educational demonstration introducing computer science principles to secondary school students in their last two years of school and consequently motivating them for computer science studies in general. The aim was to create a reusable teaching unit for occasions such as the prior science week. The result was a 1,5 hour interactive lecture that was held in a secondary school in Essen, Belgium. About 20 students attending the school's computer science class, between 16 and 17 years old, took part in the lecture.

3.3.1 Case Description

The robot developed for this demonstration copies an image consisting of non-crossing, connected lines by scanning it with a light sensor and drawing it on a sheet of paper with a attached pen. The choice of a copy robot was motivated by the idea that copying and scanning are well known mechanisms that can be understood instantly. Additionally, the copy robot offers a physically compact setup (i.e., does not

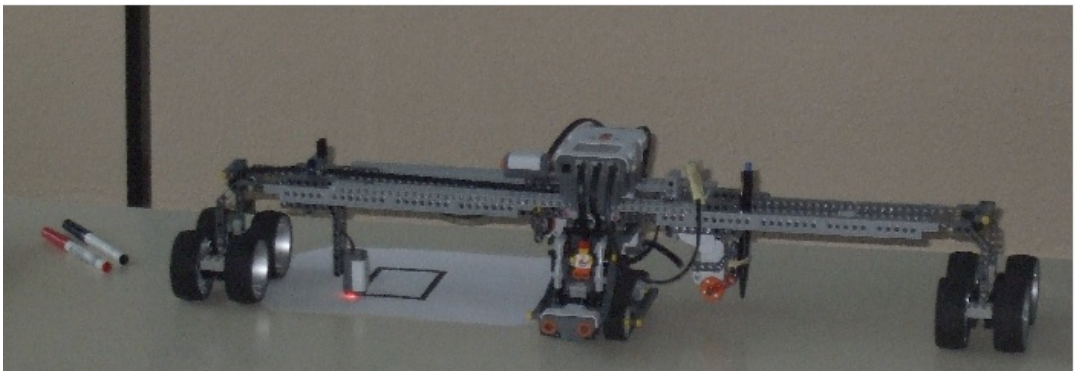

Figure 9: Copy Robot in action.

drive around) that is well suited for the intended interactive lecture setup (see Figure 9).

The lecture was structured around the evident technological issues of the robot's hardware and software setup. The lecture consisted of two parts, one introducing the hardware and a second introducing software aspects of the robot's design. The first part of the lecture familiarized the students with the given robot by introducing hardware of the robot. This introduction ranged from issues of the motor's movement, to the different attached sensors and their different functionalities. The students were shown the concrete hardware setup and introduced to some problems faced with the given LEGO robot setup, e.g., the dependence of the light sensor on the light source and light intensity. This section concluded with showing the class diagram of the now fully explained copy robot.

In the second part the students were introduced to the algorithms used, mainly the scan-line algorithm. The algorithm was explained step by step guided by slides and based on interaction of the students. The presenters asked questions such as, what kind of lines can be scanned, what are the limitations of the scan-line algorithm and how efficient are the proposed in between solutions. The slides contained story diagrams as well as pseudo code supporting the explanation of parts of the algorithms. Also in second part, the students were given two exercises to solve on their own. The first exercise was to model a simple loop that gets values until the value is above a specific threshold. The second exercise was an extension of exercise one that resembled a simple scan-line algorithm. To solve the exercises students were modeling the intended behavior in flow charts that they were already familiar with from their class. When comparing the results the flow charts were presented next to the story diagrams.

3.3.2 Models

Supporting the lecture, class diagrams, object diagrams and story diagrams were used. While a simple object diagram showed the current sensor and motor setup, the class diagram summarized the hardware introduction of part one. In part two of the lecture, introducing the algorithms, mainly snippets of story diagrams were used to explain the involved algorithms. Since the secondary school students were accustomed to flow charts from their class, with only a little explanation and comparing flow charts and story diagrams, it was reasonably easy for them to understand the story diagrams. However, it is interesting to note that the university students setting up the lecture were using story diagrams mainly to give an overview of the algorithm. When going into detail however, they were using pseudo code explaining method details and loop behavior. Also when programming the robot's behavior they were using this mixed approach of modeling and coding.

3.3.3 Analysis

The authors observe the following strengths in this example:

- The demonstration fitted well in the curriculum of the class since the high school students were studying behavioral modeling with flow charts at the moment. Positive feedback was given about the connection of the classes subject matter to a well-understandable example. The students claimed to now better understand the subject matter.
- The copy robot itself and its example setup are a compact, well portable teaching unit. In the current setup it can be used independently of group size and location (except a projector no further equipment is needed).
- By covering both hardware and software aspects the lecture is quite demanding and reaches a coverage of different aspects. We think that this is a challenge appreciated by secondary school students of this age.

There are some weaknesses however:

- The interactive lecture was a welcome change in the student's daily routine. However, we found in evaluation of the lecture that students with no prior interest in computer science also did not show any increased interest in the lecture.
- Although students were encouraged to contribute to the discussion, there were no activities that strictly required interaction. To solve the exercises the students were left on their own which resulted in unsolved exercises where time did not suffice or students had problems.

- Finally, a better usage of the available 1,5 hour time slot would improve the lecture. In the 1,5 hours of the lecture both hardware and software issues were covered in high detail. While this is interesting for students wishing to get a broad view of all possible issues involved, it harmed the interaction in the class and was especially visible in the number of unsolved second exercises.

3.4 Wall•E Rescues Eve

Less than one year after the science week at the University of Antwerp, one of the instructors was asked to organize at Eindhoven University of Technology (TU/e) a science-related workshop for families with kids. It seemed promising to use this event as an opportunity to tackle some of the weaknesses of the Factory example described above. The so-called "open day" targeted kids between the age of 8 and 12 and a session should take about half an hour at most.

The following considerations have driven the design of a new Fujaba NXT example: (i) all existing examples were too complicated for the children younger than 12 years of age, (ii) the 2008 forklift design and its multi-step turning approach was a promising basis, (iii) several participants to the 2008 Factory Robot workshop had made enthusiastic references to the Wall•E movie.

3.4.1 Case Description

Wall•E is a 2008 computer-animated film produced by Pixar Animation Studios that follows the story of a robot named Wall•E [5]. This robot is designed to clean up a waste-covered Earth far in the future. The robot has transformed an old container in a cosy home where he keeps valuable waste items organized in shelves. He eventually falls in love with another robot named EVE, and follows her into outer space on an adventure that changes the destiny of both his kind and humanity [8]. The movie includes scenes about a sandstorm that puts Wall•E and EVE into trouble.

The Fujaba NXT example picks up from here: EVE gets stuck in a big pile of waste and Wall•E also gets damaged in the storm. Fortunately, he manages to return to his home, where he has some spare parts to repair his broken wheels. Wall•E does not have spare set of his original continuous-track wheels and is forced to use three wheels instead. He mounts two wheels in front and one wheel in the back. One motor is attached to the axis on which the front wheels are mounted whereas a second motor enables Wall•E to turn the rear wheel. How should Wall•E program his navigation computer to find his way back to EVE? The children are invited to assist him in this quest.

The instructor can show some motivating pictures and scenes from the movie. In fact, for the "open day", a lively set with waste bags and pictures of large dumps is constructed. The interested reader can access a dedicated instructor's manual for further details [7].

Figure 10 shows the track design that has been used to demonstrate that the robot could in fact find his way through the imaginary duststorm. Figure 11 shows an example execution. Note how the robot drives from the black line several times and backs up iteratively to find a state where his both light-sensors are on top of the black line again.

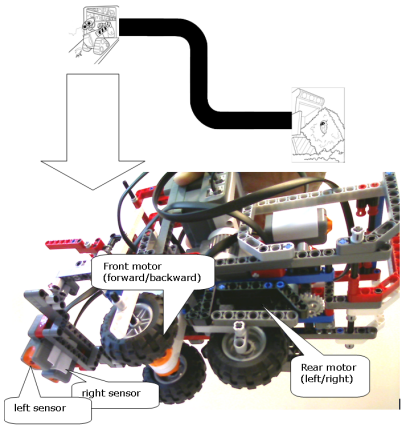

Figure 10: hardware setup for the Wall•E example.

Figure 12: Wall•E session with kids from 12 years of age.

When arranging tables as a corridor in the shape of the path from Figure 10, 12 year old kids need about 5 minutes to walk from start to end. In order to avoid cheating, one can blindfold particular role players. As shown on Figure 12, we decided to blindfold the child playing the robot instance only, since the robot instance needs to map light sensor events to navigator calls. The pictures shown in Figure 10 are from a tryout session in an elementary school in Mol, Belgium.

3.4.2 Models

Figure 13 shows a story diagram that models how to make the trike robot (i.e., Wall•E) move forward. The diagram is specified in the context of a navigator class that specializes navigation behavior for the three-wheel (trike) design. Such specialization is necessary since the default FNavigator class assumes a differentially wheeled robot design (such as the UniKassel continuous track robot of the Hanoi example).

The *this* node shown on Figure 13 represents the specialized trike navigator. Story diagrams of the shown level of complexity can be used for explaining to kids (or the parents that accompany them) that the role playing game is more than just a game: these implementation models of the robot turn out to be so close to the walkthroughs that almost anyone who is able to read can comprehend them (at least partially).

It is the responsibility of the instructor though to make the

Figure 11: Example run of the Wall•E Rescues EVE scenario.

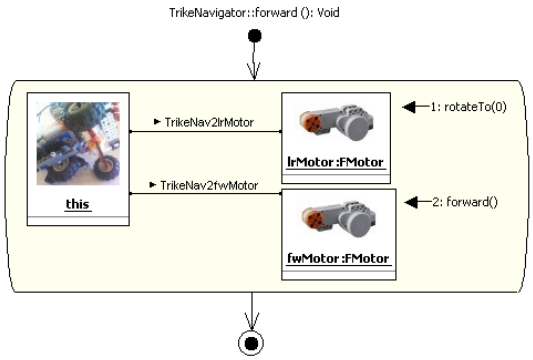

Figure 13: Story Diagram: move the trike forward.

children aware of which node on the diagram they have been playing. Similarly, the instructor should indicate explicitly the mapping from diagram edges to the cords from the role playing game and (roughly speaking) the physical wires on the LEGO robot. Finally, we have perceived that the instructor should not take for granted that all children grasp immediately the order of execution of collaboration statements.

Remark that the icon mechanism has turned out to be very practical in this context. Unfortunately, the icons associated to classes are not used automatically at runtime by eDOBS. Instead, a *getIcon* method (as shown for example on Figure 6) needs to return the image location as a string explicitly. Obviously, we have filed the automatic generation of such a simple method implementation as a feature request.

3.4.3 Analysis

We have collected feedback from a group of 12 year old kids. We asked the children (i) which part of the session they liked the most, (ii) which part of the session they disliked the most, (iii) what they had learned, (iv) what they did not comprehend, and (v) what they would have liked to have seen or done additionally.

Some results are hard to map to improvements. For example, some kids like most the part were they have to guess how Wall•E finds its way to EVE by means of looking at the actual LEGO device in action. Such kids find the role-playing rather boring. Others indicate exactly the opposite. The trial session has also exposed significant differences in mental capabilities, even though the trial session has been performed with kids from the same class: some children can specify exactly how many degrees the rear motor has to turn (+45 degrees or -45 degrees for turning right or left respectively) whereas others indicate that "they simply do not understand why there are so many motors involved".

The survey does provide feedback that can be used directly: a very large portion of students has indicated that they disliked that only one child per group of three children was actively role-playing. We did not get this feedback when performing the Factory session, which targeted adolescents and take the difference into account. More specifically, we aim to have more than one instructor available during future workshop sessions for kids, such that multiple role-playing sessions can be organized in parallel. We can still keep the T-shirts available for those kids (if any) that fear to be exposed in group.

Additionally, a significant amount of children indicated that

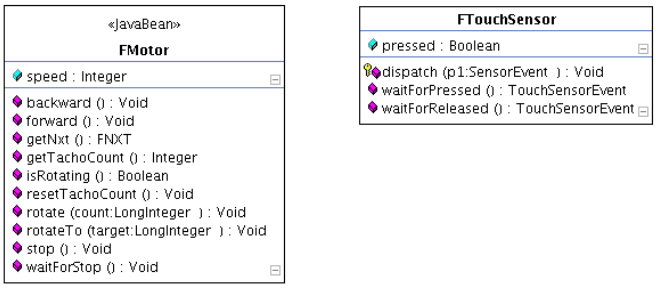

Figure 14: Class diagram extract of the Fujaba NXT library layer.

they would have liked to have "driven" the robot by themselves. It is hard to figure out whether they simply wish to instruct the navigator explicitly (e.g., via eDOBS) or whether they wish to change some diagrams (i.e., do actual programming). We acknowledge that both directions are quite interesting but so far we have not managed to perform such activities within the available time of half an hour. Obviously, we look forward to exploring this further when more time is available and when the groups are smaller.

4. LESSONS LEARNED

This section presents the main lessons that the authors have learned after the publication of [4]. First of all, we present new modeling guidelines, secondly we describe an improvement of the underlying engine. Finally, this section presents a list of known open issues.

4.1 Modeling Guidelines

The common approach to model a custom robot is to subclass one or more classes from the core API, like FNXT. This way, default behaviour can be easily overridden. A statechart should be used to model the global robot states. From that, story diagram methods are called to execute actual robot actions. Furthermore, Statecharts can be used to react to sensor events. Figure 14 shows an extract of the Fujaba NXT API:

Some methods of these classes are synchronous whereas others are not. This is neccessary to model efficiently, but for inexpierenced developers, it is very difficult to distinguish between these. For example, all *waitFor...()*-methods are blocking, *forward()* and *backward()* not, whereas *rotate(long)* is synchronous as well: it returns when the rotate task is completed. A *waitFor...()* method implicitly encodes an event handler.

4.1.1 Bad Style in Story Driven Modeling

When using the *waitFor...()* methods frequently in story diagrams, one is scattering implicit event handlers. Figure 15 for example contains some, rather hard to find, *waitIfLast* and *waitIfEmpty* calls.

In general, Figure 15 illustrates the authors' negative experiences with the Story Driven Modeling of multi-threaded event handling. These experiences have eventually lead to the combined use of Story Diagrams and Statecharts. Obviously, Figure 15 is not intended to be readable. In fact,

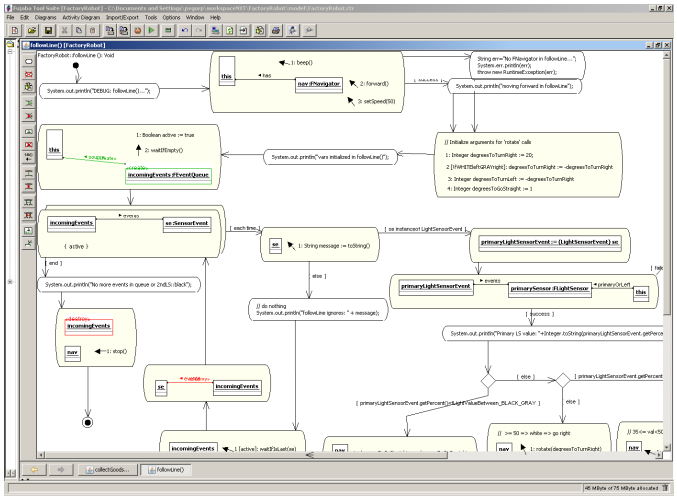

Figure 15: Example of SDM bad style.

it does not even fit on a mainstream notebook screen. One could work around this limitation by refactoring some fragments to separate methods. However, the core complexity, which relates to the explicit modeling of an event queue, cannot be removed using Story Driven Modeling constructs. Mind that this is not only an issue of forced over-specification (i.e, a lot of work) but more dramatically we have learned the hard way that this modeling style is very error-prone and hard to debug.

4.1.2 Proposed Style: Combine SDM and Statecharts

Statecharts seem to be a feasible solution to overcome the problems related to the Story Driven Modeling of event handling: each statechart runs in a separate thread implicitly while wait and notify methods are called when expected, behind the scenes. The current sensor adapter implementation ensures that all sensor events can be used to trigger state transitions. Currently, we use a very simple string encoding of the event triggers: <sensor-port> _ <state>, where sensor-port is s1, s2, s3 or s4 referring the hardware ports at the NXT brick, and state is black, white, pressed or released, depending on the connected sensor type.

Figure 16 shows the statechart of a common task for a driving robot: Follow a black line on the ground. The robot has two light sensors attached, which both should follow a wide black line. When one sensor goes off the line, it will read white, and an event trigger either *s2_white* or *s3_white* is fired. The statechart goes in one of the turn states and the robot will turn until both sensors read black again.

Recall from Section 3 that for sequential domain algorithms (e.g., let a robot collect goods from shelves) and basic navigation modeling (e.g., make a trike turn right), Story Diagrams do not expose the bad style that is discussed in the previous section. In fact, it does not make sense to replace the use of Story Diagrams by Statecharts completely: in several cases, the pattern-based specification approach and the use of icons appears a perfect fit for all children between 8 and 18 years of age. Therefore, a multi-formalism modeling approach seems most appropriate.

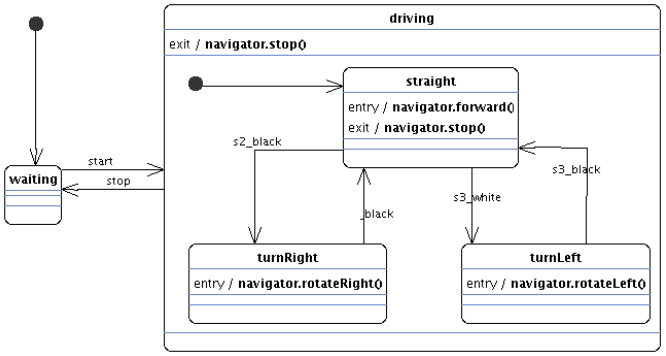

Figure 16: Line Follower algorithm as a statechart.

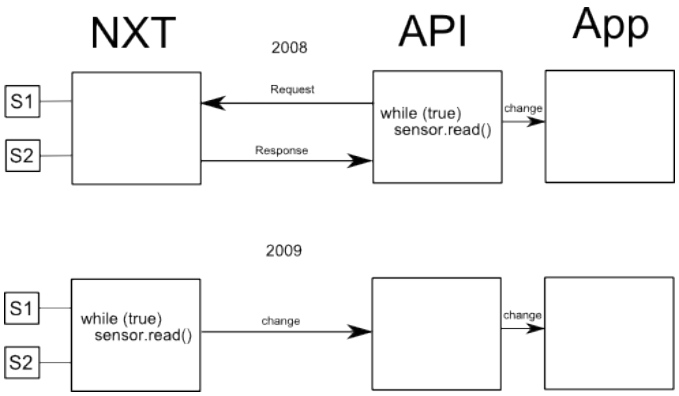

Figure 17: Architectural change of the event polling mechanism.

Also mind that we decided only show to the children those diagrams that illustrated good modeling style (e.g., the one shown on Figure 13). In the authors' opinion, it is not yet relevant for the children to learn from advanced pitfalls in the context of sessions that are primarily intended to stimulate motivation and a basic understanding.

4.2 Engine Improvements: ActiveSensors

The initial architecture of the Library as presented in 2008 suffered from the fact that the controlling PC had to poll all sensors actively in a loop, generating a permanent load on the bluetooth connection, which interfered with host commands. Furthermore, the latency until a sensor change was detected could be up to 50ms in the worst case, which makes quick reactions difficult.

In 2009, the authors from the University of Kassel started to investigate running parts of the library locally on the NXT. Figure 17 visualizes the architectural restructuring: the sensor poll loop now is now executed by the TinyJava VM in the NXT brick.

Because even the NXT-local LeJOS sensor implementation does not offer a notification mechanism upon sensor changes, the framework still relies on polling. This has not resulted in performance problems so far as there is no other code running on the NXT yet. The polling is already more efficient than in 2008 due to the reduced bluetooth traffic. Each time a sensor value changes, an event is generated and send to the controlling PC (as well as the initial sensor value). The same approach is done for motors: The motor state is monitored permanently and when a motor state changes (running or stopped), an event is send. Furthermore, when pressing one of the NXT buttons, a button event is send to the PC.

The PC still controls the NXT over the bluetooth connection with host commands using the LegoControlProtocol (LCP). Our program running locally on the NXT still needs to handle LCP requests now. A full-duplex stream connection between the NXT and the PC will be opened when the host application starts. As some of the LCP commands require answer packets, our program is responsible for sending those back to the PC. Because of the limited NXT hardware bluetooth module, which supports only a single full-duplex stream connection, we cannot use a secondary connection for sending the events to the PC. We have to send the events using the same connection: This requires a demultiplexer at the receiving PC to distinguish between LCP answer packets and event packets. After receiving such a event, a corresponding SensorEvent is generated and dispatched to the application layer.

Except button events, which the Pccomm API doesn't allow, all ActiveSensors functionality is fully transparent to the application layer. Thanks to the layered design discussed in Section 2.1, an application can still choose between the 2008 remote polling mechanism and the 2009 approach. The latter one requires a local program to be deployed on the NXT, which is part of the Fujaba NXT framework distribution.

4.3 Open Issues

Work on the Fujaba NXT framework and the construction of the examples described in this paper has so far been quite challenging. Although this should no longer be visible in a finished example, it does have an impact on the number of examples that has been completed successfully. The authors are convinced that a core issue has been tackled with the new statechart approach. However, the reader should be aware that other challenging issues are still open. Therefore, this section provides an overview of such issues and some known workarounds. Although these issues have not been solved yet, new developers of Fujaba NXT examples can save time by taking into account the known dangers and pitfalls, as experienced by the authors.

- when debugging Fujaba NXT applications (or embedded software in general), one can often not easily make erroneous behavior available for replay by other developers. This is due to the large amount of electrical components that can have bugs or thoughput limitations too (bluetooth dongles, electric cables, ...). So far, the first and second author managed to collaborate remotely by means of video sharing. In the case that the other developer could not reproduce the undesirable behavior, the first developer understood he had to consider deploying on another machine before spending more time on software debugging. In future work, best practices for using the simulation adapters will be investigated to make the debugging cycle more reliable and less time consuming.

- debugging multi-threaded software is a challenging task in general. The following characteristics make debugging the multi-threaded Fujaba NXT applications particularly hard to debug: when suspending a thread running on the host computer, the related threads on the LEGO components may continue to execute. To tackle this, special debugging features should be added to the Fujaba NXT framework (e.g., stopping all motors when suspending the VM on a breakpoint). This can be tightly integrated with Design Level Debugging [6]. Also mind that Design Level Debugging facilities would be very valuable too for the Statechart models.

- The underlying LeJOS framework still exposes bugs (obviously where they are least expected) and has a counter-intuitive design. For example, the bluetooth startup code is in the static initializer of the Motor class.

5. SUMMARY AND CONCLUSIONS

This paper illustrates how the Fujaba NXT framework is applied with reasonable success in several educational projects. An intended outcome of these projects is that the children acknowledge that role playing can provide a useful basis for programming. The underlying hypothesis is that this will break perceptions (if any) of software development as a pure asocial activity and we hope this attracts additional bright people to the software industry that would otherwise have choosen another path.

The framework consists of (i) Fujaba models that provide some basic data structures and building blocks of primitive sensor and motor functionality, and (ii) a set of hand-coded wrapper classes that integrate third party (LeJOS) binaries. The object-oriented nature of the Fujaba models enables one to easily apply inheritance and delegation techniques to deal with variability in robot hardware designs (e.g., continuous track designs versus the discussed trike design). The underlying LeJOS framework does not have this characteristic.

The Fujaba NXT framework turns out to be useful for role-playing sessions (at least) with kids between 8 and 18 years old. From 2002 onwards, behavioral modeling was done with story diagrams that generalized the behavior of story boards, that in turn represented snapshots from a role playing game [2]. The authors of this paper still acknowledge that the unique strengths of Story Driven Modeling remain (i) its semi-structured method for role-playing, (ii) the simple mental mapping for role players, since several domain alogithms involve rewriting steps, (iii) its syntax specializability by means of a simple icon mechanism, and (iv) its tool support for runtime visualization (eDOBS).

In 2008, the inherent limitations of this approach became better understood and since 2009 one can bypass these limitations by expressing event-based behavior using Statecharts.

6. FUTURE WORK

Section 4.3 already lists the technical issues that remain to be solved. This section focuses on future work on the conceptual level.

From the Copy Robot example, we learn the following: instructors cannot expect that bachelor or master students without Fujaba training will use story diagrams (or statecharts) spontaneously for the development of new Fujaba NXT projects. Such students do rely on a story boarding process but resort to pseudocode for generalizing method behavior. The authors consider it a challenge for the Fujaba community to provide more, and more elaborately documented, examples of story diagrams (and statecharts). The authors themselves will continue their collaboration to contribute such examples to the community. On the sort term, the Fujaba NXT library will be extended with more ready-to-reuse basic functionality.

Additionally, some university student projects related to Fujaba NXT are being supervised and even more will be supervized on the longer term. These projects should result in (i) stable hardware/robot/sensor setups (with building instructions) and (ii) multiple modeling solutions for one given problem and one or more robot designs.

7. REFERENCES

[1] T. Fischer, J. Niere, L. Torunski, and A. Zündorf. Story Diagrams: A New Graph RewriteLanguage Based on the Unified Modeling Language and Java. In *Proceedings of the 6th International Workshop on Theory and Application of Graph Transformation (TAGT)* , volume 1764 of *LNCS*, pages 296–309. Springer Verlag, Nov 1998.

[2] I. Diethelm, L. Geiger, A. Zündorf. UML im Unterricht: Systematische objektorientierte Problemlösung mit Hilfe von Szenarien am Beispiel der Türme von Hanoi. *Erster Workshop der GI-Fachgruppe Didaktik der Informatik, Bommerholz, Germany*, Oct. 2002.

[3] I. Diethelm, L. Geiger, A. Zündorf. Fujaba goes Mindstorms. *Objektorientierte Modellierung zum Anfassen; in Informatik und Schule (INFOS) 2003, München, Germany*, Sept. 2003.

[4] R. Jubeh. Simple robotics with Fujaba. In *Fujaba Days*. Technische Universität Dresden, Sept. 2008.

[5] P. Kanyuk. Brain springs: Fast physics for large crowds in WALL-E. *IEEE Computer Graphics and Applications*, 29(4):19–25, 2009.

[6] Leif Geiger. Design Level Debugging mit Fujaba. In *Informatiktage*, Bad Schussenried, Germany, 2002. der Gesellschaft für Informatik.

[7] P. Van Gorp, R. Mans, and J. Samalikova. Help Wall•E door de vuilnisberg heen! handleiding voor publieksdag project, Sep 2009.

[8] Wikipedia. WALL-E. http://en.wikipedia.org/wiki/WALL-E.

[9] LeJOS, Lego Mindstorms NXT. http://mindstorms.lego.com/, 2009.

[10] LeJOS, Java for Lego Mindstorms. http://lejos.sourceforge.net/, 2009.

[11] A. Zündorf. Story driven modeling: a practical guide to model driven software development. In G.-C. Roman, W. G. Griswold, and B. Nuseibeh, editors, *ICSE*, pages 714–715. ACM, 2005.