A Visual Interface for Concretizing Sorting Algorithms

Ilkka Jormanainen

22.09.2004

University of Joensuu
Department of Computer Science
Master’s Thesis

Abstract

Algorithm visualization is an efficient way to teach programming. Several different visualization techniques have been developed in the past decades. The Concretization Environment Framework, CEF, combines algorithm visualization with concrete objects (e.g. Lego Mindstorms robots). CELM, Concretization Environment for Lego Mindstorms is an application of this framework. By using the framework, the user can turn the mental model the user has into a concrete one. User feedback on the framework and its application has confirmed the functionality of the concept and the usefulness of the approach.

ACM-classification (ACM Computing Classification System, 1998 version): K.3.2 **[Computer and Education]**: Computer and Information Science Education - *Computer Science Education*; I.6.8 **[Simulation and Modeling]**: Types of Simulation - *Distributed, Parallel*

Keywords: algorithm, concretization, robotics, role-based, visualization

Preface

I have almost finished the first step to the interesting world of computer science. Now, it is time to thank you all for supporting me to achieve the goal I set about five years ago.

First of all, I would like to thank my supervisor professor Erkki Sutinen who has always given a new, refreshing idea when I have not been able to found it by myself. My advisor, Dr Meurig Beynon, gave me lots of valuable comments for finishing the thesis. The whole research group of Educational Technology at the Department of Computer Science, University of Joensuu, has offered an unique and creative atmosphere where to study and work. Especially I would like to thank Niko Myller, Osku Kannusmäki and Javier Lopez Gonzáles who have helped me to search the core of my thesis. The members of the Kids' Club track of the Educational Techonology Summer School in August 2004 (Chris, Marjo, Martyn, Meurig, Mike and Pasi) helped me a lot by giving the valuable feedback about the concept. Justus Randolph gave me an enormous help with the grammar of the thesis. Also, many other friends have supported and helped me. Thanks!

The support from my family has played an irreplaceable role during the whole study process. I really would like to thank them for it. However, the greatest thanks goes to my dearest, Sari, who has always believed in me when I have had rough times with my work. With a great patience, she has taught me what is important in this life.

Contents

1 Introduction

One of the main difficulties that students of computer science face is understanding algorithms. Traditionally, algorithms have been taught by verbal explanation with the use of blackboard or slides. With these it is only possible to visualize algorithms in a static way.

In past decades, researchers have developed different kinds of systems for algorithm visualization. Most of these systems allows the user to interact with the visualization and the algorithms are often visualized through animation (Stasko, 1990; Ben-Ari et al., 2002). Visualization has been used also in other fields of computer science. For example, a simulator tool has been developed for teaching computer architecture (Yehezkel, 2002). However, in this thesis I will focus only on algorithm visualization.

Robot technology has become cheaper and has been adopted widely to teach programming and computer science especially to novices. Robotics has been used to motivate students to learn programming. A student can create concrete new knowledge and learn in a constructionist way by interacting with real world objects (Ben-Ari, 1998). This can also lead more to hands-on learning with algorithms. In *algorithm concretization*, the algorithm's execution is emulated by robotics or other real world objects. In this way, robots engage the student with the algorithm thereby fostering learning.

However, it might be difficult to implement even a simple algorithm (e.g. Bubble Sort) for robots. This is because of the low-level implementation which should be done in order to get robots to produce a visualization (González, 2004).

In this thesis I will introduce a novel framework for concretizing algorithms. Furthermore, I will implement an application which is based on the framework. With this application, the user is able to make concretizations for sorting algorithms in a visual way and more easily than it has been done in (González, 2004). In this way, it is not necessary to rewrite the whole code of the algorithm for every single object. The framework and the application are not developed for algorithm visualization primarily but for constructing concretizations in a visual way. However, the framework supports a simulator tool, which can be used for visualizing algorithms. Furthermore, the framework allows the user to

construct the concretizations with a *role-based concretization* approach. This approach uses *interesting events* of the algorithm (Demetrescu et al., 2002; Stasko 1990).

Questions.
The research questions in this thesis are as follows.

1. What should be the general characteristics of a framework to support algorithm concretization by visualization?
2. What kind of architecture should the framework have in order to be used in diverse concretizing platforms such as Lego Mindstorms or EK Japan Co., Ltd.'s Soccer Robo® 915?
3. What kind of additional features does such a framework facilitate?

Methods.
As the questions illustrate, the current study analyzes novel concepts. To assess the potential usefulness of the approach, I use concept implementation as the research methodology. To get answers to the questions above, I design the framework and implement an application of it. Finally, I gather feedback from users of the application. Based on this feedback and the experiences I have gathered during the planning and the implementation, I will draw conclusions to the research questions.

It is important to notice that this work uses an existing work as its basis. The protocol used for communication between the Lego Mindstorms robots has been defined in (González, 2004) and it constructs the object layer of the application described in Chapter 4. It also gives inspiration for discussion concerning the role-based concretization approach. Although I have not implemented the communication protocol, I will describe it to make it clear how the object layer of the application works.

The structure of this thesis will be as follows. In Chapter 2, I will examine the background of the framework. I will also provide an overview of the field and I will analyze existing systems. Moreover, some problems on the field of concretization will be examined.

In Chapter 3, I describe the design of the framework on the abstract level. Chapter 4 describes the exact implementation of this application. I will discuss the user feedback on the framework and the application in Chapter 5. I will present the results in Chapter 6.

2 Background: Using Illustrations in Computer Science Education

The ways that algorithm visualization can be used have changed rapidly during the past few decades. However, it is not so clear whether visualization improves the learning process or not. Petre and Green concluded that the information contained in secondary notation is the main advantage of the visualization. This notation can be, for example, the placement of visualized elements, the colors or the indentation (Petre 1995; Petre and Green 1993). In the light of these studies, I will discuss how visualizations improve learning of computer science concepts. I will also present some techniques and tools for visualization which have been used in computer science education. At the end of this chapter, I will present the theoretical background to the framework which I have developed for this thesis. I will present the framework thoroughly in Chapters 3 and 4. The framework is based on the interesting event approach which I describe in Sections 2.2.2 and 2.2.3.

2.1 Algorithm and Program Animation in Computer Science Education

Learning algorithms has been seen as one of the main difficulties that students of computer science face during their studies. Therefore algorithm animation has been used for decades to aid in the learning process. The first purely educational algorithm animation was the videotape "Sorting Out Sorting" in 1981 (Baecker, 1981). This video explains nine sorting algorithms and also makes a comparison between the algorithms' running time.

In general, the field of software visualization has been divided into two domains. In the program visualization domain (PV), views of program structures are generated automatically (Korhonen, 2003). Views of this visualization type can be dynamic or static (Brown, 1988). These views trace the execution of the algorithm step-by-step; they are

low-level views that are not expressive enough to adequately convey how the algorithm works (Korhonen, 2003). The second domain, algorithm visualization (AV), visualizes all the states of the data structures during the execution of an algorithm. These visualizations are required to fully understand the behavior of the algorithm (Korhonen, 2003). Figure 1 presents how the different domains of software visualizations are related to each other (Price et al., 1993). The size of each circle is not significant, only the intersections between them.

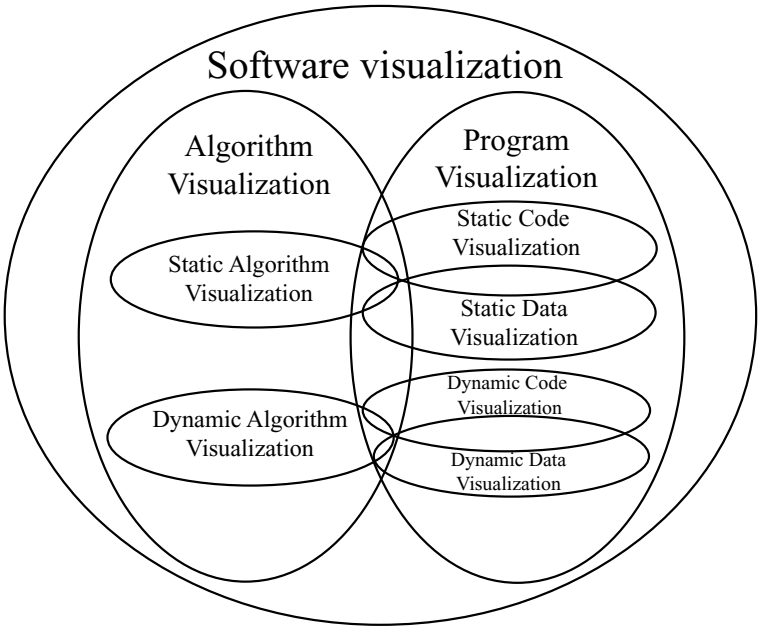

Figure 1: Venn diagram for different domains in software visualization (Price et al., 1993).

In the field of AV, it is crucial to preserve only those characteristics of data structures that are essential. Thus, some trivial data types or variables which do not offer any additional information about the algorithm's behavior, can be omitted (Korhonen, 2003). Price et al. (1993) write that algorithm animation is considered to be the dynamic visualizations of algorithms that are implemented later, and program animation is considered to be the dynamic visualization of the actual implementation of programs. Many systems contain features from both of these fields. However, our framework and application focuses only on algorithm animation.

The methods of algorithm visualization can be divided into three categories. Different tools use typically one of these three methods for visualization. In the following list, I

will go through these different methods and discuss their advantages and disadvantages.

1. **Hand-coded visualization.** At the lowest level, the algorithm is *re-written* to produce the visualization during the execution of the algorithm. This approach has been used, for example, by González (2004). Usually, this is a quite hard task because it might even take hours to produce a good animation for a simple algorithm (e.g. Bubble Sort).

2. **Visualization library.** At the second level visualization is done by *adding function calls* to some external visualization library. This is done in order to get interesting events visualized. This level of visualization is used in many tools. For example Tango (Stasko, 1990), XTango (Stasko, 1992) and Balsa (Brown, 1988) use this method of visualization. I will use this method in my framework.

3. **Automatic visualization.** At the third level visualization tools use *self-animated* algorithms. In this approach there is no need to add function calls to the code. Instead, the code is interpreted by the visualization system and the system producesthe visualization based on the real code of the algorithm. This is the easiest way for end user to produce a visualization. For example, the Jeliot family (Ben-Ari et al., 2002) uses this method.

Creating a good animation is usually a difficult and time-consuming task. For this reason, many tools for aiding this work have been developed. However, some doubts about these tools has been presented. For example, Fleischer and Kucera (2002) argued that the approach of trying to generate program animations automatically is futile. Instead they belive that a good animation must be designed and implemented by hand. Also, many empirical studies do not support the claim that visualizations unequivocally improve the learning of computer science concepts. For example, in the study by Stasko et al. (1993) the group that used animation to learn a complicated algrorithm did not perform better than the control group. They found that students had difficulties mapping the graphic elements of the animation to the algorithm since students had not used animation before.

According to Petre (1995) novice and expert programmers notice and concentrate on different graphical details. Petre defines *secondary notation* as an informal part of graphics (e.g. placement or color). Novices misinterpret secondary notation and cannot use graphics in on efficient way. Petre and Green concluded also that novices must be taught to read graphics (e.g. through visualization).

Ben-Ari et al. (2002) noticed that using Jeliot 2000 animations in the classroom was most beneficial for mediocre students. They noticed that the stronger students in the animation group had the same kind of difficulties as the stronger students in the control group. Students in the animation group believed that they could use the material and learn without Jeliot. In light of these studies, one can conclude that it is not certain if algorithm visualization is a good way to learn in all cases. However, there are several indicators that visualization can positively affect to the learning process.

2.2 Visualization techniques

The first step during the development process of algorithm visualization system is to define what kind of technique one would like to use. There are two main approach in the field of visualization: the *event driven* and the *state driven* approach. The self-animated approach has also been used in some visualization tools.

2.2.1 Event-Driven approach

The event-driven approach is probably the most commonly used approach to visualize algorithms. In this technique, an author of the visualization defines the *interesting events* of an algorithm. This event might be, for example, a swap operation in the code of a sorting algorithm. It is crucial to remember when defining these events that they have to be meaningful for visualization purposes. Then, these interesting events have to be associated with a suitable algorithm scene (Demetrescu et al., 2002).

Listing 1 presents a bubblesort algorithm in Java. In Listing 2 some interesting events

have been added to the code in order to get a visualization. In line 3 of Listing 2 the visualization is initialized with an array to be sorted and the size of this array. The interesting event generating the visualization has been added to line 10 just after a swap operation at the algorithm level. Note that this is only an example and it does not represent any real system. In Section 2.2.3, I will present an existing visualization system called Polka where visualization is based on interesting events.

```
1  int tmp;
2  int[] intArray = {12, 4, 5, 8, 7, 9, 45, 11};
3  for (int i=0;i<(intArray.length-1);i++) {
4      for (int j=(i+1); j< intArray.length; j++) {
5          if intArray[i]< intArray[j] {
6              tmp = intArray[i];
7              intArray[i] = intArray[j];
8              intArray[j]=tmp;
9          }
10     }
11 }
```

Listing 1: Bubble Sort algorithm.

2.2.2 Data-Driven approach

In the data-driven approach an animation is based on a graphical interpretation of the *interesting data structures*. The animation should reflect, at any time, the state of the program and its computation. This approach has been also used with commonly used debuggers. Debuggers update the display after each change of the program, for example in sorting algorithms, when a variable gets a new value or when two cells of an array are swapped.

```
1  int[] intArray = {12, 4, 5, 8, 7, 9, 45, 11};
2  int tmp;
3  cl.Init(intArray, intArray.length);
4  for (int i=0;i<(intArray.length-1);i++) {
5      for (int j=(i+1); j< intArray.length; j++) {
6          if intarray[i]< intArray[j] {
7              tmp = intArray[i];
8              intArray[i] = intArray[j];
9              intArray[j]=tmp;
10             cl.Swap(i,j); // An interesting event
11         }
12     }
13 }
```

Listing 2: Bubble Sort algorithm with an interesting event.

2.2.3 Interesting event example: Polka

Polka is a system for visualizing programs written in C++ (Demetrescu et al., 2002). Polka was originally written for X Window System, but a version for Microsoft Windows (PolkaW) has been released too. To create a visualization with Polka, the user has to carry out the following steps:

1. The user has to annotate the program source with *Algorithm Operations*, which are Polka's version of interesting events.
2. The user has to create *Animation Scenes* which perform an animation chunk.
3. The user has to specify a mapping between algorithm operations and animation scenes.

Listing 3 presents the Bubble Sort algorithm implemented in C++ (Demetrescu et al., 2002). To visualize this algorithm with Polka, the user has to specify which pieces of information related to the algorithm's execution should be visualized and how it should

be done (e.g. what kind of graphical elements there should be). A possible approach is to use horizontal rectangles to represent the elements to be sorted and animate the swap operation which is at line 6 in Listing 3. Figure 2 presents a PolkaW visualization of this algorithm.

```
1 int v[]={3,5,2,9,6,5,1,8,0,7}, n=10, i, j;
2 void main(void) {
3     for (j=n; j>0; j--)
4         for (i=1; i<j; i++)
5             if (v[i-1]>v[j]) {
6                 int temp=v[i]; v[i]=v[i-1]; v[i-1]=temp;
7             }
8 }
```

Listing 3: Bubble Sort algorithm in C++ (Demetrescu et al., 2002).

To achieve the visualization presented in Figure 2, the designer has added some interesting event calls to the code. These calls initialize and invoke the animation scene. At line 3 in Listing 4, the event call "Input" signifies that all the array values to be sorted are set and that the animation should draw the initial configuration of the array (Demetrescu et al., 2002). Basically, this event creates and lays out the set of the horizontal rectangles and scales them according to the corresponding array values. The second event, "Exchange", signifies that a swap operation between two elements of the array has occurred. Array elements which were exchanged are passed as parameters.

In the PolkaW visualization, the "Exchange" event invokes the `Exchange` method in the `Rects` class. This class produces the animation. The code of the `Exchange` method is presented in Listing 5.

2.2.4 Interesting data structure example: Leonardo

Leonardo is an integrated developing environment (IDE) for developing, executing and visualizing C programs (Demetrescu, 2001). Leonardo provides two major improvements

```
1  int v[]={3,5,2,9,6,5,1,8,0,7}, n=10, i, j;
2  void main(void) {
3      bsort.SendAlgoEvt("Input",n,v);
4      for (j=n; j>0; j--)
5          for (i=1; i<j; i++)
6              if (v[i-1]>v[j]) {
7                  int temp=v[i]; v[i]=v[i-1]; v[i-1]=temp;
8                  bsort.SendAlgoEvt("Exchange",i,i-1);
9              }
10 }
```

Listing 4: Bubble Sort algorithm in C++ with interesting event calls (Demetrescu et al., 2002).

which traditional IDEs do not have (Demetrescu et al., 2002):

1. Support for visualizing computation of the program graphically by attaching graphical representations to key variables.
2. A run-time environment that supports fully reversible execution of C programs.

With this visualization technique, basic animation can be obtained typically by adding a few lines of additional code to the original source. However, Leonardo does not realize the real state mapping technique since it allows users to choose and control which visualization declarations are active at any time (Demetrescu et al., 2002). The Leonardo system is distributed with a collection of animations of algorithms and data structures (see Demetrescu (2001) for more information).

Visualization declarations in Leonardo are written in *ALPHA*, which is a simple declarative language. Declarations are enclosed within separators `/**` and `**/`. To produce a visualization of Bubble Sort, the user has to append the code presented in Listing 6 to the code of the algorithm.

```
1  int Rects::Exchange(int i, int j) {
2      Loc *loc1 = blocks[i]->Where(Part_NW);
3      Loc *loc2 = blocks[j]->Where(Part_NW);
4      Action a("MOVE",loc1,loc2,1);
5      Action *b = a.Reverse();
6      int len = blocks[i]->Program(time,&a);
7      time = Animate(time,len);
8      len = blocks[j]->Program(time,b);
9      time = Animate(time,len);
10     Rectangle *t = blocks[i];
11     blocks[i] = blocks[j];
12     blocks[j] = t;
13     return len;
14 }
```

Listing 5: The code of the method which animates the swap operation of Bubble Sort (Demetrescu et al., 2002).

```
1  /**
2      View(Out 1);
3      Rectangle(Out ID,Out X,Out Y,Out L,Out H,1)
4          For N:InRange(N,0,n-1)
5          Assign X=20+20*N Y=20 L=15 H=15*v[N] ID=N;
6  **/
```

Listing 6: The ALPHA code which produces the visualization for Bubble Sort (Demetrescu et al., 2002).

2.2.5 Self-animated algorithm example: Jeliot

The history of the Jeliot family goes back to the 1990's. Researchers at the University of Helsinki created an animation library that could be used to animate programs written in C. Also, a library of self-animating data types was created (Sutinen et al., 1997). In this approach, there is no need to add extra notation to the program code. *Eliot* was the

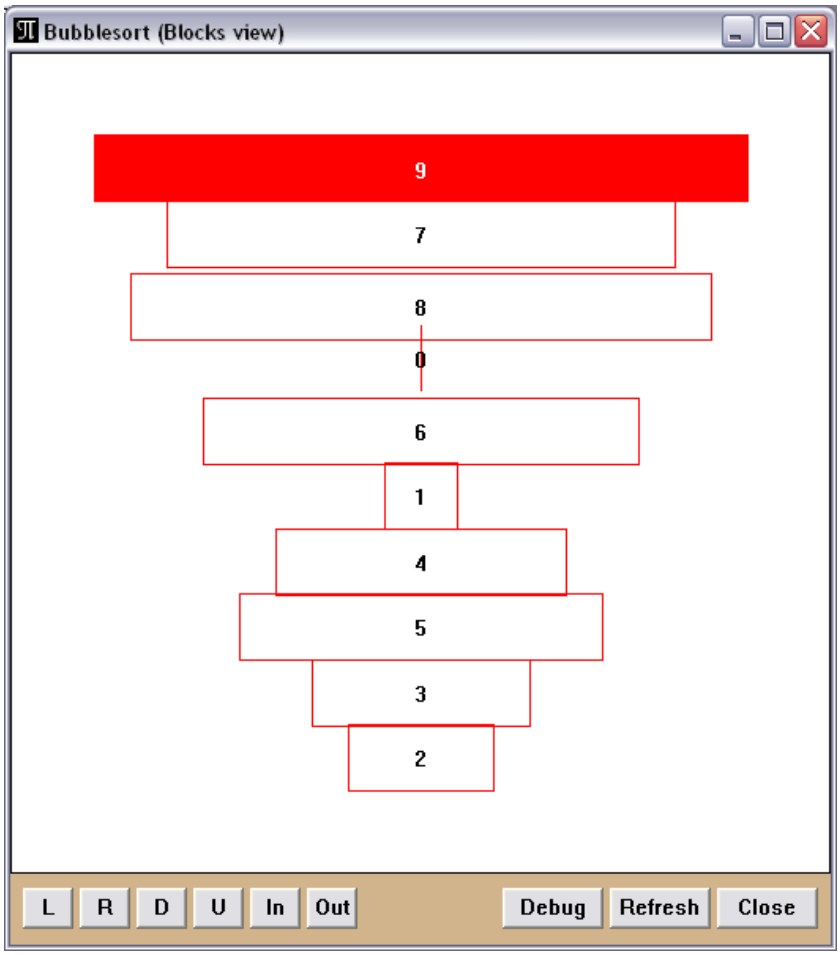

Figure 2: PolkaW visualization of Bubble Sort.

first product of the Jeliot family (Sutinen et al., 1997). After that, Jeliot I, Jeliot 2000 and Jeliot 3 have been released (Myller, 2004).

Developers of Jeliot 3 continue their work by adding new features to Jeliot. For example, by combining program visualization and collaborative authoring tools, it was possible to bring new aspects to the field of visualization. JeCo (JEliot COllaborative), introduced a novel concept that supports both program visualization and peer-to-peer collaboration (Moreno et al., 2004). This concept is called *collaborative program visualization*, and it supports the theory of *socio-cultural constructivism* where a learner should have possibilities to communicate with other members in a learning community (Duffy and Cunningham, 1996).

BlueJ is an integrated Java development environment specifically designed for introduc-

tory teaching. The environment presents object-oriented structures graphically and it is targeted for teaching programming with the "objects-first approach" (Kölling et al., 2003). Authors of Jeliot have developed an extension for BlueJ which allows the user to animate the BlueJ project (Jeliot 3, 2004).

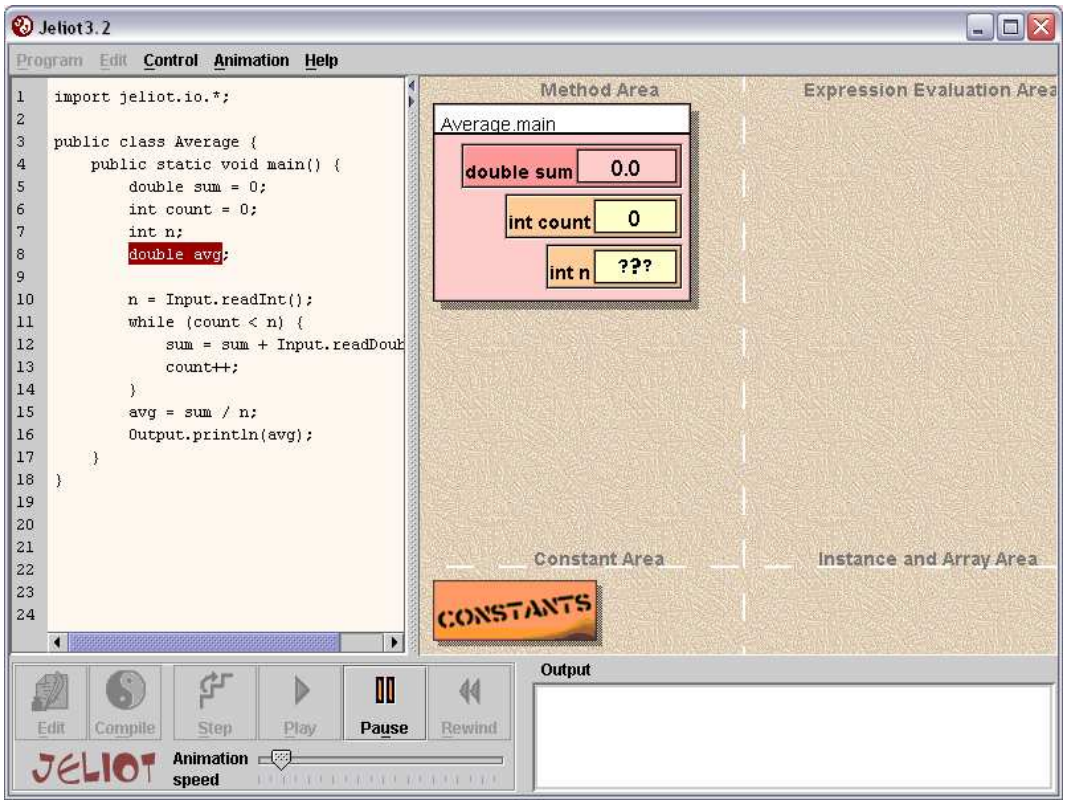

Figure 3: Visualization of a program by Jeliot 3

2.3 Concretization in Computer Science Education

In the few past decades, researchers and industries have developed a number of different robot kits designed to help learning in scientific fields such as mathematics, physics and computer science. These kits typically contain all the parts which are needed to construct a robot: motors, sensors, wheels, gearwheels and belts. Some of the kits (for example LEGO Dacta and LEGO CyberMaster) include cable or radio equipment that make it possible to connect the device to the computer. This allows the user to control the robot. Another approach uses autonomous robots. There is a small computer inside autonomous

robots, so they can communicate and move independently according the program the user has constructed. These kits have been built according to educational principles which have been derived from Jean Piaget's theories of cognitive development (Miglino and Lund, 1999). Seymour Papert has revised these theories. According to this approach the active learner is the center of the learning process. Learners enlarge their knowledge by manipulating and constructing objects (Miglino and Lund, 1999).

These rather cheap robotic kits can be used, for example, for teaching Java programming to novices (Barnes, 2002). Artificial organisms have also been used for teaching the design and construction of industrial prototypes to engineers who have a bachelor's degree. Often they have excellent knowledge of fundamental theoretical concepts, but they lack experience in construction (Miglino and Lund, 1999).

However, there might be some problems when teaching, for example, object-oriented problem solving with robots. For example, in the object-oriented programming approach, if we want to model a car, it might be necessary to model wheels, axles, petrol tanks, windows and so on. However, when using the LeJOS environment with Lego Mindstorms and RCX, motor and sensor objects are obtained via a pre-created static reference. This is not an appropriate way to teach object-oriented programming (Barnes, 2002). Also, physical restrictions of the RCX unit might prevent one from using it teaching programming. Lack of memory and difficulties in debugging programs with a RCX unit must be taken into account when planning courses and instructional materials.

Fagin and Merkle (2003) argued that the use of robots in teaching computer science is ineffective. They ran a year-long quantitative experiment in which they noticed that test scores were lower in the robotics laboratory sections than in the non-robotics sections. However, they believed that the most significant factor accounting for this result was the lack of a simulator for the programming environment that was in use. Students were unable to practice the write-run-debug loop that seems to be an important part of the learning process.

In the study of Fagin and Merkle, the authors also discussed the role of teacher experience as a factor in the negative result. All of the teachers in the experiment had only one

semester experience teaching the robotic sections. Although the authors had planned the robot exercises very carefully. Fagin and Merkle (2003) did not believe that they were completely successful in controlling for the lack of teacher experience with the robotics as a factor in student learning.

2.3.1 Algorithm concretization with robotics

Robotics have been used widely to teach computer science concepts. Programming, networking, artificial intelligence and many other topics are taught with robots. However, *algorithms* have typically been taught in traditional ways. González et al. (2004) present a novel way to teach sorting algorithms with robots. The authors ran an empirical study where they taught a bubblesort algorithm to 13-to-15-year-old students. The data was collected with a questionnaire and by taping the lesson. Results showed that at least some of students understood the sorting algorithm taught with robotics.

One can say that the added value of concretization, compared to the visualization, is the hands-on character of robotics which may positively affect a certain type of student. This issue is further addressed in this thesis.

2.3.2 Concretizing Bubble Sort algorithm with Lego Mindstorms

In his research, Javier Gonzales developed some concretizations of sorting algorithms with Lego Mindstorms (González, 2004). The main idea in his work was to use a master robot which controls other robots (slaves). In this scheme, every robot has an individual id and a weight. This information is used to sort the robots with a sorting algorithm. At the moment, Bubble Sort and Selection Sort have been implemented. The main idea for handling the communication is to use a pre-defined protocol which the robots use when communcating with each other. Furthermore, some kind of synchronization is needed to ensure, that the excecution of the concretization proceeds smoothly. Algorithms for the robots were developed in NQC (Not Quite C), which is a C-like programming language for Lego Mindstorms robots (Baum et al., 2000), and in Java with LeJOS. There is a

certain algorithm for the master and the slave robot. All slaves use the same code with only two slight differences: all slaves have an individual id and weight. This infomation is changed via the communication protocol. For more information about implementation, see González (2004).

As it can be seen in González (2004), algorithms with robots are complicated. This makes it difficult to use concretization in an efficient way, for example, when teaching programming or algorithms to novices. It needs much code to implement even a simple sorting algorithm, like Bubble Sort (see González (2004)). However, as has been stated in González et al. (2004), this method of teaching is promising and worth further study. With the environment I am developing for this thesis it is possible to construct concretization in a more sophisticated way.

The protocol defined in González (2004) is based on the behaviours of robots. In sorting algorithms, two particular behaviours, which repeat on every round of the algorithms, can be found. We call them `left` and `right` since they are associated with the physical position of the robot. In González (2004), the behaviour is one pre-defined movement of the robot. Behaviours are implemented as NQC tasks. The code of the slave robot contains three tasks: the main task, which starts the execution, and tasks for left and right behaviour.

One round of concretization of the sorting algorithm contains the following steps:

1. The master sends a message to the first slave.
2. The slave confirms this message by answering the master with its own id.
3. The master sends the slave a message which contains information about the slave's role (whether it is left or right).
4. The slave moves in the direction specified by the master.
5. The slave sends a signal, which shows that they have done the work, to the master.
6. The master sends a confirmation signal to the slave.

After one round of the loop the master sends the ids and behaviours to new slaves. Figure 4 illustrates this *protocol*. The technical details of the protocol and communication can be found in Section 4.6.

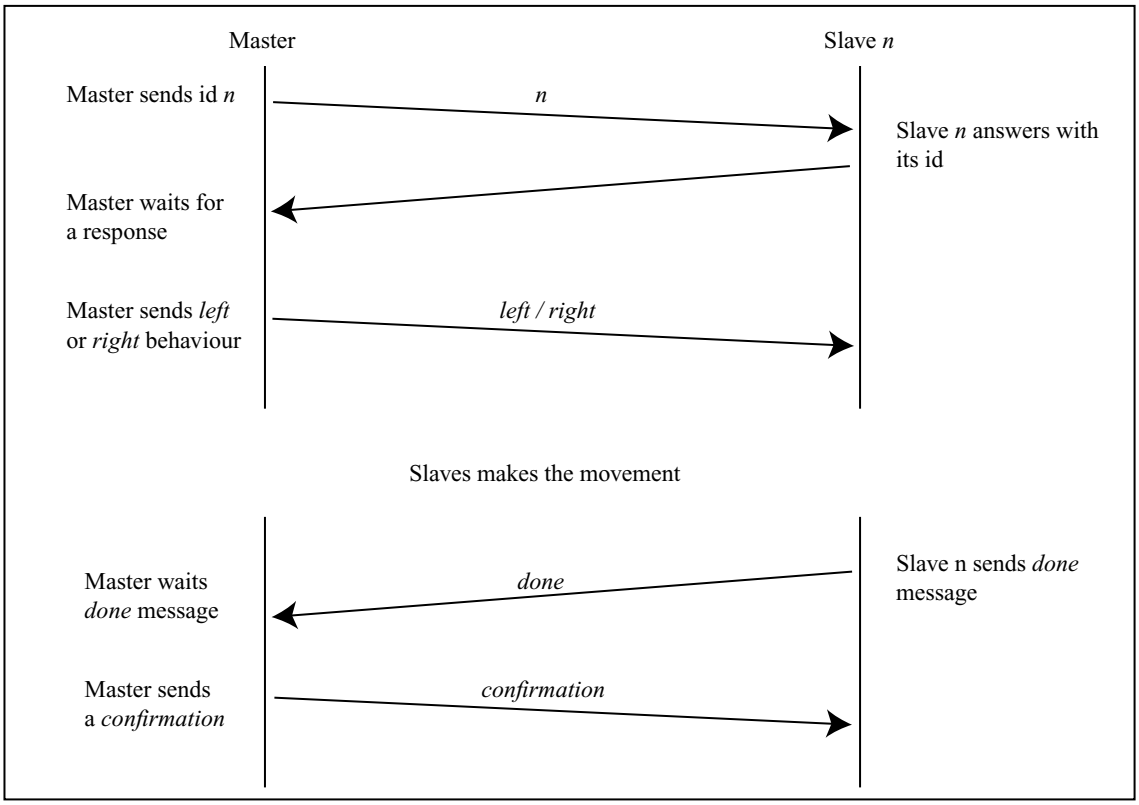

Figure 4: The protocol between master and slaves (González, 2004).

2.4 Summary

In this chapter, I have described an existing communication protocol which allows the robots to sort themselves according different sorting algorithms. However, the protocol is hard to implement. Also, it is a time consuming task to develop concretizations for new algorithms. For this reason, there is need for a framework, which defines the parts which are needed for the application which allows the user to construct concretizations for robots in a visual way.

In this chapter, I have also examined existing program and algorithm visualization approaches and tools. By using interesting events of the algorithm, it is possible to define

a framework which allows the user to develop concretizations easily. I will present this framework in the next chapter.

3 Design of the Framework

In this chapter, I will explain the main design issues for a general Concretization Environment Framework (CEF). I will discuss the principles of the design in the Section 3.1, sketch the CEF in Sections 3.2 and 3.3 and discuss the structure of the whole framework in Sections 3.4 and 3.5. In Section 3.6, I will present the limitations of the architecture and the approach. The CEF itself, and the principles behind its design are discussed in this chapter and illustrated by using some examples from the CELM environment. The technical issues concerning implementation are discussed in Chapter 4.

3.1 Principles of design

The main goal of this project was to develop an application which may help the user (teacher, instructor) construct concretizations for sorting algorithms. The application is directed, especially, at teachers and those who would like to get a better understanding of a particular algorithm which they already know at least on some level. However, the application is not primarily targeted at novices who aim to learn themselves, because the user has to know how the algorithm works in order produce a suitable concretization of it (see Section 3.6).

The environment uses an event-driven approach where interesting events of an algorithm are visualized (for more information about this approach, see Chapter 2.2.1).

The main difference between the work described in this thesis and existing systems that use the same technique (for example Tango (Stasko, 1990)) is that interesting events and their visualizations (and concretizations) are constructed using a graphical interface.

In the Jeliot family, one can also use a graphical user interface. However, Jeliot I is the member of Jeliot family that is most targeted to algorithm visualization. The Jeliot products after Jeliot I are primarily involved in the field of programming visualization. The main difference between this framework and Jeliot I is that Jeliot I uses a *self-animating approach*. The visualization of Jeliot is also based more on data structures of an algorithm

than interesting events.

3.2 Sketch of the framework and the application

In the application I have implemented for this thesis, I will use the *theater metaphor* in the same way that the Jeliot family uses it (Myller, 2004). Parts of this metaphor in this application are:

- Action: An interesting event in the algorithm in which the actors participates.
- Actor: A concrete object (e.g. a robot).
- Director: A user who leads the actors.
- Stage: A place where actors conduct actions.
- Storyboard: A place for gathering actions.

Figure 5 presents an outline of the application. All actions are played on a stage. The director can move actors over the stage by dragging with the mouse. Actors will try to realize the path the director has prescribed but will necessarily end up performing an approximation because of the inevitable noise in a physical surrounding environment. There can be numerous actors on the stage at the same time and these actors can communicate and interact with each other. These movements and discussions determine the code which can be sent to concrete objects (e.g. robots). The user can produce as many events as it is necessary to concretize a particular part of the algorithm. These events are collected on a *storyboard*. This can be seen as a filmstrip which contains short animated films.

The user can load a code containing the algorithm in the *code* window. In that window, user can mark an interesting event with a mouse. The content of a particular event is listed in an *eventlist*.

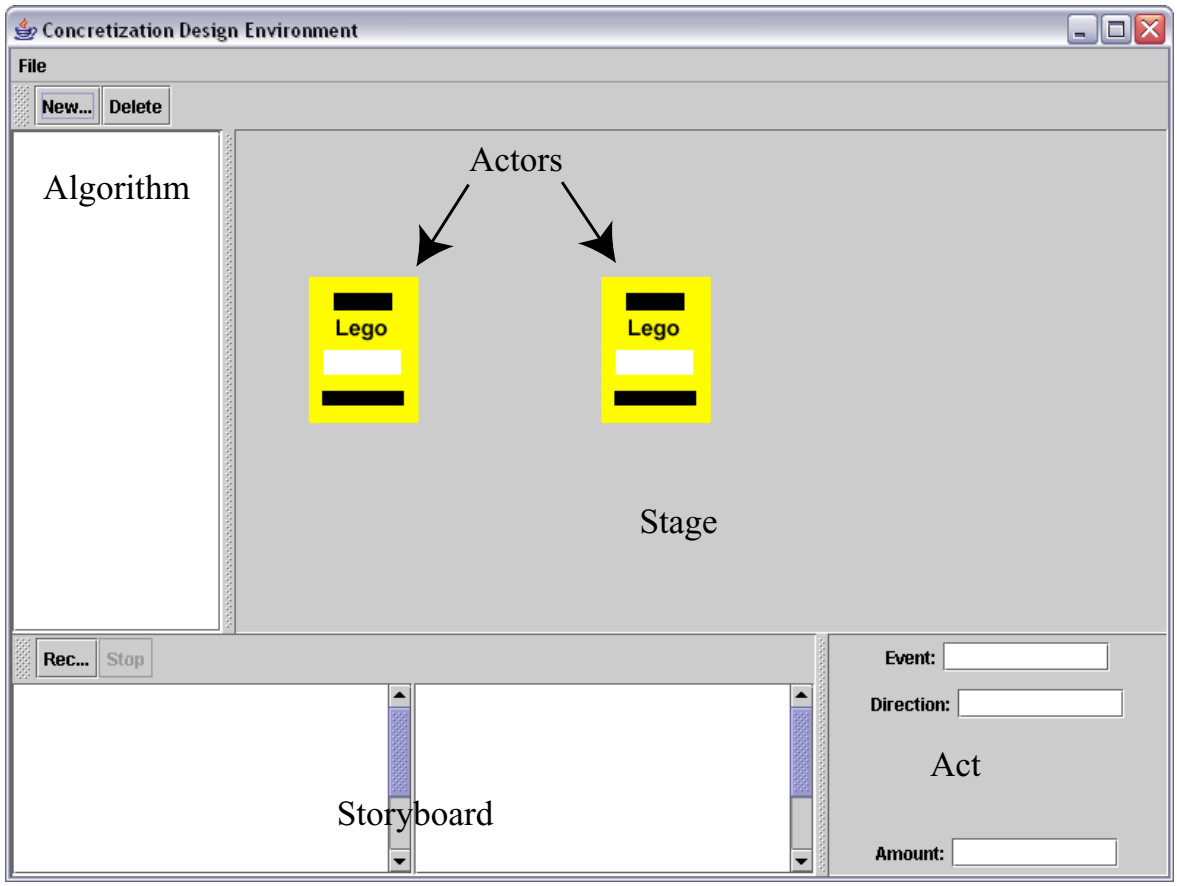

Figure 5: Outlines of the system.

3.3 Working with the framework

A generic workflow of the application is presented in Figure 6. Figure 6 shows how a user's mental model becomes concrete by using robots as a concrete model. In the beginning, users have only a mental model of how the algorithm *should* work in their minds.

The user defines the interesting events of the application and visualizations for these events. Visualizations are defined by dragging robots on the stage of the application. After that, the user allows the application to upload the code to the robots. Then, the robots will execute the visualization. In this way, the mental models which users have in their minds become concretized in the material world.

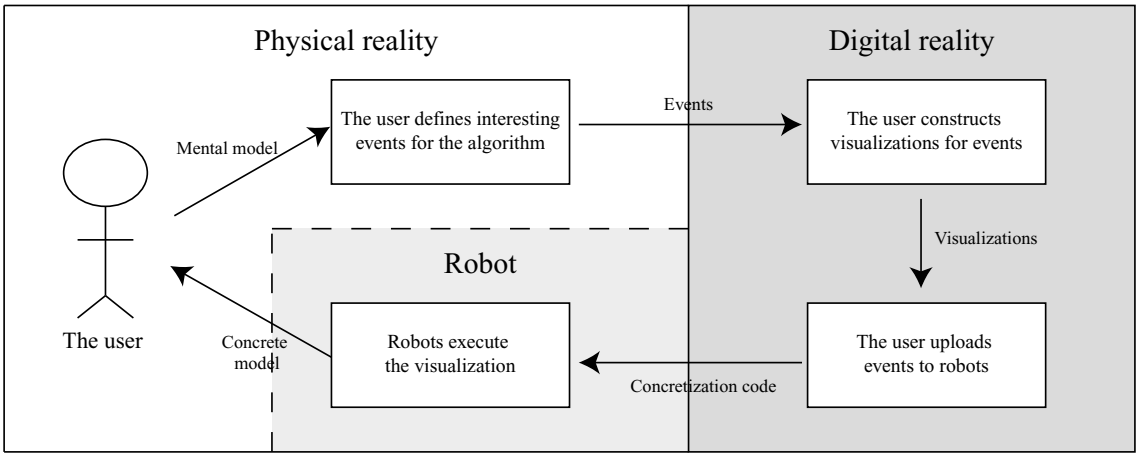

Figure 6: A workflow of the system.

3.4 Architecture of the framework

The architecture of the framework is designed in a way that it can be used in different domains. For example, controlling a Lego Mindstorms robot with a dedicated environment is one application of this framework. This thesis contains a discussion about this field. Another example could be controlling an enterprise robot via a network. These possible applications are presented in Chapter 6.

The architecture of the framework contains three separate layers. For each layer, there is some output which serves as input in the next layer. Communication between layers is bi-directional. This means that physical objects can communicate and send information about their states to the application. In this way, it is possible to track the movement of the robot. Figure 7 presents this structure and communication between layers. The figure also shows the idea how this kind of framework should be suitable in different domain areas. The communication protocol and other technical issues are presented in the Chapter 4.

The *environment layer* (EL) takes care of the communication between the user and robots. Results of these actions are sent to a *transfer layer* (TL) which takes care of transferring and converting the code produced by the environment layer to a *object layer* (OL). This layer represents the physical objects which concretize the algorithm. This object might be, for example, a Lego Mindstorms robot (Ferrari, 2001).

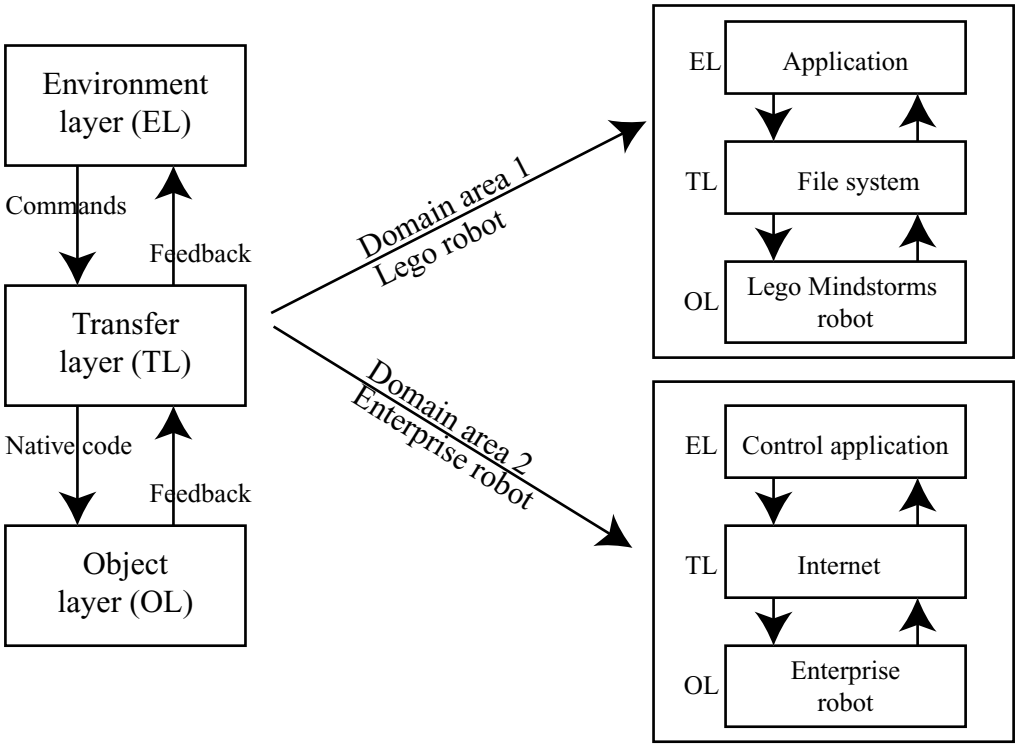

Figure 7: Architecture of the system

3.4.1 Environment layer

This layer is visible to the user as an application. The user can define interesting events and visualizations by using the interface of the application. Basically, there can be several objects on the stage at the same time. However, it might be difficult to handle, for example, discussion between objects if there are more than four of them present at the same time.

3.4.2 Transfer layer

This layer transfers the concretization constructed by the user to the object layer. Basically, this layer interprets the code for the robot that is in use (see Chapter 3.5). In this way, it is rather simple to replace only this layer according to the robot which is in use. For example, with Lego Mindstorms robots, this layer turns the code to LeJOS-code (Laverdae, 2001) and then it sends the LeJOS-code to the robot via an infrared transmitter. This layer also transfers the feedback given by object layer to the application.

3.4.3 Object layer

The purpose of this layer is to concretize the algorithm in the same way that users desire their mental models concretized with physical objects. These objects (robots etc.) can be seen as *physical learning objects* (Eronen et al., 2004). There have been some studies done which show that physical objects can facilitate students' building of mental models during the learning process (e.g. see Poon (2000)). In our case, robots represent the data of the algorithm (e.g. cells of an array or an individual variable). One robot can represent one variable at a time. The user can change this mapping during the execution of the program. Robots can exchange data in order to concretize, for example, a swap operation, which is commonly used in different sorting algorithms.

The object layer gets instructions for the concretization from the transfer layer. The format of instructions depends on what kind of concretization tool is in use. In the case of our application, the interpreter turns the robot code to LeJOS-code (see the Chapter 4).

3.5 Code

The environment layer produces code which it sends to the object layer via the transfer layer. The code contains instructions on how the robot should move and behave. The code uses object-oriented notation as in Java, where an individual robot is identified by a name. A dot follows the name and, after the dot, a method call is invoked with some parameters. For example, a simple instruction for movement may look like this:

```
robotA.move(20, FORWARD)
```

In this example, the command contains the following parts:

This instruction makes a robot named `robotA`, move 20 units forward. The user has to define the unit amount in the application. This *calibration* is necessary to ensure accurate movement of robots (González, 2004). The whole set of commands is rather small and simple. The complete list of commands is presented in Table 2.

Table 1: The structure of the command.

Part	**Purpose**
robotA	Name of the robot (representing a variable)
move	A method, which makes the robot to move
20	Quantity to move
FORWARD	A direction to move

Table 2: Set of commands.

Command	**Purpose**	**Example**
stop	Stops movement	robotA.stop
move(quantity, direction)	Move a robot a certain amount in a certain *direction*	robotA.move(20, FORWARD)
rotation(direction, degrees)	Turns a robot to *direction* amount of *degrees*	robotA.turn(RIGHT, 90)
send(value)	Sends some value	robotA.send(25)
send(value, robot)	Sends some *value* to a specific *robot*	robotA.send(25, robotB)
receive(robot)	Receive any message from a specific *robot*	robotA.receive(robotB)
receive(msg, robot)	Receive a specific message (*msg*) from a specific *robot*	robotA.receive(25, robotB)

3.6 Restrictions of the approach

The main problem with this approach is how to concretize loop and condition structures. For example, a condition statement basically needs two different concretizations - one for the case that the statement gets the `true` value and another one for the `false` value.

However, this is quite easy to handle by constructing two possible concretizations for this particular statement.

The situation is more complicated when there is a *loop structure*. Loops are very important control structures in programs. However, in this concretization approach loops cause a very difficult problem. The number of different possible execution paths of the algorithm is so large that it is impossible to produce a concretization for each of them.

Listing 7 presents a Bubble Sort algorithm. During the execution of the algorithm, lines 6-8 are invoked approximately $O(n^2)$ times, where n is the number of robots. Basically, every iteration of the lines 6-8 has to be represented by a quite distinct concrete action. Table 3 shows the approximate numbers of steps (concretization events) for different sorting algorithms when n represents the number of robots.

Table 3: Steps in different sorting algorithms.

Algorithm	**Average time consumption**	n	**Steps**
Bubble Sort	$O(n^2)$	3	9
Selection Sort	$O(n^2)$	4	16
Shell Sort	$O(n^{\frac{5}{4}})$	5	7

Possible approaches to solve this problem are:

1. A concretization is constructed separately for each event inside the loop. This is problematic, because there might be too many steps to produce (see Table 3). The developer of the visualization has to explicitly model the precise positions and arrangement of the robot for each step of the algorithm. Unless this is possible, the expressive power of concretization using robots cannot be exploited.

2. The user defines *roles* for robots. For example, in the sorting algorithm, robots which represent cells of an array always have a role called *left* or a role called *right* (González, 2004). It is possible to define a generic behaviour for both of these roles. After that, the movement of the robot depends on its role and on where it is located.

The direction and the quantity of movement are calculated based on the robot's index in the array. In Listing 7, variables `i` and `j` define indexes and furthermore, identify the robots which are used in one particular step of the execution. Later on, this approach is called *role-based concretization* (see Chapter 4.2).

```
1  int tmp;
2  int[] intArray = {12, 4, 5, 8, 7, 9, 45, 11};
3  for (int i=0;i<(intArray.length-1);i++) {
4      for (int j=(i+1); j< intArray.length; j++) {
5          if intArray[i]< intArray[j] {
6              tmp = intArray[i];
7              intArray[i] = intArray[j];
8              intArray[j]=tmp;
9          }
10     }
11 }
```

Listing 7: Bubble Sort algorithm.

There are also two other categories of restriction to which this approach is subject. Noise in the physical environment or lack of user ability to understand the algorithm to be concretized or a lack of programming knowledge may lead to difficulties.

3.6.1 Physical environment

The fact that robots operate in a concrete world causes problems due the limitations of working in a physical environment. For example, the movement of the robot may vary on different kinds of surfaces. Also the voltage level of batteries may cause different speeds between similar robots. Light conditions may change radically during the execution of the concretization and this may cause some strange results if light sensors have been used. More information about these problems and methods to reduce them can be found in González (2004).

3.6.2 User knowledge

The user has to construct all the movements and communication events for the robots with the application. Thus, the user has to know, at least on some level, how the algorithm works. This means that the system is more useful for those who are teaching programming than to those who are studying it. In the investigation conducted by González (2004), using of robots in teaching seemed to lead more deeper understanding. However, to be sure about this issue, investigations which are more structured, with good research methods, are needed.

From a technical point of view, although it seems quite easy for the expert programmer to replace, for example, the object layer (physical objects) with another one, this might be difficult for the nonprofessional programmer.

4 Implementation

In this chapter, I will discuss one option for the application of the framework. The application is called *CELM* (**C**oncretization **E**nvironment with **L**ego **M**indstorms). The framework and the approach itself are so complicated that it was impossible to totally implement it for this thesis (see Chapter 3.6). In the application I use the role-based concretization approach. In this approach, the user defines roles for the robot. When concretizing sorting algorithms, only two roles (left and right) are needed. The aim of the application is to enable the user to construct movements for these roles. Later on, the aim is to make the application less specific by allowing the user to invoke roles in a more general way. Some inspiration for this might be drawn from the work of Sajaniemi and Kuittinen on the role of variables in general procedural programs (Sajaniemi and Kuittinen, 2003). However, we will need more studies for this. I will present the limitations of the application in Chapter 4.3. In this chapter, I will discuss the technical implementation of the application.

In Section 4.4, I will go through the issues concerning the application layer. In Section 4.5, I will discuss the implementation of the transfer layer and in the Section 4.6, the object layer. The last layer is discussed in more detail by González (2004).

4.1 An overview of CELM

I have developed an application for the framework described in the Chapter 3. The application is called CELM, which stands for *Concretization Environment with Lego Mindstorms*. As will be shown, I have implemented this application for Lego Mindstorms robots. Figure 8 shows how CELM (the application) and CEF (the framework) are related to each other.

Figure 9 illustrates the application and how it turns the user's mental model to a concrete one. The code in the Figure 9 is a Bubble Sort algorithm. The user has constructed a behaviour for this algorithm. The behaviour contains several small pieces of commands,

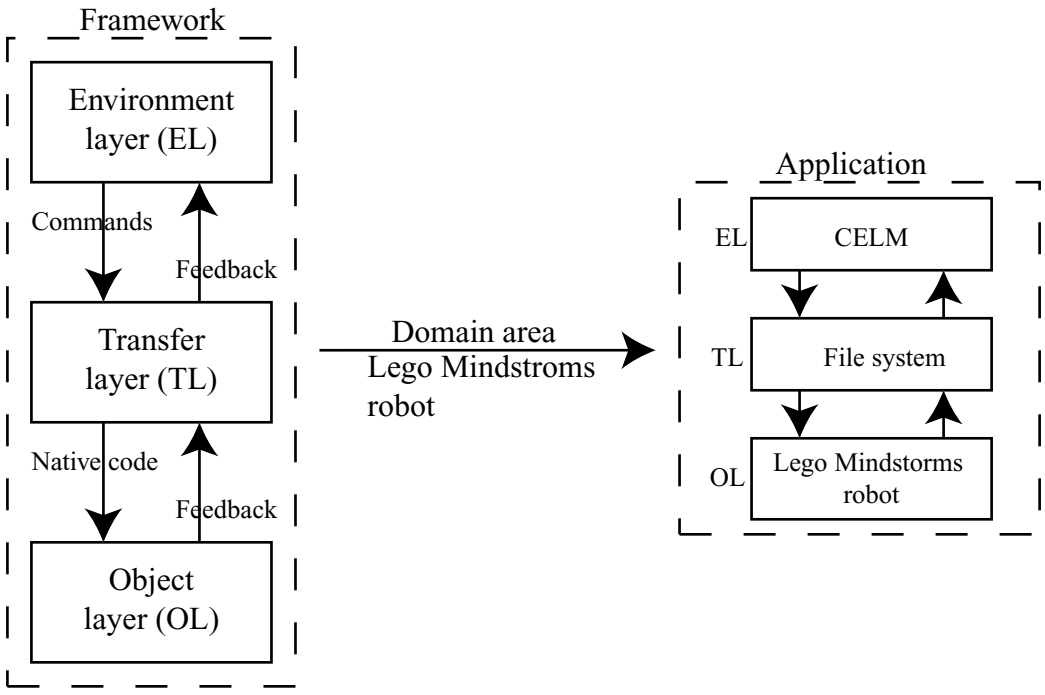

Figure 8: The relation between the framework and the application.

that will be sent to the robot by the application. The robots then concretize the algorithm according to the movements the user has defined.

4.2 Role-based concretization

Role-based concretization is a novel way to define concretizations with robots. The concept is based on idea that the data of a program or an algorithm has a certain *role*. It has been found that the following list of roles of variables covers 99% of all variables in novice-level programs (Sajaniemi, 2002): constants, stepper, follower, most-recent holder, most-wanted holder, gatherer, one-way flag, temporary and organizer. With these roles, it is possible to define a representation for each variable in the program. Plan-Ani (Sajaniemi and Kuittinen, 2003) is a tool for representing roles with graphic visualization. In addition to images, Plan-Ani uses animation for visualization as well.

However, in this application we see the concept of role in a different way. González (2004) has defined two roles for concretizing sorting algorithms: left and right. This is

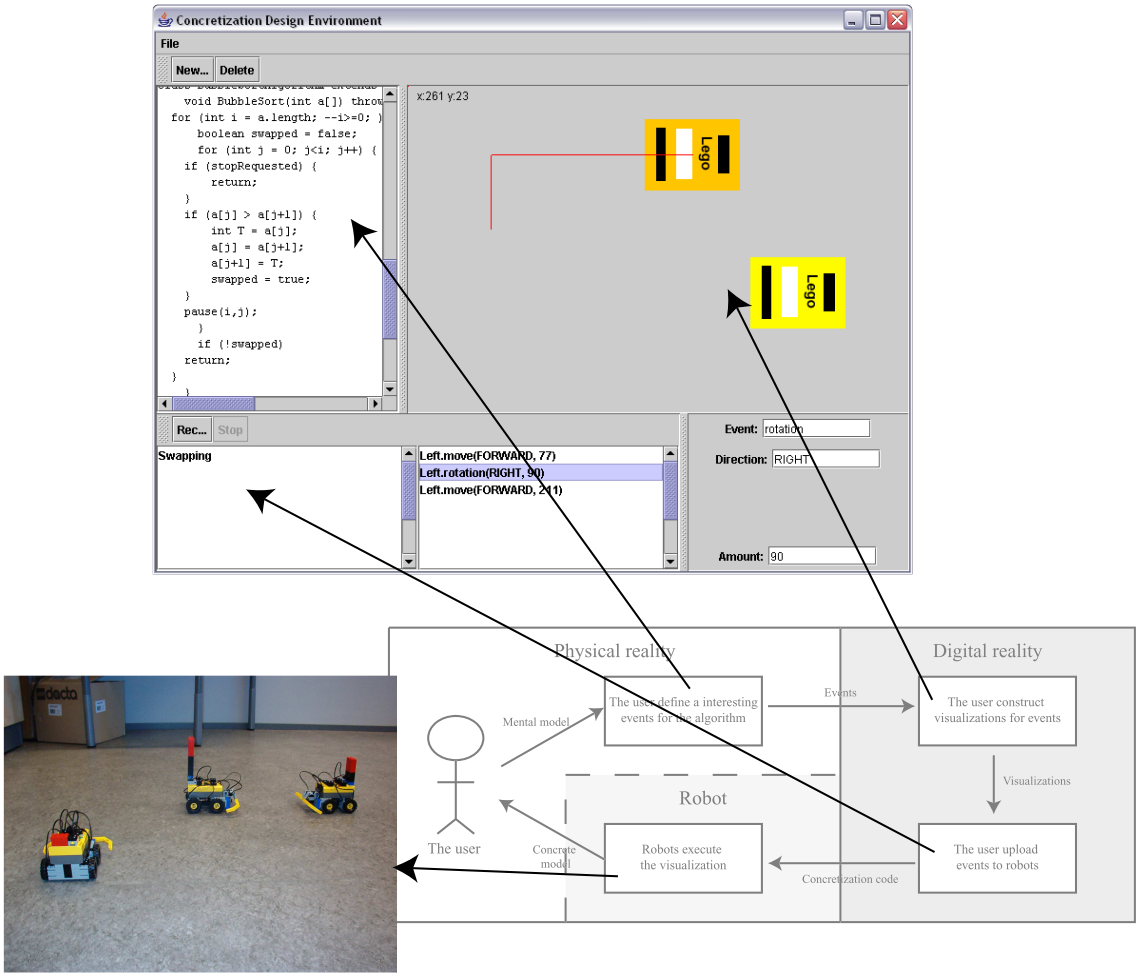

Figure 9: Mental model turns to the concrete one with the application.

based on the idea that, in the sorting algorithms, the data to compare can be illustarated as it has been presented in the Figure 10. In the figure, each square represents an item to sort. Typically, this item is a data in the cell of an array or in other data structure in a program. In the concrete world, for example, a robot represents this item. Items to sort are in one row and items to compare have been taken out from the row. The role is defined based on physical position of the robot or other object. In Figure 10, `Item 2` has the role *left* and `Item 5` has the role *right*.

The difference between González's roles and the roles in Sajaniemi (2002) is that the role in González (2004) is defined by the object's physical position related to other objects, whereas the role in Sajaniemi (2002) gets its definition from the semantics of the program. Therefore role-based concretization could be *location-based* or *position-based*

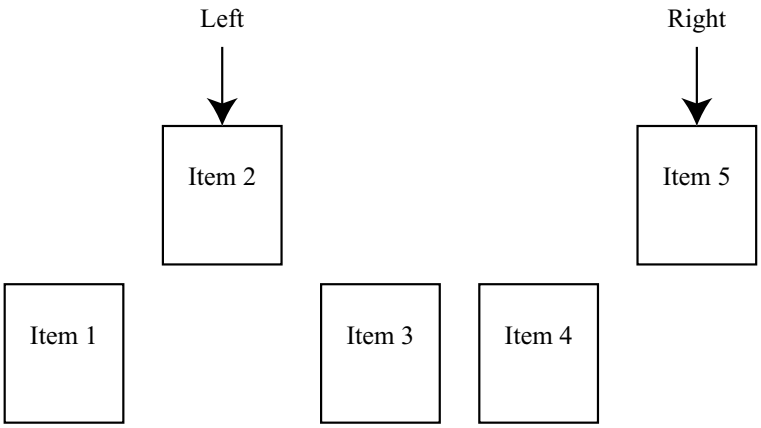

Figure 10: Roles left and right are defined by their physical position.

concretization as well. However, this terminology needs more studies, so from now on in this thesis I use the term *role-based concretization* for the approach I present in this thesis.

In role-based concretization, the user defines some movement or other action for the robot or other concretization tool. During the execution of the algorithm, this role is invoked when needed. For example, González (2004) has implemented roles as NQC sub-routines. The master robot leads the execution of concretization and it tells the slave robots which role to adopt (see Chapter 4.6 for more information).

In this approach, the user creates concretizations for these roles with the application. For common sorting algorithms (Bubble Sort, Selection Sort) the user has to define four different movements (Table 4). The environment produces the complete LeJOS-code which is needed for moving robots based on these movements. In this implementation, one role is implemented as one method. Movements `R1` and `R2` will be inserted to the method `right_behaviour`, which implements the `right` role. In the same way, movements `L1` and `L2` will be inserted to the method `left_behaviour`, which implements the `left` role.

Table 4: Movements the user has to define for sorting algorithms.

Movement	**Role**	**Swapping**
R1	Right	No
R2	Right	Yes
L1	Left	No
L2	Left	Yes

4.3 Constraints on CELM

This thesis contains a description of the framework for an application discussed in Chapter 3. The implementation of the application is described in this chapter. However, it is important to note that the application does not implement all the features I have described in Chapter 3. Because of the problem with loops, I decided that the application should be targeted only for concretization of sorting algorithms (see Chapter 3.6 for more information about the difficulties with loops). Also, recursive sorting algorithms (for example Quick Sort) are not included in this study because

1. They are not so important for novices, and
2. It is difficult to implement a recursive visualization or concretization.

These limitations change the generic workflow. The application itself makes it possible to work with sorting algorithms only, but not in as generic way as it has been described in Chapter 3.3.

4.4 Environment layer

The environment layer contains an application which is dedicated to the control of one or more robots. With the application, the user can construct movements and other behaviours for robots, and send them to the robot. The application has been implemented purely

with Java, so it can be used in diverse platforms, such as Windows, Linux or Macintosh. The only requirement is the need for a LeJOS environment, which is used in the transfer layer to compile the code produced by the environment layer to the native code of the object layer. It is also possible to produce any layer of the framework with some other programming language. The most important issue is to ensure that the layers give output in the right format. Also, each layer has to have the capability to use the output of the previous level as its input. Figure 11 presents the part of the workflow which belongs to the environment layer.

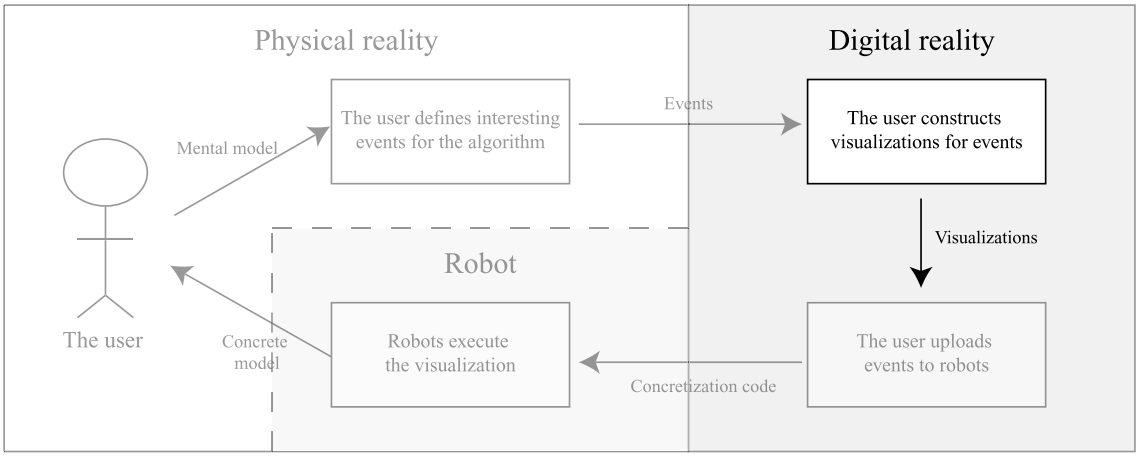

Figure 11: The part of the workflow which belongs to the environment layer.

The user interface of the application (Figure 12) allows users to interact with the robots presented on the screen in the normal way that user use computers (such as a drag-n-drop and a context menu from the rightmost mouse button).

The program has been implemented with Java using an object-oriented approach. This means, for example, that it is easy to replace a robot with another one. The shape of the robot can be replaced just by replacing the existing picture with new one. However, it is not very difficult to extend the behaviour of the robot because a single class represents the robot. Table 5 presents the most important classes which belong to the environment layer, and the purposes for them. There are also a few more classes which are not mentioned here because they are only for constructing the graphical user interface (GUI) of the application.

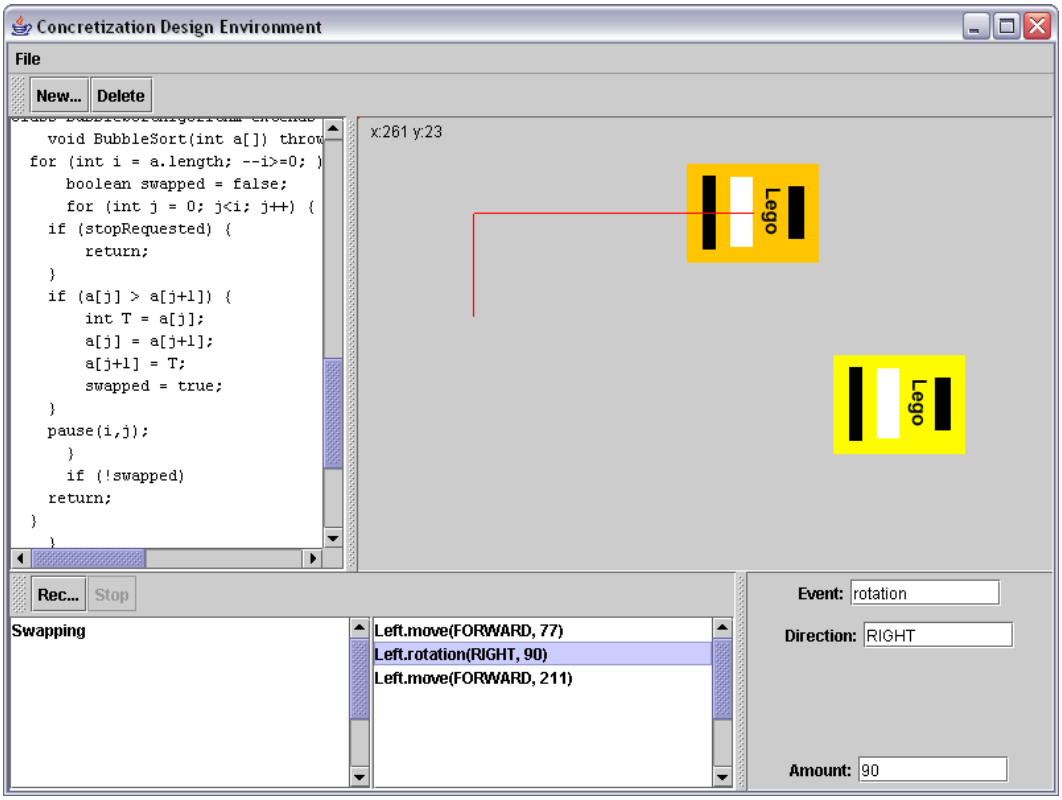

Figure 12: The user interface of the application.

Table 5: Java classes that construct the application.

Class	**Purpose**
Behaviour	Represents a single behaviour of the robot. Contains a vector where instances of the `Movement` class are stored
Movement	Contains one single movement (moving, turning, sending etc.)
RoboData	Contains the information about the single robot
Robot	Represents the robot. Each robot has its own instance of this class
RoboPanel	Receives user interactions with the robot and delivers them forward if necessary
Robots	Contains all robot objects

4.5 Transfer layer

The transfer layer contains those part of the framework which are needed to transfer the code produced by the environment layer to the object layer. This layer can contain programs, data transfer devices (network, hard drive) and different protocols for them. The transfer layer of the CELM application contains parts of the program and an infrared transmitter which is connected to the computer. Figure 13 illustrates the transfer layer as a part of the general workflow of CEF.

In the application, I have used Lego Mindstorms robot kits. The kit contains a RCX unit (see Chapter 4.6 for more information), which uses infrared for communication. By using this infrared techinique, the user uploads the programs to the robots which the user has produced on the computer. The infrared transmitter can also work as a receiver and, in this way, the robots can communicate and give feedback about their states to the application. There are some advantages and disadvantages to this approach: one advantage is that robots can work independently and move freely because there are no wires which connect them to the computer. On the other hand, a disadvantage is that the infrared link is rather slow and there might be problems during the communication if the robot and the infrared transmitter are far away from each other. Also, the concurrent communcation produced by several robots can be a problem for the application. Therefore it is a key issue also to develop a strong enough protocol for this layer.

There is particular default firmware[1] in the RCX unit. This firmware also can be replaced in order to get more control or other features to the device. The default firmware of the RCX unit can be replaced for example with LeJOS, which is a JVM (Java Virtual Machine) for the RCX unit (LeJOS, 2004). The LeJOS firmware offers the programmer a more rich API than the default firmware. The LeJOS interface implements a subset of features of the standard JVM (Java 2 Standard Edition, Sun Microsystems (2004b)), as does J2ME (Java 2 Micro Edition, Sun Microsystems (2004a)).

The workflow of the transfer layer is as follows:

[1] Firmware is like an operating system and it can be replaced with other firmware.

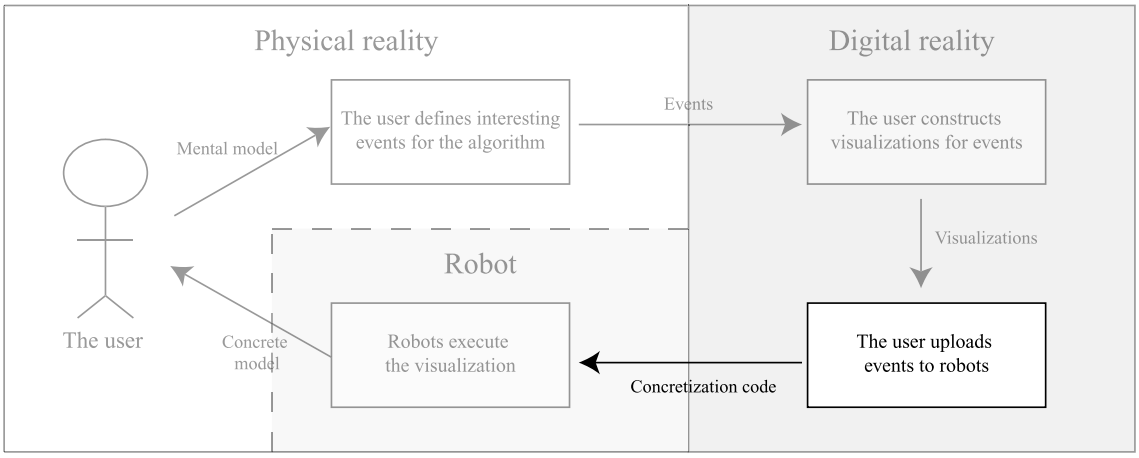

Figure 13: The part of the workflow which belongs to the transfer layer.

1. The user constructs the movement of robot within the CELM application.
2. The user compiles the code to the LeJOS code. In practice, this is done by passing the code to the `CodeGenerator` class which interprets the code.
3. The application compiles the LeJOS-code to the Java bytecode which the LeJOS environment sends to the robot via an infrared device.

Code interpretation is needed to run the CELM code in Lego Mindstorms robots. This interpretation is done by the `CodeGenerator` class of the application. An interface of the class is follows:

```
public CodeGenerator ()
public String Generate (Vector m)
```

The constructor does not take any argument. Instead, it creates an empty instance of the class. `Generate` method takes a vector as an argument. This vector contains instances of the `Movement` class, which contains information about the action made by a single robot (type, direction and amount). The `CodeGenerator` class interprets these movements to the LeJOS code, and the method returns this code as a string. This string will be saved to file and then sent to the robot by the LeJOS environment.

For example, the user constructs the code presented in Listing 8 with the CELM appli-

cation. The `CodeGenerator` class interprets it to LeJOS code. In this example, the robot's name is A, and its internal id is 1. The robot's weight is 20.

```
1 A.move(FORWARD, 132)
2 A. rotation (RIGHT, 90)
3 A.move(FORWARD, 141)
```

Listing 8: The original CELM code.

These commands are converted into LeJOS code (the complete code is presented in the Appendix 1). The most interesting lines of the code are presented in Listing 9.

```
1 nav. travel (132);
2 nav. rotate (−90);
3 nav. travel (141);
```

Listing 9: The CELM code has been turned to the LeJOS code.

In this implementation, a `TimingNavigator` class is used (`nav` is an instance of this class). The class is a part of the LeJOS class library, and it contains methods for performing basic navigational movements (LeJOS API, 2004). This code is compiled to Java bytecode and that code is then sent to the object layer.

4.6 Object layer

The object layer is the part of the framework which makes the visualization concrete. It is possible to use different robots or other concrete objects in this layer. In our research group, we have used Lego Mindstroms robots for many years (see Jormanainen et al. (2002)). Therefore, it was easy to decide which technique to use at the object layer of our application. Figure 14 presents these robots as a part of the workflow of the framework.

The name of the application, CELM, came from Lego Mindstorms actually. The history of Lego Mindstorms goes back to 1986 when a research group supervised by Seymort Papert and Mitchel Resnick, started to develop *Programmable Brick*, a small unit capable

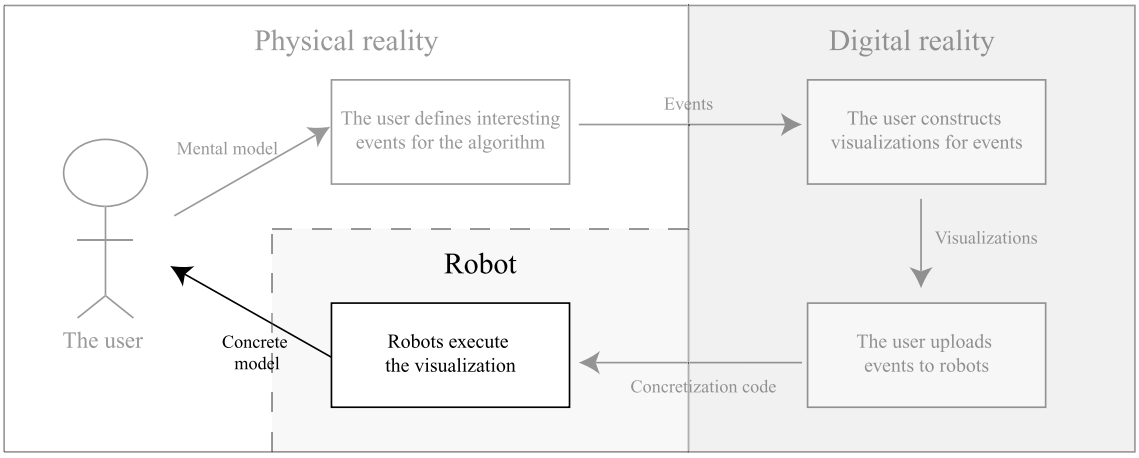

Figure 14: The part of the workflow which belongs to the object layer.

of connecting to the external world through a variety of sensors and actuators. The brick was designed for the creation of robots and other applications in which a computer might interact with everyday objects (Laverdae, 2001).

With Lego Mindstorms kits, it is possible to construct an independent and autonomous robot. The Mindstorms kit includes a RCX unit (Robotic Command Explorer), which is the core of the kit. The RCX unit is flexible, because it can be programmed by using many programming languages (NQC, Java, Visual Basic etc.). The Lego Mindstorms package also contains a programming environment, the *RCX Code*, which is a RCX specific programming language. Figure 15 shows the RCX unit.

The RCX unit contains the following parts (Laverdae, 2001):

- **Processor:** 8bit Hitachi H8/ 3292 microprocessor, running a 16 MHz CPU speed
- **ROM:** (Read Only Memory) 16 Kb
- **SRAM:** 512 bytes on chip, 32 Kb external
- **Outputs:** 3 motor ports, 9V 500 mA
- **Inputs:** 3 sensor ports
- **Display:** LCD display

- **Timers:** Four system timers (8bit)
- **Batteries:** 6x 1.5V
- **Communications:** IR port (transmitter + receiver)

These parts compose an individual computer which has a simple structure. Figure 16 presents the block diagram of the RCX unit (Laverdae, 2001).

With this hardware, it is possible to produce autonomous robots, which can also communicate, during execution, with a computer or with each other via an infrared device. However, the communication is rather slow and there is a need for a strong protocol in order to ensure that the communication is reliable (González, 2004). In the CELM application, slave robots realize the concretization independently. This means that they do not receive any messages from the computer. Instead, slaves communicate with each other

Figure 15: The RCX unit from Lego Mindstorms Robotic Invention System kit.

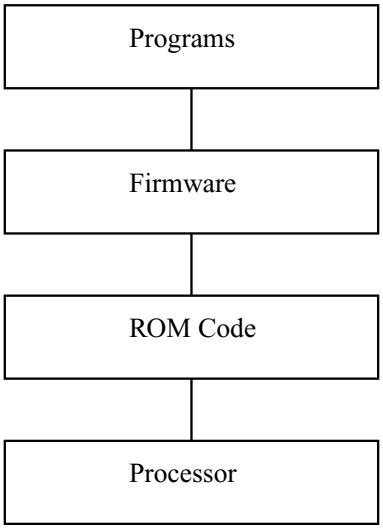

Figure 16: The logic structure of RCX unit (Laverdae, 2001)

and with the master robot. Figure 17 shows robots during the concretization of a sorting algorithm. In the figure, two robots are just exchanging messages in order to find out which one is bigger and whether they should change places or not.

Figure 17: Robots communicating with each other during the execution of a sorting algorithm.

Lego Mindstorms robots communicate with each other via an infrared transmitter/receiver. Therefore, there might be some problems during the communication. For example,

a message can get lost if robots can not "see" each other (the infrared transmitter/receiver is located in the front of the RCX unit. It is the black part of the unit in Figure 15). Furthermore, only one robot is allowed to send a message at one time (González, 2004). To avoid these problems, a well defined protocol[2] must be used for communication between robots. The protocol in this application ensures that robots get the messages and that the message sending will happen in the right order. The complete communication protocol for the Bubble Sort algorithm implemented in González (2004) is defined in Figure 18.

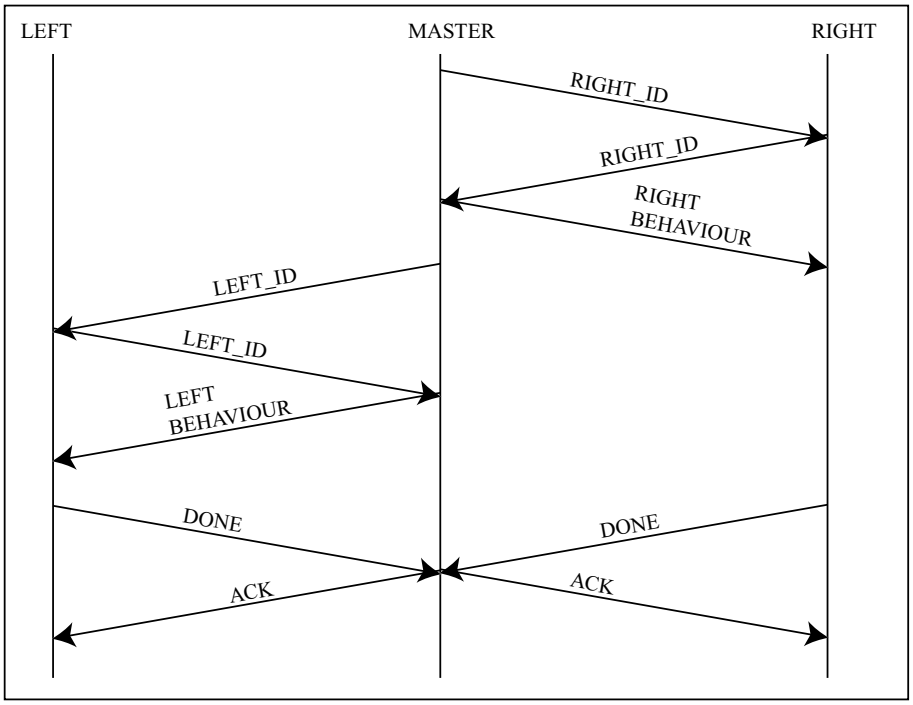

Figure 18: The complete protocol for the Bubble Sort algorithm (González, 2004).

The original work by González (2004) is based on message passing between master and slave robots. The system has been implemented with NQC. The communication protocol is simple: the master contacts a slave and send the slave's role (left or right) to it. After the slaves have done their work, they send a confirmation. Between these steps, there are of course some defined messages which help the robots to be sure that all the messages have passed. The protocol can be defined as follows:

1. The master sends a message containing the slave's `ID` to the first slave.

[2]Protocol is a pre-defined set of rules, which all parts of communication must accept and use.

2. The slave confirms this message by answering the master with its own `ID`.

3. The master sends the slave a message which contains information about the slave's role (whether it is `LEFT` or `RIGHT`).

4. The slave makes the movement that has been defined in the behaviour corresponding to the message.

5. The slave sends a `DONE` message, which shows that it has done the work, to the master.

6. The master sends an `ACK` (confirmation) signal to the slave.

In the protocol, it has been decided that the master will send an id and a behaviour to the right slave first. After that, the master sends the same information to the left slave. There is no defined order in which to retrieve the `DONE` message, because the execution speed of the algorithm may vary depending on the robot.

When the slaves have started their execution, they make pre-defined movement to certain locations defined by the user. After moving, these robots start to compare their data with each other. In Figure 17, the robot in the center and the robot on the right are sending messages to each other. The part of the protocol, in the case that the robots are not going to swap their places (the left robot's weight is smaller than right's weight), is presented in Figure 19.

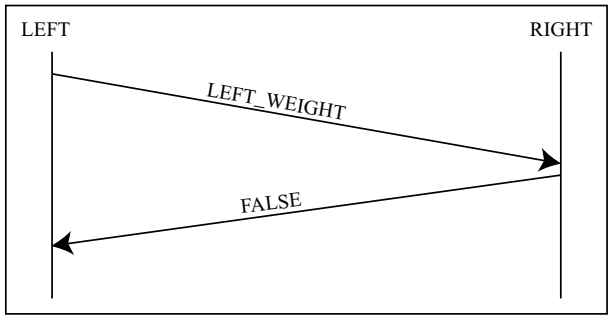

Figure 19: The part of the protocol which is used to compare the weight of the robots (González, 2004).

Furthermore, if the left robot's weight is bigger than right's weight, robots exchange their ids (see Figure 20). After that, the robots change places by using the pre-defined movement.

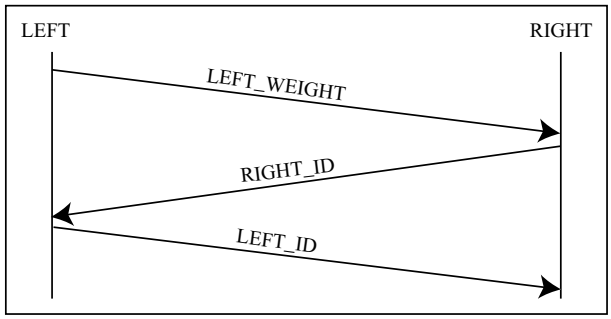

Figure 20: The part of the protocol which is used to compare the weight of the robots (González, 2004). With this protocol, robots are going to change their places.

5 Evaluation and future directions

To assess the potential usefulness of the approach and to get an objective view of its merits, limitations and prospects, I made a questionnaire in which I asked for opinions and suggestions about the application and the framework. The group ($n = 6$) completed the questionnaire during the Educational Techonology Summer School in August 2004 at the Mekrijärvi Research Station, Finland. The members of the group were participants in the *Kids' Club track* of the summer school; they all were CS educators and they had experience in the field of educational techonology. Questions were in paper form and they were answered as part of track activities. The following questions were used:

1. Mention three issues where the application could help one when learning an algorithm by concretizing it with the robotics.
2. By the term *role* we refer to... How do you understand the concept of role in the context of this application? Could you come up with some other term for this concept?
3. Suggest one idea in which direction this project could continue (techniques, approaches etc.).
4. Mention one other application area which may use this concretization approach and the framework.

According to the answers, the application and robotics can be used to illustrate abstract concepts, such as sorting and searching algorithms especially when teaching children or novices. Furthermore, participants believed that the explicit nature of the robotics gives understanding of what is happening in a clear way. However, according to the answers it is not so clear that the concrete objects in the real world makes it easier to really learn an abstract concept like algorithm. At least, the way in which to use robotics needs to be planned in more detail.

According to the answers, the concept of role can be used with this approach. However, the term should be defined more clearly. For example, terms right and left are suitable for sorting algorithms, but it is not possible to use them in general. Also, it is not so clear whether the robotics can be utilised as comprehensively as visualizations for example in Jeliot.

Also, some directions for future developement were presented in answers. The usage of the application as a first programming tool for children could be an interesting approach. The approach might be useful also when programming industrial robots. The combination of this approach and Empirical Modelling (Roe, 2003) is definitely an interesting approach. To expand the power of the expression of the robots, it might be a good idea to see the robots as agents with a rich set of actions and allowed interactions.

Some very concrete suggestions were also presented in the answers. For example, an important feature to add to the application is a time scale. This feature allows user to examine the position of the robot at a certain moment of time. Furthermore, according to the answers, there is a need for implementation of bi-directional communication between layers. This communication allows the robots to send data to the application (e.g. positions at the certain moment of time). The framework defines this feature, but the CELM application does not implement it.

6 Conclusion

The concept of concretizing algorithms with the robotics is based on research done at the Department of Computer Science, University of Joensuu (González 2004; González et al. 2004). However, it has been found that it is hard to produce even a simple concretization because of the low level implementation that robots need. Therefore there is a need for a framework and for an application which a user (for example a teacher or other instructor) can use to produce concretizations. The research questions I had in this thesis, were:

1. What should be the general characteristics of a framework to support algorithm concretization by visualization?
2. What kind of architecture should the framework have in order to be used in diverse concretizing platforms such as Lego Mindstorms or EK Japan Co., Ltd.'s Soccer Robo® 915?
3. What kind of additional features does such a framework facilitate?

To answer the first and the second question, I used concept implementation as the research methodology. I developed a framework which makes it possible to use diverse platforms in an easy way. The framework is based on *layers* which can be replaced with another one. Layers are:

1. **The environment layer:** Takes care of the communication between the user and robots.
2. **The transfer layer:** Takes care of transferring and converting the code produced by the environment layer to the object layer.
3. **The object layer:** Represents physical objects which concretize the algorithm. These objects might be, for example, Lego Mindstorms robots.

For example, Lego robots in the object layer can be replaced with some other robot kit. The only requirement is that these robots understand the code which the transfer layer

gives to them. When replacing the object layer, it might be necessary also replace the transfer layer, or part of it. However, one can replace the whole transfer layer or part of it without changing the object layer or the environment layer at all.

To achieve full support for these features, definitions for interfaces between layers must be developed further. Especially, interfaces from the object layer to the environment layer (via the transfer layer) have to be developed carefully in the future. For this thesis, I decided on an interface from the environment layer to the transfer layer. This interface contains codes which the application at the environment layer has to produce. The transfer layer has to have the capability to transfer this code to the native code for robots (or other objects) at the object layer.

In this thesis, I have developed a novel concept, *role-based concretization*, which could be a topic for further studys. The schema for roles of variables presented in Sajaniemi (2002) could be an interesting research topic. These roles could also be implemented for robotics: it would also be interesting to study whether roles (used with robotics) positively affect the learning process, as it has been found in Sajaniemi and Kuittinen (2003).

To answer the third question and to get a view of the concept of *role*, I conducted a questionnaire. Six researchers in computer science education answered the questionnaire. Answers indicated that the concept *role* can be used with this approach. However, some doubts about the approach were presented. Answers indicated also some ideas for future development concerning the framework and the application.

In this thesis, I have presented one possible application (CELM) for the CEF framework. Another possibility for the application could be use as a collaborative tool, which allows several student to work with the same concretizations over the Internet. For example, a user could produce a concretization for a robot, and then the user could share or upload it to network. Other users could download the concretization and use it with their own robots. From a techical point of view, the implementation of this kind of application could use a client-server model or peer-to-peer networking for communication between users. Furthermore, the application in the environment layer could allow users to work with the same concretization in collaboration with each others.

One really interesting approach could be to combine concretization with robotics and Empirical Modelling (EM). The EM approach has been developed at the University of Warwick since 1983 by Dr. Meurig Beynon. The Empirical Modelling is an approach for constructing computer based models that can assist in the understanding of a phenomenon. The approach has an emphasis on experiment, observation and interaction during the developement process (Roe, 2003). For example, a simulator that could observe the behaviours of robots, based on EM, could be a very interesting topic for further study.

References

Baecker, R., 1981. Sorting out Sorting. Videotape, 30 minutes, presented at ACM SIGGRAPH ’81 and excerpted in ACM SIGGRAPH Video Review #7.

Barnes, D. J., 2002. Teaching introductory Java through LEGO MINDSTORMS models. In: Proceedings of the 33rd SIGCSE technical symposium on Computer science education. ACM Press, pp. 147–151.

Baum, D., Baum, D., Gasperi, M., Hempel, R., Villa, L., 2000. Extreme Mindstorms: an Advanced Guide to LEGO MINDSTORMS. APress.

Ben-Ari, M., 1998. Constructivism in computer science education. SIGCSE Bulletin 30 (1), 257–261.

Ben-Ari, M., Myller, N., Sutinen, E., Tarhio, J., 2002. Perspectives on program animation with Jeliot. In: Software Visualization: International Seminar. Lecture Notes in Computer Science 2269. Dagstuhl Castle, Germany, pp. 31–45.

Brown, M. H., 1988. Perspectives on algorithm animation. In: Proceedings of the SIGCHI conference on Human factors in computing systems. ACM Press, pp. 33–38.

Demetrescu, C., 2001. Leonardo IDE: C Compiler and Software Visualization System. WWW-page, `http://www.dis.uniroma1.it/~demetres/Leonardo/` (Accessed 2004-06-17).

Demetrescu, C., Finocchi, I., Stasko, J. T., 2002. Specifying Algorithm Visualizations: Interesting Events or State Mapping? In: Diehl, S. (Ed.), Software Visualization State-of-the-Art-Survey. Vol. LNCS 2269 of Lecture Notes in Computer Science. Springer-Verlag, pp. 16–30.

Duffy, T. M., Cunningham, D. J., 1996. Constructivism: Implications for the design and delivery of instruction. In: Jonassen, D. H. (Ed.), Handbook of Research for Educational Communications and Technology. pp. 170–198.

Eronen, P. J., Silander, P., Sutinen, E., Virnes, M., 2004. Keksivä oppiminen ja fyysiset oppimisaihiot teknologiaympäristön rikastuttajina (in Finnish). Symposium presentation at ITK '04 conference, abstract: `http://www.hameenkesayliopisto.fi/itk04/eronen_etal.html` (Accessed 2004-08-31).

Fagin, B., Merkle, L., 2003. Measuring the effectiveness of robots in teaching computer science. In: Proceedings of the 34th SIGCSE technical symposium on Computer science education. ACM Press, pp. 307–311.

Ferrari, M., 2001. Building Robots with Lego Mindstorms. Syngress Publishing, Rockland, MA, USA.

Fleischer, R., Kucera, L., 2002. Algorithm Animation For Teaching. In: Software Visualization: International Seminar. Lecture Notes in Computer Science 2269. Dagstuhl Castle, Germany, pp. 113–128.

González, J. L., 2004. Software Visualization with Lego Mindstorms. Master's thesis, University of Joensuu, Department of Computer Science.

González, J. L., Myller, N., Sutinen, E., 2004. Sorting out sorting through concretization with robotics. In: Proceedings of the working conference on Advanced visual interfaces. ACM Press, pp. 377–380.

Jeliot 3, 2004. Jeliot 3 - BlueJ extension. WWW-page, `http://cs.joensuu.fi/jeliot/downloads/bluej.php` (Accessed 2004-08-13).

Jormanainen, I., Kannusmäki, O., Sutinen, E., 2002. IPPE - How to Visualize Programming with Robots. In: Ben-Ari, M. (Ed.), Second Program Visualization Workshop. HornstrupCentert, Denmark, pp. 69–73.

Kölling, M., Quig, B., Patterson, A., Rosenberg, J., 2003. The BlueJ system and its pedagogy. Journal of Computer Science Education, Special issue on Learning and Teaching Object Technology 13 (4), 243–247.

Korhonen, A., 2003. Algorithm Visualization and Simulation. Ph.D. thesis, Helsinki University of Technology.

Laverdae, D. (Ed.), 2001. Programming Lego Mindstorms with Java. Syngress Publishing, Rockland, MA, USA.

LeJOS, 2004. leJOS, Java for RCX. WWW-page, `http://www.lejos.org` (Accessed 2004-06-15).

LeJOS API, 2004. LeJOS API documentation. WWW-page, `http://www.lejos.org/apidocs/` (Accessed 2004-06-16).

Miglino, O., Lund, H. H., 1999. Robotics as an Educational Tool. Journal of Interactive Learning Research 10 (1), 25–47.

Moreno, A., Myller, N., Sutinen, E., 2004. Collaborative Program Visualization with Woven Stories and Jeliot 3. In: Proceedings of the IADIS International Conference on Web Based Communities. pp. 482–485.

Myller, N., 2004. The Fundamental Design Issues of Jeliot 3. Master's thesis, University of Joensuu, Department of Computer Science.

Petre, M., 1995. Why Looking Isn't Always Seeing: Readership Skills and Graphical Programming. Communications of the ACM 38 (6), 33–44.

Poon, J., 2000. Java meets teletubbies: an interaction between program codes and physical props. In: Proceedings of the Australasian conference on Computing education. ACM Press, pp. 195–202.

Price, B. A., Baecker, R. M., Small, I. S., 1993. A Principled Taxonomy of Software Visualization. Journal of Visual Languages & Computing 4 (3), 211–266.

Roe, C., 2003. Computers for Learning: An Empirical Modelling perspective. Ph.D. thesis, Department of Computer Science, University of Warwick.

Sajaniemi, J., 2002. An Empirical Analysis of Roles of Variables in Novice-Level Procedural Programs. In: Proceedings of IEEE 2002 Symposia on Human Centric Computing Lanuguages and Environments (HCC'02). IEEE Computer Society, pp. 37–39.

Sajaniemi, J., Kuittinen, M., 2003. Program animation based on the roles of variables. In: Proceedings of the 2003 ACM symposium on Software visualization. ACM Press, pp. 7–16.

Stasko, J., September 1990. Tango: A framework and system for algorithm animation. IEEE Computer 23 (9), 27–39.

Stasko, J., 1992. Animating algorithms with XTANGO. ACM SIGACT News 23 (2), 67–71.

Stasko, J., Badre, A., Lewis, C., 1993. Do algorithm animations assist learning?: an empirical study and analysis. In: Proceedings of the SIGCHI conference on Human factors in computing systems. ACM Press, pp. 61–66.

Sun Microsystems, 2004a. Java 2 Platform, Micro Edition (J2ME). WWW-page, `http://java.sun.com/j2me/index.jsp` Accessed 2004-06-15.

Sun Microsystems, 2004b. Java 2 Platform, Standard Edition (J2SE). WWW-page, `http://java.sun.com/j2se/index.jsp` Accessed 2004-06-15.

Sutinen, E., Tarhio, J., Lahtinen, S.-P., Tuovinen, A.-P., Rautama, E., Meisalo, V., 1997. Eliot – an Algorithm Animation Environment. Report A-1997-4, Department of Computer Science, University of Helsinki, Helsinki, Finland, `http://www.cs.helsinki.fi/TR/A-1997/4/A-1997-4.ps.gz`.

Yehezkel, C., 2002. Visualization of computer architechture. In: Second Program Visualization Workshop. HornstrupCentert, Denmark, pp. 113–117.

Appendix 1: The complete LeJOS code

```
1  import josx.platform.rcx.*;
2  import josx.util.*;
3  import josx.robotics.*;
4
5  public class Robot1 {
6      public static byte id; // robot's identifier
7      public static byte weight; // robot's weight
8      public static float speed; // robot's linear speed
9      public static float rotation_speed; // robot's rotation speed
10     private static TimingNavigator nav;
11
12     Robot1 (byte id, byte weight, float speed, float rotation_speed) {
13         this.id = id;
14         this.weight = weight;
15         this.speed = speed;
16         this.rotation_speed = rotation_speed;
17         nav = new TimingNavigator( Motor.C,
18             Motor.A, speed, rotation_speed );
19         nav.setMomentumDelay ((short) 95);
20     }
21     public static void main (String [] args)
22     throws InterruptedException {
23         Robot1 robot = new Robot1 ((byte) 1, (byte)20, 7.8f, 6.55f);
24         Motor.A.setPower (3);
25         Motor.C.setPower (3);
26         nav.travel (132);
27         nav.rotate (-90);
28         nav.travel (141);
29         nav.stop ();
30         Sound.systemSound (true, 5);
31         TextLCD.print ("END");
```

```
32          }
33  }
```

Appendix 2: Answers from the Evaluation

Questions:

1. Mention three issues where the application could help one when learning an algorithm by concretizing it with the robotics.
2. By the term *role* we refer to... How do you understand the concept of role in the context of this application? Could you come up with some other term for this concept?
3. Suggest one idea in which direction this project could continue (techniques, approaches etc.).
4. Mention one other application area which may use this concretization approach and the framework.

Answers:

Answer 1

1. Helping to expose the operational interpretation of algs, i.e. concept of a virtual machine; giving access to the internal state of the executing algorithms, to assist debugging; promoting the awareness of the significance of observation in understanding algorithms
2. Role is a good term, provided that it is understand as context-dependent. The theatre analogy helps, but the notion of role you invoke is much more specific than "playing a particular character". Perhaps phrases like "role in sorting / in the exchange/comparison" might help.
3. Definitely an interesting link with Empirical Modeling. Several ideas here; not all in scope of short-term research. Possibilities:
   (a) extend the EM bubblesort

(b) take model further with EM heapsort

(c) try to make a more general study relating capabilities of agents to algs they can concretize

4. Other sorting algorithms (as above) variations - Showing concurrent/autonomous sorting; playing games, eg. variations of noughts-&-cross.

Answer 2

1. Difficult issues, when someone needs some concretizing. This is useful maybe to young childrens (understanding of a difficult structure that is not a part of their everyday knowledge); Planning of a program = designing a program; algorithm by using the application = no need to write a code; testing of algorithms that are not able to do with a real life robots; Learning of concepts of programming, but this is needed to plan more detailed... how to do it?

2. Role of robot is related to the robot's behavior and robot's role in an algorithm... I'm thinking... :) actually the role can be both static and dynamic, but it depens on whether you think a role as an actor or operator (onkohan tämä oikea termi?) = like an independent robot doing a thing as an actor or a task that a robot is doing. In this case "a role" is the task. Siis kohdistuu joko robottiin tai toimintoon.

3. Could a robot write a code or a pseudocode by moving a robot by a player/programmer? Or is it already doing so? Like programming by building, but you'll get a code (I-Blocks does not give a code but a result). Can robot do the same as concretizing "animation" = trasferring the behaviour of the "virtual robot" to a real robot?

4. Could the application be used for learning of concepts of programming? Well, this is maybe not a answer to the question... :)

Answer 3

1. First of it helps one to understand that basic algorithms are executing one command after another; It shows the connection between the abstract command and the concrete action it triggers; Perhaps playing wih robots in concrete world could be directed back to the application, which would show those acions animated with information about the actors (ID, the weight of the variable etc.) <= kind of like vice-versa -approach.

2. You have used the right and the left, which are suitable for concretizing algorithms like sorting. But it is an another question, wheter we can have robotsics in a role of "general concretization tool" similar to Jeliot? Perhaps, robots carrying displays (PDA's or small LCD's?) with changing images, i.e. picture of roles, could serve there being actors on the stage in the reality.

3. I find it interesting to use this application as a first tool for children to a kind of animations in the real world. They could orchestrate movements of several small robots to create for example robot ballet. I would also suggest including a time scale to a program, where you could see the position of robot at certain moment of time.

4. I would think this would be VERY fruitful, if combined to a model of real environment presented in scale. Then you could for example program industrial robot (robots by moving them and seeing how do they behave in that model in an environment with a number of other similar robots. NOTE: Answers (3) and (4) are a kind of combined idea from animating of a model to concrete world movement without a programming phase in between. Programming by animating. It would be more than shear visualization. It should also be two-directed from animation to a real world movement and from real world movement to animated model.

Answer 4

1. The explicit nature of the robotics gives clear understanding of what is happening; Does the explicit nature make it easier to 'learn' the algorithm?

2. It seems that when the robots move to do their actions they are like actors in a play who come forward to tolk the stage for this 'act'. The 'performance' is the algorithm and each step is an 'act'.

3. Can you show the educationally benefit of the explicit nature of the robotics? Valdemar Setzer has some possibly relevant work on manual sorting by children using playcards.

4. Could you use robotics to illustrate mathematical concepts?

Answer 5

1. Visualization of abstract concepts (sorting/searching); able to explain in own language the algorithm (by explaining robotic actions); by storing convete perhaps will help later recall when trying to use algorithm in future.

2. Behaviour? Procedure?

3. Given a concrete model of an algorithm can students then derive a procedural algorithm, ie. after watching robots do a sort can they spot main features + code. Later on, do students remember the concrete model to recall the algorithm??

4. searching? Object interaction (communication); networking theory?

Answer 6

1. Visualizing the algorithm;

2. The type of move/action allowed to a given robot at a given point in time.

3. Maybe expand moving robots (with limited actions and degrees of freedom) to agents with a rich set of actions and allowed interactions.

4. Describing / visualizing the behaviour / performance of a system. For example, the process in an OS, or threads in a distributed system.