IT 11 058
Examensarbete 15 hp
Maj 2011

mJeliot - ICT Support for Interactive Teaching of Programming

Moritz Rogalli

Institutionen för informationsteknologi
Department of Information Technology

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

mJeliot - ICT Support for Interactive Teaching of Programming

Moritz Rogalli

Conceptual weaknesses and learning difficulties in early computer science courses have been identified in previous research. Several factors have been identified and addressed by earlier projects, leading to visualization frameworks and the postulation of active learning and the engagement taxonomy. High failure rates in basic computer science courses suggest that there is room and need for improvement in computer science teaching. This report describes a tool called mJeliot, that is developed to address this need. It combines visualization with theories in active and deep learning. mJeliot enables students to make predictions on their smart phones about code executing in the Jeliot visualization framework. The instructor can send out assignments to the audience. The audience can predict method parameter and return values on their phone and send their solutions back to the instructor. Individual feedback to every member of the audience is given based on his/her predictions and the instructor is presented with a statistical overview on the audience's result. The results enable both the audience and the instructor to reflect on them and ask questions directly. mJeliot aims to improve participation amongst the audience by reducing the threshold , encouraging members of the audience to train their predictive and intuitive capabilities. mJeliot works over any available network infrastructure, which connects a mobile application for the audience, a command- and visualization interface for the instructor and a server software. The server offers sessions called lectures to the clients and distributes assignments, predictions and results between them.

Handledare: Arnold Pears
Ämnesgranskare: Arnold Pears
Examinator: Anders Jansson
IT 11 058
Tryckt av: Reprocentralen ITC

Contents

Chapter 1

Introduction

1.1 Motivation

Teaching to program to novice students has been a topic of research for several decades [1, 2, 3, 4, 5] and the high percentages of failing students in basic programming courses suggest that teaching of programming still has to be improved. The goal of mJeliot is to contribute to that improvement. It provides a new way of interacting with novice programming students in classroom settings. It is built to help teachers to improve the students' experience in the classroom, improve their learning and to give teachers a direct feedback on the learning progress of their students. mJeliot is designed to be adaptable and easily deployable in most settings while keeping deployment and usage costs and at a minimum. The main cost factor for interactive tools like clickers is the need for specialized hardware. Furthermore are clickers limited to multiple choice questions. Today however most students carry a smart-phone capable of running applications during lectures. We utilize this fact to propose mJeliot, a system based on a mobile client [1] for mobile handheld devices and on Jeliot, a program visualization application.

1.2 Goal

The goal of mJeliot is to involve students interactively in lectures. A lecturer shall be able to distribute assignments to members of the audience, who run the mobile mJeliot client on their smart-phones or similar devices. Members of the audience shall be able to send in a prediction of the future state of method parameters and return values back to the lecturer. The lecturer shall then be able to analyze the result together with the audience and every participating member of the audience is presented with a personal result. The system shall be easy to use and provide a transparent user interface avoiding the need to configure network parameters. There shall be no need to register on the server before lectures. The client devices of choice for the project are smart-phones because they have become more and more popular and many students have their smart-phones activated during lectures anyway. The software shall be

[1] For more information and current releases see the mJeliot website [7]

easily deployable to maximize student participation. The system shall also be usable without the need to install new hardware in lecture halls, without special knowledge in networking and shall be rapidly deployable in any location. The overall goal is a cost effective, simple and extend-able architecture which is accessible for as wide an audience as possible.

The scope for this thesis project is not to create a ready-to-deploy platform but to create a simple working prototype that can be used to test the system and its concept in a test session with students to get input, experiences and ideas for the future development. It is not meant to be feature-complete and not meant to work on all devices or under all circumstances. The parameters for a test session are a working network infrastructure, a few Android-based devices, a presentation device running both the server and the lecturer's command interface and a few students to participate.

1.3 Jeliot

Jeliot [6] is a tool that visualizes Java program code execution. It is developed at the University of Helsinki, Finland and the Weizmann Institute, Israel [6]. Jeliot animates the components of a Java program and their interpretation at execution time. It allows students to see the structure of an object-oriented program and enables them to get a deeper understanding of underlying concepts. The program text to visualize can be a regular Java program without the need for special constructs or functions. Jeliot uses a Java interpreter to generate an intermediate code called mCode that supports most of the features of Java [11]. Based on the generated mCode Jeliot offers a variety of possibilities to visualize and interpret programs. Jeliot has been successfully used in classroom settings and in lab sessions [10] to teach object-oriented programming to novice students, it is published under the General Public License (GPL) [8] and can therefore be modified and adapted to support new tasks. The long-term goal of the mJeliot project is to contribute the additions made to Jeliot back to the original code base [2].

1.4 Android

Android [12] is a Linux-based operating system for mobile devices. It is developed by Google and supported by the Open Handset Alliance, a consortium of hardware companies, software companies and network operators. Android aims at offering a standardized software platform for mobile devices. It supports user-written applications, usually coded in Java. The market share of Android-based phones has been on the rise for the last few years. It was at 22.7% by the end of 2010, it is predicted to grow further and to become the most popular platform for mobile phones in 2011 [13].

[2] For more information on Jeliot see [6, 9, 10]

Chapter 2

Literature

This project is mainly inspired by work in two fields of research. The first research field is about finding causes for high failure rates in introductory programming courses which identified conceptual weaknesses in understanding the structures of programming languages and computer systems. The other field deals with the importance of active learning and visualization in learning abstract concepts.

2.1 Conceptual Weaknesses

There are several conceptual weaknesses identified in previous research that contribute to student learning difficulties, three of them are:

1. Machine model: ability to use an intuition based in an understanding of von Neumann computer architecture abstractions to reason about program execution is under-developed in many students [14].

2. Students focus on syntactic elements of the language, and fail to see larger patterns and structures[15, 16, 17].

3. Debugging is a poorly understood activity, students have a weak understanding of what debugging is about [18, 19, 20].

Having identified aspects on why students have difficulties in learning how to program the aspects have to be addressed in teaching. The next section reasons why mJeliot addresses these aspects.

2.2 Active Learning and Visualization

The development of mJeliot is based on theory in three areas, namely, the visual engagement taxonomy [25], research into active learning and promoting student deep learning tendencies and the role of prediction and reflection in strengthening intuition. The underlying learning model is inspired by work on conceptual change and threshold concepts[21, 22, 23].

To encourage change in the student approach to learning to program we concentrate on developing tools that assist students in experiencing hitherto

tacit aspects of programming activity in the classroom. We argue that underlying theories of conceptual development support the view that students' existing conceptions about programming will be challenged more directly using this approach, and that they will consequently be more likely to accommodate expert conceptions of programming into their conceptual frameworks.

The role of visualization in learning computer programming has also been studied extensively. Theory regarding the role of visualization tools in scaffolding learning in computing lead to the development of the Visual Engagement Taxonomy[24, 25]. This research concludes that student interaction is a key feature of successful learning systems based on visualization.

This research informs the student centric and inherently interactive nature of the tool. mJeliot addresses both the conceptual weaknesses and the importance of active learning and visualization at the same time by letting the audience predict results or propose small pieces of code in collaborative code development situations. The predictions are integrated directly into the teaching process, thereby giving the teacher and the students an instant feedback. Keeping in mind the shortcomings identified in 2.1 we combine this with the visualization provided by Jeliot.

Jeliot [6] visualizes the abstract concepts that programming languages base on. It can be used by students in lab sessions and by teachers in frontal lectures, the participation of students is however limited to ask and answer questions. It is hard to involve shy or insecure students, speaking in front of the whole audience especially poses an obstacle for struggling students. mJeliot is aimed at lowering the threshold for active participation. Everybody can give an answer on his phone and giving an answer anonymously without having to speak up in front of everyone might help to overcome insecurities or shyness.

Chapter 3

mJeliot

3.1 Overview

mJeliot has four components that, connected through any TCP/IP enabled network, create an interactive element in an otherwise frontal lecture. They all base on the same code for the data model, they use the same protocol parser for generating and parsing protocol messages and both the mobile client and Jeliot use the same client component to communicate with the server. The four parts are:

- A mobile client that:
  - Connects the user to the server.
  - Receives assignments.
  - Sends in user predictions.
  - Displays user-specific results.

The mobile client consists of a GUI written for Android, a controller dispatching events, the generic client and the parser. See 3.2 for a detailed description of the mobile client.

- A modified version of Jeliot that:

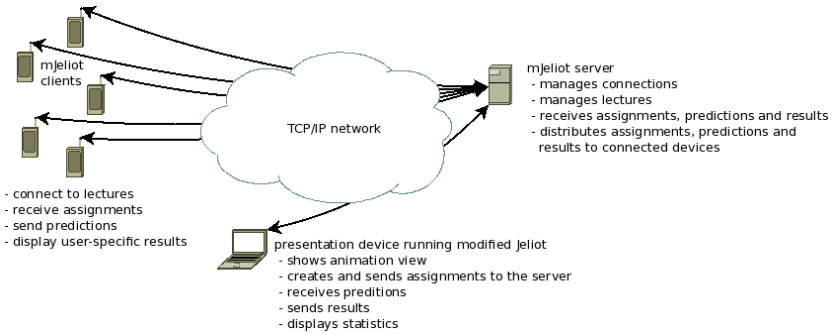

Figure 3.1: The architecture of mJeliot

  - Sends out assignments to the mobile clients.
  - Collects the users' answers.
  - Shows statistics.

Jeliot is basically a client as well, with an extended interface for initiating the interaction with the mobile clients. It is equal to the regular version of Jeliot with some GUI adaptations, a controller for dispatching events and the same generic client and parser components as the mobile client. The functionality is described in detail in 3.3.

- A server that:
  - Manages the lectures.
  - Manages users and the connections to the corresponding clients .
  - Collects and distributes:
    * Assignments.
    * Predictions.
    * Results.

The server contains a controller for managing clients, assignments and the corresponding predictions and results. It has a server for accepting connections and spawns a new thread for every connected client. Generation and parsing of messages is done by the same parser as for the mobile client and the modified Jeliot version. The server and its usage are described in more detail in 3.4.

- An XML-based communication protocol

The protocol is implemented by the parser that is also responsible for generating protocol messages. See 3.5 for details on the protocol.

For now assignments are limited to the prediction of primitive and string parameters and, if applicable, return values of method calls. There are plans to extend both the available data types and the types of available assignments. This is however beyond the scope of this thesis, for a list of plans and ideas for the future see the sections 5.4 and 5.5.

The standard work-flow in mJeliot is as follows[1]:

1. The lecturer creates a lecture on the server.
2. The students' mobile clients and the lecturer's modified Jeliot version connect to the server.
3. The lecturer sends out an assignment.
4. The students solve the assignment and send in their solutions.
5. Jeliot displays statistics on the received solutions.
6. Jeliot sends out the actual result to all clients.
7. The mobile clients show the students' personal results.

The work flow is also visible in the sequence diagram in 3.6.

[1] For more details on how to use and to set up mJeliot see the manual, available on [7]

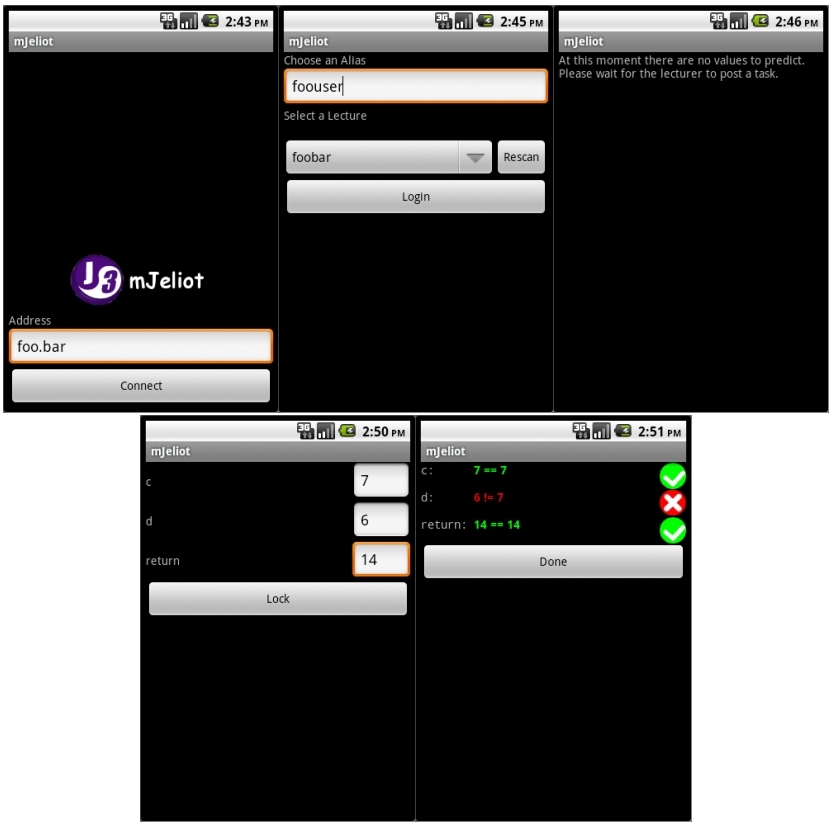

Figure 3.2: Start screen, Login screen, Wait screen, Prediction screen and Result screen of the android client

3.2 Mobile Client

The mobile client enables students to predict and send in the values of parameters and return values of a method call. It also gives the user personal feedback on his predictions. It can utilize any TCP-capable network connection to connect to the server that is available on the mobile device it is running on. This means that in class rooms already equipped with WiFi access, which is quite common nowadays, there is no need to deploy any network infrastructure or to buy any new hardware to use mJeliot. If the clients are 2/3/4G-enabled you can even utilize available mobile phone networks to connect the clients to the server, provided that the server is not behind a firewall blocking access from the outside. The reason to choose Android as a platform for the mobile clients are its market share and the projected growth over the next few years [13]. The developer resources available on the internet are extensive and detailed, the available libraries make the standard components of the platform easily usable and the development process fast.

When the user starts the application he has to enter the server's address to connect to. As soon as the connection to the server is established the user is presented with a login view. In the login view the user can select an alias and a lecture to log in to. The alias is for future use and has no function at

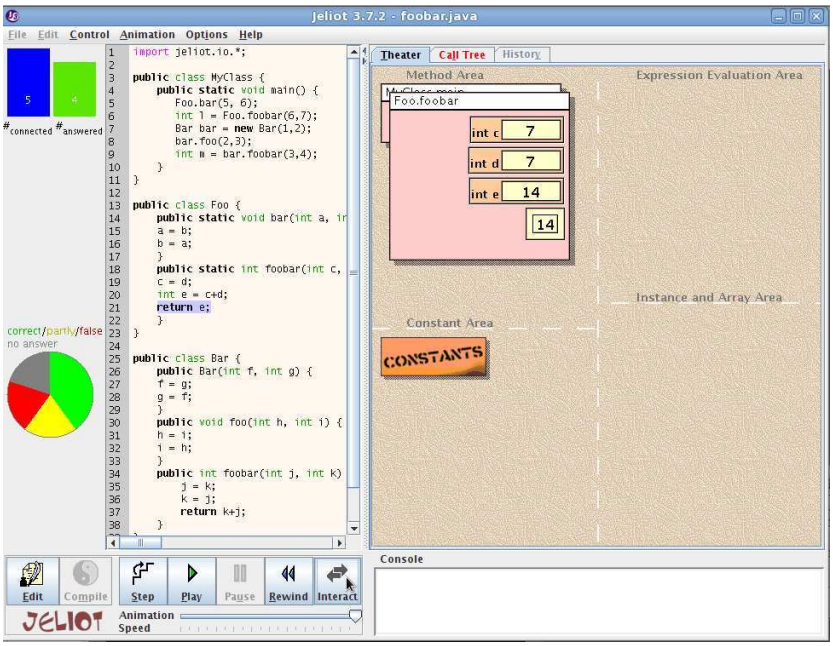

Figure 3.3: The extended Jeliot interface

the moment. When the user logs in he is presented with a wait screen until an assignment for the current lecture is posted by the lecturer and received by the client.

When the client receives an assignment the user is presented with a form to predict all the function's parameters and its return value if applicable. The user can enter his prediction into form fields and send in his solution by pressing the lock-button. As soon as the result is received from the server the user is presented with his personal result showing which parameters he predicted correctly and which falsely. If the user did not send in his prediction he is still presented with a personal result based on the entered values at the moment the correct solution was received.

3.3 Jeliot

To be able to support mJeliot Jeliot has been modified. There is a new statistics bar to the left of the code editor which shows status information on how many clients are connected, how many clients have sent in a prediction and statistics on the parameters and return value as soon as the result is posted. It also shows the percentage of completely correct predictions (i.e. the percentage of users that predicted all values correctly), the percentage of partly correct predictions (i.e. the percentage of users that predicted a subset of values correctly) and the percentage of completely false predictions.

Right of the already available control buttons there is a new button to handle connecting to the server, logging into a lecture and sending out predictions. Sending out predictions is possible for all methods that are currently executed

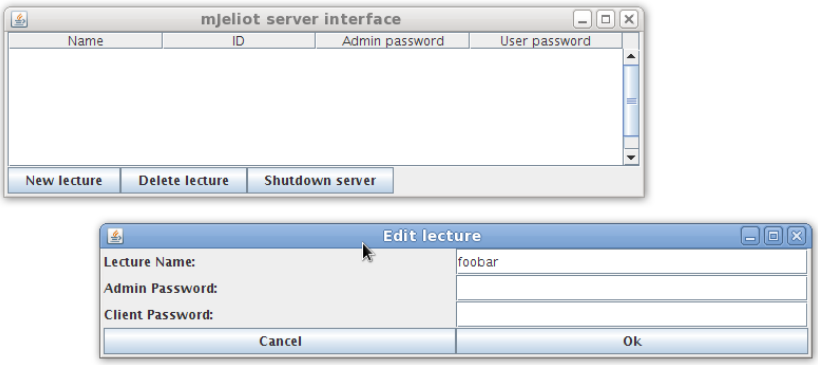

Figure 3.4: The mJeliot server interface

from the moment they are called to the moment when the method returns. As soon as a method that was sent to be predicted returns Jeliot sends the result to all server which forwards it to all clients that are logged in to the same lecture. The statistics bar can be hidden to support regular use of Jeliot without showing the statistics bar for mJeliot.

3.4 Server

The server is a standard Java application. It keeps track of running lectures, assignments, logged in users and received predictions. It listens on TCP port 43711 for incoming connections and serves every connected client in its own thread. The data structure and the interaction between clients are managed by a single working thread to simplify data coherence. Lectures are set up through a graphical user interface shown in 3.4 during run time. This requires the server to have support for a windowing system like an X-Server and/or some remote desktop or X-forwarding mechanism. The reason for requiring this was to make usage of the server as simple as possible and to keep the time necessary to set up a working server at a minimum. Combining the server with a simple graphical interface on the same machine makes it possible to start a server instance and start using the system on a standalone PC or notebook, possibly the presentation device in less than a minute, thereby enabling lecturers to use the system ad hoc in any networked environment without the need for access to a dedicated server and without the need for support from the local IT staff. It is also possible to set up a simple ad hoc WiFi network on the presentation device or using a simple access point or router as the network for mJeliot. Future versions could separate the control GUI from the actual server and manage the server through a network connection. The password fields are for future use and are not used at the moment.

3.5 Protocol

The protocol for mJeliot is XML-based to keep it platform independent and human-readable. It consists of a header that states the version of the proto-

```
<?xml version="1.0" ?>
<ict version="1.0" action="predictSendout">  <!-- Header with version and action -->

      <lectureId>123456</lectureId>           <!-- The lecture's id -->
      <className>Foo</className>              <!-- The method's class' name -->
      <methodName>bar</methodName>            <!-- The method's name -->
      <methodId>654321</methodId>             <!-- The method's id -->
      <parameterCount>2</parameterCount>      <!-- The number of parameters -->
      <parameter name="a" />                  <!-- The first parameter -->
      <parameter name="b" />                  <!-- The second parameter -->

</ict>                                       <!-- Closing the header tag -->
```

Figure 3.5: Protocol example

col and an action. Depending on the action there are more sub-nodes in the protocol. 3.5 shows an example for sending out a method parameter predict assignment. It references the corresponding lecture's id, the method's class, name and id, the number of parameters and a list containing the parameters' names.

The sequence diagram shown in 3.6 shows the protocol and its available messages. After connecting to the server a client queries a list with available lectures and the client logs in to a lecture on user selection. The server informs all other clients that a new client has logged in and then confirms the login with a user logged in message. The newly connected client then receives a list with all users currently logged in to the same lecture. The list of users currently only matters to the lecturer's interface but it is sent anyway to keep the system flexible for future additions. Since programming courses rarely are bigger than 100 people this should not pose a performance problem. A client can log out of a lecture by sending a logout message to the server, the server passes the logout message on to all connected clients. The server generates a logout message on behalf of the client in case the connection between a client and the server gets interrupted. A sendout message is sent to the server as soon as the lecturer posts a new assignment. The server passes it on to the clients which can send their solution to the server which is then distributed to all clients. This again is done to keep the system open for new functionality on the client side. When the result gets posted a result message is generated by Jeliot and distributed to all clients by the server.

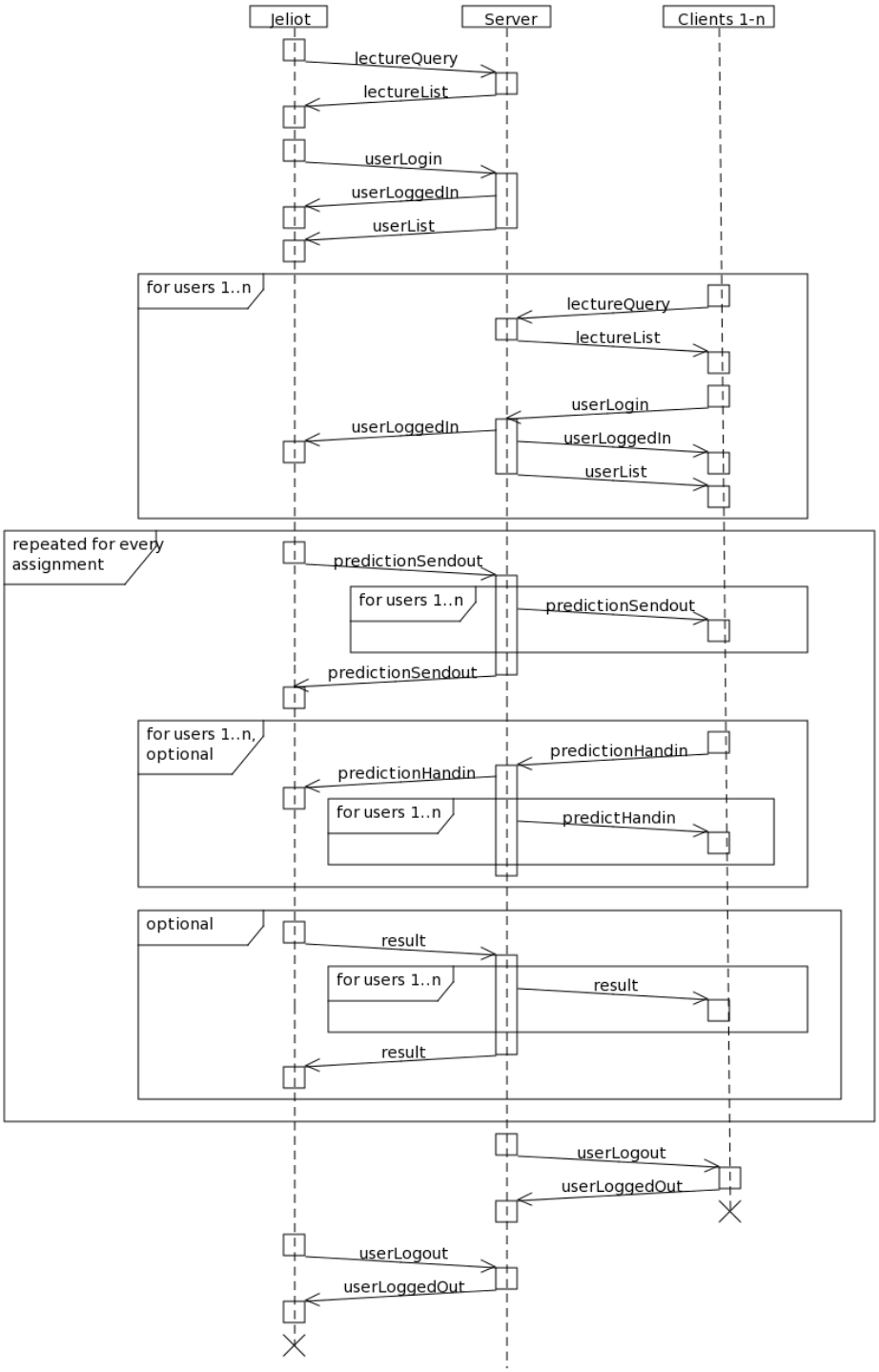

Figure 3.6: Protocol sequence diagram

Chapter 4

Development

4.1 Development Process

The development was based on a very light-weight rapid-prototyping process based on daily increments. After an initial research phase on Android applications, some meetings to define the goals and requirements and some small programming exercises to familiarize myself with Android the goal was to have a running current version every evening, thereby making sure that additions since the last version are tested and working before implementing new parts. First, an early version of the mobile client was developed. Then Jeliot was modified to build the data structure needed to support sending out assignments. Then a first version of the protocol was designed and implemented and last the network components and the server were implemented. Regular meetings with my supervisor made sure that the requirements were achieved and that difficulties that came up during the development were taken into account. During the meetings several of the original goals were refined or changed due to obstacles and ideas that came up during the development. This lead to further development on all parts during the process, converging the four components to a coherent system. The switch from an ad hoc WiFi network on the presentation device to a more flexible (and often simpler) approach utilizing already existing network infrastructures lead to a major rewrite of the protocol since the server and the clients now had to support multiple simultaneous lectures.

To compensate for missing parts during the development phase missing parts were replaced by dummy procedures delivering input to test already implemented parts and dummy buttons to initiate events to trigger the inputs. The code was re-factored regularly to reflect the increased understanding of the requirements, Android and the Jeliot mCode interpreter. The mobile client, the server and the modified Jeliot version follow a model-view-controller scheme to keep the code structured. The model is used by all parts of the system to store the necessary data. The controller reacts on input events from the parser, the client and the user interface, updates the model and throws events on which the views react. The views present the user interfaces and react on events initiated by the controller.

Testing and debugging of the mobile client was done on an Android emulator, the network components were tested by running the server on a remote Unix

system with X-forwarding, the emulators and the Jeliot client were run on a local machine. Running both the Android clients and Jeliot on the same system does however not make a difference to running them on different machines since there is no direct communication between clients. Due to the lack of an Android phone during the development phase the mobile client was not tested on a real Android device.

4.2 Challenges

4.2.1 Android

There were several challenges that had to be addressed during the mobile client development process. Android has a unique concept for structuring applications and as discussed in 5.4 there are some energy- and memory-saving mechanisms that can interfere with the software. Not having access to a real Android device to test the application, these issues could not be tested during the development phase. The performance of the mobile application could not be tested either. However since the emulator and the standard applications shipped with Android were behaving quite slowly while the mobile client was adequately fast performance should not be an issue. The system was tested with six mobile clients logged in simultaneously and no significant drop in performance could be seen. If that holds true for bigger numbers of clients or not remains to be investigated. But even with bigger numbers of clients the number of received messages per second that have to be parsed and processed should be manageable as shown in 5.2.

4.2.2 Jeliot and mCode

The main challenge in modifying Jeliot was writing a new mCode interpreter. Since the Java interpreter converts all names to unique ids during the interpretation of the Java code there are no names available during execution. To be able to refer to methods, parameters and variables by name, we have to be stored in data structures from which the names can be retrieved by their id. Understanding the interpretation process of the Java code and the interpretation of the resulting mCode during execution took the longest. The rest of the functionality was regular GUI coding with no surprises. The Jeliot code is clearly structured and documented, adding elements to Jeliot was therefore not a problem.

4.2.3 Server

The server software is the simplest part of software of mJeliot. It keeps track of connected clients, their state and their messages and distributes the received messages to all other clients connected to the same lecture. After having developed both the Android application and modifying Jeliot this task was straight forward.

4.2.4 Protocol

The protocol is based on XML. It was extended and restructured several times before it covered all needs. Otherwise there were no bigger surprises.

Chapter 5

Results and Discussion

5.1 Reached Goals

The main goal, to create a working prototype for the mobile client, the server software and the adaptation of Jeliot to be able to post simple prediction assignments has been reached. There are some missing features and shortcomings discussed in section 5.4, however, the system has progressed far enough to be tested on a real Android device and in a small testing session.

5.2 Performance and Scalability

mJeliot does not require much bandwidth or performance. The amount of data sent and received for every message is quite low. The expected number of messages is also quite low, see the next paragraph for upper bounds on the number of messages sent and received. There are no complex algorithms run in any component so the system should not run into any performance issues. mJeliot is also not time critical, even slow or high latency network connections should not pose a problem since the user should need many times the time to answer on assignments than the delay created by a slow network connection.

Let n be the expected maximum number of connected mobile clients per lecture, let m be the expected maximum number of assignments in one lecture and let o be the number of concurrent lectures on the server. The expected number of messages received per connected client and assignment is bound by the upper limit $O(n)$, since the client receives 1 assignment, up to n predictions from the mobile clients and 1 result from the lecturer. The number of assignments m should stay reasonably small since the number of assignments that can be handled during a regular lecture is bound by the time constraints of the lecture and the time needed to solve the assignments. The mobile client can receive up to $n+1$ login messages and $n+1$ logout messages during a lecture, provided that the users do not log out and in again during the lecture. This behavior can be expected from the most clients. This means that the client has to handle $O(n \cdot m)$ messages, with m and n being reasonably small. Every mobile client sends $O(m)$messages during a lecture.

The server can be expected to receive $n+1$ login messages and $n+1$ logout messages during 1 lecture. For every assignment it receives up to $n+2$ messages,

1 assignment, up to n predictions from the mobile clients and maximum1 result. This means that the number of received messages for the server is bound by $O(n \cdot m \cdot o)$.

The number of messages sent by the server is bound by $O\left(n^2 \cdot m \cdot o\right)$, since every client receives $O(n \cdot m)$ messages and there are n clients in every lecture. For reasonably small n this is still not a problem. If however the number of clients increase drastically the performance could be improved drastically by not sending all predictions to all clients and just to the lecturer instead. This would reduce the number of received messages per client to $O(m)$with the exception of the lecturer's client which still would receive $O(n \cdot m)$ messages. The number of messages sent by the server would be reduced to $O(n \cdot m \cdot o)$.

5.3 Trade offs

During the development there were several trade offs to consider. There were several different possibilities on how the network connection to the clients could be realized. The original idea was to create an ad hoc WiFi network on the presentation device and use the wireless network's ESSID to connect the clients to the server. This solution would require just a notebook to create a lecture and make it available to WiFi-capable mobile clients. This solution was however never implemented. The wide variety of notebooks, their hardware and their operating systems would have made this solution hard to implement and error-prone. The setup of such an ad hoc WiFi network on a vaguely defined hard- and software basis would have required hardware, software and network knowledge on the lecturer's side. Connected client devices would often not be able to access the internet as long as the device is connected to the ad hoc network which would limit the participation on the students' side. Instead we decided on requiring a working network infrastructure to deploy mJeliot. Most classrooms at universities already have working WiFi networks. 2/3/4G coverage in the classroom could also be utilized to provide communications between the clients and the server. The lecturer can still create his/her own wireless network if there is no existing wireless network or if the utilization of the network is not feasible. Either by creating a network through his/her presentation device or by using a standard wireless home router or access point.

The choice of XML as basis for the transport protocol leads to a significant protocol overhead for messages sent between devices. However, the messages are small and bandwidth consumption and performance should not be an issue as shown in 5.2. XML-parser libraries are available for almost every platform which makes the development of new clients easy. The protocol is human-readable which makes debugging easy as well.

mJeliot supports only simple types of assignments. The main reason for keeping assignments simple at the moment is directly connected to the limited screen size and limited input speed on most current mobile devices. Moreover, expanding the types of available assignments pose a difficult task. The more complex assignments get the more it has to be researched how to keep mJeliot easily usable. To maximize participation the user experience should not be frustrating or deterrent. If novice programming students have to struggle with the tool they are supposed to learn with they might get demotivated and stop participating. There are ideas on expanding the types of assignments, formulated

in 5.4 and 5.5 however without any experience in a real classroom setting and research in how to realize those features in a good way the gained effect on the students' learning success can not be assessed.

5.4 Missing Features/To Do

The main goal of this project was to develop a working prototype for testing purposes and to create new input for further development. Four features that do not add functionality to the current client but might be considered for upcoming prototypes are listed below. Some features that add new functionality and can be considered for future versions of the system are discussed in 5.5. Due to the lack of a phone for testing during the development phase the mobile client was tested on an Android emulator only. Since I had no previous experience with software development for Android devices the application might show unexpected behavior when running it on real Android devices. The fact that Android has a mechanism to kill parts of a running application to conserve memory might lead to gaps and logical errors in navigation. The mobile client might also react unexpectedly to Energy-conserving mechanisms and to disrupted or delayed network connections. Since however the connection is based on TCP-sockets the connection between client and server should be reliable as long as the client does not loose network connectivity completely. This however might happen when an Android device switches between different available network connections. There might also be unwanted sideeffects caused by other Android applications.

5.4.1 Access Control

The current prototype does not support any access control. This is because access control is not needed for early tests, however as soon as the system is run on a bigger scale, in a more course-oriented setting, on a institution-wide platform or on a publicly reachable server controlling access to a lecture might be desirable. The server interface and the data model already have fields for a lecturer and a student password. However neither the protocol parser nor the clients make use of this at the moment.

5.4.2 Remote Server Administration

The current server can only be configured on the machine it is running on. For early tests where the server most probably runs on the presentation device or another desktop system this is a simple and satisfying solution. However, to be able to run it on a server a remote administration interface without granting people creating lectures direct access to the server is desirable. But having a remote interface makes the system more complex and harder to set up. Moreover, building a good remote administration tool is beyond the scope of this project.

5.4.3 Customization and Configuration

Most of the parameters are hard-coded into the program. Exposing parameters like a default URL or the default TCP port to listen on makes it easier to adapt

the system to special needs. However since the system is designed to require as little configuration as possible those options are not available for the moment. They could however be moved into a configuration file and/or a settings dialogue in the clients and the server.

5.4.4 Support for More Data Structures

The prototype is limited to predicting parameters and return values of static and non-static methods with simple data types. There is no support yet for complex data types like arrays or lists and no support for constructor calls. There is also no support for value prediction on local variables. These extensions require a thorough investigation on how they can be represented especially on the mobile client's side both efficiently and systematically without loosing the simplicity of the interface. This is not just outside the scope of this thesis but should also be based on practical experience with the current prototype.

5.4.5 Support for More Platforms

The current mobile client application is programmed for Android devices. But there are other popular software platforms available for mobile devices that have big market shares as well. Extending mJeliot to other mobile operating systems to maximize the participation of students should therefore be considered as soon as the prototype has been tested and before the system is used on a big scale. Programming new mobile clients to support the protocol should be feasible for almost all modern network-capable devices.

5.5 Proposed Future Work

There are several possible future features for the system and there are surely more feature ideas to appear when the system is in use. However there are a few future work areas that came up during planning and developing this project.

5.5.1 Mobile Code Editor

This is an advanced feature that will probably require a lot of planning and a lot of work. In this scenario students are given the opportunity to write simple and small code snippets on their clients. The snippets can be sent in to the lecturer who can select a student's solution and integrate it directly into the existing code. Programming on smart phones or similar devices is, however, limited to small and simple programs. The typing speed, especially on keyboard-less devices, and the limitations on entering non-alphanumeric characters pose an obstacle that has to be overcome by the editor. A possible solution could be a context-sensitive interface that offers templates for common structures. These templates would limit the input to selecting templates and defining names and values. This should increase input speed but limits the users to a predefined subset of the programming language. This subset will probably be limited to a few very common structures and expressions. Otherwise users might get lost and of course due to space restrictions on most handheld devices. However, since the intended audience is a novice audience keeping the number of possibilities

to a few might even be wanted. Other problems to solve is how the replies are presented, selected and integrated into the code.

5.5.2 More Statistical Data

A feature that is especially interesting for research and course evaluation is the collection and evaluation of more statistically relevant data. Analyzing data like the correctness of replies, the speed of answering, the participation rate, etc. enables a more thorough evaluation of participation and could be useful to improve both mJeliot and the teaching. Storing statistical data enables evaluation after the actual session and comparing it to other sessions.

5.5.3 User Identification and Selection

Different modes of registration could extend possible applications. The system could for example have support for selecting a certain user for tasks or to send different assignments to different users. There could also be different modes for lectures: open lectures everyone can participate in under a pseudonym, lectures with semi-closed user groups, limited for example by a group password shared only with a subset of potential users or even lectures with a mandatory registration and identification to enable teachers to quiz students and asses them individually. These features have however to be implemented very carefully, since as discussed earlier the user's anonymity is essential to increase the spontaneous and honest participation. It has to be clearly visible when the user is identifiable or when a specific answer can be singled out to be discussed and when not.

Chapter 6

Conclusion

This project emphasizes that computer science and studying it is not just about the facts but also about how it is teached. It aims to improve the students' learning and understanding by giving teachers a tool to involve and engage students. Hopefully improving interactivity and engagement amongst students can help to reduce failure rates in computer science and help students to develop a deeper understanding of programming and strengthen their abilities to predict and deduct. Extending mJeliot to support real collaborative code development in a classroom setting will hopefully lead to live program development exercises where students propose code that is included into the exercise directly.

For me personally the project was an interesting experience. I came in contact with Android which for me was a new and very interesting platform that offers a big potential. Getting familiar with the concepts and the design of Android and starting to develop an application from scratch was fun and I learned a lot. Getting a grip on Jeliot in general and the mCode interpretation engine was challenging but rewarding. Developing the protocol made me think about the structure of protocols I already knew in more detail and to come to a new level of insight in their structure.

The different trade offs that had to be considered during development and the changing requirements showed me that even a project that seems pretty straight forward in the beginning has its hidden complexities. There are always more than one solution to a problem with different results and implications than meet the eye at first glance. The pedagogical scope of this project demonstrated that there are more things to consider than the technical aspects.

Finally I came to the conclusion that studying and being involved in other projects while trying to write a report like this is challenging in itself but leads in my opinion to better results. It forces you to take a break, take a few steps back and rethink your ideas and solutions whenever other commitments have to be addressed.

Bibliography

[1] E. Soloway, "Learning to program = learning to construct mechanisms and explanations", *Communications of the ACM*, vol. 29, no. 9, pp. 850-858, 1986.

[2] M. McCracken, V. Almstrum, D. Diaz, M. Guzdial, D. Hagan, Y. B.-D. Kolikant, C. Laxer, L. Thomas, I. Utting, and T. Wilusz, "A multi-national, multi-institutional study of assessment of programming skills of first-year cs-students", *SIGCSE Bull.*, vol. 33, no. 4, pp. 125-180, 2001.

[3] R. Lister, E. S. Adams, S. Fitzgerald, W. Fone, J. Hamer, M. Lindholm, R. McCartney, J. E. Moström, K. Sanders, O. Seppälä, B. Simon, and L. Thomas, "A multi-national study of reading and tracing skills in novice programmers", *ITiCSE-WGR '04: Working group reports from ITiCSE on Innovation and technology in computer science education*, New York, NY, USA: ACM, pp. 119-150, 2004.

[4] A. Robins, J. Rountree, and N. Rountree, "Learning and teaching programming: a review and discussion", *Computer Science Education*, vol. 13, no. 2, pp. 137-172, 2003.

[5] A. Pears, S. Seidman, L. Malmi, L. Mannila, E. Adams, J. Bennedsen, M. Devlin, and J. Paterson, "A survey of literature on the teaching of introductory programming" *SIGCSE Bull.*, vol. 39, no. 4, pp. 204-223, 2007.

[6] M. Ben-Ari. (2011) Jeliot 3.0. [Online]. Available: `http://cs.joensuu.fi/jeliot/index.php`

[7] (2011) mJeliot project website. [Online]. Available: `http://www.it.uu.se/research/group/upcerg_new/projects/ICT_prog%20`

[8] (2011) Jeliot 3.0 license. [Online]. Available: `http://cs.joensuu.fi/jeliot/faq.php#q2`

[9] R. B. Levy, M. Ben-Ari, and Pekka A. Uronen "The Jeliot 2000 program animation system", *Computers & Education*, Volume 40, Issue 1, January 2003, pp. 1-15.

[10] M. Ben-Ari "The Effect Of The Jeliot Animation System On Learning Elementary Programming Mordechai", [Online], Available: http://200.17.137.110:8080/licomp/Members/jeanemelo/plonelocalfolderng.2006-04-10.7475913377/PEP/PEP2009/Aula6/Grupo5/1_Artigo_sobre_Jeliot.pdf.

[11] A. Moreno "The Design and Implementation of Intermediate Codes for Software Visualization", Master thesis, Department of Computer Science, University of Joensuu, Joensuu, Finland, 2005.

[12] (2011) Android os. [Online]. Available: http://www.android.com/

[13] Gartner. Gartner says android to command nearly half of worldwide smartphone operating system market by year-end 2012. [Online] Available: http://www.gartner.com/it/page.jsp?id=1622614

[14] N. Ragonis, and M. Ben-Ari "On understanding the statics and dynamics of object-oriented programs", SIGCSE '05: *Proceedings of the 36th SIGSCE technical symposium on Computer science education*. New York NY, USA: ACM Press, 2005, pp. 226-230.

[15] A. Pears, "Conveying conceptions of quality through instruction", *7th International Conference on the Quality of Information and Communications Technology*, 2010, [Online], Available: http://urn.kb.se/resolve?urn=urn:nbn:se:uu:diva-130865%+UppsalaUniversity,DivisionOfComputerSystems

[16] J. C. Spohrer, E. Soloway and E. Pope "A Goal/Plan Analysis of Buggy Pascal Programs", *HUMAN-COMPUTER INTERACTION*, Volume 1, pp. 163-207, 1985.

[17] T. R. G. Green and M. Petre, "Usability Analysis of Visual Programming Environments: A 'cognitive dimensions' framework", *Journal of Visual Languages and Computing*, vol. 7, no. 2, pp. 163-207, 1985.

[18] Y. B.-D. Kolikant and M. Mussai, ""So my program doesn't run!" definition, origins and practical expressions of students' (mis)conceptions of correctness", *Computer Science Education*, vol. 18, no. 2, pp. 135-151, 2008. [Online]. Available: http://www.informaworld.com/10.1080/08993400802156400

[19] S. H. Edwards, "Improving student performance by evaluating how well students test their own programs", *ACM Journal of Educational Resources in Computing*, vol. 3, no. 3, pp. 390-396, 2010.

[20] S. Fitzgerald, R. McCauley, B. Hanks, L. Murphy, B. Simon and C. Zander "Debugging From the Student Perspective", *IEEE Transactions on Education*, Vol. 53, No. 3, pp. 390-396, 2010.

[21] J. Meyer and R. Land "Threshold Concepts and Troublesome Knowledge: Linkages to Ways of Thinking and Practicing within the Disciplines".

[22] S. Vosniadou, A. Baltas and X. Vamvakoussi "Re-framing the conceptual change approach in learning and instruction", *Advances in learning and instruction,* Amsterdam, The Netherlands, 2007.

[23] G. Özdemir, D. Clark "An Overview of Conceptual Change An Overview of Conceptual Change Theories", *Eurasia Journal of Mathematics, Science & Technology Education,* pp. 351-361, 2007.

[24] T.L. Naps, G. Rössling, V. Almstrum, W. Dann, R. Fleischer, C. Hundhausen, A. Korhonen, L. Malmi, M. McNally, S. Rodger and J. Á. Velázques-Iturbide "Exploring the role of visualization and engagement in computer science education", *ITiCSE on Innovation and Technology in Computer Science Education (ITiCSE'02)* (Working Groups Report). Aarhus, Denmark: ACM Press, pp. 131–152, 2002.

[25] N. Myller, R. Bednarik and E. Sutinen "Extending the Engagement Taxonomy: Software and Visualization and Collaborative Learning", *ACM Transactions on Computing Education, Vol. 9*, No. 1, Article 7, pp. 1-27, 2009.