Programming Life - Final report

Group E:
Felix Akkermans
Niels Doekemeijer
Thomas van Helden
Albert ten Napel
Jan Pieter Waagmeester

June 18, 2012

Contents

Preface

This final report concludes our effort that started almost half a year ago on the longest project we've participated on at TU Delft; the Context-project, where team can choose from several directions to develop a software application very specific for the domain (context) they choose. The project ran for a whole semester, in which the first served for mostly preparation, a seminar and domain study, and the second for the actual implementation. This report is our final conclusion on the project, with a high emphasis on reflection.

We would like to thank all the people that helped us in the development, the testers, the TA's who have stood by us and provided us with very good feedback on often very short terms, and the tutors that provided us with an inspiring challenge and seminar.

Delft, June 18, 2012

Jan Pieter Waagmeester
Felix Akkermans
Niels Doekenmeijer
Thomas van Helden
Albert ten Napel

Abstract

This document concludes the whole *Contextproject* and especially the development of our product: Zelula. This piece of software provides synthetic biologists with a visual tool to design, simulate and validate biological logic circuits.

When we compare the final product to its design, the GUI looks a lot like the sketches made in the design phase. Not only that: every must-have is recognizable in the final product. The proposed tests are not always implemented, but a fair amount of the code is covered. We discovered that testing certains things is quite hard, so we had to rely more on acceptance testing than anticipated.

During the development we encountered a number of problems, ranging from planning and design to bugs in libraries used. One particular case of the latter was the problems we had connecting our program with an external solver. It took quite some time to debug the SBML files used in that communication.

As a team, we really think we delivered a nice product. We are also satisfied by the way the team operated.

A number of features did not fit in our schedule and will improve the program significantly, to name a few: a more thorough grouping/compound implementation, highlighting errors in the actual GUI and different workspace improvements.

1 Introduction

This report will detail our design and implementation of Zelula for the *Contextproject*. Zelula is a piece of software for synthetic biologists with a visual tool to design, validate and simulate biological circuits.

Chapter 2 will contain a description of the product, every part of the finished product will be detailed. Chapter 3 contains a description of the design and implementation process, detailing how the design compares to the implementation. Chapter 4 describes the key problems we had and explains the solutions. Chapter 5 contains a general reflection on the teamwork, followed by chapter 6 which contains individual reflections of the team members.

Lastly appendix A lists our scrumplans.

2 Description of Zelula

Zelula is a modelling and simulation package which enables users to simulate the protein *logic* in a cell. With a provided set of BioBricks representing the basic AND and NOT gates, the user is able to design more complex circuits of interactions between the different BioBricks. The resulting circuits can be simulated and the results will be presented as a concentration/time graph.

2.1 Circuit editor

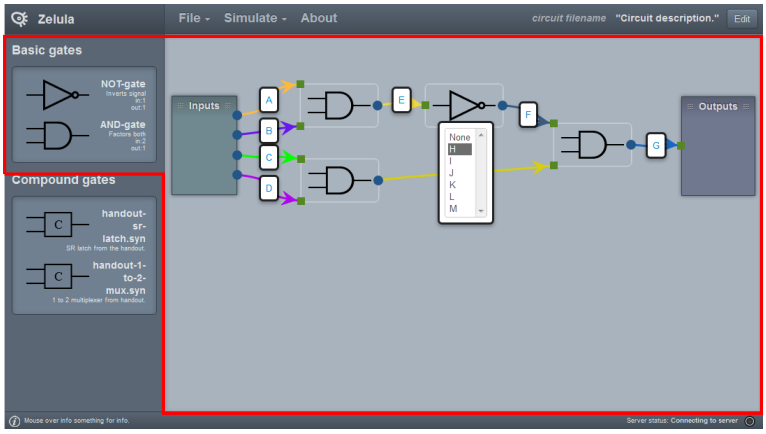

Figure 1: GUI mainscreen; highlighted is the workspace with the regular (draggable) gates.

Zelula enables the user to intuitively construct circuits from the basic available building blocks (see figure 1 for the main screen). Gates can be dragged from the sidebar to arbitrary positions in the working area. Connections between the gates can easily be dragged from and to the *input* and *output* connectors and the endpoints on gates.

When a connection is made, no protein is selected for it by default. The user may select which protein to use. Visual feedback on protein assignment is provided through the colors of the connections.

The interface prevents the user to make some simple mistakes. For example, it's impossible to connect two inputs. If the user connects one output to two inputs, the interface makes sure the protein on those connections is the same.

Even the position of the circuits' input and output connectors can be freely chosen by dragging them using the text as an handle.

Removing gates and connections is a matter of double clicking on them.

2.2 Compound library

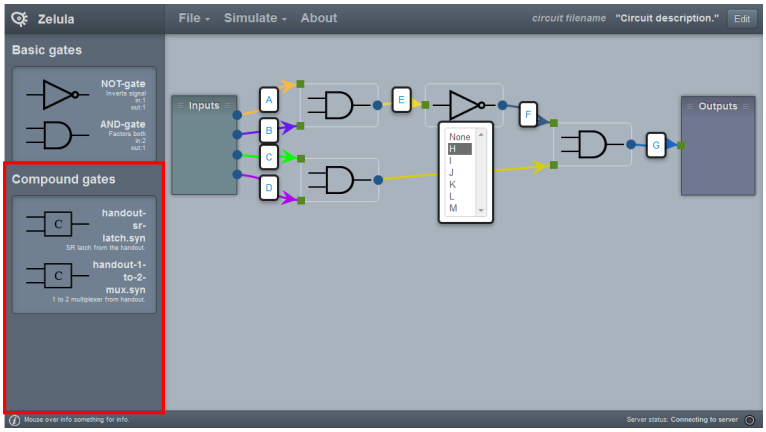

Figure 2: GUI mainscreen; highlighted is the box with the (draggable) compound gates.

Every circuit can be saved as a compound gate, which means it can easily be reused in later circuits. The saved compound gates are presented to the user in the sidebar, below the basic gates (figure 2). When a user drags a compound gate to the working area, it will expand to the original arrangement of gates, with the inner connections in place. The only thing left for the user is connecting it to the rest of its circuit.

2.3 Validation

When a user is satisfied with his work, he can validate the circuit. Different errors like unassigned proteins or inclomplete connections will be reported. The user will see the error message below the circuit, providing excellent overview to both the circuit and the errors to solve the problem.

2.4 Simulation

A valid circuit can be simulated. In order to simulate a circuit, the input must be defined as a function of time. Zelula provides two ways to accomplish this. Firstly, in an input editor the level for each input can be defined as a function of time in a visual way, secondly, a CSV text can be provided describing the transitions as a function of time.

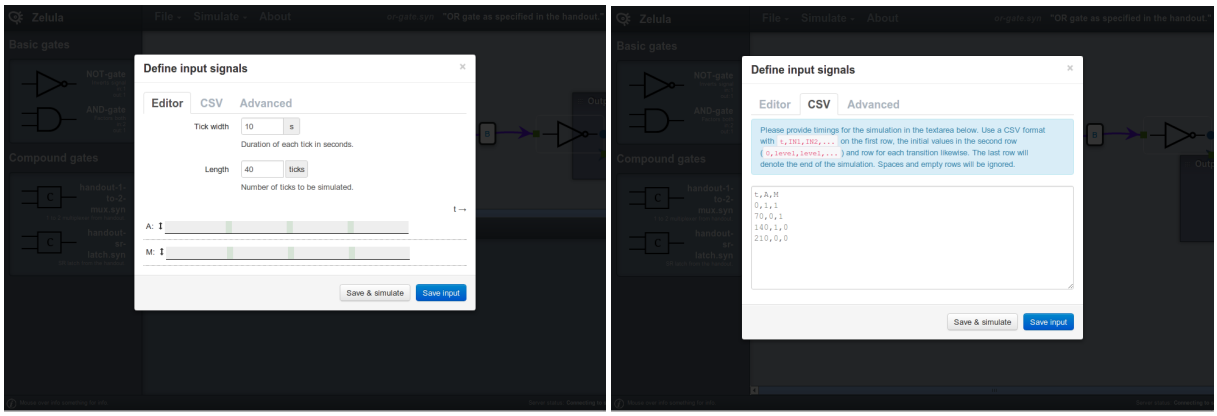

Figure 3: Two different ways to define input values: an editor (A) and by CSV (B).

Input editor

The input editor (figure 3.A) enables the user to define each input signal in a visual way. Each tick represents a number of seconds in the simulation. The user can select this number of seconds, and also the number of ticks available.

When a user clicks on a tick, it is toggled. The user may also select ranges by clicking on a certain tick and dragging the mouse to another tick while holding the mouse button.

CSV input

The input may also be defined by a simple CSV format (figure 3.B), as defined by the TA.

Simulation result

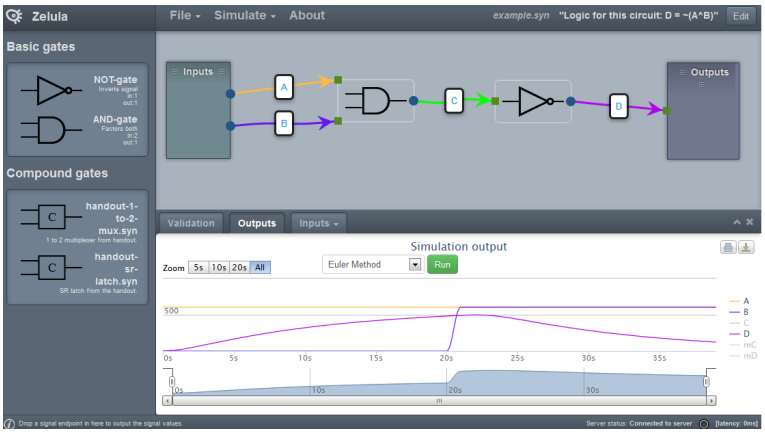

Figure 4: Split screen view with the editor and the simulation result in a graph.

2.5 Exporting

The work done in Zelula can be shared in two different ways:

- The circuit can be exported to a SBML file, a common format used in synthetic biology.
- The the graph that visualizes thee simulation output can be exported to a number of different image formats.

Of course if the user has access to the server like when it is ran locally, the user can simply save the circuit using the editor to obtain it in our application specific format (`.syn`) that contains all information present in the application.

2.6 Server

The server operates in the background, serving the files and data needed for the GUI, providing persistent data storage and simulation services. The user does not interact with it in a direct way, and no server is required to be present for modelling and design. Each service of the server is provided by a Java servlet. These servlets depend on the Apache Tomcat webserver and might be distributed as a `.war` archive.

3 Design and implementation process

The first half of the project was focused on design, while the second part was all about implementation. In this chapter we will reflect on the consistency between design and implementation. We will clarify what was implemented according to the documents and what was changed or perhaps not implemented. This will done with help of a reflection on the initial GUI sketches (3.1) and our list of requirements (3.2). The chapter is conclude with a short section about how we finally tested our work (3.3).

3.1 Client GUI

In our RAD document[1] we gave a few simple sketches of what we had in mind for the user interface of the client. The final implementation is greatly influenced by these sketches, as can be seen from the following screenshots.

The matching letters in the “before” (figure 5) and “after” (figure 6) pictures show the same functionality. So A and B provide all the gates, D is the workspace, and so on. There have not been any great changes from what we had in mind for the user interface. We did, however, add a lot of features along the way; small features we did not think of at the beginning. For example, we colored the proteins so that they are easily distinguishable and we added general gates which serve as input or output. Another example is that used proteins cannot be chosen from during protein selection.

It is according to the SCRUM-workflow that we have added these little features. We started off with a basic working product and added them along the way.

Implementation was greatly accelerated by the use of frameworks and libraries such as jQuery, Bootstrap, HighCharts, jsPlumb and JSON-Java. The use of such frameworks of course also implies the study of their use, and solving problems or limitations encountered with them. This costed considerable time as eluded in section 4.

[1] Requirements Analysis and Design @Github

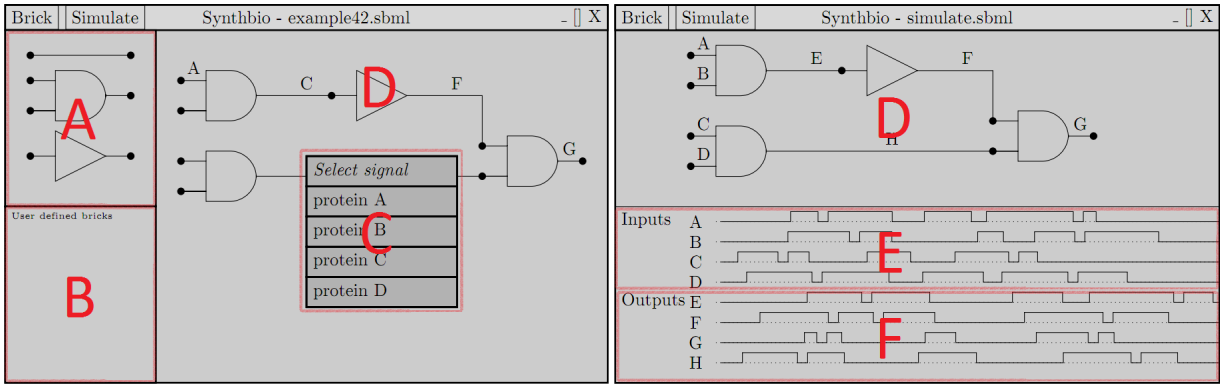

Figure 5: Sketches from RAD document: model and simulate.

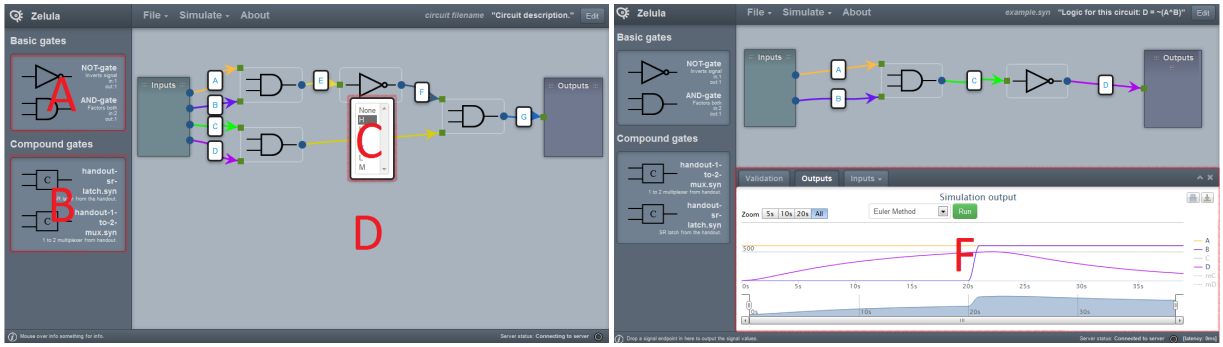

Figure 6: Screenshots of the final GUI implementation: main screen and outputs (F).

3.2 Requirements

In the Test & Implementation plan[2] we set up a list of requirements according to the MoSCoW model. What follows is a copy of this list with comments on how (and if) it is implemented.

Must-Haves

1. **Connection** *Client and server must be able to communicate. If there is no connection, the user should be notified.*
   This was one of the first finished tasks, as it is essential for both client and server. The client will show an error dialog when no connection to the server can be established. While there is a connection, the user can view the connection delay (in milliseconds) in the bottom right corner of the screen.
2. **Available gates** *The application must be able to present a list of available gates to the user. These gates can be used to model the circuit.*
   As can be seen in figure 6.A, the user is presented with a list of draggable (regular and compound) gates.
3. **Design circuit** *The user must be able to design a circuit by specifying gates (using a drag-and-drop) and the relations between these gates.*
   - **3.1** *The application must be able to visualize a gate using a simplified image. This image should relate to the function of the gate. For example, for the AND gate, it is logical to use the distinctive AND symbol normally used in circuit design.*
     Gates are visualized using a simple, yet distinctive image (figure 6.A). We simply used SVG images in the HTML to display these.
   - **3.2** *The user must be able to drag and drop gates from the list into the working area.*
     Gates are visualized using a simple, yet distinctive image (figure 6.A). They can be dragged from the list and dropped into the workspace (figure 6.D).

[2] Test and Implemention plan @Github

- **3.3** *The user must be able freely move the gate around in the working area, but gates will snap to grid points on the working area.*
  After the gates are dropped into the workspace, the description of the gate disappears and the image remains. The gates can still be dragged around the screen.
  At first, we got the system working without a grid snapping system. This worked very well and we did not see the need to implement a grid. That is why we postponed this requirement, and decided to drop it near the end of the project.
- **3.4** *The user must be able to draw connections between the gates in the form of wires.*
  In the workspace, gates get endpoints for input and output (the little green and blue dots in figure 6.D). Wires can be dragged from the output endpoints to the input endpoints. Wires from input to input and output to output endpoints are not possible. Output endpoints can contain multiple wires, while inputs only allow one.
- **3.5** *The user must be able to draw input and output wires for the circuit, to explicitly state which proteins will be used as input.*
  We have added input and output gates which serve as big input or output endpoint. Wires can be dragged from the input and dropped into the output.

4. **Available proteins** *The application must be able to present the user with an overview of available proteins to assign to signals (visualized by the wires).*
   Clicking Simulate > Show proteins in the menu will display a table with the available proteins and matching parameters. This table is sortable and can be filtered. Each wire will also have an overlay with the selected protein. Clicking on this overlay displays a list of proteins that can still be used for selection (figure 6.D).
5. **Protein specification** *The user must be able to specify which protein is used for a certain signal.*
   After opening the list of unselected proteins (figure 6.D), clicking on a protein results in this protein being specified for the connection.
6. **Export circuit** *The application must to able to save a circuit to a .syn file.*
   A circuit can be saved using File > Save and exported to SBML using the File > Export menu item.
7. **Import circuit** *The application must be able to load an exported circuit from a .syn file.*
   Saved files can be loaded using the File > Open menu item.
8. **Input values specification** *The user must be able to specify the input values used for the simulation of the circuit.*
   Clicking Simulate > Define inputs (or pressing F7) will open a dialog in which the input values can be defined (figure 6). These input values can also be loaded from a .csv file.
9. **Circuit validation** *The user must to be able to validate his circuit in the application and get feedback over where there are conflicts.*
   Pressing F8 (or clicking Simulate > Validate circuit) will give a validation of the circuit. This validation is done server side, and conflicts are grounded by useful feedback.
10. **Circuit simulation** *The application must be able to simulate a valid circuit and present the output values to the user.*
    Pressing F9 (or clicking Simulate > Run solver) will simulate on the server and present the user with a plot of the output values (figure 6). The user can zoom in/out and specify which proteins are visible in the plot (only the input and output values are shown by default). Because we use a simulation library which supports multiple solving methods, we give the user the options to select a method.

Should-Haves

11. **Re-use circuits** *The application can import pre-defined circuits as extra gates. This is not a necessity, but would be a great addition to the program (and will ease building circuits). Among others, protein specification, importing and exporting will be more difficult to implement.*
    This was a requirement we really wanted to add, but decided to implement a simpler version due to time restrictions. We wanted to make it so that compound gates were visualized the same as regular gates, but decided this would be too much work. We can now mark circuits as compound gates at the time of saving. These gates will then show up in the list of compound gates and can be dragged into the workspace. When dropped, the old circuit will show up (minus the input/output signals).

3.3 Testing and test coverage

Since the development of this project was to be test-driven, we did put some effort in writing a test and implementation plan[3]. This section reflects on the use of that document, and on the amount of testing we actually did.

Unit testing

Unit testing was a major part of our test plan. Having written a test plan implies the actual use of it, however, we did not really use the plan in deciding what to test. Tests were mainly written together with the code, not necessarily before implementing it.

Unit testing client side JavaScript code was planned, but not implemented in the same depth as intended. GUI interaction was not tested in any automatized way, mainly because we did not know how to do it. It might have been possible to invest more time in it, but since the structure of the interface did change a lot it would have been really cumbersome. Testing these aspects of the program was done through acceptance tests, with a detailed description below.

Judging from the past, we lacked the experience to write a decent doable test plan. We made some bold and broad plans but both too precise and too vague. More emphasis on the testing during the development could have fixed it, but was omitted.

Test coverage

Testing for coverage was not really covered in our testing plan, however, during the selection of tasks for iteration 4 we decided to explicitly check and improve coverage. We used Cobertura to report on Java coverage, and JSCoverage to report on JavaScript coverage.

As can be seen in figure 7, branch coverage figures are moderately high, but not exceptional. Apart from that, coverage on issues not easily recognizable by coverage reports like these is not always very good. For example, a more detailed plan on the things a test for a solver should cover could have exposed the simulator problems earlier in the process[4].

Acceptance testing

The acceptance tests are test if the system meets the requirements made, and how the system performs on pseudo-requirements. In particular we test these using the user interface to see if the system can perform the required tasks without problems. With these test we also manually test the user interface to see if there any display artifacts or bugs. Because these acceptance test call all the functionality throughout the whole system from the interface, they also serve as an important part of the integration testing.

The acceptance testing included the running of several specified scenarios, and a grading on several usability aspects to measure how well the application performs there.

In the end these tests were write quite late, and we only came to performing these test in the last iteration (SCRUM iteration 5). We had several people perform these tests, some developers, and some other computer science students, as these fall within the category of target users. Despite this we these test runs revealed a lot of bugs and gave us valuable feedback that influenced us in making changed in the application during the last iteration. Some structured reports can be found attached at the end of this document.

[3] Test and Implemention plan @Github

[4] For a detailed explanation on this problem, refer to section 4.1

Coverage Report - All Packages

Package	# Classes	Line Coverage	Branch Coverage	Complexity
All Packages	32	64% 688/1062	65% 298/454	2.444
synthbio	1	85% 18/21	87% 7/8	1.667
synthbio.files	2	96% 62/64	77% 37/48	2
synthbio.json	1	78% 15/19	N/A N/A	1
synthbio.models	11	86% 366/423	75% 194/258	2.468
synthbio.servlets	11	0% 0/258	0% 0/42	4.75
synthbio.simulator	6	81% 227/277	61% 60/98	2.128

Coverage Report - synthbio.models

Package	# Classes	Line Coverage	Branch Coverage	Complexity
synthbio.models	11	86% 366/423	75% 194/258	2.468

Classes in this Package	Line Coverage	Branch Coverage	Complexity
AndPromotor	90% 19/21	88% 16/18	2
BioBrick	100% 2/2	N/A N/A	1
CDS	93% 15/16	50% 1/2	1.286
Circuit	84% 81/96	75% 54/72	2.682
CircuitException	100% 2/2	N/A N/A	1
CircuitFactory	84% 79/94	76% 55/72	12.6
Gate	79% 42/53	71% 10/14	1.706
NotPromotor	61% 11/18	16% 1/6	1.75
Position	100% 16/16	83% 5/6	1.333
Promotor	100% 13/13	62% 5/8	1.5
SimulationSetting	93% 86/92	78% 47/60	2.333

Report generated by Cobertura 1.9.4.1 on 6/17/12 9:23 PM.

Figure 7: Overview of Java package coverage and the coverage of a selected package.

JSCoverage

Browser | Summary | Source | Store | About

☐ Show missing statements column

File	Statements	Executed	Coverage
Total: 12	9400	4381	46%
/lib/jquery-1.7.2.min.js	4224	667	15%
/lib/qunit-1.7.0.js	2239	1534	68%
/models.js	1009	788	78%
/test/models-test.js	376	371	98%
/test/util-test.js	112	107	95%
/util.js	169	122	72%
lib/jquery-1.7.2.min.js	3	3	100%
lib/qunit-1.7.0.js	740	390	52%
models.js	330	222	67%
test/models-test.js	117	117	100%
test/util-test.js	31	31	100%
util.js	50	29	58%

Figure 8: Overview of JavaScript coverage.

4 Key problems and solutions - highlights

During this project we have encountered some problems. In this section we will have a look at some of the highlights of these problems and how we fixed them.

Planning and design

Since, for some of us, this was the first time working with SCRUM, it took some getting used to. At first, we had to get used to working towards a result in two weeks and estimating what we could do. This quickly became easier as the project progressed. What we really noticed is that our design was not complete. We hadn't thought of the complete structure and failed to set some standards of coding. This led to less readable code, which is annoying when reviewing each others code, and no clear idea which function was implemented where. Especially later on, some functions turned out to be double implemented or unused. This point is also discussed in the paragraph about restructuring.

Setup

A common issue using server-client structure is the actual connection between the two. In the beginning we had some trouble with this but this was mainly caused by out of sync code and trouble with Apache Tomcat. We decided on a standard version of Tomcat and wrote a setup page on the wiki[5]. Afterwards, the problem was solved.

QUnit

Our program was developed test driven. This makes for very neat code, but it also comes with some problems. We mainly encountered problems with QUnit, a JQuery testing framework similar to Java's JUnit. Although the testing was done properly, the feedback on the results was not very helpful. Eventually we managed to tackle this problem with some extra manual testing. We chose to do some extra manual testing instead of switching testing framework because of all the work we already put into it. Also, QUnit works really easy with JQuery code and the code is readable.

Drag and drop

Another problem client side was the dragging and dropping of gates and connecting them with wires. We had anticipated this as being tough and found a solution in jsPlumb, a solution with its downsides. The framework jsPlumb makes it easy to create connectors and draw wires between gates, but it didn't fully meet our needs so we had to customize it, leading to quite some hours of extra work.

Dropdown menus

At one point, we figured we wanted to select our proteins from dropdown menus on our wires. Regular HTML dropdown menus are not an issue, however combining this with jsPlumb and bootstrap proved to be a little more difficult. This took more time than we had anticipated but was eventually solved by fully generating the menus every time we needed it. When opening a menu we just adjust its size, to simulate its dropdown effect.

Communication and github merging

Throughout this project we worked with Git and GitHub to share our code. We also used the ticket mechanism of GitHub to keep track of what had to be done. Git is exceptionally good at merging code but cannot always prevent conflicts. If there is too little communication between two people who are working on the same code, merging can become difficult. However, with some reverts and recommitting these problems were resolved.

Restructuring

We had a lot of code clutched together. We had that idea ourselves and the SIG evaluation gave the same result. So towards the end of our project, we did some major restructuring. We shifted some functions around, deleted unused functions, changed some names and did some more testing and commenting. On the client side we put every function into files named exactly after the part of the program they run in. This made it a lot easier to look for functions when you wanted to adjust code, or notice an error in some part of the program.

[5] https://github.com/FelixAkk/synthbio/wiki/Tomcat-installation

Solver issues

During the second to last iteration we discovered a problem with the connection to the solver which was implemented. It didn't show correct results for the transcription and translation reactions, because degradation didn't work. After discussing things with Alexey, Jan Pieter decided to write a solver on its own, which initially took about 12 hours of work. Cleaning up the code and some other adjustments took another 8 hours. During the development of our own solver, Albert worked together with Alexey to fix the problems in the first approach. The result of this is the availability of two solvers in our GUI.

Graph rasterisation, servlet issues

The graph library used in our GUI provides a way to export the graph to different image formats. This exporting requires a special server which is provided by the author of the library, but we wanted to be independent of that, so we decided to provide the rasterisation service by our own server code. At first, it seemed very straight-forward, but quite some time was absorbed by the confusion about Tomcat's class-path. It turns out Tomcat only looks in `WEB-INF/lib/*` and not in it's subdirectories, resulting in a huge stack of `NoClassDefFoundErrors` to fix.

Migrating GUI elements from modals to a workspace tab

The GUI was initially implemented with several key GUI elements such as the output, input and validation results in modals. Although this was not according to the final designs, and has several drawbacks on usability when working on a circuit alongside, this was far easier to implement than in an element that was always present in the workspace. This element had high requirements on flexibility (being able to hide, show again without overlapping the workspace, resize to meet size requirements, go fullscreen in a usable way etc.), which was not provided by any ready made framework or library. This meant implement a lot of workspace management and crafting on these tabs. As work progressed on this it turned out to be a more and more complex task, going far over the estimated time. At some point we had to decide which workspace we'd write the report about and deliver at the last scrum. Although work was nearly completed, there was just not enough time and we left this feature out.

4.1 Simulator problems in detail

In the second to last iteration we discovered that our simulator didn't degrade anything (Figure 9), this was discovered so late because non of the tests targeted degradation, we only tested if the reaction speed changed. Either the solver library (SBMLsimulator) was wrong or the SBML format used was. After some testing and help from Alexey, the problem found was indeed the SBML format.

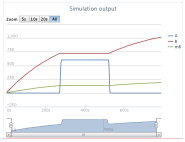

Figure 9: Wrong output, B doesn't degrade.

The SBML format that is used for the simulation requires a list of species and a list of reactions. These reactions each have a set of reactants, products and modifiers:

- reactants are the fuel of the reactions, the reactants decrease as the reaction occurs,
- products are what the reaction produces in the end, the products increase as the reaction occurs,
- modifiers can speed up or slow down the reaction, but their concentration doesn't change in the reaction.

In our simulation we have two reactions: transcription and translation. The transcription reaction produces mRNA (that encodes for a specific protein) from a gene sequence with help from transcription factors. The translation reaction produces the protein from the mRNA. On top of this degradation takes place: the proteins and mRNA degrade over time.

In our SBML file the degradation didn't have a separate reaction, which caused a problem given that the degradation reaction required a different set of reactants, products and modifiers from the transcription and translation reactions.
The transcription reaction produces something from a gene sequence. The degradation reactions need to degrade

to something, but it's not producing something from a gene sequence, so putting the degradation in the same reaction as the transcription (or translation) reaction would be wrong since they have different reactants and different products.

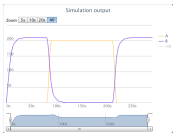

Figure 10: Correct output, B degrades.

To fix this (Figure 10) we split the transcription and translation reactions such that the degradations got their own reactions, this allowed us to specify different reactants, products and modifiers for the degradation.

4.2 Late external requirements

During the project we received several new or changed requirements from the project leaders/TA's. Because these were late this often mean we had to suddenly change our planning to accommodate for the unexpected work. Some of these requirements were:

1. The capability of the user to specify the input values for the simulator in a CSV[6] format, somewhere in the GUI. It was later communicated that the format was amended due to limitations of the initial format, and required that the application was adjusted accordingly.
2. Some requirements and guidelines on this final document were communicated recently.

Although this sometimes brought up some inconveniences in the development process, we then realized that this is actually quite typical of a real world development scenario, and accepted that this is an aspect where flexibility from our part is rightly to be expected. In this line we quickly managed to adjust our system to these new requirements.

In the end this was probably a good exercise, and something we coped with properly.

[6]Comma seperated value. A simple, generic scheme for structuring information for transport. The format is outlined in Handout 2, available on the Blackboard course.

5 Reflection on the teamwork

The teamwork during this project grew better as time passed. We got more used to each other and everyone became a little more enthusiastic during the implementation phase. The first phase, the design phase in which we wrote a lot of documents, passed without any big problems. We had a meeting every week in which we divided tasks and had little brainstorms. These brainstorms brought up a lot of ideas and we were eager to start implementing.

The second phase, implementation, started off very well with good results after the first iteration. Again we had weekly meetings to divide work and to discuss ideas, but a lot of the communication was now done casually. We came together more often while implementing, so the threshold to ask questions and throw out ideas lowered. The rate at which code emerged was pretty high.

This high activity occasionally resulted in work being done twice, but these miscommunications were resolved well. There were no other notable problems, other than perhaps some implementation problems. Such implementation problems were handled by having multiple people look at the problem and discussing possible fixes.

Every project member is proud of the delivered product and is satisfied with how the project elapsed. The next chapter will give a personal reflection of each team member on the project.

6 Individual reflection on the project

In this section each team member gives a summary of their experience and analysis of the project.

6.1 Felix Akkermans

The project was the longest spanning one I've participated in thus far. Especially the synthetic biology/programming life direction really appealed to me. I had very high hopes, and am very happy that with this project we managed to realize something great.

During the start the project took a very different direction than expected, which caused the necessary confusion during the documentation phase when it came definitions and work that was to be expected. At the same time we had a course in project management skills, which I think certainly helped us cope with this, although not ideally as fast and flexible as we could have. But then again this was the first time we were actively and consciously trying to apply these skills. At several occasions I really noticed this made a difference in insight and outcome.

The documentation phase went quite smooth, with an increase of quality in our work as we managed feedback from the tutors.

When we started on the implementation we paid special attention to the novel project management method (SCRUM) we were going to use. I think we coped with this very well, and got up to speed with it's iterative rhythm very well. From the first iteration we have a working presentable product, and managed to stick to this pattern throughout the project. During each iteration we had our share of teamwork and implementation problems. The implementation problems we coped with reasonable I think. Be it slow and gradual, in general the teamwork and improved. I felt that for some the dedication to the project declined quite at some point, but in the end everyone delivered a good contribution to the project.

My work has mostly focussed on the GUI. I feel that my work and criticism was appreciated here and this also made it a great job to work on. I felt that I made a useful and good contribution with directing the GUI, but I wish I had more time to spend at reviewing work of others and helping in their fields. This often left me with a feeling of a lack of overview of the project.

6.2 Niels Doekemeijer

Looking back at this project, I am quite pleased with the overall teamwork and final result. I feel like we had a bit of a rough start, having very few meetings in the first part of the project. This resulted in somewhat lower grades than expected for the first few documents. Writing reports is not my favourite part of projects, but it has to be done and I did my share.

After starting implementation of the product, I feel like the group came more at ease (having more weekly meetings definitely helped, too). I had never worked with SCRUM before, but I can see why it's accepted as a development method as it definitely helps to always have a working product. Demonstrations after each iteration helped us confirm that we were well on our way. We put a lot more work in the implementation than in the reports, but I think both were of very decent level.

I started taking more initiative in the implementation phase and tried to always have an overview over what was happening and what needed to be done. My opinion is that every group member had a phase in which they

did a lot of work and a phase in which they did less than usual. I think the overall activity and devotion to the project was good. This has led to a decent product which has exceeded my initial expectations.

This high activity and devotion sometimes lead to "code-clashes" as more people accidentally worked on the same bit of code. I personally think these little conflicts are no big deal and I rather have work being done twice than work not being done at all. It did not affect our team work and I think we settled these few miss-communications well.

6.3 Thomas van Helden

I am quite amazed about the product we managed to put together. I'm certainly glad we made it with a nice JS GUI. Previous projects were all made in Java leading to horrific GUIs. This project is also the most complex product I've ever made. So in several ways, this was the most challenging IT project I've worked on. Once again, I've learned that a good preparation works a million times better than just starting to code.

My reflection on the teamwork can be divided into three parts: A general part, feedback on my work and communication about the product. In general I was very pleased with the team. We all started every sprint with a meeting. We discussed the issues in the previous sprint and we divided the work. Although some did more than other, we all had an understanding mindset towards each other.

I believe the biggest part was the feedback on me. I personally feel as if I really didn't do the job I could normally do. I was occupied with a lot of things, personal and professional, and so I was unable to do my job properly. This was discussed in the team and I got some really good feedback on how to improve my attitude towards the team.

Communication, in what I've learned so far, is always an issue. Although we did a good job, there was still some miscommunication leading to extra work and stress. It happened a few times that people were working on the same thing without the other one knowing. This resulted in two people doing double work, wasting hours of time. This was no real disaster but it was a shame. Apart from these few points there was nothing to mention with regard to our teamwork.

6.4 Albert ten Napel

I think the project has been very successful, the resulting product is a beautiful, easy to use application that has quite a few useful functions. The teamwork itself was pretty good, we had a meeting each scrum iteration planning on what we would do that iteration. This way I knew what I should do and what was expected of me. In previous projects I was often not sure on what I was to do and the planning was really weak. We also used GitHub, which allowed us to create issues detailing things like optional tasks, bugs and unassigned work. When I was finished with a task and I didn't have anything to do, I could check the issues and pick a new task.

Communication was mostly good, if somebody had a problem he could tell it to the group and we would all think about it. There was some miscommunication regarding tasks that two people were doing at the same time without the other knowing. But in the end this was solved fairly quickly and wasn't that big of a problem.

During the project I learn to better manage things, for example using tools like ant. I also learned how to set up a server-side system with Tomcat and Java Servlets and how to manage libraries in Java. Furthermore we used git instead of svn (which was used in earlier projects), I learned a lot about how to use git.

Test driven design wasn't something that I liked a lot. In theory I like the idea but in practice I find writing all of the tests upfront more of a burden and a lot of time was spent actually fixing tests instead of fixing code.

I didn't do a lot of client side, JavaScript things, because early on I wasn't very interested in that part of the application. Later on I didn't feel confident that I could help with the JavaScript stuff because I wasn't very familiar with the client side. I think I should have done a bit more, since I would've learned some things and with a next project I would like to look more into the GUI side of things.

6.5 Jan Pieter Waagmeester

The first thing I would like to note is that I am surprised by the final product we managed to deliver, it exceeds my expectation. It looks good, works well, has clean code and all the must-haves we defined. Overall, I am also satisfied with the way the team worked together. There were some frustrations now and then, but always room for feedback.

I will reflect in more detail on two areas: the technical development and the team collaboration.

Development & tool use

It was quite some time ago since I did collaborative Java development on larger scale. So I did not have strong opinions about the tool chain to be used, however, during the Software testing class I got my share of Eclipse frustration, so glad to leave that for a while.

I'm used to the command line and enjoyed learning to use new tools like `tomcat`, `ant`, `git`. Tomcat is a very complicated way to have simple server functionality. Since we only needed exactly that, it might have been more efficient to select something more lightweight, however, lots of documentation is available. Ant is a powerful way of to automate boring tasks and made me smile from time to time. Git, especially in combination with GitHub beats every other SCM toolchain I have used before. Easy of branching makes it useful, local commits make code review by peers way easier, but like any other SCM depends on how it's used.

The actual development was often quite straightforward. No very complicated algorithms, mostly pushing simple data structures around. That said, testing things is not always easy. For example, testing JSON responses is not a trivial task if the testing tools have no notion about its structure. I think we could have put more focus on exactly what to test, not only on line or branch coverage.

I really did like the test driven design approach, or more precisely, the fact that tests existed. Some refactoring made the importance of tests very clear: the tested parts worked like before because problems were identified by the tests, but the untested functionality lagged behind.

Team collaboration

Balancing the work during projects is an always an issue. Some people have less time available, some like to invest more time. Some are perfectionists, others do not care about brace placement. I think being one of the more active people does not always justify complaints about others contributing less. Some people could have displayed more initiative from time to time.

Working together in a source code management system can be a source of frustrations: braking *the make*, little care when committing unintentional changes or merges. I sometimes found it hard to patiently correct them and explain the how and why.

7 Future implementations

Throughout this project we implemented a lot of features, but we were not able to implement all our ideas. In this section we will explain some features which could be implemented in the future. Some of these are would-likes we thought of during the design phase, others were ideas we though up during implementation. We tried to make our application in such a way that there is a good base for these ideas.

7.1 Acceptance testing results

From our acceptance tests we got a lot of feedback. One of the main goals is to improve the things mentioned there, because those are the bugs and inconsistencies that users noticed. One example of this, is that saving an unnamed file with a certain name, does not change the circuits name in the description. This was mentioned in the acceptance test as being a feature, but it didn't work. We already fixed some issues we noticed during our earlier acceptance tests. In the appendix you can find more about the results of these acceptance tests.

7.2 Groupings

One of the ideas we had was the concept of groupings. We would use groupings to specify the structure of a circuit. This would be easily converted to JSON and would help to load files to the workspace, directly displaying every circuit with proper spacing and placement. This was not implemented mainly because of time deficiency.

7.3 Protein concentration

Another idea was the ability to set the protein concentrations. Within a nice interface, one could easily specify which proteins would have which concentration, and perhaps even at which time the protein would start. This idea was also encouraged by the TA's but unfortunately we didn't have enough time.

7.4 Highlighting validation error

When a circuit validates it says in green: "Circuit validates!". When there are some errors, it displays in text which errors are present. What would be nice if the element causing the error would light up in the workspace. So for instance, a wire has no protein specified, then that wire would light up. If a gate has an open input slot, that gate would light up, or maybe that input slot.

7.5 The green dot

Currently, there is a bug in the GUI. Sometimes the end point of the last output wire, connected to the output panel, is displayed at the top left corner of the workspace. This bug only occurs once in a while and after hovering over the wire or shifting gates or panels, it disappears. We did not fix this issue because it is probably an issue within the jsPlumb library we use and we did not have time to fix such a small bug.

7.6 Compound and normal gate distinction

Compound gate are now saved as regular gates, but also separately in a compound folder. So to change a compound gate you have to change the regular circuit and resave it as a compound gate. It would nice to have this separated, so compound gates could be altered by itself. We chose not to do this because of time restrictions. We do have the foundation for this to work, so implementing it will be rather easy.

7.7 Compound gate display

This was one of the major things we would like to add. If you drag and drop a compound gate, it shows the circuit it is buildup of. Our original plan was to display these gates as a small gate, on which you could zoom in or double click to see the entire circuit. This was rather complex, and we did not have enough time to implement this unfortunately.

7.8 Input and output fields display open wires

What would be nice for our input and output panel, is to have connection slots. So per available connection there would be an open connection slot generated on the input and output panel. This would make it more visible when a connection has not been made yet. Also it would generate standard points from where they wires would start. This would prevent the weird shifting of wires, which sometimes happens when resizing the window or dragging gates. We saw this point of improvement as too small to actually implement it, with regard to our available time.

7.9 Zoom workspace

Right now we use the browsers standard zoom to zoom in and out. But this also changes the menu and side panel size. This is undesirable, so what we would like is to have a zoom which only affects the workspace. Perhaps in combination with the groupings, you could zoom out to groupings level. In combination with the compound gates you would zoom in on them to see their inner circuit. This was also too big to implement.

A Lightweight SCRUM Plans

The following pages will include the relevant parts of our lightweight scrumplan for each iteration. The colored links and numbers with a hashtag refer to GitHub issues on https://github.com/FelixAkk/synthbio/issues. For each iteration, a milestone is available providing a nice overview of the issues for each iteration.

Comparing actual time used to the original estimation shows something about how accurate the estimates were. A comparison can be found in the following table. The comparison shows that we managed to estimate the work quite precisely, however, uncompleted tasks which were moved to the next iteration are not shown. The last iteration shows a major difference because we could not postpone tasks to another iteration anymore so we invested more time to finish everything.

	Effort		
Iteration	Estimated	Actual	Comparison
#1	65	68	4,62 %
#2	92	87	-5,43 %
#3	93	98	5,38 %
#4	73	72	-1,37 %
#5	77	112	45,4 %

Table 1: List of estimations, actual times and a comparison for each iteration

	Task	Developer	Effort estimated	actual
#2	Server: Tomcat server that handles HTTP requests HELLO WORLD and class scaffolding	Niels/Albert	2*10	17
#3	Server/Client: Design and document JSON format for circuit (`.syn`) and protein list.	Jan Pieter	4	3
#4	Client: JavaScript scaffolding	Thomas	8	10
#5	Client: Create basic GUI	Felix	8	12
#6	Server/client: create and display connection state by ping/pong heartbeat	Niels/Albert	2	3
#7	Client: Show available basic gates	Jan Pieter	3	1 + 2
		Total	45	48
	Optional tasks[1]			
#8	Server: Parse and serve list of Proteins[2]	Jan Pieter[3]	4	16
#11	Cient: Display list of Proteins	Jan Pieter[4]	2	2
		Total	6	18
	Added during iteration[5]			
-	Ant build evironment	Jan Pieter	-	2
		Total	-	2
		Grand Total	65	68

3 Reflection on this iteration

In this section we will give a quick review on this iteration. First we will comment on our initial planning and second we will review the implementation phase and the workflow.

3.1 Planning

The estimated efforts were not that far off the actual time needed for implementation and we are confident that our estimations will improve as the project continues. We did, however, forget to plan a few obvious tasks. For example, code reviewing, tool selection (testing HTML responses and JSON-serialisation) and creating a building environment are tasks that could have been foreseen. Also the time needed for everyone to setup their working environment was underestimated.

A lot of these tasks were one-time efforts, but they can consume precious time. We think the number of these small tasks will reduce, as this was the very first iteration. We now have a starting point that can be built on. For next iterations, we really have to think things through, to make sure there will be as little unforeseen subtasks as possible.

[1] Things from next iterations that could be done if sufficient time is available
[2] Only after list is supplied by Alexey
[3] Original plan was *Niels/Albert*, but Jan Pieter finished his tasks...
[4] Initially unassigned
[5] Significant tasks not planned before.

3.2 Implementation

The implementation phase for this iteration didn't go as smoothly as we hoped it would go, but it did gave us a few points to work on. One of these points is communication; there was little communication between members. This, combined with a low number of commits, made it hard for members to see what everyone was working on and what the overall progress was. Only when the deadline was approaching, the commits started to flow and the project advanced quickly. Late commits make code reviewing and integration testing harder. That is why we want to commit early and commit often in the next iteration.

The client-side scaffolding didn't go as planned either. One of the issues was the QUnit testing. Implementing tests works well, but the feedback once something goes wrong is often rather useless. So the debugging took more time than expected. The Javascript scaffolding was not completed within the scheduled time because of a lack of communication (as more people could have helped). This has been pointed out and will be improved. In the next iteration, this point has priority and will be finished as soon as possible.

We are happy with the delivered code and product. Test-driven development was applied correctly and we think we have built a solid base. For this base we needed a few (previously undocumented) tools:

Build environment

We decided to use Apache ant to build the sources and provide easy access to the JUnit test suites. Currently, four targets are present.

JSON serialisation

Selected the library from http://json.org/java since it is lightweight.

HTML/JSON Unit testing

We have selected HtmlUnit (http://htmlunit.sourceforge.net/) to help test the server applets. This unit acts as a browser (without the GUI) and fetches pages from the server. This helps testing the JSON responses. Unfortunately, we have found no easy way to compare JSON objects in Java. That is why we are comparing the objects as strings. JSON objects are unordered by definition, but we make sure the objects have a predictable layout (alphabetical order).

Project title

After the name confusion about BioBricks we wanted to clear this issue once and for all. In this scrum the term still floated around, for example in the GUI. So we decided on a name for the project to be used from here on everywhere in our work; **project Zelula** (meaning cell in a foreign language [6]). In Scrum 2 this should be apparent in the GUI, documentation, code comment headers.

[6] http://eu.wikipedia.org/wiki/Zelula

	Task	Developer	Effort estimated	Effort actual
#18	Server: Save circuit	Jan Pieter	12	16
#19	Server: Load circuit	Jan Pieter		
#20	Server: Validate circuit	Jan Pieter		
#26	Server: Serve saved circuits	Jan Pieter	3	2
#21	Server: Connection to simulator	Albert	12	15
#27	Server: Convert circuit to simulator input	Albert	3	-
#4	Client: Javascript scaffolding	Thomas	10	12
#22	Client: Gate scaffolding and rendering	Felix	14	- [1]
#24	Client: Drag-and-drop gates to working area	Felix+Niels		7+3
#25	Client: Move gates in the working area	Felix		- [2]
#30	Client: Draw wires between gates	Niels	14	12
#31	Client: Draw input/output wires	Niels		8
#28	Reflection scrum plan 2	Thomas	3	2
#29	Scaffolding scrum plan 3	Thomas	1	1
#32	Code review	Everyone	5 * 4	11
		Total	92	89

	Optional tasks[3]			
-	Server: Simulate circuit	-	-	-
-	Client: Validate circuit	-	-	-
-	Client: Load circuit	-	-	-
-	Client: Save circuit	-	-	-
-	Client: Specify proteins for wires	-	-	-
		Total	0	0

	Grand Total	92	87

3 Reflection on this iteration

In this section we will give a quick review on this iteration.

3.1 Planning

From the beginning of the iteration we spend a decent amount of time on planning. During the first meeting we mainly established our plan, dividing tasks and estimating how much time it would take us. From the previous iteration we learned that things will always take more time, especially if you take into account the extra hours you spend documenting and reading in on your work. So we added a small amount of hours on top of our expectations which brought us closer to our real time spent. We added the hours for code reviewing. So overall we improved on the planning part with respect to the first iteration.

[1] This issue is moved to scrum #3. It was not worked on due to issue #22 taking longer than expected.
[2] This issue was not implemented because it's included in the jsPlumb framework. See paragraph **??**.
[3] Things from next iterations that could be done if sufficient time is available

3.2 Implementation

As for the implementation, things went well. We did run into some issues, but eventually they were fixed. Main problems were drag and drop for our gates and old documentation.

Dragging and dropping gates had some issues. This also had to do with jsPlumb implementation. We managed to drag and drop a gate, so the rendering part is finished. However, we cannot extract a JSON representation of our circuit yet. Old documentation was one of the reasons our work was getting out of sync. We established an HTTP API so we could work based on that.

During the previous iteration we had already thought ahead about one specific problem. We thought that drawing wires would prove to be one of the major issues during this iteration. However, during this iteration we discovered jsPlumb which made things a lot easier and good looking. Using this framework also meant some features are implemented out-of-the-box, saving time and bringing focus more to high-level functionality.

Furthermore, we really noticed the benefits of code review during this iteration. Many small bugs were easily fixed by just having another person review your code.

Connecting to the simulator input took extra time as well, which lead a delayed implementation of the conversion of a circuit to simulator input. This task will be pushed on to the next sprint.

The test driven development for the JavaScript scaffolding didn't go according to plan, because we instantly apply functions to our Ajax results, which makes it hard to test the intermediate results. However, through manual testing we still managed to do some proper testing.

jsPlumb

We chose to use one extra tool which was not mentioned before. This is because we only recently discovered it. We used jsPlumb (`http://jsplumb.org/`) as framework for the (visual) connections in the GUI. This framework makes it easy to define anchors and draw wires between these anchors. Unfortunately, the framework did not match up to all our wishes. Small changes to the library were necessary to allow jsPlumb to automatically create anchors from which multiple wires can be drawn (these kind of anchors are used in the connector from which input signals originate).

	Task	Developer	Effort estimated	Effort actual
#42	Client: Specify protein for wires by drop-down menu	Thomas	3	12
#44	Client: Specify input signals	Jan-Pieter	16	20
#22	Client: Circuit bookkeeping	Felix+Niels	2*15	5+15
#45	Client: Save Circuit	Felix+Niels		5+0
#46	Client: Load Circuit	Felix+Niels		8+4
#47	Server: Simulate Circuit	Albert	2	2
#48	Server: Convert circuit to simulator input	Albert	14	16
#49	Wiki: Simulator	Albert	1	0
-	Code Review	All	5*4	2+3+4+1+1
#50	Final Scrumplan-3	Thomas	2	3
#51	Scaffolding Scrumplan-4	Thomas	1	1
#52	Final report: Key Problems/Solutions	Thomas	4	0
		Total	93	92
	Optional tasks[1]			
#43	Polishing modelling/grid styles and workflow	Felix+Niels	-	1+1
#53	Client: Output visualisation	-	-	-
#54	Client: Simulate Circuit	-	-	-
#55	Final report: Reflection on teamwork	-	-	-
		Total	-	2
	Added during iteration			
-	JSLint ant target and style review.[2]	Jan Pieter	-	2
-	Client: validate Circuit[3]	Jan Pieter	-	2
		Total	-	4
		Grand Total	93	98

[1] Things from next iterations that could be done if sufficient time is available
[2] To ensure some best practices in our JavaScript code, including coding style, we use JSLint/JSHint
[3] For some reason, this task was not in the initial scrum plan #3, but it was an optional task in #2.

3 Reflection on this iteration

During this iteration one of the main problems we encountered were in late commits and a lack of communication when someone was getting behind on schedule. This would lead to a lack of time to correct late work or to help someone catch up with the schedule. During the scrum-reflection we discussed this and noticed that as before this is still a problem. Some improvements were noticed this scrum iteration, and we further emphasized that this is still to be worked on and requires more attention during in our group-process. Everyone agreed on this and plans to show improvements in this aspect.

Extending on this issue we also noticed that at the end of the scrum iteration there were still quite some issues open that should have been closed. This requires more attention so the state of the project issues on GitHub more accurately reflects it's current state in the code.

Another thing we noticed is that our actual time expenditure on code review was below of that what was planned. We adjusted our expectations for the next scrum iterations so this would be more realistic.

3.1 Planning

We had planned to quite some work this iteration even though there was a holiday during this scrum run. We were requested to have a working program during the next sprint, so we looked at what was still missing and aimed to complete a great proportion of that during this sprint. We mainly shoved the Circuit Simulation on the client side to the next scrum run. The rest of the big chunks we tried to tackle this iteration. We also had a little bit of work left from the previous sprint so we planned that in as well. We acknowledged that our final report will be a big piece of work as well, so we decided to work a little bit in advance on that.

3.2 Implementation

During the implementation of all of our functions we encountered some difficulties. We had some difficulties implementing dropdown menus on wires. We didn't expect this but it seems jsPlumb combined with bootstrap gave quite some errors. We didn't manage to fully complete this part so we pushed the remainder on to the next sprint.

Other than this we mainly made a lot of small improvements to our program. We did not encounter great problems, instead we found a lot of room for minor improvements. We took these enhancements and set them as optional in our next sprint.

	Task	Developer	Effort estimated	actual
#73	Client: Resize	Felix	6	-
#45	Client: Finish Save Circuit	Felix	1	5
#66	Client: Acceptance testing	Felix	8	4
#72	Client: Extend input definition	Jan Pieter	6	6
#54	Client: Simulate Circuit	Jan Pieter	2	2
#53	Client: Output visualization	Niels	2	6
#68	Client: Refactor JavaScript	Niels	6	6
#67	Test Coverage	Albert + Thomas	9	4+4
#69	Finish Scrum plan 4	Albert	2	2
#70	Scrum plan 5 scaffolding	Albert	2	1
#42	Leftover Thomas	Thomas + Felix	5	9+8
#52	Final report: Key Problems/Solutions	Thomas	4	1
#55	Final report: Reflection Teamwork	Everyone	5*1	-
-	Code Review	Everyone	5*3	14
		Total	73	72

	Optional tasks			
#71	Client: Compound Gates		12	-
#65	Client: Save input/output gate positions		-	-
#43	Client: Polish circuit styles		-	-
#43	Client: Style of gate while dragging		-	-
#43	Client: Size and drop area of endpoints		-	-
#43	Client: Deleting of wires and gates using delete		-	-
-	Client: Highlighting/selecting gates to move multiple or delete multiple		-	-
-	Client: Input and output fields resize and display open connections		-	-
		Total	-	-

	Added tasks			
#61	Extended input signal modification: toggle ranges	Jan Pieter	-	5
-	Created `new` menu item	Jan Pieter	-	1
#74	Edit circuit (file)name and description	Felix	-	4
		Total	-	10

		Grand Total	73	82

3 Reflection on this iteration

In this section we will give a quick review on this iteration.

Planning

We planned to work on the client side simulating and starting with the simulation results. Furthermore we wanted to fix different small gui issues and refactor the Javascript so it would be easier to work with. We also wanted to get more work done on the final report, since there was still a lot to be done in the report.

Implementation

During this iteration one of the problems we encountered were inadequately communicated commits. This resulted in two persons working on the same thing and committing the work simultaneously. It took some time to figure out what went wrong and how to fix it.
Another problem we encountered was that as soon as the output visualisation was in a usable state, we discovered that the simulation results weren't completely correct. The problem still hasn't been found and we have to spent time in the next iteration to find a solution.

	Task	Developer	Effort estimated	Effort actual
#81	Server: Fix simulation results	Albert	8	5
#86	Refactoring circuit servlet	Jieter	3	4
#87	Server: Read new CSV input	Jieter	2	4
#88	Client: Tooltip graph	Niels	2	1
#73	Client: Resize	Felix	6	4
#66	Client: Acceptance testing	Felix + Thomas	8	7
#71	Client: Compound Gates	Niels + Thomas	12	14
#77	Finish Scrum plan 5	Felix	2	2
#52	Final report: Key Problems/Solutions	Thomas + rest	4	2+1
#55	Final report: Reflection Teamwork	Everyone	5 * 1	5 * 1
#89	Final report: Introduction	Jieter	1	0.1
#90	Final report: Description of product	Jieter	4	4
#92	Final report: Design and implementation process	Niels	5	5
#62	Confirmation on close		0	dropped
-	Code Review	Everyone	5 * 3	6+2+3+2+1
		Total	77	70,1

	Optional tasks			
#43	Client: Polish circuit styles	-	-	dropped
#60	Automated QUnit tests from ant	Albert	-	1
#14	Client: Compile .less server side	-	-	dropped
#13	Client: Assign keyboard shortcuts	Felix	-	3
#84	Client: Input and output fields resize and display open connection	-	-	dropped
#82	Client: Migrate stuff to simulation tab	Felix	-	12
#65	Save input/output gate position	Niels	-	2
#43	Client: Deleting of wires and gates using delete	-	-	dropped
#43	Client: Highlighting/selecting gates to move multiple or delete multiple	-	-	dropped
#91	Client: Table with simulation results	-	-	dropped
		Total	-	18

	Extra tasks			
-	Fixing the simulator: writing our own simulator	Jan Pieter	-	20
-	Exporting graph: serve our own SVG rasterizer	Jan Pieter	-	4
		Total	-	24

		Grand Total	77	112,1

3 Reflection on this iteration

Solver issues

During this iteration we discovered a problem with the connection to the solver which was implemented. It showed correct results for the transcription and translation reactions, but degradation did not work. After discussing things with Alexey, Jan Pieter decided to write a solver on its own, which initially took about 12 hours of work. Cleaning up the code and some other adjustments took another 8 hours.

During the development of our own solver, Albert worked together with Alexey to fix the problems in the first approach. The result of this is the availability of two solvers in our GUI.

Graph rasterisation, servlet issues

The graph library used in our GUI provides a way to export the graph to different image formats. This exporting requires a special server which is provided by the author of the library, but we wanted to be independent of that, so we decided to provide the rasterisation service by our own server code.

At first, it seemed very straight-forward, but quite some time was absorbed by the confusion about Tomcat's class-path. It turns out Tomcat only looks in `WEB-INF/lib/*` and not in it's subdirectories, resulting in a huge stack of `NoClassDefFoundErrors` to fix.

Overtime

It's obvious that in this iteration we went far over our estimated time. This is due to several .factors.

Firstly we found ourselves in an awkward situation when we found the simulator was not working correctly, we set out to fix this, , and because this was an issue that could not be pushed back to another SCRUM iteration, we also set out to write our own solver as an insurance. We managed to fix both the 3rd party solver and complete our own. This was of course a good outcome, but it did take a big bite out of our time expenditure.

Furthermore, this iteration we really started performing the acceptance testing, which brought up some bugs, which required fixing.

Also, because this was quite a long iteration and we really wanted to deliver a great product, we pushed very hard at implementing some should-haves/nice-to-haves.

Acceptance testing results

Thomas van Helden

17 juni 2012

1 Acceptance testing results

These are the results of the acceptance tests we ran at the end of our fifth SCRUM sprint. We looked at all the basics for our program, searching for possible bugs and errors. We went through all the startup processes to check if all menus would display correctly in FireFox 13.0 and Chrome 19.0. We then had a look at if we can open, save and make new circuits. Afterwards we looked at validating and simulating our circuits after setting the inputs. We managed to find some errors and we intend to fix them.

1.1 Basic startup and workspace initialization

This is where we looked at the basic setup of our application. We went to the URL and the page loaded correctly. The connection to the server did not show correctly, but we had it disabled at the time of this test. The handle bars also worked properly. There was one bug in this section: We found that in FireFox the editing of the circuit name and description in the upper right corner didn't work. When pressing the edit option, the name and description should become editable, and they did not. All the other features like tooltips did work according to our expectations.

1.2 File opening

When we tested the opening of files everything worked perfectly. The File menu displayed correctly. Same went for the open file dialog. We were able to open the file with CTRL+O and loaded files we displayed correctly in the workspace, with all the preset proteins. When trying to open an ERROR file it gives an error saying it can't be loaded.

1.3 File saving

With saving we found one bug. When opening a file in FireFox and then using SaveAs on the same file, it should ask confirmation if you want to override this file, but it doesn't. All features work including hotkeys and opening the SaveAs dialog when saving a file for the first time.

1.4 Create simple circuit

With the application you can create a simple circuit. This worked in both browsers. Choosing proteins was also not an issue. There are no double proteins possibilities. Also when deleting gates or wire, the proteins reset correctly. When having a wire as input for multiple gates, the proteins were set automatically. The process of validating the circuit also went smoothly.

1.5 Create and simulate a 2-to-1 multiplexer:

This works fine in both Chrome and FireFox. Building circuits is no problem. The Define Inputs menu looks great and you can set concentrations to different levels. Before simulating the circuit is first validated. If it is invalid is shown correctly.

1.6 Compound Gates

Opening a circuit was not problem. Saving is not a problem. The circuit was correctly loaded into the compound panel and was draggable. When dragging it into the screen it all worked. This was to be expected since it is the same mechanism as for regular circuits.

1.7 General bugs

In general we found some bugs in resizing and in dialog modals.

Any dialog modal which can open with a hotkey one bug. When you open a dialog and close it, it darkens a bit and then it becomes normal again. However, when you repeat this process fast enough, the new dialog opens over the fog of the previous one, which makes the screen darker. When you repeat this process you can fully blackout your screen. In some cases the application freezes, in some cases all usability is still there, you just can't see it.

When resizing the application there are some bugs as well. Although they are hard to fix, since they are not our own libraries, we do acknowledge them. For instance, when simply resizing the browser, the input and output shift along, but the circuit doesn't. This causes the wires to be displayed incorrectly. Also, the result panel does not resize correctly, to fit the graph nicely, at first. When resizing it one bit or rerunning the simulation from the bottom panel, it does resize. Final example is when you run a simulation, then go from the output tab (bottom panel) to the validation panel, and then rerun the simulation, the graph and the bar for the graph are not shown correctly.

1.8 Usability grading

Here the tester can give grades for specific and overall aspects of the usability of the application. Of course written remarks and feedback on certain aspects are also very valuable, and we encourage the tester to make these along with providing grades. On some aspects an explanation is provided in the footnotes.

In the following table, the grades correspond to the following valuations;
1 = very bad, 3 = bad, 5 = moderate/average, 7 = good, 9 = very good.

Aspect	**Score on aspect**
Usability of circuit modelling	8
Usability of circuit management [1]	8
Responsiveness	7
Performance	7
Affordance [2]	7
Presentation of data [3]	6
Visual appeal	7
Overal application usability	**7**

[1] The ease of use and learability of importing, exporting, saving, opening, browsing and editing of circuit details.
[2] The degree in which the UI intuitively implies it's functionality and use.
[3] The quality of communication of, for example; simulation output data, validation results, input data, protein listings