Programming Life - Requirements Analysis and Design

Group 5/E:
Felix Akkermans
Niels Doekemeijer
Thomas van Helden
Albert ten Napel
Jan Pieter Waagmeester

March 9, 2012

Contents

1 Introduction

The purpose of biotechnology is to use micro-organisms like moulds and bacteria for the production of chemical substances. These chemical substances are used as energy (biofuels), in products (bioplastics), in food and in medicine. Not only can these micro-organisms produce, they can also be used to break down chemical substances, for example in the purification of water[1].

Using molecular biology, these micro-organisms can be altered to add, change or remove functionality. This is done by mutating the DNA of the organism. Synthetic biology is the science in which biology and engineering are combined to create new biological functionality[2]. The engineering part is made easier by using biobricks[3]. These "bricks" are simple genetic circuits which provide a basic functionality (for example the imitation of an OR or AND gate) and share a common interface. By combining these biobricks, a new biological systems can be engineered.

In its abstract form, this system can be visualized as a logical circuit, but in the organism the circuit corresponds with a group of molecules (proteins, genes, RNA) that react with each other. How (and if) they will react depends on many different elements, like the concentration and reaction speed of a protein. This reaction can be simulated using a (heavily) simplified model[1].

A logical circuit of biobricks can be defined using the System Biology Markup Language (SBML). SBML is a free and open interchange format for computer models of biological processes[4]. It is a widely supported format that continues to evolve and expand.

In this report, we will describe the design of a visual modeling environment for synthetic biology, in which biotechnologists can design, simulate and validate a logical circuit built using biobricks. We will clarify our proposed system using a list of requirements (2.2-2.4), a few analysis models (2.5), a business object model (2.6), a few dynamic models (2.7) and a preliminary drawing of the interface (2.8). In chapter 3 we will give a planning of the following of the project.

2 Proposed system

2.1 Overview

In this document, we will provide an analysis and design proposal for a visual modeling environment for synthetic biology, in which biotechnologists can design, simulate and validate a logical circuit built using biobricks.

We have made up a list of requirements which answer the following questions. What does the application have to do? What doesn't belong in the basic functionality? What kind of programming language will we use to develop this program and when is it due? First we will go into detail on functional requirements (2.2), so what does the program do? Secondly, we will discuss non-functional requirements (2.3). These answer questions like: What programming language will we use, but also how we will increase the usability of the application. Finally we will have a look at constraints (2.4).

Following the requirements we will specify a few use case models (2.5). These models describe specific scenarios of the program, what steps the user has to take to reach a goal and how the system should react to these steps. The main scenarios we will specify are loading/saving, modeling and simulating. Dynamic models such as sequence diagrams and activity diagrams will visualize the steps and interaction of the user and system in chapter 2.7.

Our business object model (2.5) will clarify the key concepts and their roles of our application. It will give a simplified overview of how proteins, biobricks and the System Biology Markup Language (SBML) relate to each other.

Last but not least, we will show a preliminary drawing of our interface (2.8) and explain why we have chosen for this interface and how it will work. Chapter 3 contains a planning for the rest of the project. This schedule is built up around the deadlines for the other documents and has a preliminary planning for the implementation fase.

[1] Programming Life – Contextproject assignment (Dick de Ridder, Marcel Reinders)
[2] Programming Life – Contextproject kickoff (Dick de Ridder, Marcel Reinders)
[3] http://biobricks.org/
[4] http://sbml.org/Main_Page

2.2 Functional requirements

The following are use cases the application must be able of completing in whatever way.

1. **Circuit abstraction** The application must be able to abstract a designed circuit into a BioBrick, and then treat this as a gate, so it can be used in building of subsequent circuits/BioBricks.
2. **Protein specification** The user must be able to manually specify which protein is used to represent which signal in a circuit.
3. **Available proteins** The application must be able to present the user with an overview of available proteins to assign to signals. The available proteins must be predefined by a user.
4. **Export XML** The application must be able to export a built circuit as a BioBrick in XML (in the SBML schema).
5. **Import XML** The application must be able to import XML (in the SBML schema) into the application as a BioBrick.
6. **Interfering/invalid signals** The application must be able to notify the user that the current circuit design has protein(s) assigned to multiple signals, or still has signals which do not have a protein assigned.

2.3 Non-functional requirements

Non-functional requirements are requirements about how the application behaves and operates (usability, accessibility, availability, performance). In this section we will also cover constraints on the source-code and development (programming language, documentation, tools/frameworks, compatibility, extensibility, maintainability, testability, deadlines).

1. **Documentation** For documentation we will use LaTeXto finalize the reports and will be written in English. GIT will be used for hosting and collaborating on documentation.
2. **Tools/frameworks** As mentioned before, we will use GIT and LaTeX, but also other tools for this project. The program will be written in mutiple languages. We will create the GUI and frontend in JavaScript. The backend will be implemented in Java. The tools we will use to test this are JUnit and Crawljax. Furthermore, we will use the Java library LibSBML to implement the saving/exporting of our BioBricks.
3. **Testability** We want to be able to test every detail. We will look at three major subjects: user-friendliness, errors in the web application and errors in the java backend. To test user-friendliness we will have normal people try our program to test the feel of the program. We will use crawljax to find as many errors in the web application as possible. Crawljax is a sort of bot which tries every possible option in a webbased application. We will use JUnit to test the java backend.
4. **Usability** Because the program is not made for information technology students, we will spend some time increasing the usability. However, the target group consists of scientists, and to be more specific, mainly bio-informatics experts. Because of this, we will not create step-by-step tutorials. Our aim is to make the application basic enough so a regular scientist can work with the application, without detailed tutorials.
5. **Extensibility** We want to create our application in such a way that BioBricks can be exported and reused later as part of even greater BioBricks. As for the application, it is always best practice to make the application extendable, and so we shall aim for that.
6. **Maintainability** Maintainability is also very important, not only for future use but also for the development process. Because of the agile workform we will be maintaining, it is crucial to keep an oversight and a well defined structure within our program. If we lose this, there is a high risk of bugs and superfluous code.
7. **Accessibility** Because our application frontend will be written for the web, we will host it somewhere and it should be accessible from anywhere, if there is internet. From the time of launch of our first version until the end of the project, there should always be a testable version hosted somewhere. It's also likely that we will use modern frameworks such as Bootstrap CSS to provide a high start-level of usability, although this has not been decided yet. The use of such well developed frameworks guarantees a high accessibility of the website.
8. **Deadline** The architectural design will be finished on the 16th of March (16-03-2012). The test and implementation plan will be done on the 23rd of March (23-03-2012). The first version is up for evaluation on the 4th of May (04-05-2012). The final version of the application and the final report are due on the 15th of June (15-06-2012).
9. **Biological plausibility** The application does *not* have to give an biologically accurate or plausible simulation.

10. **Frontend** The GUI frontend of the application will be a website written in HTML+CSS+JavaScript. This means our application frontend will be compatible with all the big operating systems that support web browsers.
11. **Backend** The backend will be written in Java Server Pages.

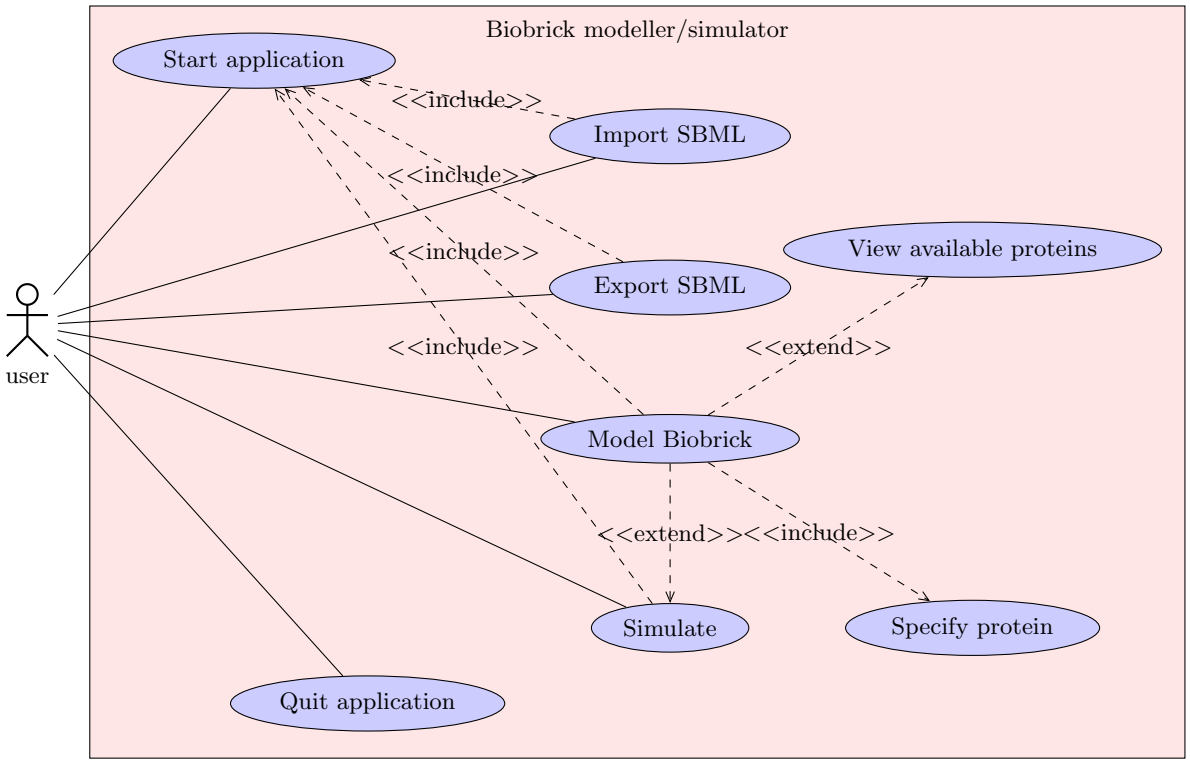
2.4 Analysis models

In this section, we first present some use cases, a Business Object model and some dynamic models.

2.4.1 Use case model, descriptions and scenarios

In the usecase diagram in figure 1 you can see all the different option available for our user. Inbetween starting and closing the application, the user can model BioBricks. Once these BioBricks are modeled they can be exported to SBML. These SBML files can be imported again and used to create more complex BioBricks. During the modeling, the user can specify which proteins will be used as input. Once the BioBricks have been modeled, their output can be simulated. We will describe some example scenario's to further explain this diagram.

Figure 1: Usecase diagram



Start application

Primary Actor	User		
Pre-condition	The application was closed.		
Post-condition	The application is ready for modeling.		
Flow of events		Actor Input	System Response
	1.	Start application.	
	2.		The application opens and displays the initial program state.

Quit application

Primary Actor	User		
Pre-condition	The application is running.		
Post-condition	The application has stopped running and is closed.		
Flow of events		Actor Input	System Response
	1.	Press close button.	
	2.		The application will abort its actions and close.

Specify protein for signal

Primary Actor	User		
Pre-condition	The working area is showing at least one wire.		
Post-condition	The wire has a specific protein appointed to it.		
Flow of events		Actor Input	System Response
	1.	Click the wire or its connector.	
	2.		The selection dialog with available proteins is showed.
	3.	Double click the wanted protein.	
	4.		The selected protein is appointed to the wire.

Model BioBrick

Primary Actor	User		
Pre-condition	The application is in its initial state.		
Post-condition	The application is showing the modeled BioBrick.		
Flow of events		Actor Input	System Response
	1.	Drag and drop elemental parts into the working area.	
	2.		The working area is showing the specified elemental parts.
	3.	Connect the gates with wires.	
	4.		The working area is showing a connected biological circuit.
	5.	Select the protein signals for each wire.	
	6.		The working area is showing a connected biological circuit.

Simulate BioBrick model

Primary Actor	User		
Pre-condition	The application is showing a valid model.		
Post-condition	The application is showing a simulation of the output signal(s) over time.		
Flow of events		Actor Input	System Response
	1.	Press simulate button in menu.	
	2.		The simulate menu is displayed.
	3.	Press the start simulation button	
	4.		The application will show a simulation of the output signal(s) over time.

Import SMBL

Primary Actor	User		
Pre-condition	The application is ready to model and file is a valid SBML file.		
Post-condition	The application has imported an SBML file and the circuit is displayed.		
Flow of events		Actor Input	System Response
	1.	Press brick button in menu.	
	2.		The brick menu is displayed.
	3.	Press the import SBML button.	
	4.		Choose file dialog is displayed.
	5.	Choose file and press OK.	
	6.		The SBML model is loaded from the specified file and displayed.

Import invalid SMBL

Primary Actor	User		
Pre-condition	The application is ready to model and file is an invalid SBML file.		
Post-condition	The application has shown an error message and has not imported anything.		
Flow of events		Actor Input	System Response
	1.	Press brick button in menu.	
	2.		The brick menu is displayed.
	3.	Press the import SBML button.	
	4.		Choose file dialog is displayed.
	5.	Choose file and press OK.	
	6.		An error message is shown stating that given file is corrupt and the application has not imported anything.

Export SBML

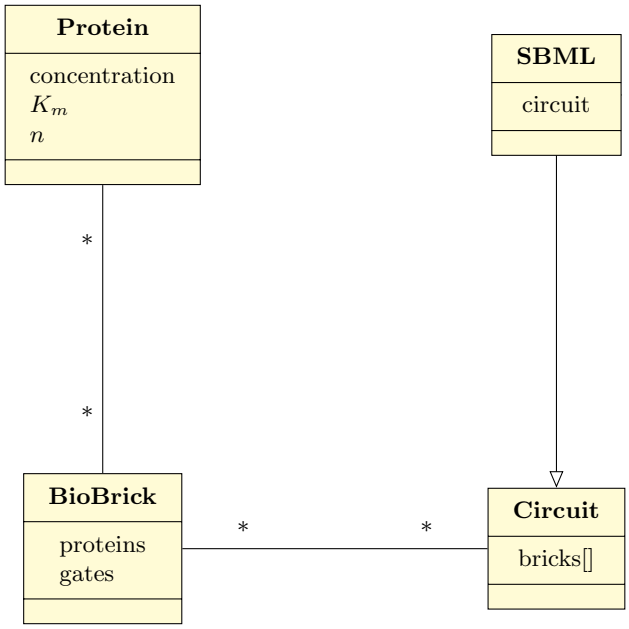
Primary Actor	User		
Pre-condition	The application is showing a model.		
Post-condition	The application has exported an SBML file.		
Flow of events		Actor Input	System Response
	1.	Press brick button in menu.	
	2.		The brick menu is displayed.
	3.	Press the export SBML button.	
	4.		Choose file dialog is displayed.
	5.	Choose file and press OK.	
	6.		The SBML is saved to the specified file.

2.4.2 Business Object model

The objects of our model are the proteins, the biobricks, the circuits and the Systems Biology Markup Language.

The protein is the smallest building block of the model, with proteins and gates we can form Biobricks. These Biobricks can then be connected to each other to form a circuit. The Systems Biology Markup Language (SBML) can be used to represent and describe these circuits. So from the circuit the resulting SBML can be obtained and vice versa.

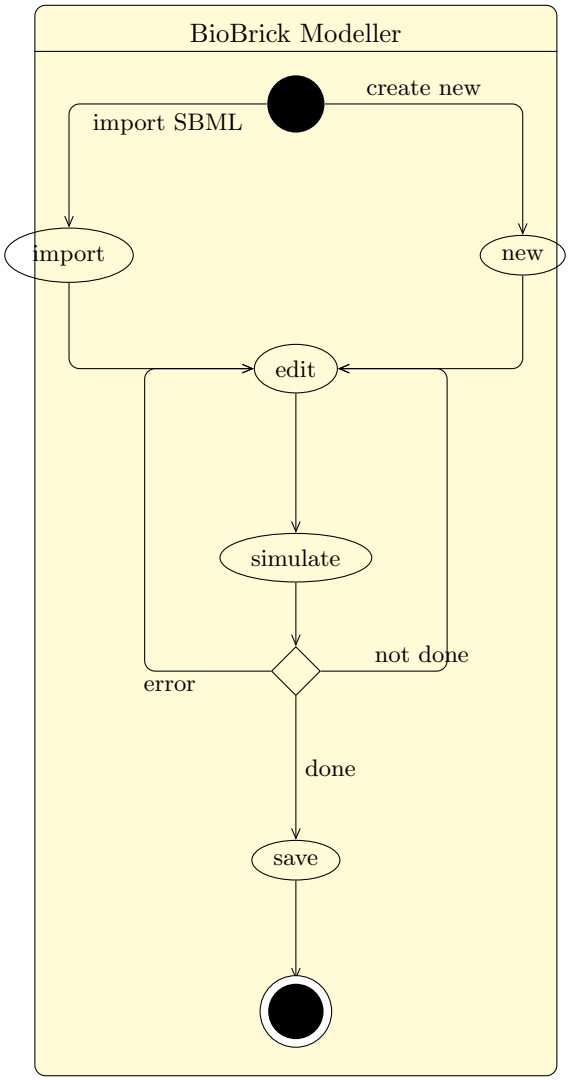
Figure 2: Business Object model



2.4.3 Dynamic models

When the user enters the program, two options are available, he can import an existing circuit (SBML-file) or create a new one. The user then edits the circuit to suit his needs. After the editing the circuit can be simulated, if the user choses the proteins wrong an error will occur and more editing will be in order, if the user decides he is not done yet with editing he can continue after the simulation. When the user is done he can save the circuit as a SBML-file.

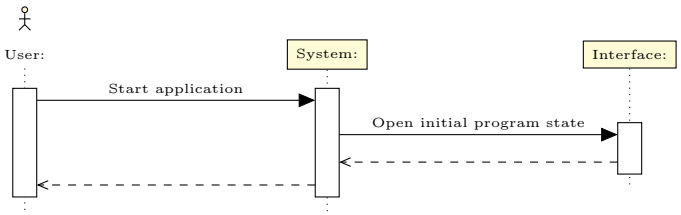
Figure 3: Activity diagram



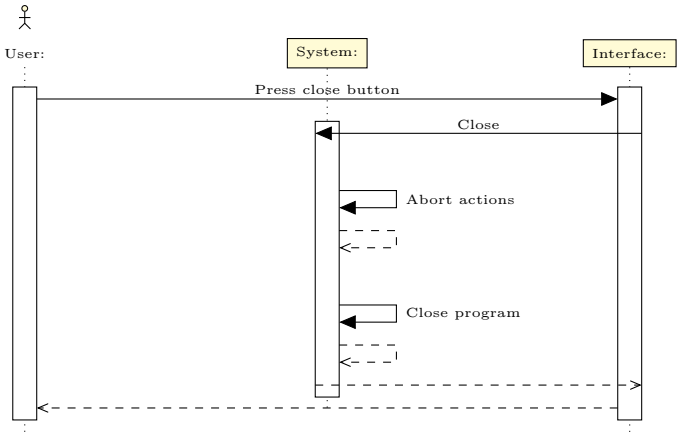
Sequence Diagrams

Following are sequence diagrams for each of the use cases. They visually describe the flow of events. Note that these diagrams are all high level and the technicalities might differ in the architectural report.

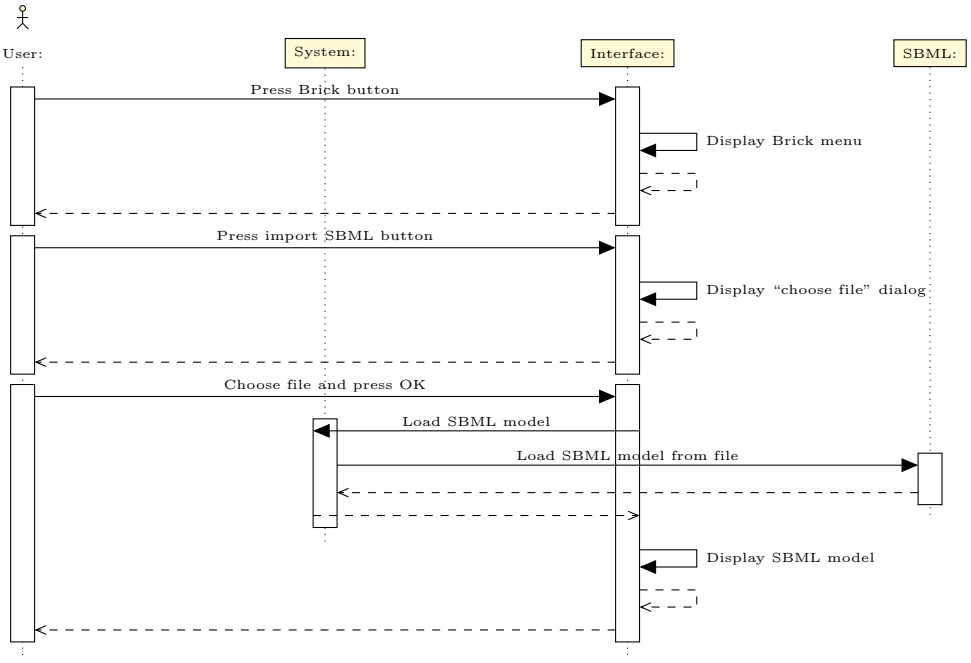
Use case #1: Start application



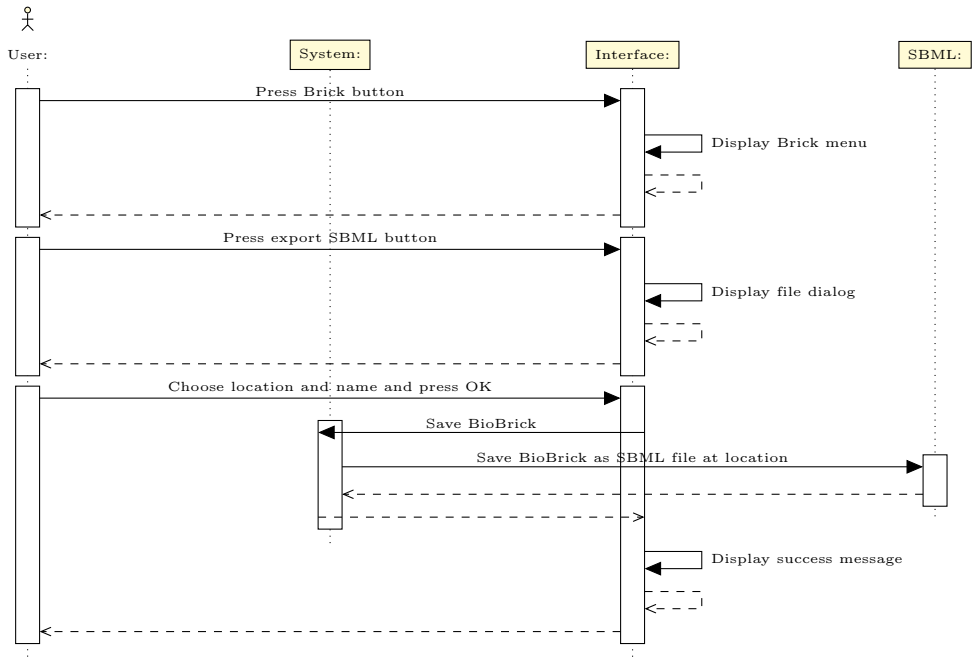
Use case #2: Quit application



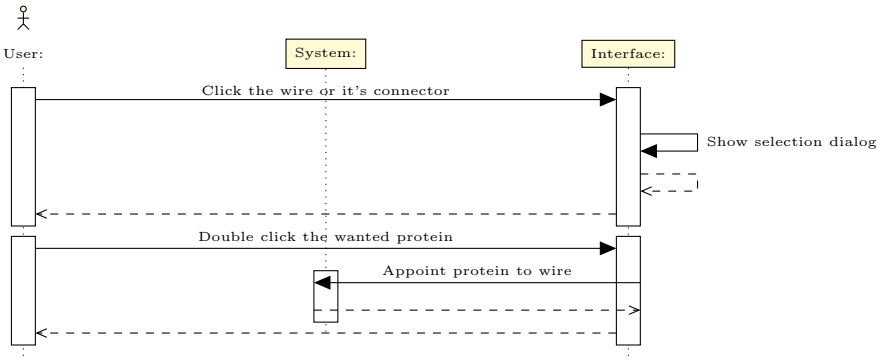
Use case #3: Import SBML



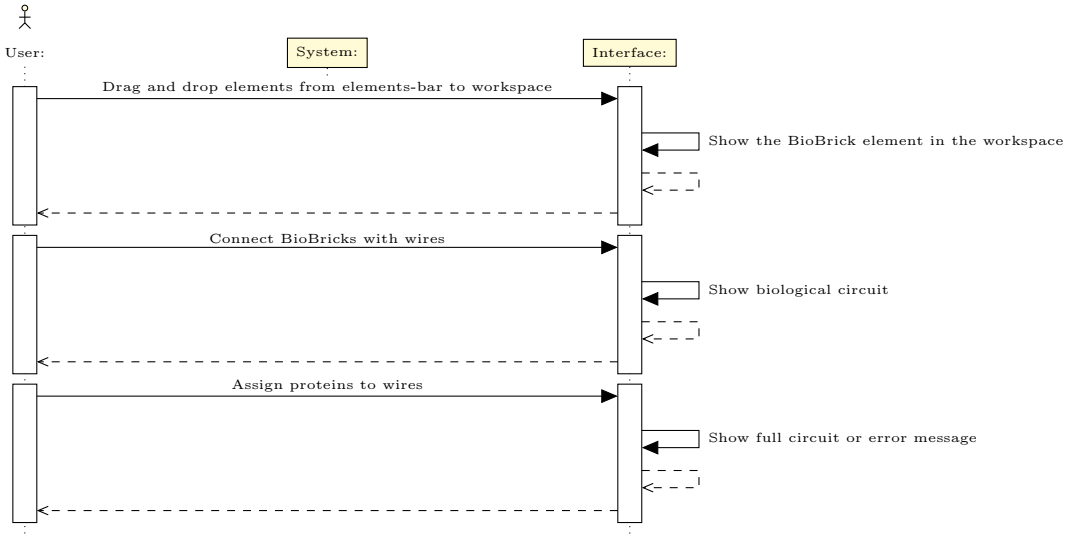
Use case #4: Export SBML



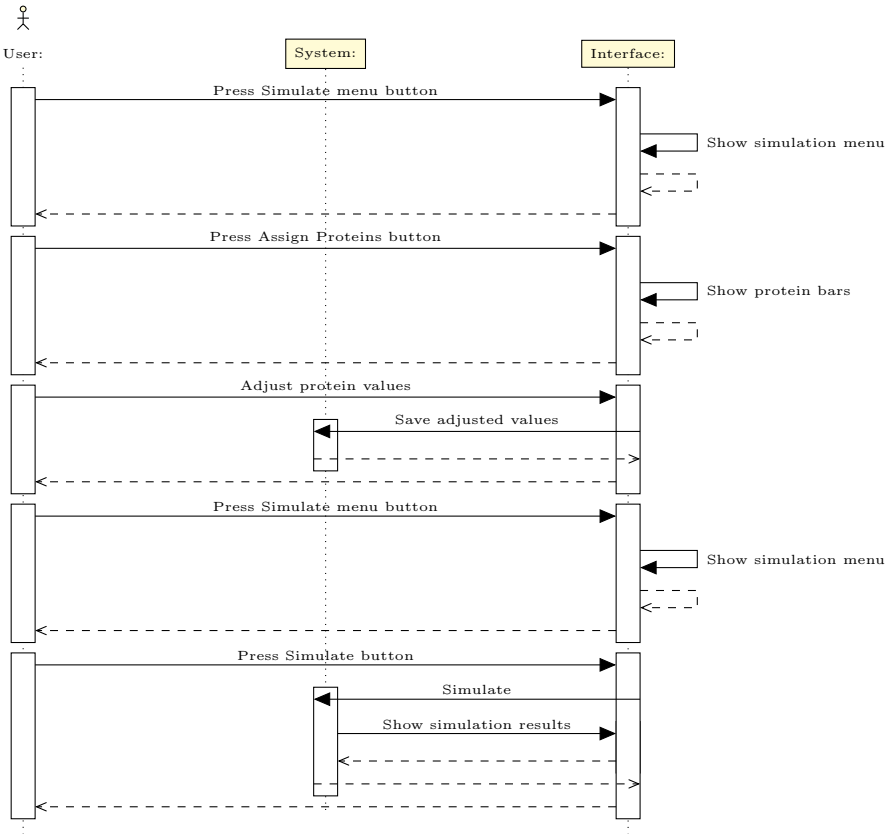
Use case #5: Specify protein for signal



Use case #6: Model BioBrick



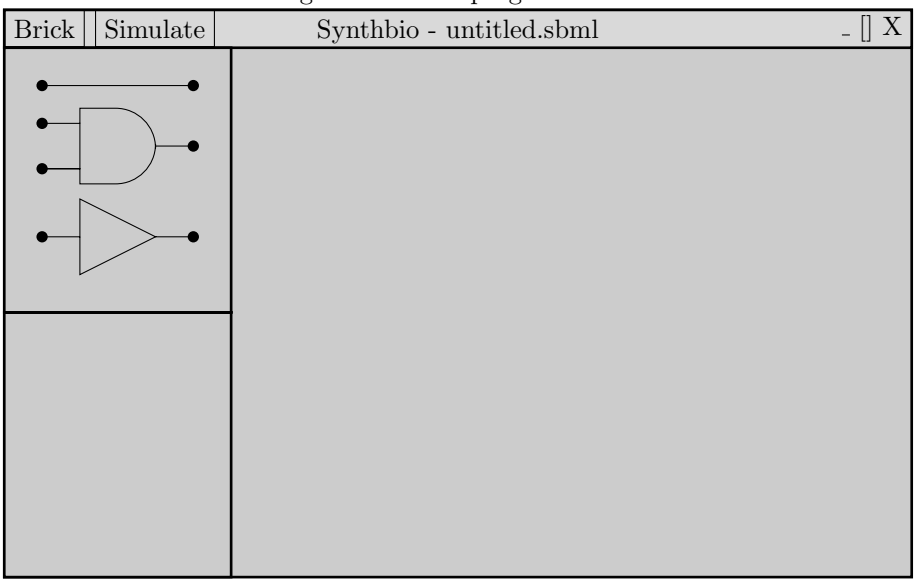
Use case #7: Simulate BioBrick



2.5 Interface

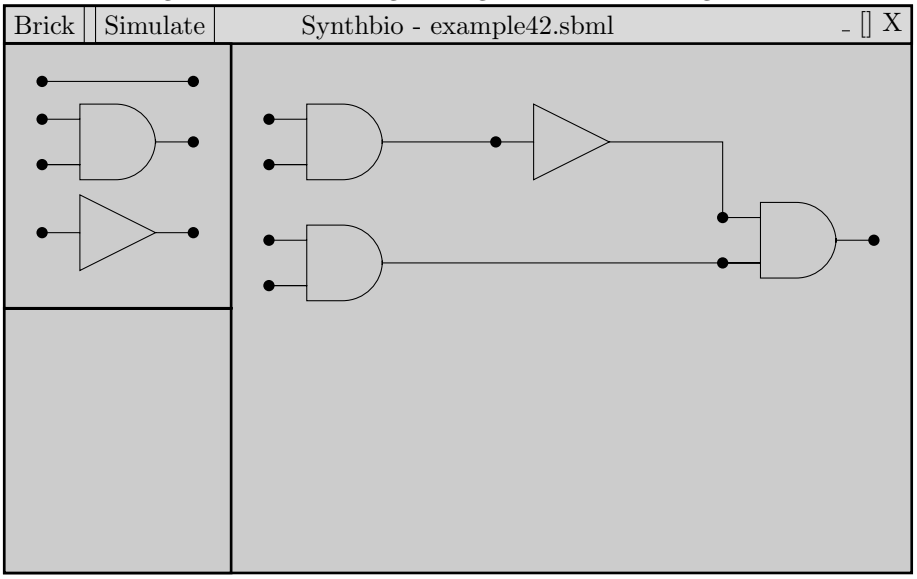
When starting the program for the first time, the user is presented with window in build mode (figure 4) containing a sidebar with the three elemental building blocks: wire, an AND-gate and a NOT-gate. The lower part of the sidebar is reserved for user-defined *substructures*, as mentioned in functional requirement 1. The remainder of the window is a working area which is initially empty. On the top bar two menu items are present. One for basic file actions like export, save and import, the second for actions concerning the simulation of the brick.

Figure 4: Initial program state.



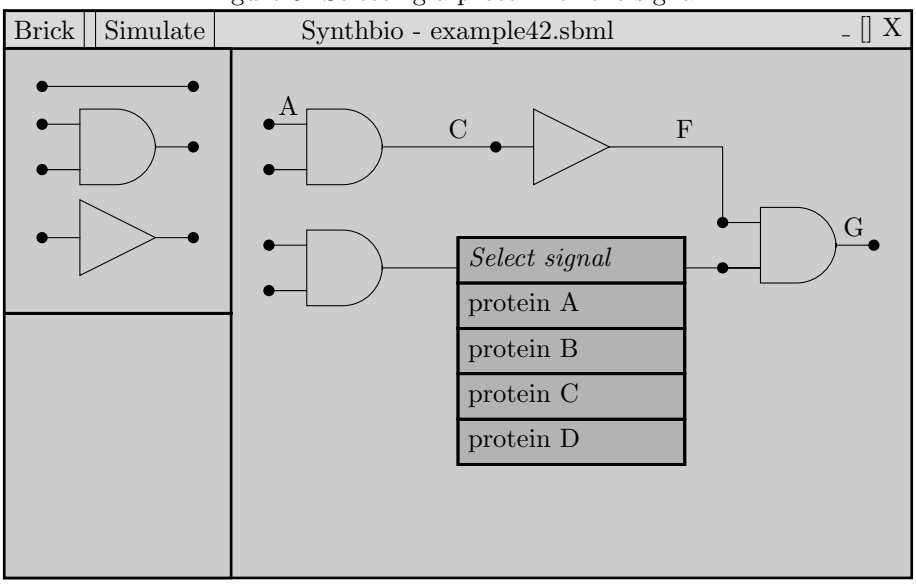
To model a circuit, the elemental parts can be dragged into the working area, the gates may be connected using wires. The interface state after adding some wires is proposed in figure 5.

Figure 5: After adding some gates to the working area.



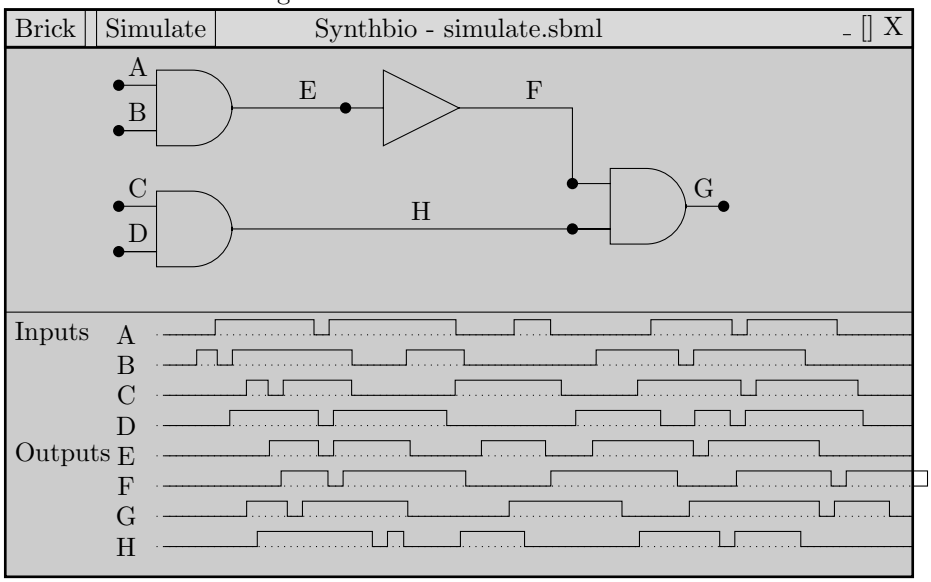
After building the circuit, the user has to select which protein to use for each signal. This can be done during the building process aswell. In figure 6 the protein selection dialog is shown for one signal, for some other signals the representing protein is shown above.

Figure 6: Selecting a protein for the signal.



If the user is satisfied with the circuit, the simulate mode can be selected. For each input protein, the concentration as a function of time can be changed. The program will calculate and display the results as shown in figure 7.

Figure 7: Simulation of the BioBrick



3 Planning

Date	Deliverable/Event
02.03.2012	Deadline **draft** RAD document
09.03.2012	Deadline RAD document
Q3 W5	First Peer-Review
10.03.2012	Deadline **draft** Architectural Design document
14.03.2012	Deadline **draft** Test and Implementation plan
16.03.2012	Deadline Architectural Design document
23.03.2012	Deadline Test and Implementation plan
04.05.2012	Deadline upload source code to SIG for first evaluation)
Q4 W4	Second Peer-Review
24.05.2012	Feedback SIG evaluation by Eric Bouwers (10:00-11:00, room TBA)
06.06.2012	Deadline **draft** Final Report
15.06.2012	Deadline Final Report
15.06.2012	Deadline upload source code to SIG for final evaluation
20.06.2012	(morning) student presentations within each context
20.06.2012	(afternoon) plenary presentations of the best projects per context

4 Glossary

BioBrick Isolated and documented cell function to be reused in future projects. For example, the production of a light emitting protein when some other protein is available.

Circuit What we think of as a circuit is actually just a cell wherein all signals and gates are proteins and parts of the DNA, mixed together without any separation.

DNA Deoxyribonucleic acid is a very long molecule containing the information needed to support the life in allmost all creatures.

Protein a complex molecule produces by certain processes in the cell. In turn it can activate other processes, or perform functions such as emission of light or change color.

RNA Working copy of the information in the DNA. This copy is then used to synthesize proteins.

SBML *Systems Biology Markup Language* is a XML-based format for storage of various computermodels of biological processes.

Simulation Execution of a set of differential equations in order to predict the output of the modelled circuit

Transcription Process of making the mRNA-copy of the DNA master.