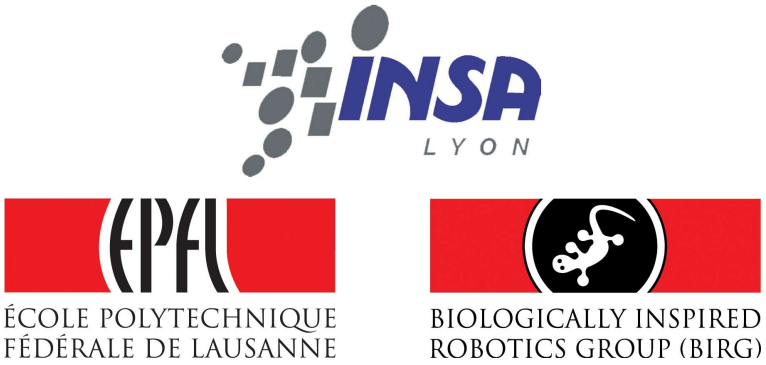
DRAFT COPY ONLY

Roombots Robot-User 3D Interface

Project Specifications



Sébastien GAY

Department of Computer Science

Institute of applied sciences (INSA) Lyon

A document submitted for the degree of

Engineer diploma

Yet to be decided

DRAFT COPY ONLY

Contents

Chapter 1

Introduction

THE Biologically Inspired Robotic Group has been chosen to design a full system of adaptive auto-organized furniture. The main application of this project would be to be able to design and arrange rooms at distance without needing anyone to move or change the furniture. The Learning Center of the EPFL is particularly interested by this features. This implies that the the furniture should be able to move by themselves and change shape. For this it was decided to use modular robots (i.e. a set of identical robot units) as building blocks for this furniture. Our work is included in this project. Our goal is to design a graphical interface enabling the users to compose robot structures by easily assemble modules (in simulation) to compose custom furniture and then arrange it in a room. A possible application would be for the organizer of a meeting to arrange the room on his personal computer or on a dedicated terminal and then send this arrangement (via TCP-IP for instance) to the controller of the robot modules which would automatically reconfigure and reproduce the simulation model of the room at real scale.

The purpose of this document is to sum up the requirements of the customer expressed during the interviews. We will also present the general structure of our solution so that the customer can validate that it is in accord with his expectations.

Chapter 2

Expectations of the Project

IN this chapter we summarize the requirements that have been specified by the customer during the interview. In the first section we present the functional requirements, and in the second one the non functional ones.

2.1 Functional Requirements

In this section we present the demanding in terms of functionality of the software. These have been classed into four groups described next.

2.1.1 Building Robot Structures

The user should be able to build complex robot structures using virtual modules as building blocks. It should be possible to select the modules to use, move them in the world (i.e. translate and rotate them) and connect them to one another. The user should also be able to move some parts of the modules according to their degrees of freedom, to orient some part of the structures with respect to others. For our application the user would typically use modules to build furniture like structures. The modules would be designed in advance using Webots© and be imported to the software somehow. It could be by importing a file created using Webots© or by directly interfacing our software with Webots© as a plugin for example.

2.1.2 Saving Structures

The robot structures that have been built by the user should be possibly saved to be reused before. This could be done in a database containing all the structures of one user or in files (i.e.

one file per structure). It should anyway be possible to export the configurations to Webots© either by saving them as a Webots© files or by a direct export.

2.1.3 Building Full Rooms

The user should be able to use the robot structures that he or another user built as new building blocks to compose either more complex structures or arrange these structures in the world. For our main application this shall enable the user to arrange rooms using preliminary build furniture. This "worlds" shall also be saved similarly to robot structures.

2.1.4 Visualizing Reconfiguration Sequences

The ability to move modules around the degrees of freedom of others should enable the user to explore the space of configurations. This can be done manually by moving the joints of the modules one by one by hand or automatically by importing sequences of movements. These sequences could be imported from files or directly from the software generating them.

2.2 Non Functional Requirements

In this section we present the expectations that are not anymore linked to functionalities but constitute inner characteristics of the solution.

(modify this with real Mac Cay (or whatever) names)

2.2.1 Accessibility

The users being likely to have no knowledge in computer science or robotics, the user interface that we are designing should be simple enough to be used by anyone. The interface should involve common and intuitive controls i.e. buttons, check-boxes, text zones etc. For instance the mouse should be used intensively for movements of the world, modules etc. since its use is very intuitive to most users.

2.2.2 Portability

We want our software to be usable by the greatest number so it should be operating system independent. The user interface should be portable to Unix and Windows systems. So we will use only ANSII coding languages and portable libraries. This could result a single version for all OS (like with Java for example) or more likely one version per OS (for instance if we use C++ and OpenGL/Glut).

2.2.3 Interoperability

Our software should be fully compatible with Webots© and the software generating reconfiguration sequences. This compatibility could be achieved by directly integrating our graphical interface into Webots© by making it communicate with Webots© (using TCP-IP or inter threads communication systems) or by ways of common files. The choice of the final solution is discussed in Section 3

2.2.4 Generality

Our software should be very general. That means it should allow the user to import any predesigned module independently of its shape and degree(s) of freedom.

2.2.5 Extensibility

Due to the tight time-line of the project, we might not be able to develop all the functionalities expressed in this specifications document. So, the project may be taken over by some other developer. Our solution should thus be thought to be modified and extended easily. Typically, the final structure of the solution, communication model, shall be described precisely. The source code shall be clearly commented, the name of the variables, functions and classes of the solution shall be meaningful and precise conventions of naming shall be used.

Chapter 3

Comparative of Two Solutions

IN this chapter we discuss the advantages and drawbacks of two solutions to interface our solution with Webots©. The first solution, called *Plugin Solution* consists of including our solution directly in Webots© as a plugin, a dynamic library for example. The second solution would be a *Standalone Application* interfaced with Webots© and other necessary applications by files or inter-process communication.

The next section summarizes the pros and cons of both solutions and the following one explains our choice.

3.1 Pros and Cons of the Two Solutions

Table 3.1 presents the good and bad points of both solutions.

	Pros	Cons
Plugin Solution	Included in a well know application : Webots©.	Necessitates the understanding and modification of the Webots©source code.
	Full compatibility with Webots©.	The full solution has to be installed to use one feature.
	Part of a homogeneous complete solution.	
Standalone Solution	Independent from any other application.	Can be different from what the users are used to.
	Fast to develop.	Compatibility is not implicit.
	Can be installed without Webots©, on a dedicated terminal for example.	

Table 3.1: The main pros and cons of the plugin and standalone solutions

3.2 Choice and Justification

Also the final choice will be let to the customer, we recommend to use the standalone solution. Indeed we think that it would be useful to have our application (or at least part of it) available for every users. These users may have no knowledge about Webots©and thus it would be more a burden that an advantage for them to have to use Webots©. Furthermore, we believe that it would be a good feature to have our application embedded in dedicated terminals or PDAs. These devices may not have enough storage capacity and computation performances to run Webots©.

The compatibility with Webots© and other applications could be achieved by importing and exporting proper files. This necessitates no changes in the source codes of the other applications.

Finally Webots© is a big application and getting into its source might be time consuming. A standalone solution allows us to only care about the structure of a Webots© file.

We chose to use C++ with OpenGL and Glut for their good performance, full portability and because these libraries are largely used in the 3D imagery community.

Chapter 4

Description of the New System

OUR application will be divided into several packages, each of which should be dedicated to one class of users. Before defining these modules the first thing to do is thus to sum up the actors of the system and their roles, rights and goals.

4.1 Actors

This section describes the specificities of the different classes of users of our system.

4.1.1 Users of the Learning Center

These people can be anyone and thus have any kind of knowledge in the use of computers. They are not supposed to have received any training in robotics, computer science or more specifically on our application. They are not supposed to know about the details of the robots functions. Thus they should not be able to interfere with the reconfiguration sequences or locomotion gaits. They goal is to quickly and simply arrange rooms, chose and position furniture. Some special users (any user ?) could be given the right to design new types of furniture.

4.1.2 Members of the BIRG lab

The members of the BIRG lab have all received academic education and are specialists in robotics. They are directly linked to the reconfiguration and locomotion processes since they have take part to the design of the system. These people may thus want to be given more possibilities than the lame user. These users will be able to animate the furniture they have composed to explore the space of possible configurations or import precomputed reconfiguration sequences to visualize them.

4.2 Description of the modules

To satisfy the needs each of these users and be able to define rights, our system will be decomposed into three main modules :

- *Furniture Design* module
- *Room Arrangement* module
- *Reconfiguration Sequences Visualization* module

Furniture Design module contains all functionalities to design new furniture by easily assembling modules. This includes loading predefined modules, moving them in the world, rotating them, connect and disconnect them, moving their degrees of freedom. The designed furnitures can then be saved. It will also be possible to load a furniture to modify it.

Room Arrangement module enables the user to load preliminary designed furniture (generic or custom) and position them in a room. The dimensions and specific constraints of the room should be loadable so that furniture could be placed at valid positions. These arrangements shall also be saved and sent to the robots.

Reconfiguration Sequences Visualization module is meant to enable advanced users to import precomputed reconfiguration sequences for a given furniture and visualize the robot changing shape. The user shall also be able to move subsets of the structure to manually explore the space of possible configurations.

Figure 4.1 represents the communications between the modules and the rest of the systems. Files imported and produced are specified.

4.3 UML model

In this section we present a general UML model of the system to give a first idea about it's global mechanism. Figure 4.2 represents the class diagram of our application, with the parts belonging to each of the modules.

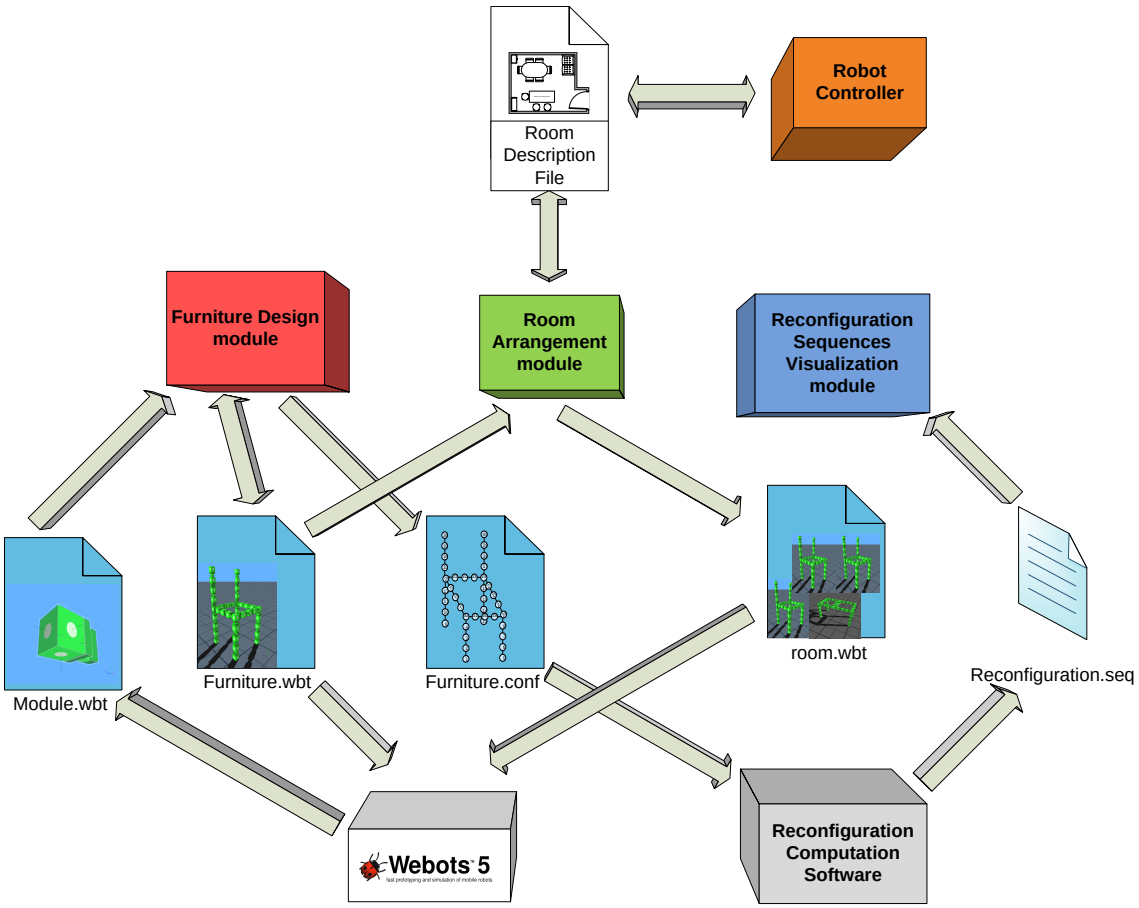


Figure 4.1: The communication diagram between our application and the rest of the system.

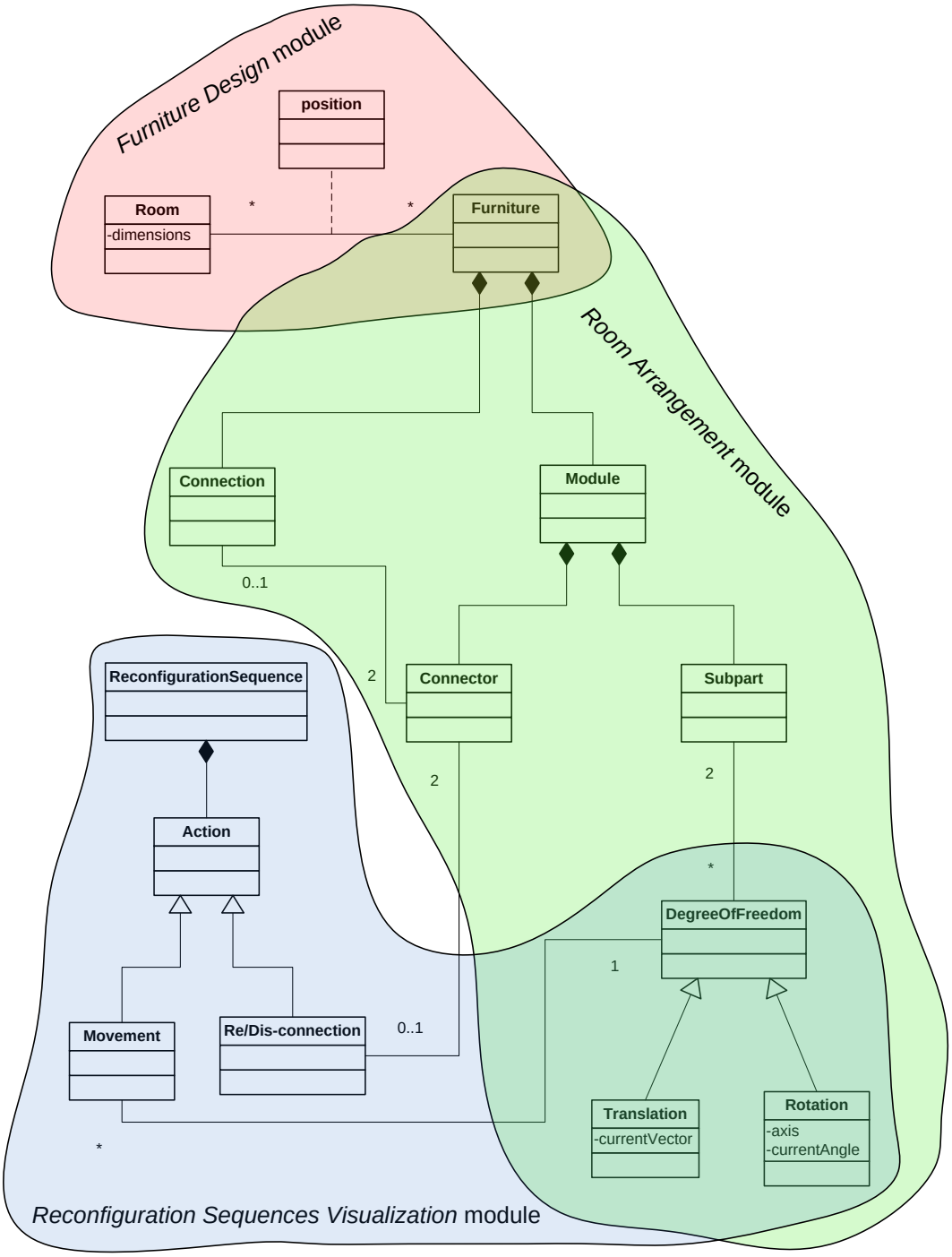


Figure 4.2: The UML class diagram of the system. The parts belonging to each module are precised

Chapter 5

Conclusion

IN this document we summarized the requirements of the customer. For better identifying and formalizing these requirements we sorted them into functional and non-functional requirements.

Then we describe a plugin solution and a standalone solution and compare their advantages and drawbacks.

We made a list of the users of our application, their specificities and needs.

We finished by describing the general structure of our solution, the different modules, communication model and class diagram.

This documents initiates the project and shall be used by the customer to confirm his demands and by the engineer as a reference.

Appendix A

Interface Layout

THIS appendix presents models of the interface that we aim at. Figure A.1 represents the main window, where the user can chose what he wants to do. The next figures present the interface layout of each of the modules. Figure A.2 represents the interface of the *Furniture Design* module, Figure A.3 the one of the *Room Arrangement* module and Figure A.4 the one of the *Reconfiguration Sequences Visualization* module.

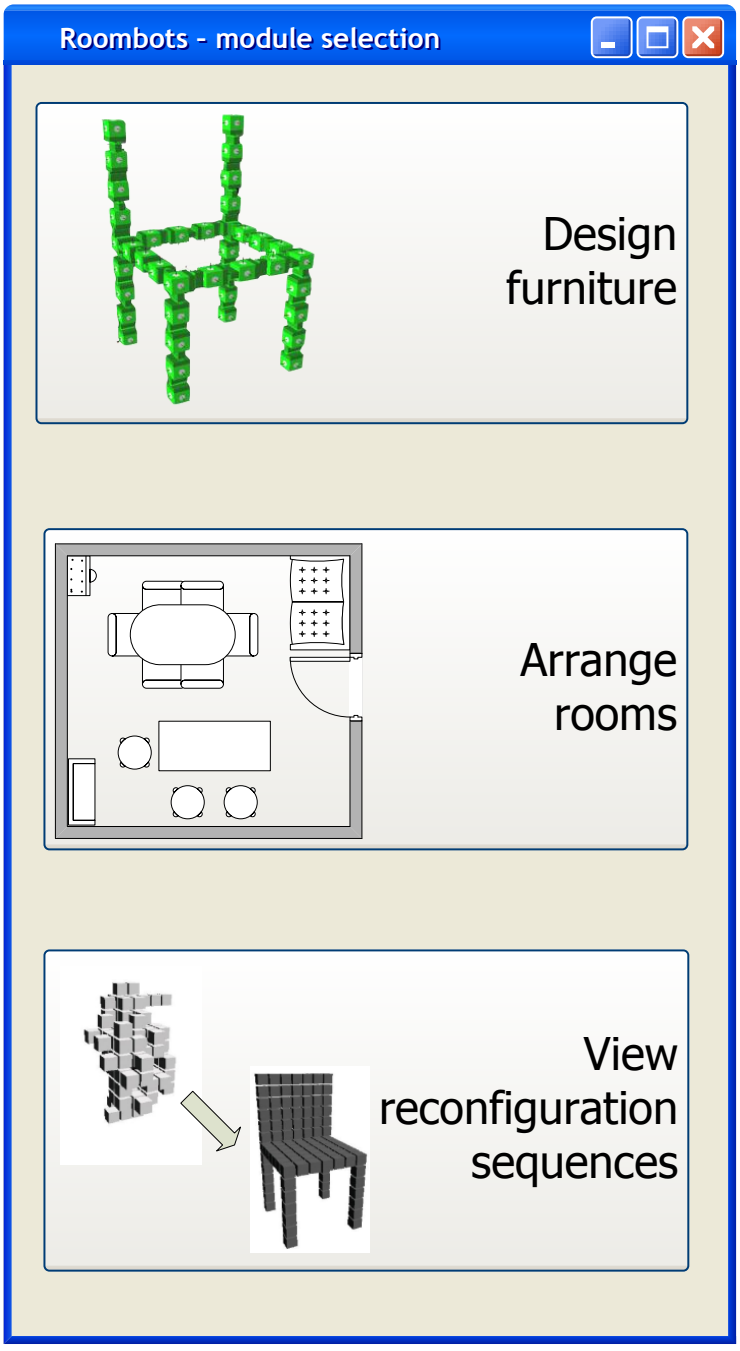


Figure A.1: The main window : the user can chose which module he wants to use.

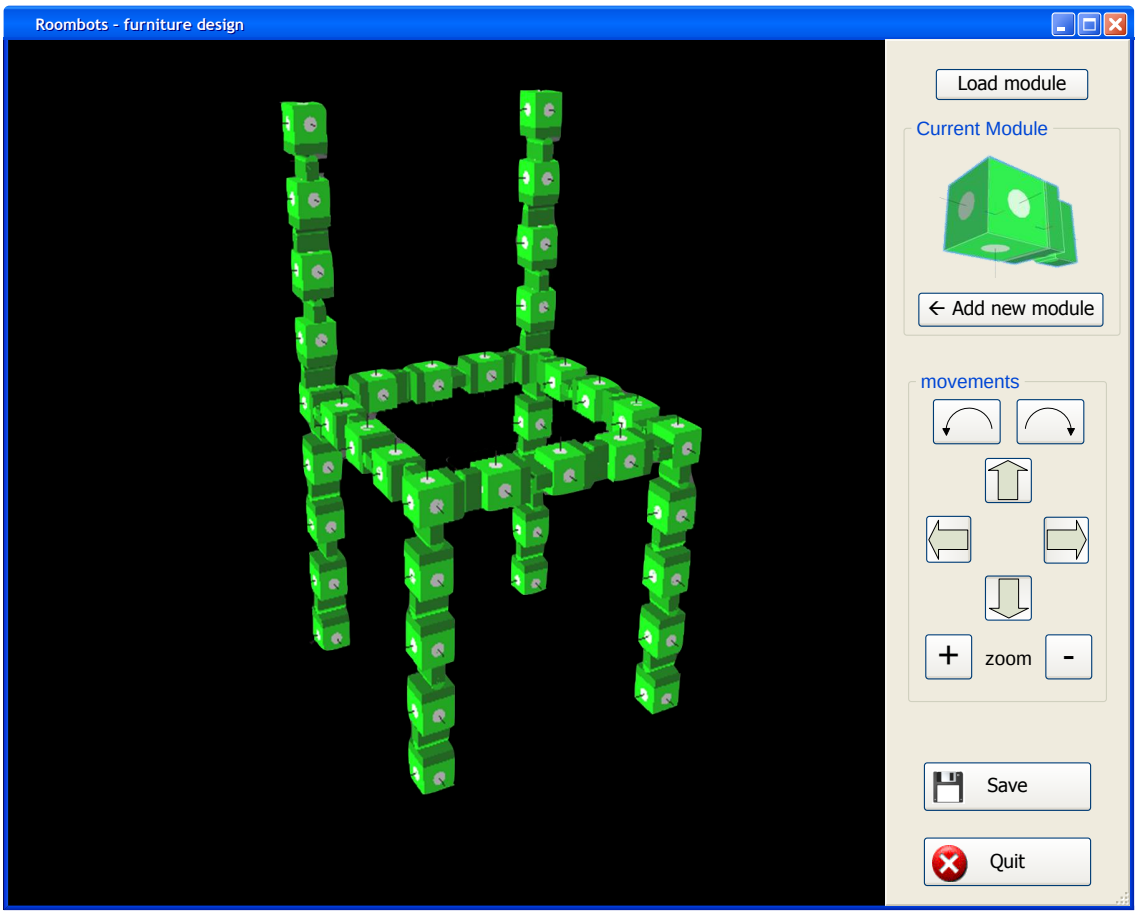


Figure A.2: The furniture design window : modules can be loaded and a mini-model of the module is shown on the top right corner. The modules are assembled on the left sub-window. Movements can be made with the mouse or with the dedicated buttons on the right.

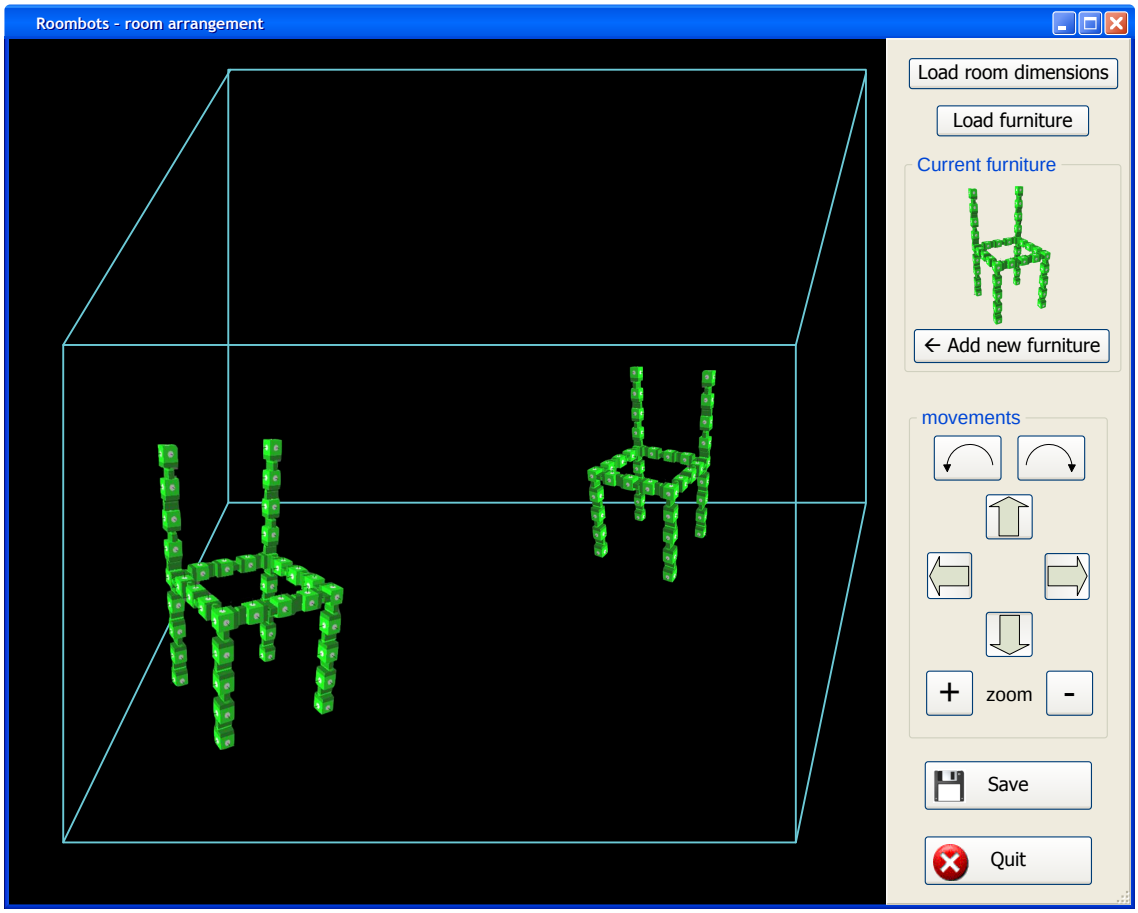


Figure A.3: The room arrangement window : The dimensions of the room can be loaded from a file. Furniture can be loaded and a mini-model of the furniture is shown on the top right corner. The room is arranged on the left sub-window. Movements can be made with the mouse or with the dedicated buttons on the right.

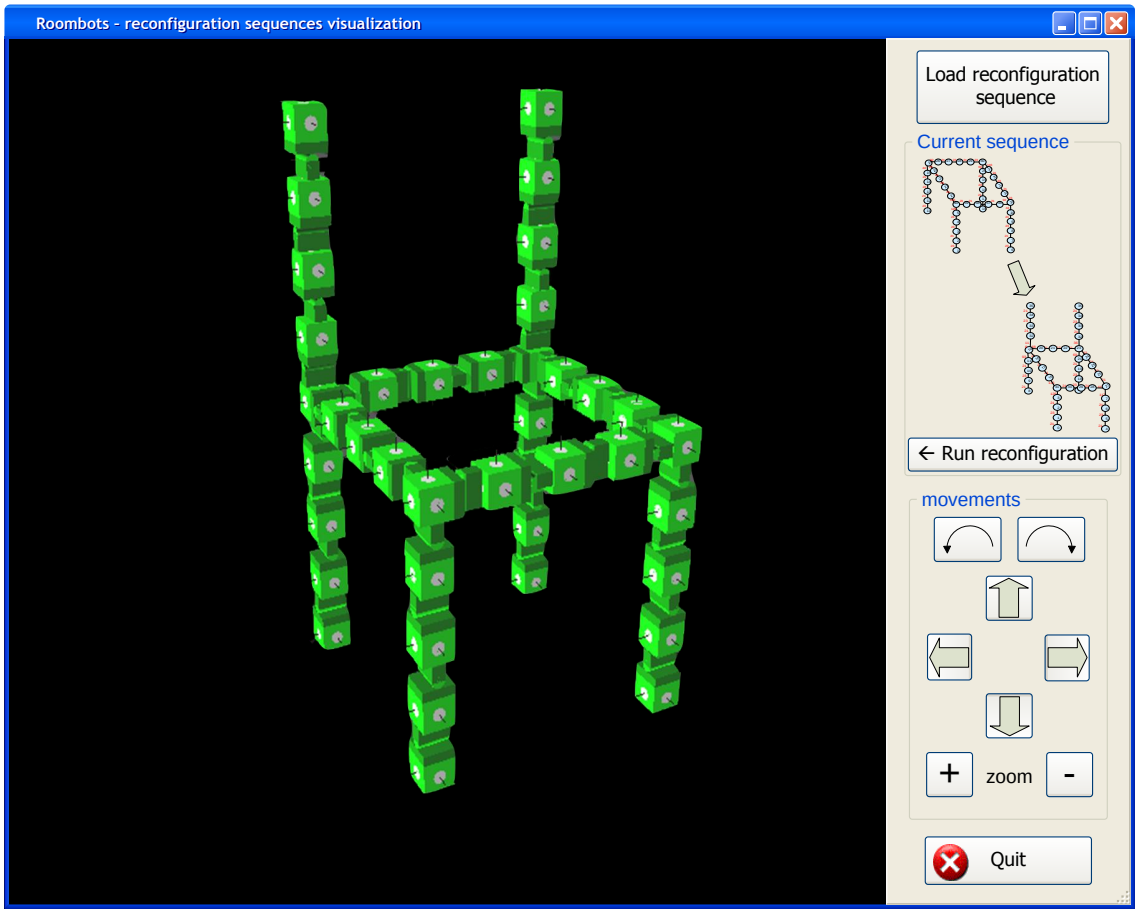


Figure A.4: The reconfiguration sequences visualization window : sequences can be loaded and a mini-model of the sequence is shown on the top right corner. The sequences can be viewed and manipulated from on the left sub-window. Movements can be made with the mouse or with the dedicated buttons on the right.