Chapter 1

1 Introduction

"The only true wisdom is in knowing you know nothing."
— Socrates

Chapter Objectives

1. Explain the motivation and vision for a project.
2. Introduce the methodology used during the project.
3. Familiarize the reader with writing style and terminology used in report.
4. Emphasize the contribution of the report to science.

1.1 Motivation and Vision

Nowadays simulators are essential tool for experimenting with robots. Using a simulator, developers can experiment with hardware design, programming controller, interaction of robot with environment and so on. The main reason for doing so is to minimize the development costs of robot. At the same time, simulators are developed for each or group of robots specifically. This increases the time needed to develop the robot, because development of a simulator is also a time-consuming task. In the specific case of modular robots, a generic simulation tool named USSR (Unified Simulator for Self-Reconfigurable Robots, [1]) has been developed. The main advantage of a generic simulator is that, robots are constructed and experimented with quicker than developing a new simulator for each robot. Currently USSR and Webots [2] are the only two such representatives from the field of self-reconfigurable modular robots. The problem noticed by users of USSR is that the time to construct the modular robot morphology[1] from modules, assign behaviors to modules and starting a primitive simulation in three dimensional space takes a significant amount of time. Furthermore, it is difficult for beginners and experienced programmers to create advanced simulation scenarios in USSR due to specifics of simulation framework.

There are two well-known ways to approach the problem with construction of graphical robot's morphology from modules in simulation environment and they are involving the physical robot and not. When the robot is involved, the morphology of the robot is constructed by simulator, based on data send from the physical robot. Here the robot is sending the data to a simulator over the wireless, cable communication or storage device. After that simulator interprets this data and constructs the morphology of the robot. In case when the physical robot is not involved, the user of simulator constructs the morphology of the robot, by interacting with GUI (Graphical User Interface) options and simulation environment. The best case scenario is when simulator supports both, like for instance Cube Interface [3]. Coming from the fact that hardware of modular robots supported by USSR is currently modified and the main idea behind USSR is to develop it as modular robot generic, the decision was made to pursue the implementation without involving physical robot.

In general, the problem was named as QPSS (Quick prototyping of Simulation Scenarios) and divided into two parts: 1) construction of modular robot's morphology

[1] Morphology refers to outward appearance of a modular robot and its modules.

(shape) and 2) assignment of behaviors (controllers) for simulation scenarios. Essentially, the first is the process of assembling the morphology of a modular robot from modules. The main problematic issue here is that often in the simulation scenarios, the morphologies of modular robots consist of a large number of modules. This results into time-consuming and ineffective process of constructing the morphology of a modular robot. Similar case is during assignment of behaviors, because each module in the morphology of a modular robot should be assigned a controller. It is time-consuming to assign different controllers to different modules in the morphology of a modular robot. As experimental platform was used USSR currently supporting the following modular robots: ATRON [5], M-Tran [6] and Odin [53]. The main reason why USSR was chosen is the fact that both problems manifest themselves significantly here.

The solution is the software implemented for USSR and improving on both problems introduced above. The software was given the same name as the problem and it is QPSS. Here several innovative ideas are implemented and evaluated. Innovative means that to the author's best knowledge these ideas were not previously addressed in simulators for modular robots. First, implement QPSS as object-oriented framework currently supporting three instantiations (ATRON, M-Tran and Odin) and providing possibility to add the support for new instantiations. This means that new modular robots can be supported by reusing existing functionality of the framework. Second idea is partially innovative and these are several tools for interactive construction of modular robot's morphology. Here the user interacts with the GUI of QPSS and simulation environment during construction of morphology. Third idea is interactive assignment of behaviors using the library of behaviors. Here, the user implements controller and as a result is able to choose it in GUI and assign it to the module(s) in the morphology of modular robot by simply selecting it. By assigning different controllers to different modules in the morphology of a modular robot the user is able to observe overall behavior of modular robot. Forth idea is the use of labels for easier identification of entities (modules, connectors and so on) in the morphology of a modular robot. By using labels the user assigns meaningful name to the specific part of the modular robot and later refers to this label during implementation of controller. The main advantage of the labels is the fact that they can be used to abstract the morphologies and their behaviors.

1.2 Methodology

In order to keep the project on track and utilize limited time effectively one should consider employing project management techniques and methods. Being aware of this fact, author and his supervisor Ulrik Pagh Schultz decided to hold meetings approximately every second week, which were dedicated to discuss project related tasks. Knowing that the best practice is to come to the meeting prepared, before each meeting author delivered the draft version of the project report and document containing questions of interest. These documents served as an agenda for a meeting. In this way during the meeting, there was always defined goal, like for instance to discuss around ten and more questions about the project and other questions about the project report structure, typography, context and so on. The answers to the questions were kept in electronic form in order to quickly review them later and improve on them. At the end of the project, it seems that this practice proved itself as being extremely effective because in author's opinion he had an interesting time during the meetings to consider the opinion of Ulrik Pagh Schultz about different questions of interest, learn scientific style of work and being motivated to prepare for each meeting.

As for managing the time span of the project, author chosen to employ Gant[2] chart to schedule tasks and milestones. At the beginning of the project, rough Gant chart was created where two major tasks were: programming application and writing report. As it is always the case for the project schedules, they are not something static, but rather dynamic due to unpredictable events and changes in flow of the project. Considering this, every month author modified the schedule of the project and added subtasks to the three above for a month forward. Therefore, every month author was clearly aware what should be done. In summary, both tasks took approximately two months each. At least two subtasks were executed in parallel in specific time span of the project to spread the workload evenly. In addition to parallelization of tasks and subtasks, one more practice called iteration was employed. After each meeting with Ulrik Pagh Schultz (every two weeks) author reviewed the report, source code and literature to read. Moreover, roughly planned what should be done for the next meeting.

Finally, it is possible to conclude that creation of Gant chart is useful technique in order to keep the priorities right and manage the different subtasks during each month of the project. Furthermore, be able to estimate the passing time and achieved results.

The last method used to improve on writing style of the reports is analysis of previously written reports and wiring styles for writing research papers [4]. Here were analyzed such aspects like structure of the report, typography and style of writing. As a result, several of them gave inspiration and were adopted or modified to meet the requirements of current report.

1.3 Reader's Guide

This report encompasses three major domains, see Figure 1.1. The first is modular robots and serves as an inspiration for defining the answer on question: What is simulated in modular robots nowadays? Next are simulators for modular and mobile robots, where several chosen representatives and interaction techniques are analyzed, for achieving quick prototyping of simulation scenarios (QPSS). Mobile robotics field is a secondary domain and is chosen for discussion, because there are not so many simulators for modular robots with high interaction necessary to support QPSS. Human-Computer Interaction (HCI) is primary domain for investigation of current techniques for QPSS and their application in general.

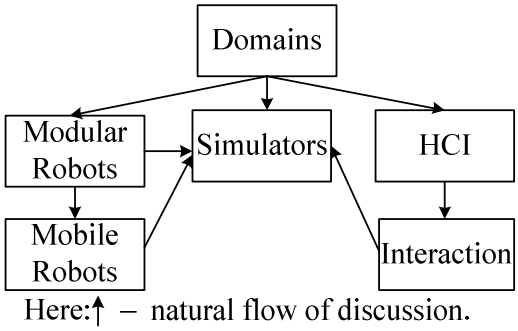


Figure 1.1 Relation of project domains

In order to discuss all the domains mentioned earlier the structure of the report was split into nine chapters with specific purpose.

Chapter 1 briefly explains the motivation and vision for a project. Moreover, familiarizes the reader with methodology, writing style, terminology and contribution of

[2] Gantt chart is a type of bar chart illustrating project's schedule. Gantt chart illustrates the start and finish dates of the tasks as well as the summary of them in project's time span.

the project.

Chapter 2 is dedicated for introduction into the modular robots. The intent here is to familiarize the reader with their historical context of appearance, taxonomy, design approaches, architecture, and control. Chapter in general covers major characteristics of modular robots, furthermore going into detailed discussion in particular cases. The motto here is to discuss the modular robots in relation to the project.

Chapter 3 concentrates on discussion about simulators for modular and mobile robots. These are overviewed from two main perspectives called underlying constituents and interaction. As a result, chapter is split into two parts, where the first overviews underlying engines and related properties of simulators in general and specifically for simulators of modular and mobile robots. The second stresses on discussion about interaction in the same simulators and related properties of them. The goal here is to compare USSR to simulators for modular and mobile robots from different perspectives. Moreover, present interaction techniques serving as inspiration for devising solution to QPSS problem.

Chapter 4 is dedicated to present the analysis of the problem tackled during the Master thesis. In general, the problem can be divided into two parts: construction of morphology and assignment of behaviors, before and during simulation of modular robot scenarios. The main problems here are complex and inefficient way of defining the morphology of modular robot and assignment of controllers (behaviors) to the modules in the morphology. USSR is used as an example to define and describe the problem more thoroughly. Simulators like Webots 5 and Cube Interface are used for comparison and evaluation of existing approaches to the problem.

Chapter 5 is introducing the software implemented for USSR, which is improving on the problem presented in previous chapter. It was named the same as the problem and it is QPSS. Chapter begins with explanation and evaluation of the main features of QPSS software. Next, describes the case study in order to give the feeling of QPSS usage. Finally, discusses the software architecture.

Chapter 6 takes the user perspective and presents the functionality of QPSS on concrete examples. Here are discussed the features (tools) for both interactive construction of modular robot's morphology and assignment of behaviors. Moreover, each feature is evaluated in the context of modular robots by discussing advantages and disadvantages of it. The chapter is quite detailed and can be considered as partial user guide.

Chapter 7 is concerned with introducing the usability tests which were undertaken in order to improve QPSS. Chapter begins with describing the motivation for usability tests. Next presents the test plan used during the testing and concluding discusses how the usability tests influenced QPSS.

Chapter 8 is dedicated to explain implementation and design of QPSS. The main goal here is to discuss the main principles, on which QPSS is based on and evaluate the design by means of evaluating partial design class diagrams.

Chapter 9 is presenting the main tasks, which should be addressed in near future in order for QPSS to be deliverable for the use by the end users. Moreover, the chapter is suggesting several new ideas, which could be considered in the future evolution of QPSS.

1.3.1 Writing Style

Most, if not all readers should agree that each person on planet earth is in a way unique. Similar statement can be applied to different styles of writing used by authors of various types of literature: Each writing style on planet earth is in a way unique. The main arguments for that are: each person's unique way of thinking, writing, presenting, background, communicating knowledge, and so on. At the same time, this uniqueness can be considered as disadvantage, because it is difficult for the reader to make transition from

his/hers style of writing to other person's. In order to ease the process of transition the standard forms, templates, rules and so on are used. At first glance, it seems that it is a perfect solution. However, it is not completely true, because there are huge amount of different rules, templates, and forms followed by authors. Consequently, in order to ease the process of reading, reader should still recognize and even more difficult, to know most of them. As a result, the best authors can do is introduce his/hers style of writing before using it. That is why, beneath are presented main techniques applied in this report.

On the wide scale of report, each chapter begins with citation, which is author's attempt to motivate the reader by means of presenting the wisdom of other authors. In essence, each citation expresses the attitude or definition taken by author. Next follows the list of chapter objectives, where the content of chapter is introduced in shortest form. Already here the reader can decide if he/she is interested in reading current chapter, basing the decision on his/hers previous background. If reader considers himself/herself an expert of information presented beneath, the best way is just to scan it and read the sections of interest. The body of the chapter starts with more detailed introduction into the chapter, the rest varies depending on the intent of the chapter. The approach in writing the body can be called top-down, meaning that first major concepts are introduced and later related to concrete examples. Moreover, knowing that humans accept ninthly percent of information in form of pictures and only ten percent in textual form, the emphasis is to use figures. Finally, chapter ends with summary, emphasizing the main points of discussion.

Several typographical techniques should be also explained. The footnotes in the document are used to explain the terms or abbreviations possibly unfamiliar to the reader. Additionally the list of abbreviations is included just before the first chapter in order to avoid misleading meaning. Moreover, the abbreviations are explained only once, first time met in the report or chapter and the letters of abbreviations are capitalized in explanation. Section References is split into two subsections in order to distinguish web pages and published and draft literature. Additionally, each link to webpage is supported with last date of update and number of references in order to estimate how recent the data is and on how many resources it is based on. As for published and draft references the evaluation criteria for quality of scientific papers and books is a number of citations according to Google scholar. The literature used indirectly (not referenced) during writing the report can be found in Appendix A. Listing of text items is accomplished in two major ways, which are "letter" and "number" listing. Letter listing means that there is no significance in sequence of listing. For example: a) text item 1, b) text item 2 and so on. Number listing means that sequence of text items have significance and priority. For example: 1) text item 1, 2) text item 2 and so on. The links to references (for example [1]) are used in a number of different ways. Most common is the reference to literature on which the current text is based on. These references are situated at the end of the paragraph. Special reference cases are: a) citation reference, indicating the source from which the citation was taken from, b) newly introduced terms reference, where reader is referenced to a source of literature serving as initial and more detailed source of information, c) dedicated column of the table for references, where the references are listed in one of the columns of the table. Here the reader is directed to the literature on which this table relies on, as well as concrete concepts of the table, d) figure reference, which is reference in the title of the figure. This indicates the literature from where the concrete diagram or photo is taken from.

1.3.2 Terminology

Variation in naming of terms describing the same concept from different perspectives often leads to confusion and misunderstanding. In order to avoid that, let us define and clarify the terms specific to this project. These are simulation environment and objects in it,

interaction, controller of the robot, and QPSS.

The term “simulation environment and objects in it” consists of two sub terms called “simulation environment” and “objects in it”. The sub term “simulation environment” encompasses computer-simulated graphical model of physical environment, which is close to physical real life environment specific to the case study. For instance in physical world, this can be any chosen physical environment like forest, room, landscape and so on. In case of simulation, these are computer-represented graphical models of forest, room, landscape and so on. The sub term “objects in it” points to the physical objects computer-simulated in simulation environment as graphical models of objects. For instance in physical world, these can be tree, chair, sofa, and so on. In case of simulation, these are computer-represented graphical models of tree, chair, sofa, and so on. The both sub terms together define computer-simulated graphical model of physical environment and physical objects in it.

In general, interaction is a kind of action that appears when two or more objects affect each other. The main idea here is the two-way interaction. In this report, the term interaction is related to the interaction with simulation environment and objects in it. Meaning that user interacts with the simulation environment and objects in it by means of directly selecting them or selecting GUI elements as intermediate medium.

The term controller (behavior) of the robot can be used in two major ways: a) when referring to software application running on the physical robot and controlling the actions of it in physical environment and b) when the controller is implemented for the graphical model of physical robot to control the actions of it in simulation environment. The later is the case in this report.

QPSS is an abstract term covering tasks related to quick prototyping of scenarios before and during simulation. In the context of this report, it is term covering interactive construction of modular robot morphology from modules and assignment of behaviors (controllers).

1.4 Contribution

One of the evaluation factors for the quality of scientific paper is its contribution to science fields. The main contributions of this report are: analysis of QPSS problem domain in and experimentation with different mechanisms to improve on it.

As for concrete contributions to specific science fields, these are: a) attempt to refine the taxonomy of modular robots, b) create rough taxonomy of control for modular robots, c) create taxonomy of interaction techniques used in simulators for modular and mobile robots, d) implementation of QPSS as object-oriented framework, including experimentation with interactive assignment of behaviors and labels, for identification of entities in the morphology of modular robot. To the best author's knowledge, the last four attempts were not considered previously in the context of modular robots.

Chapter 2

2 Modular Robots

"Science can only ascertain what is, but not what should be, and outside of its domain value judgments of all kinds remain necessary."
— Albert Einstein

Chapter Objectives
1. Briefly introduce history of mobile robots in historical context of robotics science field.
2. Discuss and attempt to improve taxonomy for modular robots.
3. Explain major criteria for classification of modular robots.
4. Construct the taxonomy of control for modular robots.

2.1 Introduction

The intent of this chapter is to familiarize the reader with the modular robotics field by means of starting from a brief introduction into the history of the robotics science field and emphasizing the recent appearance of the modular robots field. Next, we go through the general taxonomy of modular robots, taking the top-down approach, meaning that the main concepts of the field are introduced first and later they are related to concrete representatives. They are ATRON, M-Tran III (Modular Transformer), SuperBot [7], Catom (Claytronics ATOM) [8], Molecube [3], and Odin. At the same time, these representatives are related to criteria for classification of modular robots, like type of their modules, design characteristics, and architecture. Finally, the taxonomy of control for modular robots is presented in general, as well as expanding discussion by means of concrete examples.

2.2 From Science Fiction to Scientific Field

The true inventor of word "robot" was Joseph Čapek born in Czech Republic. Nevertheless, it was first introduced and popularized by his brother Karel Čapek – author of science fiction play named "Rossum's Universal Robots" in 1920. The word robot comes from Slavic word "robota" and means "self labor" and "hard work" [9]. Russian-born American author and professor of biochemistry Isaac Asimov was the first to use the word "robotics" in printing in his short science fiction story named "Liar", published in May 1941. In his other story named "Runaround" (published in 1942) he formulated the Three Laws of Robotics [10], [33]. Writing these science fiction stories, Asimov was not even aware that he was coining the term for a new scientific field – as design of electronic devices is called electronics, so the design of robots is appropriately called robotics [9], [10], [33].

From that time, until now, the robotics scientific field went through number of scientific research stages and currently encompasses several branches. These produced different types of robots like: industrial, mobile, modular and more others. The first patent of industrial robot belongs to George Devol (in 1954), who together with Joseph F. Engelberger produced first industrial robot in 1956. The purpose of the robot was to transfer the object from one point to another (pick and place operation). According to ISO[3]

[3] ISO – International Organization for Standardization.

an industrial robot is defined as "automatically controlled, reprogrammable, multipurpose manipulator programmable in three or more axes" [11]. The appearance of mobile robots is closely related to World War II (1939-1945), when cruise missiles V1 and V2 were invented in Germany and used to bomb London and Antwerp. These are the first examples of mobile robots, also called smart bombs, because they detonated at a certain range to the target. The term mobile robot is defined as "automatic machine that is capable of movement in a given environment" [12]. However, the largest popularity mobile robots reached in the context of AI (Artificial Intelligence), when the first paradigm called GOFAI appeared (Good Old Fashioned AI, also called Symbol-Based). Here the robot's knowledge is represented in the form of rules and facts. In other words, the robot is aware of all the aspects of the environment (or part of it) surrounding it. The disadvantage of this approach is the fact that the robot is pre-programmed for a specific environment and is not able to operate in another one. Moreover, it is operating sequentially in the loop called "sense-model-plan-act". In late eighties, a new paradigm appeared and is called Behavior Based approach. Here the robot is considered as an agent in a dynamic environment and it is relying on execution of pre-programmed behaviors, executed in somewhat concurrent fashion. Moreover, the behaviors can be combined together and executed depending on the level of their priority. For example, the behavior called "avoid collisions" for a mobile robot is the highest priority and it is combined with behavior called "detect obstacles". In this case, the behaviors "detect obstacles" and "avoid collisions" are of highest priority to others and are executed all the time while the robot is operational [34-36].

The origin of the modular robot concept is often referred to the Prof. Toshio Fukuda (in late eighties), who imagined robots consisting of cells (modules) and able to split into their constituents, later reassembling into the new form of robot. The term used for this kind of robot was Dynamically Reconfigurable Robotic System (DRRS). Fukuda's well-known example of modular (cellular) robot is a robot able to get inside a tank through a narrow entrance in the form of different modules (joints, branching and work cells) and self-assemble into the robot suitable for performing the task, see Figure 2.1.

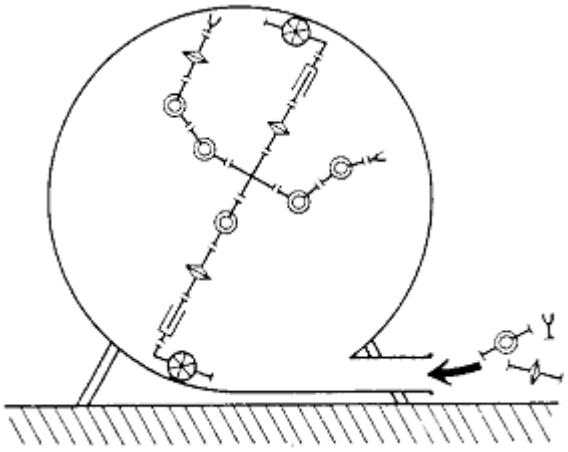


Figure 2.1 An example of maintenance work in a storage tank, by Prof. Toshio Fukuda [37]

The figure above depicts Fukuda's example of DRRS able to get into the storage tank in form of its constituents and inside assemble itself into the modular robot capable to carry out maintenance. The main argument for this type of robots was their versatility[4] and robustness[5] [33], [37].

[4] Versatility by Kasper Støy: "To be versatile means to be able to adapt or be adapted to many different functions and activities."

[5] Robustness by Kasper Støy: "The robot is robust if it is continuing to work despite hardware and software failures"

2.3 Modular Robots

As mentioned above, the pioneer of modular robotics was Fukuda, who mainly concentrated his work on the conceptual side of the field (late eighties). Moreover, he considered different types of modules needed to construct the robot. In parallel, he was working on CEBOT (CEllular roBOT) system, which became a multi-robot system consisting of mobile robots [33].

Fukuda's work gave an inspiration and rise for new categories of modular robots, see Figure 2.2, where the criteria for categorization are types of modular robots. The hierarchy and meaning of each category is as follows.

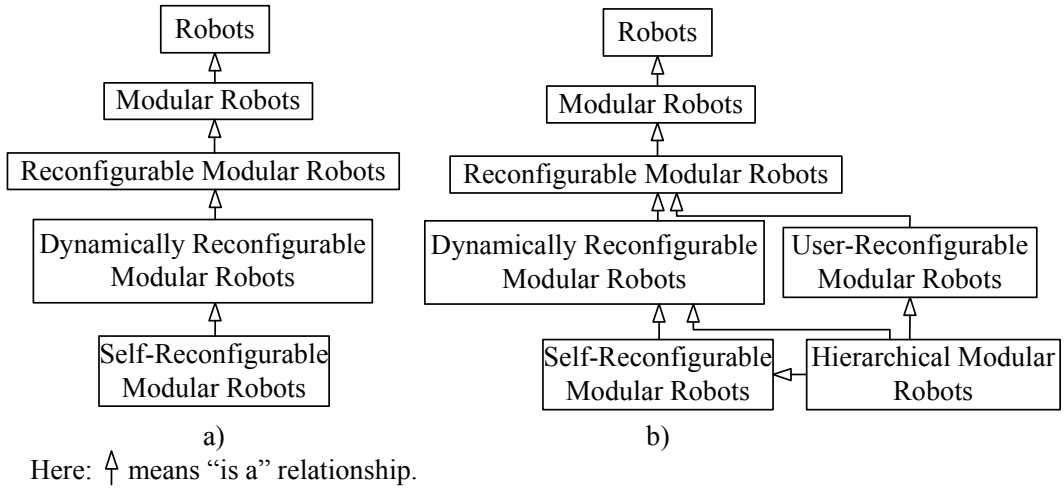


Figure 2.2 A taxonomy for modular robots: a) According to Esben H. Østergaard [38] and b) Inspired by Esben H. Østergaard's point of view

Modular Robots (MR) are robots built from several independent units (modules), which encapsulate some of their complex functionality. Moreover, they have a potential for being easier to maintain and repair, because broken module can be identified and replaced by new ones within relatively short amount of time. In case of non-modular robot, the whole robot may need to be replaced due to the failure of one or several parts.

Reconfigurable Modular Robots (RMR) are the robots consisting of modules that can be connected in several different ways to form different kinds of robot in terms of size, shape (morphology) or function. Modularity however adds more design requirements for modules and the control software.

Dynamically Reconfigurable Modular Robots (DRMR) are robots, where the modules can be connected and disconnected at runtime. The main characteristic of such a robot is that the robot should detect attachments and de-attachments of modules at runtime. According to Esben H. Østergaard this subcategory can be further divided into two subcategories: User-Reconfigurable (URMR) and Self-Reconfigurable Modular Robots (SRMR). In the first case, the user manually attaches or de-attaches the modules at runtime. This emphasizes the role of the user in assembly of the robot. It seems that it is not completely correct, because this definition describes URMR as modules, which are running all the time during the process of assembly. That is not necessary the case for URMR. The idea behind these robots is that a user first assembles the robot from modules in specific architecture, for specific task and after that starts it.

Self-Reconfigurable Modular Robots (SRMR) is a subcategory of robots, which can be characterized by means of three previous subcategories and one additional term called self-reconfiguration (SR), meaning that the robots are able to change the way their modules are connected (change the morphology) by their own actions at runtime [33], [38].

Hierarchical Modular Robots (HMR) are robots consisting of a hierarchy of modules. Essentially, several basic modules form a module with higher complexity and this process repeats until desirable hierarchy is reached (several levels of complex modules). The basic modules are usually simple and differ by their functionality, however the robot, assembled from them, exhibits functionality of higher degree than its constituents did. This approach is inspired by biological organisms, which are built from simple elements like cells. Cells form organs and tissues (lowest level of hierarchy) and the combination of last form the organism (highest level of hierarchy). At the end, the cells exhibit low level functionality, when the organism exhibits completely different complex functionality relying on lower level functionalities of cells and organs [39], [40].

All above-mentioned modular robots can be also classified according to several criteria, like: type of their modules, design characteristics, architecture, control, and so on.

2.3.1 Modules

From design point of view there are two types of robots, called homogeneous and heterogeneous, see Figure 2.3 and Figure 2.4 respectively, for examples.

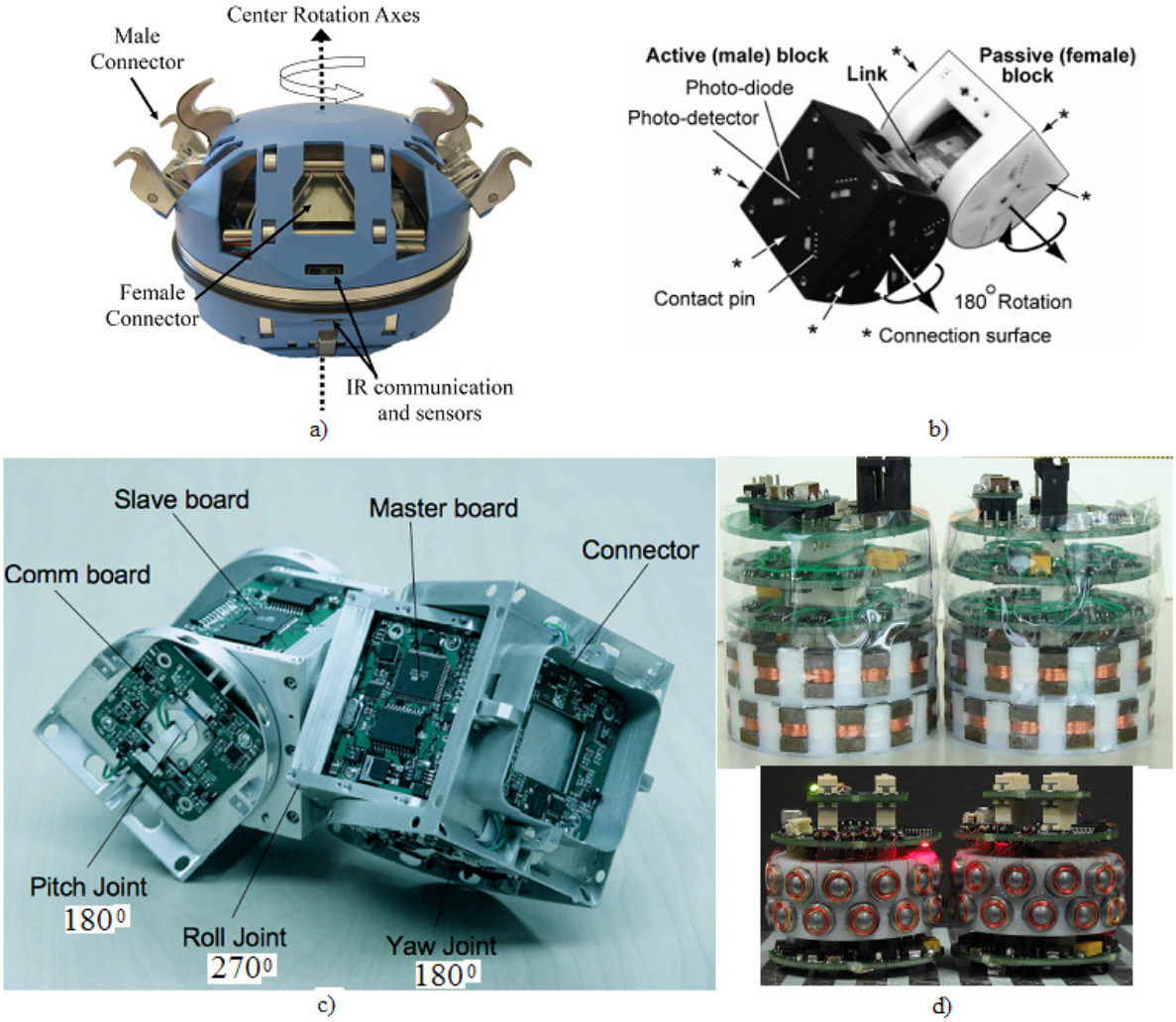


Figure 2.3 Homogeneous modules of SRMR: a) ATRON [40], b) M-Tran III [47], c) Superbot [49], and d) Catom [51]

Homogeneous robots consist of one module type. Usually these modules include all the necessary electronics, mechanics, and at the same time, they are more complex and larger in size than modules of heterogeneous robots. Most of the temporary robots are

homogeneous, due to the long-term cost of production. In the future, mass production of homogeneous modules should decrease their price. This is also one of the motivations for research in SRMR. Another aspect is a future miniaturization of electronics and mechanics, which leads to the same goal.

Several robots have additional tools. For example, ATRON have gripper, M-Tran III – camera module and SuperBot – support tools for climbing the sand hill. This fact does not mean that they become homogeneous. This is rather temporary solution for experimentation. The future design of the module should integrate all the hardware required.

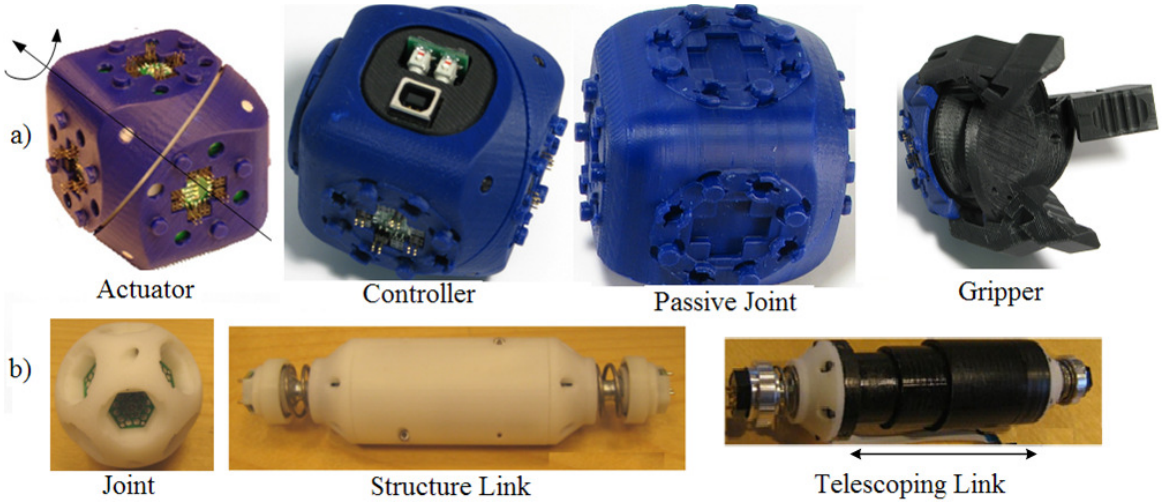


Figure 2.4 Heterogeneous modules of MR: a) Molecube [52] and b) Odin [53]

Heterogeneous robots consist of different kinds of modules and they are designed for specific purpose. For example, in the case of the Molecube modular robot: an actuator module for achieving rotational DoFs (Degree of Freedom), a controller for communication and control of modules, a passive joint for extending the structure, a gripper for grabbing objects and so on. As for the Odin modular robot, there is a joint for passing communication between the modules, a structure link for extending the structure, passing communication, in some cases supplying the power, a telescoping link for achieving linear DoF and so on.

Most important design characteristics of modular robots are related to connectors and DoFs. Connectors ensure interconnection of modules into a complete modular robot and play significant role during the SR process. They are classified according to their type as: genderless, male, and female, as well as according to technology used in their design: mechanical, magnetic, electromagnetic, and so on. For example, ATRON, M-Tran III and Odin make use of mechanical male-female connectors. These limit the range of SR and manual reconfiguration, however they are easier to control in self-reconfiguration process. The dominating solution for increasing the range of SR and reconfiguration here is deployment of more connectors or connector surfaces. Another design characteristic to consider is if they are point-to-point connectors (ATRON) or surface connectors (all above, except ATRON). Point-to-point connectors are usually designed to be stronger than surface connectors, because of mechanical torques. The last characteristics to mention are: the precision, speed of connection and power usage. Mechanical connectors are usually stronger and faster than magnetic. The only drawback here is precision: the modules of the robot must be aligned precisely for connection to be achieved. On the other hand, there are robots like Molecube, Superbot and Catom, those exploiting genderless connectors, which broaden the possibilities for SR and reconfiguration, although at the same time,

complicating the control software. In general, the main idea in the field is to make use of genderless connectors due to simplicity of connection and possibilities of SR and reconfiguration, but so far, this seems to be a difficult task. Consequently, researchers investigate advantages and tradeoffs of each.

In order for modular robots to SR and move in the environment, the modules should be actuated. As a result, the next design characteristic to consider is DoFs, which influence the final complexity of the module and the software to control sets of modules. From Cartesian coordinate systems, it is well-known that there are two types of DoFs: rotational and linear. Moreover, there is a DoF for each coordinate (x, y and z), which results in a total of six DoF. The modules of most modular robots have from one to three DoFs, but the combination of modules in a modular robot increases the amount of DoFs available. For example, ATRON module have one rotational DoF (the rotation of the northern hemisphere in relation to southern), but if the modules are connected so that their rotational axes are in parallel to the x, y, z coordinate system the robot can exploit up to three rotational DoFs. A similar case can be found in M-Tran III with two DoFs. See Table 2.1, for summary of design characteristics discussed earlier.

Table 2.1 Summary of design characteristics for MR modules. Here: m-f – male-female, p-p – point-to-point.

Name of the robot	Connectors				DoFs	
	Amount	Type	Type	Technology	Amount	Type
ATRON	4-4	m-f	p-p	mechanic	1	Rotational
M-Tran III	3-3	m-f	surface		2	
SuperBot	2	genderless			3	
Catom	24			electromagnetic	1	
Molecube	6			mechanic		
Odin	1/2-12	m-f		mechanic		Linear

From the table above it is conclude that researchers designing MR modules tend to equip modules with significant amount of genderless connectors to expand the possibilities of SR and reconfiguration process. Preferable connector type is surface, because of mechanical torques. Moreover most modules are expected to lift only one other module during SR (for example: M-Tran and SuperBot), consequently there is no need for mechanically stronger point-to-point connectors. For example, ATRON module with point-to-point connectors can lift up to two-tree modules. Technology for interconnection of modules is mechanic, because it is faster than electromagnetic. This decreases the time of SR process, which is essentially process of disconnect-connect-disconnect connectors of moving module (s). The amount of DoF for module is chosen according to two major tendencies. First, where the amount of DoF is low, the tendency is to have a module relying on the interaction with others modules to move in space (all above except SuperBot). Second, where the amount of DoF for module is high, the tendency is to have module able to move in space by its own actions (for example SuperBot). Thus, module is more autonomous. Finally, rotational DoFs are preferred over linear most likely due to simpler mechanical construction of the gearing system and the whole module.

2.3.2 Architecture

As for architecture, robots are divided into lattice, chain, hybrid, and hierarchical, see Figure 2.5 and Figure 2.6 for examples. The modules in lattice architecture are aligned in periodic structure with some geometrical symmetry. Moreover, it is similar to lattice structure of atoms in a crystal. Lattice architecture simplifies the SR due to the fact, that the modules are moving from one concrete position in the lattice structure into adjacent. In

chain (also called Tree) structure, the modules of robots are connected in serial (chain) form. However, the chain can be connected into the loop or have several branches. Most of robots are of hybrid architecture, because their modules can be connected in lattice as well as in chain architectures. The examples of such robots are ATRON, M-Tran III, SuperBot and Molecube.

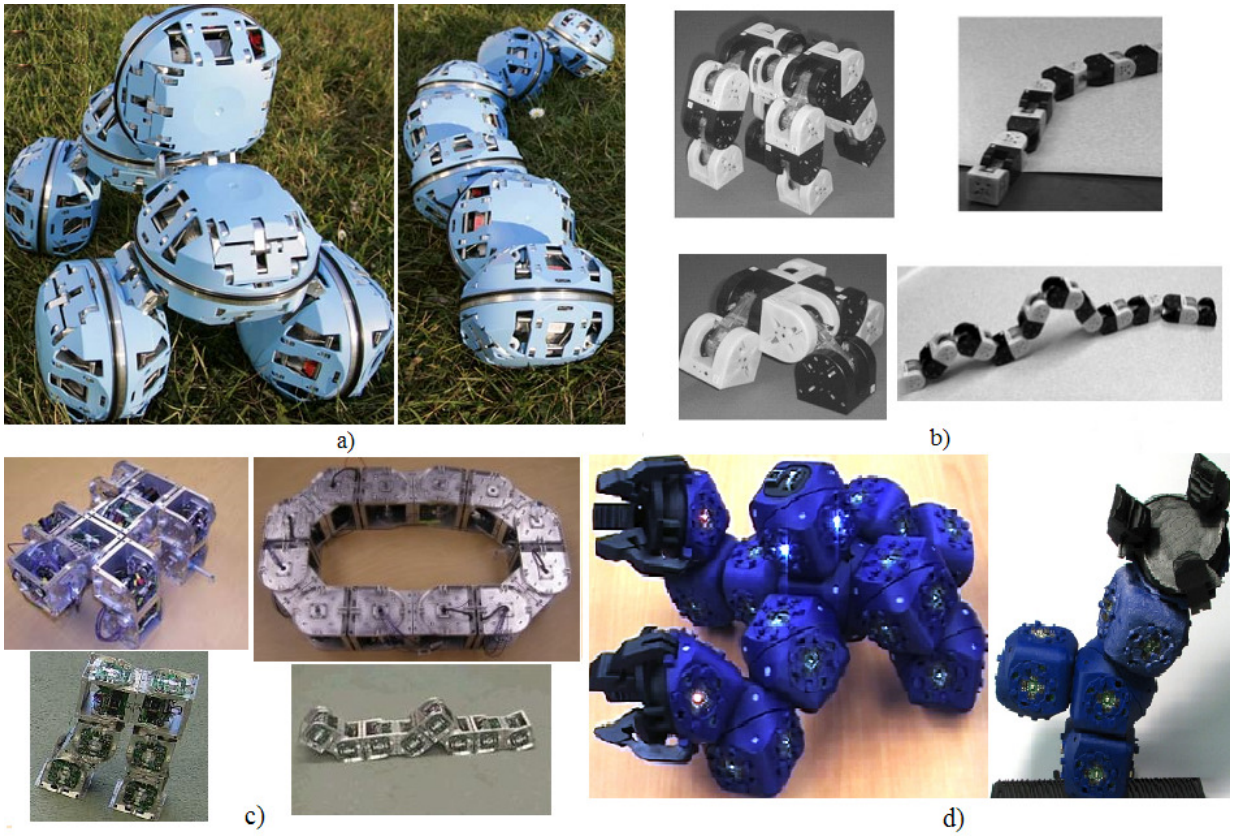
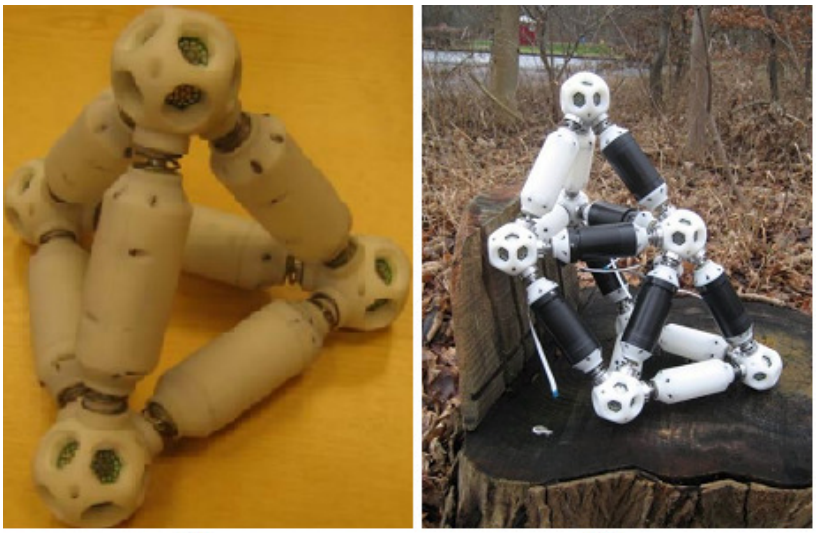


Figure 2.5 Examples of hybrid architecture, lattice (left) and chain (right) for each robot: a) ATRON [33], b) M-Tran III [47], c) SuperBot [49], and d) Molecube [52]

An interesting case is the SuperBot SRMR with three rotational DoFs in a module [50], where in one configuration are used only modules with two DoFs (like M-Tran III) and in other configuration combination of modules with both two and one DoFs. Here the modules are pre-programmed to exhibit only necessary rotational DoF for a specific case. For example in a Loop configuration (see Figure 2.5 c, top-right) SuperBot modules use only two DoFs for locomotion, however in case of H-Walker configuration (see Figure 2.5 c, top-left) one module uses one DoF (so-called spine) and the rest two DoFs (so-called legs). This is one of the best examples that modules with three DoFs interconnected into the robot, enable it to achieve different types of locomotion and SR, due to ability to shift from one to three DoF for a specific case. Moreover, the range of locomotion types is wider and the ability to recover from locomotion failures (tipping over etc.) becomes achievable. In case of SuperBot this is so far proved only in simulations [50]. The recoverability is also one of the main goals in SRMR research.

The modules in hierarchical architecture are interconnected into multi-level hierarchy of modules. The lowest level modules (for example for Odin: joint and actuation link, see Figure 2.4) are connected to form functional modules, these at same time can be interconnected to form even more complex functionality modules. For instance, after interconnection of joints and links, Odin forms tetrahedron geometrical shape or crawler configuration, see Figure 2.6. The lowest level modules have simple functionality (links

are able to expand and contract), but combination of them into the tetrahedron or crawler, enables Odin to achieve locomotion. Figure beneath depicts first version of Odin design in tetrahedron shape and second version of Odin design in crawler configuration. Both are of one-level hierarchical architecture.



a) b)

Figure 2.6 Examples of Odin's one-level hierarchical architecture a) tetrahedron [39] and b) crawler [40]

Finally let us summarize major concepts and robots of MR field discussed above, into the compact way of representation for future reference, see Table 2.2.

Table 2.2 Survey of modular robots and their properties

Name of the robot	Type	Modules, -geneous	Architecture	Simulator	Year	References
ATRON	SRMR	homo	hybrid	USSR	2003	[33], [38], [40-46]
M-Tran III	SRMR	homo	hybrid	New Unit Simulator	2005	[47], [48]
SuperBot	SRMR	homo	hybrid	Galina	2005	[49], [50]
Catom	SRMR	homo	lattice	DPRsim	2005	[51]
Molecube	URMR	hetero	hybrid	Cube interface	2005	[52]
Odin	HMR	hetero	hierarchical	USSR	2007	[39], [53], [54]

In the table above are presented the criteria for categorization, as well as categories into which fall the robots discussed earlier. Using this table, the classification (taxonomy) of modular robots can be constructed, according to specific criteria. For example: a) ATRON, M-Tran III, Catom and SuperBot fall into the category of SRMR according to the criteria "type of the robot", b) Molecube falls into the category of RMR according to the same criteria and so on (see the first and the second column from the left). This specific example is closely related to the Figure 2.2 above, where the general taxonomy of the modular robots is defined. The two more criteria: a) simulator column is added to indicate the existence of simulator, which was the primary selection criterion for discussion of these particular robots in relation to the project, and b) year column for comparison in historical context. Several other criteria were not included in the table due to their uncertain nature or ongoing research emphasis. However, they are explained beneath.

Power is necessary for the modules of the robot to operate. Most of the robots include

power supply in their modules, but there are still robots powered externally. In order for the robot to be autonomous, the power should be onboard. Currently, power plays significant role in the robots autonomous operation as it is limited. Power sharing between the modules is the most common solution in modular robots in order to keep the modules operating with the same performance (synchronization). Nevertheless, not all modular robots have this property as a major design decision.

Communication is necessary in order to keep the modules operating in synchronous or specific coordinated way. Broadcast (global) communication between the modules is not popular in robots with a large amount of modules, because the amount of communication channels is small. As an alternative, the emphasis is on the local neighbor-to-neighbor communication, but the drawback here is delay. Recently a hybrid between above two was developed for the Odin. Developers believe that it will be flexible way of communication between groups of modules [33], [38-56].

2.4 Control

Control of modular robots can be divided into centralized and distributed, see Figure 2.7. Centralized control means that the computer or one module of the robot (master) is controlling the operation of other modules (slaves). In case when the computer is controlling the operation of modules depending on the current state of the whole robot morphology, control is called online. When the behaviors of the modules are pre-computed beforehand, control is called offline. In both cases, major disadvantage is that if there is malfunctioning module in morphology of the robot as a result whole robot is malfunctioning. Most of experiments with modular robots use centralized type of control, because it is easier to pre-program. Moreover, it is better at coordinating actions of modules and ensuring certain behaviors, however, most researchers agree that it is not an optimal solution for future modular robots due to scalability and fault issues.

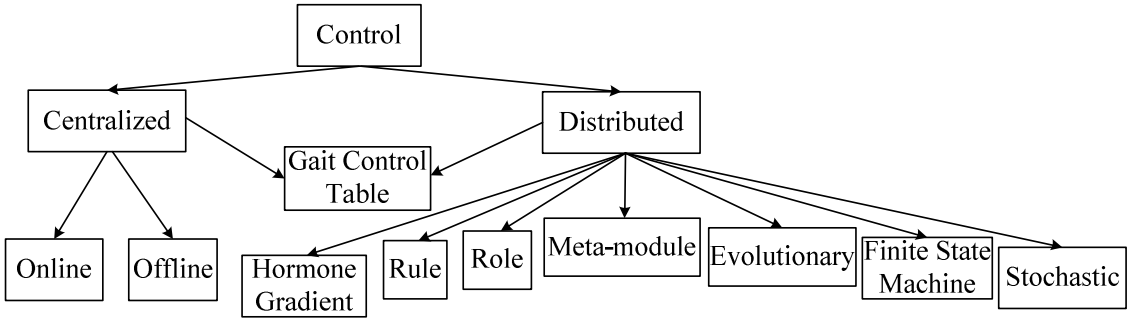


Figure 2.7 Rough taxonomy of control for modular robots

Probably one of the first centralized methods to control the locomotion of SRMR was gait control tables. Here the concept of the table is used as representation of modules and their motions. Usually the table is composed of column of steps, which should be undertaken by the modules of the robot, and rows, which contain module motions during this step. The last column is synchronization, which indicates when next step should be initiated. In this way, the synchronous operation of set of modules is achieved. In general, gait control table is relatively limited, but on other hand, it is simple. Let us consider simple example of SuperBot chain architecture (snake) consisting of six modules, where for each module there are four steps. The snake is laying on the flat surface in the straight-line form. All modules of the snake are able to achieve only rotational DoF of 90 degrees. The rotation axis is parallel to the surface. The hypothetical gait table for snake to roll over and roll back into original form should look similar to Table 2.3.

Table 2.3 Hypothetical example of gait control table for rolling snake of SuperBot MR

Steps	1	2	3	4	5	6	Trigger
1	→	→	→	→	→	→	All
2	→	→	→	→	→	→	All
3	←	←	←	←	←	←	All
4	←	←	←	←	←	←	All

The table above shows that during the first step all six modules of the snake roll over 90 degrees to the right (→) at the same time (trigger all), the second step is identical to previous. Next two (3 and 4) are opposite to two previous. Meaning that all modules roll over 90 degrees to the left (←) at each step. During the first two steps, all six modules roll over so that the snake is lying upside down. Next two steps return the snake into the original position [33].

Nowadays, researchers concentrate on experimentation with distributed control, where each module acts as individual unit and reacts on presence of the other. The main advantage of distributed control is ability tolerate faults and manage scalability issues. Pertaining problem with distributed control is that there are no general methods, which could be used to implement single controller for each module and consequently all modules in morphology will be able to perform collective task. In relation to that, there are number of implementation techniques of distributed control and they are the following: gait control tables, hormone gradient-based, rule based, role based, meta-module, evolutionary, finite state machine (FSM[6]), stochastic and so on.

Gait control tables can also be used as distributed control method. In this case, each module is assigned a specific type of function, depending on its role in morphology of the robot. For instance, M-Tran or SuperBot in H-Walker configuration (see Figure 2.5 above) consists of modules, which function as legs and spine. Here module selects its function according to its position in H-Walker configuration and acts according to it [33].

Hormone gradient-based control was inspired by cellular biology, where hormone gradients play central role in growth and self-repair. This method of control is a hybrid between hormone and gradient based controls. Essentially implementation of hormone gradient-based control relies on continuous broadcasting of hop-count number and passing the data about its current step to neighboring module(s). When module receives the hop count number it stores this number and starts broadcasting incremented or decremented hop count number and data about its current step at regular intervals. This procedure is repeated from originating module or modules through all the morphology of the robot or to a limited degree. Each module uses timer to keep track on the time when the last hop count number matching to stored one was received. If the time interval is longer than assigned, hop count number is erased and modules stop broadcasting. Using this type of control each module becomes aware of topology of the robot to the extent that it knows the distance from its position to the originating module. Moreover, each module is aware of current state of the neighbor module(s). For example, in the above snake example, first (originating module) stores number zero, after that sends number one to the next module and data about its current state, and so on, each module increases the hop number and sends data about its current state until the end of the chain. The last module in the chain is when aware that it is four modules away from the originating module. The result is very close to the gait control table, example above. The advantages of this approach are that the modules are synchronized and modules can be added or removed from the snake. The main disadvantages for this type of control are frequent broadcasting, linear increase in

[6] A finite state machine is a model of behavior composed of a finite number of states, transitions between those states, and actions.

communication load with increase in amount of the originating modules and difficulties in using it to propagate global coordinate system for the modules of the robot. The hormone gradient-based control can be further divided into closed loop and open loop. It is open-loop if the originating module is continuing its steps without knowledge if other modules finished theirs. Closed loop means that there is a hormone traveling back to originating module and in this way ensuring that, the first originating module do not continues to the next step [33], [38].

Rule based control is based on definition of rules for module according to its local neighborhood with other modules, geometric limitations, and overall design. Here main concepts are preconditions and actions. The modules execute actions according to their local neighborhood (precondition). The controller of a module is a set of rules that maps from precondition to an action. Actions are simple rotations of actuators. The approach is that when system gets into the state when there is no rule describing the current state, new rule is added manually. At some point the system have large enough set of rules to operate without adding new ones. Thus, here the central operator is user adding the rules according to which the system achieves desired behavior. This type of control has two major disadvantages: a) in complex modular robot morphologies, it is tedious work to define all the rules to achieve specific behavior. For example, experiments with 32 modules of ATRON robot in simulation show that in order for a module to move from one end of the structure to another, 927 rules are required. b) This type of control is highly dependent on the design of the module and as a result, the rules are different for each design of the robot [38].

Role-based control utilizes mathematic functions for description of locomotion for the modules. In general, the positions of modules during locomotion are devised from the function of time. In role-based control, the role of the module is defined as function, delay, and period. By means of manipulating the values of these variables for each module, the distinct locomotion type is achieved. For instance, the rolling snake example above can be defined as a sinus function of actuator rotations [33].

Meta-module control is a control strategy relying on group of interconnected modules, which is called meta-module. The actions of meta-module are composed of actions available to the modules composing the meta-module. The major advantages here are that meta-module can move by means of action of one module and there are more actions for meta-module in comparison to amount of actions available to single module. For example if three ATRON modules are connected into meta-module (three modules together) so that the rotational axis of each corresponding module is aligned with the x, y and z coordinates, as a result each module can rotate the whole meta-module around each coordinate. That is of course assuming that the meta-module is situated in some assembly of ATRON modules. The main feature of the meta-module control is that the modules can emerge when it is necessary and disassemble back into basic modules, thus the granularity of the system is not compromised. The last advantage of meta-module control to mention is that the time it takes for robot to SR is smaller in comparison to SR by each module because there are less connectors blocked by other modules [38], [40].

Evolutionary control for modular robots encompasses a number of different methods and techniques based on ANNs [7] (Artificial Neural Networks) and GAs [8] (Genetic Algorithm). Widely spread and biologically inspired approach based on neural networks

[7] An artificial neural network is a mathematical or computational model based on biological neural networks, which is interconnection of neurons.

[8] A genetic algorithm is a search technique used in computing to find solutions to optimization and search problems.

are CPGs (Central Pattern Generator), which are used to control the rotation of actuators. Typically, CPGs produce sinusoidal or other type of systematic output. GA can be used to optimize CPGs, like for instance in M-Tran [57].

FSM control is based on definition of states for the modules of modular robot, available actions and transitions from one state to another. For example in case of ATRON, hypothetical example can be. If first ATRON module is connected to another module on the same hemispheres with ninthly degrees angle and first module is connected to some base assembly of modules on opposite hemisphere. Then the first module is in state of being connected to second module and base assembly of modules. The actions available to the first module are to rotate the second module in steps of nightly degrees around axis of rotation and disconnect the second module. Most probable transition from one to another state is being connected to the first module (one state) and being disconnected from second module (second state) [38].

If above types of control are based on controlling the modules in morphology of modular robots. Then stochastic control is based on controlling the environment in which the modules are situated and modules inside it. By doing so the morphology of the robot is assembled. The main idea here is ability to eliminate the actuators from modules of modular robots and assign responsibility for reconfiguration to some external force and partially to modules. Consequently the design of modules can be minimized in dimensions, due to the fact that actuators occupy significant amount of space in the design of the module. Simple hypothetical example of such control is a control of box filled in with modules, equipped with magnetic connectors. The box is vibrating according to specific pattern. As a result the modules inside the box are connected together randomly. The whole morphology of connected modules is checked by controllers of the modules. If the module is connected in wrong place it is automatically disconnected. The whole process is running until desired morphology is reached [38].

2.5 Summary

In this chapter we started our discussion about modular robots from brief introduction into history of robotics scientific field. Next we discussed improved taxonomy of modular robots, which indicates that there is a significant diversity of modular robot types. The taxonomy is not necessary is in a complete form, due to fact that the field is dynamic and taxonomy should be improved with passing of the time, however already now it is possible to notice that the main tendency in design of modular robots is design of SRMR. The main reasons for that are their versatility, robustness, autonomy and adaptability to environment. As for modules, there are two main types followed during design: homogeneous and heterogeneous. The first are modules of the same type, when second are modules of different types. Both design approaches are explored and it seems that emphasis is on homogeneous, due to the future decrease in cost during mass production. The overall design of modules varies significantly from one modular robot to another. In general, the design of the module is based on consideration of such factors like, connector technology and type, architecture of modular robot, DoF available, ability to SR, cost and so on. Finally we familiarized ourselves with different types of control for modular robots. Here the situation is similar to diversity of modular robot types, because a number of different control strategies exists, which are best suitable for particular modular robots. The current tendency is to focus on distributed control, due to its ability to tolerate faults and scalability issues.

Chapter 3

3 Simulators for Modular Robots

"Simulation is the art of using physical or conceptual models, or computer hardware and software, to attempt to create the illusion of reality."
— Stanislaw Raczynski

Chapter Objectives

1. Discuss and construct taxonomy of simulators in general.
2. Present general aspects of simulators.
3. Discuss simulators for modular and mobile robots and their aspects.
4. Introduce two major paradigms of HCI field.
5. Evaluate interaction in simulators for modular and mobile robots.
6. Construct taxonomy of interaction techniques used in simulators for modular and mobile robots.

3.1 Introduction

This chapter is dedicated to discuss the simulators for modular robots from two main points of view: underlying components and interaction. With this intent, this chapter takes a bottom up approach, meaning that major concepts are explained first and later related to specific examples of simulators. For instance, chapter begins with introduction of taxonomy of simulators followed by main aspects of simulators, later the same aspects are used for broader discussion about simulators for modular and mobile robots. In both cases, USSR is used as example and for comparison. The simulators for mobile robots were included to improve discussion, because they exhibit wider range of properties uncommon to simulators for modular robots. Thus, wider scope of discussion is achieved. To be more specific the chosen simulators for modular robots are: USSR [1], New Unit Simulator, Galina, Cube Interface [3], Webots 5 [2] and DPRsim (Dynamic Physical Rendering SIMulator) [19]. Simulators for mobile robots were chosen to be the following: Gazebo [20], Webots 5 [2], MRS 1.5 (Microsoft Robotics Studio 1.5) [22], and Marilou (MARILOU robotics studio 2009) [23].

Next, the same simulators are discussed from the interaction point of view, applying the bottom up approach. Here the starting point is two main paradigms of HCI: Desktop and Natural Interaction. Following is exemplification of their major interaction techniques and discussion about their advantages and disadvantages. Next simulators for modular and mobile robots are evaluated by means of considering the interaction techniques supported in them. Finally in order to summarize all the interaction techniques discussed during this chapter the taxonomy of interaction techniques used in simulators for modular and mobile robots is devised.

3.2 Simulators

Traditionally simulators are classified into dynamic and static, see Figure 3.1. Dynamic simulators compute future effect of the parameter on the model of simulation by means of iterating it through the time. In contrast, static simulators employ single computation of equations representing simulated model. Consequently, static simulators ignore time

variation and cannot be used to determine the occurrence of particular events in relation to other.

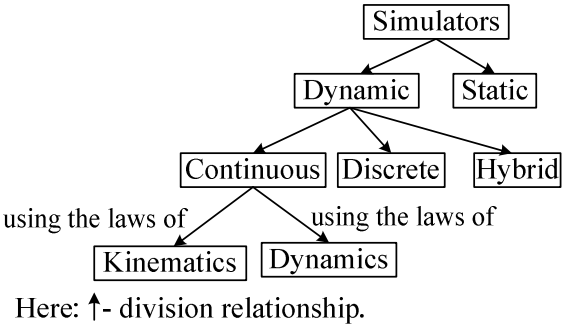


Figure 3.1 A taxonomy of simulators

With increasing power of computing and as a result, decreasing cost of computational power, simulation leans towards dynamic models. This resulted into a significant amount of different simulators, which can be further classified by criterion: how they evolve their models with respect to the time. Dynamic simulation means evolving the simulated world through the time. However, there are different ways to let the time evolve: continuously, discreetly, and in a hybrid fashion.

Continuous simulation reflects the simulated model at a particular time, taking the model through the sequence of steps, in a systematic way. An example can be simulation of physical system, description of which is dependent on the time. For instance, the tank filled in with water and water is being released.

In discrete simulation, model changes state when discrete events occur. Here the time does not anymore have direct effect and the time between the events is rarely uniform. An example can be the train station, where arrival and departure are events.

Hybrid simulation is combination of two previous and makes use of both continuous and discrete events. Typically this kind of simulators are used when the physical system is characterized being in both time variations. For instance: simulation of automobile manufacturing process, where some stages dependent on the time to carry them out.

Further classification can be taken from the level of continuous simulation, where the next criterion is the type of physics involved. Here two types are distinguished: the ones using only the laws of kinematics and the ones additionally using the laws of dynamics. The main difference between two is that the first treats the model as object without mass and directly modifies its velocity. The second on the other hand, takes into account the geometry, mass and changes the velocity by only indirectly applying forces. Therefore the simulation using laws of dynamic is more complete, however requires more computation power [58].

3.2.1 Aspects of Simulators

With increasing computational power of computers, simulators become attractive tools for investigation of objects of interest. This is due to the ability to explore and create objects that are more detailed and carry out more accurate simulations, hence get results that are closer to “real world” [59].

Currently, major application domains of simulators are computer games, academia, and military. In a way, all three are interrelated, moreover most agree that academia is the one that inspires and serves the basis of knowledge. As researchers work towards expanding their knowledge about our universe, the need for research tools arises constantly. Significant numbers of them already evaluated advantages of simulators. Moreover, they use them as major research tools. Robotics scientists are not exception, because they use simulators during their research as major tool to experiment with the design of the robot,

the behavior of them in dangerous and not environments, control, answering "what if" questions and so on. As an example consider Sojourner robot, which landed on Mars surface. However, before sending the robot to Mars, researchers tested both software and hardware behavior in simulations, because no one could predict what conditions will be on the Mars. The process of creating the robot and programming it is not an easy task at all and involves interdisciplinary efforts. Implementation of simulator for a single robot requires somewhat the same amount of effort and even more, for generic simulator. That involves consideration of such major aspects like the following (USSR is used as example): graphics, physics and collision detection, audio, AI support and so on.

Graphics (2D or 3D[9]) engine is a primary aspect when considering the fact that the simulator should be able to draw and color the lines, polygons, and so on, constituting the visual appearance of objects in simulation environment. Common approach is to make use of primary graphics engines like OpenGL (OPEN Graphics Library, [14]). Their purpose is to provide rendering functions. Higher-level engines (also called scene graph APIs[10]) like OpenSceneGraph[11] and jME (jMonkey Engine, [15]) are built on the top of primary engines and are used for processing large amount of data by sorting and identifying it. In essence, they ensure that the data, which have no noticeable effect on the scene, is not processed. For example, the object located under some surface or behind the user's viewpoint and hence invisible to the user will not be processed, because it has no effect on the appearance of the scene. This saves the computational power. There are also model formats such as VRML[12], which can be loaded by these engines. CAD tools, like 3D Studio Max[13] and others edit the model formats. The main advantage of simulators supporting this feature is the ability to share the objects (simulation worlds and robots) between each other, [60]. In USSR, the primary engine is OpenGL and jME is higher-level engine. OpenGL is a most widely adopted graphics standard specification defining a cross-language cross-platform API for developing applications producing 2D and 3D graphics. Additionally it is independent of window system and hardware operations. API consists of commands for specification of 2D and 3D geometric objects and commands to control the rendering of them. OpenGL includes geometric primitives: points, lines and polygons, combining which, it is possible to build models that are more complex. jME is a Java scenegraph API, whose primary focus is high performance 3D gaming. It is written in Java and contains support for OpenGL through the wrapper library.

In order for simulation to be as realistic as possible the dynamics of objects like gravity, mass and forces, should be accounted for. For that purpose, physics engines are used. In this case, the objects are not anymore only visual. They are closer to "real objects" with specific mass, influence of environmental gravity on them and so on. In addition to physical properties of objects, also the geometric properties should be modeled and processed. That is achieved by means of using engines for collision detection. USSR uses ODE (Open Dynamics Engine, [16]) for preparing these properties of objects, to be more precise rigid body dynamics and collision detection. ODE is a free library for simulating articulated body dynamics and moving objects in virtual reality[14] (VR). The main

[9] 2D or 3D – two or three-dimensional.
[10] API – Application Programming Interface is a set of functions, procedures, methods, or classes.
[11] OpenSceneGraph is an open source high performance 3D graphics toolkit
[12] VRML – Virtual Reality Modeling Language is a standard file format for representing 3D interactive vector graphics.
[13] 3D Studio Max is a modeling, animation and rendering package.
[14] Virtual reality is a technology, which allows a user to interact with a computer- simulated environment, be it a real or imagined one. This computer-based and simulated environment is called virtual world. Following from that it is possible to think about VR as interface, virtual world as unification of cyber and cyberspace.

characteristics of it are fast, flexible, platform independent, possesses advanced joints, contact with friction, and build in collision detection. Comparing it to other similar packages, the notable difference is its stability is not sensitive to the size of the time step. Moreover, have no difficulties in simulating articulated systems.

Audio engines can load the sound files (both compressed and uncompressed) and play them in required cases during the simulation, like for example, in OpenAL engine (OPEN Audio Library, [17]). Currently, USSR does not support this feature, as it is not a priority. However, jME engine supports OpenAL. Consequently, future implementations of USSR have the possibility of using this library.

Different AI packages are available for games and simulators, including search algorithms like A*[15] and decision trees. So far, there is no indication on the use of them in USSR as included support. However, several experiments were carried out using USSR and a separate A* implementation [40].

Some of the modern simulators, running based and client server architecture use networking for transferring data between each other. To achieve that, higher-level engines are used. Currently, there is no indication that USSR includes this type of operation.

All above aspects are considered to be the lowest level of implementation in any of the simulators and form the basis of simulator. The abstraction layers over those usually hide the details of their implementation from the developers and often are only mentioned in related literature. However, abstraction allows the developers to program the simulator in general purpose language, like Java in USSR.

Simulators are usually implemented in one of the three ways: from scratch, integrating modules and mixture of two previous. Development from scratch is often applied for so-called in-house projects, where the whole control over the software is necessary. As a result, these kinds of simulators are of significant cost and limited for use to a particular group of users. Integrating modules is one of the most widespread approaches, where several modules are collected and integrated into the simulator. Here the configuration of modules is the first difficult part of development, because the underlying engines (modules) should be evaluated and chosen according to the requirements of simulator. Simulators of this kind range from commercial to open-source. The USSR is one of them. The last development approach is the mixture of two above, usually involves development of several modules and using open-source or buying modules from other developers. Except for the first case, developers using last two approaches tend to promote the paradigm called OSS (Open Source Software). According to this paradigm, developers share the source code freely over the internet [59-61].

Finally, it is possible to conclude that simulators are not only simplifying the research in nearly all fields of science, but also require input from most of them in order to provide accurate models and simulations.

3.2.2 Simulators for Modular and Mobile Robots

After exploration of modular robotics field, it is notable that 3D simulation is a recent trend that appeared around 2000-2001. However, already now researchers discovered the main advantages of it. Like initial experimentation with hardware, communication, control and so on, at a relatively low cost [31]. Moreover, the simulators are modified together with the improvement of the robot, as the number of requirements for robot increase. Furthermore, their documentation is poor from the IT (Information Technology) point of view. That is understandable, because currently the research in modular robotics focuses on the other issues and the simulators are only support for exploring them. The major simulators of

[15] A*– (“A star”) is a best-first, graph search algorithm that finds the least-cost path from a given initial node to one goal node.

modular robotics as well as their main aspects are summarized in Table 3.1 beneath. The main arguments for choosing those particularly were: their relation to modular robots presented in previous chapter, the fact that they are the only notable representatives from the modular robotics field and existence of some form of documentation.

Table 3.1 Survey of simulators for modular robots in comparison to USSR, from perspective of their aspects

Name of simulator	Supported robot (s)	Graphics engine (s)	Physics engine	License	Download	References
USSR	Multi-robot	OpenGL and jME	ODE	BSD[16]	available	[1], [31], [32]
New Unit Simulator	M-Tran	OpenGL, Vortex[17]		proprietary		[62], [63]
Galina	Superbot	OpenGL				[50], [64]
Cube Interface	Molecube	OpenGL, ORGE[18]	PhysX[19]	GNU[20]	available	[3], [52]
Webots 5	Multi-robot	OpenGL	ODE	proprietary		[2], [18], [65-68]
DPRsim	Catom	OpenGL	ODE	GNU	available	[19]

From the table above it is obvious that in the field of modular robotics the general tendency is to develop a simulator for each specific modular robot. The only two exceptions to this are Webots 5 and USSR, where USSR has approach to develop generic simulator for modular robots. It is interesting to note that Webots 5 simulator was originally developed for mobile robotics field and is generic simulator, but nowadays work proceeds towards supporting the robots from modular robotics fields. YaMoR (Yet Another MOdular Robot) and M-Tran III are the robots from modular robotics supported in Webots 5. YaMoR is URMR with singular DoF and its own simulator called Eve. Unfortunately, for now there is not enough data about it. So far, it is known that it is also using ODE as physics engine. As for choice of graphics engine, the developers tend to use OpenGL, most likely due the fact that it is a well-known and cross-language, cross-platform API. The most interesting is that the unspoken agreement is to use ODE for rigid body dynamics and collision detection. Most simulators are OSS, except several that implement features specific to a robot (for instance: New Unit Simulator and Galina).

Concluding the discussion about simulators from modular robotics field, it is necessary to emphasize that USSR is considered to be one of a kind here, due to the fact that already now it is supporting three modular robots (ATRON, M-Tran and Odin) The only rising similar simulator is Webots 5 from mobile robotics field supporting two modular robots (YaMoR and M-Tran III).

In relation with above, let us consider the simulators for mobile robots in comparison to USSR and simulators from modular robotics field. The simulators of this field went through more software evolution stages, which are naturally dictated by the fact that the

[16] BSD – Berkeley Software Distribution BSD is family of permissive licenses and in comparison to GPL have more restrictions. Also permits integration of the code with appropriate software.
[17] Vortex is a commercial physics engine for real-time visualization and simulation.
[18] ORGE – Object-Oriented Graphics Rendering Engine is a scene-oriented, flexible 3D rendering engine implemented in C++ . It abstracts the details of using the underlying system libraries like OpenGL.
[19] PhysX is a real-time physics engine middleware SDK developed by NVIDIA.
[20] GNU – General Public License (or just GPL) is a free software license. GPL is well-known copy left license, meaning that license requires all derived implementations to be distributed under the same license, even if the primary implementation was edited or improved.

field is older than modular robotics and simulators where used earlier. The major simulators of mobile robotics field in comparison to USSR, as well as their main aspects are summarized in Table 3.2, beneath. The main arguments for choosing those specifically was the fact that they served as inspiration for finding solutions during Master thesis and availability of proper documentation.

Table 3.2 Survey of simulators for mobile robots in comparison to USSR, from perspective of their aspects

<table>
<tr><th>Name of simulator</th><th>Supported robot (s)</th><th>Graphics engine (s)</th><th>Physics engine</th><th>License</th><th>Download</th><th>References</th></tr>
<tr><td>USSR</td><td rowspan="5">Multi-robot</td><td>OpenGL, jME</td><td rowspan="3">ODE</td><td>BSD</td><td rowspan="2">available</td><td>[1], [31], [32]</td></tr>
<tr><td>Gazebo</td><td>OGRE</td><td>GNU</td><td>[20], [21], [69 – 71]</td></tr>
<tr><td>Webots 5</td><td>OpenGL</td><td colspan="2" rowspan="3">proprietary</td><td>[2], [18], [65-68]</td></tr>
<tr><td>MRS 1.5</td><td>XNA[21]</td><td>PhysX</td><td>[22]</td></tr>
<tr><td>Marilou</td><td>Engine 'M'[22]</td><td>ODE</td><td>[23], [24]</td></tr>
</table>

From the table above it is notable that in comparison to simulators for modular robotics, simulators for mobile robotics tend to be more generic. In the table, that is indicated by means of term called multi-robot, which means simulator is able to support any desired model of the robot and currently is supporting several graphical (visual) models of the robots. The underlying physics engines are nearly the same. Except, special case of MRS 1.5 where PhysX is used. The main reason for using PhysX is hardware acceleration, which relieves physics calculations from the CPU (Central Processing Unit) by means of assigning it to GPU on graphics card (Graphics Processing Unit). Typical tendency is that proprietary simulators are using graphics engines of their own development. As previously mentioned the main argument for doing so is complete control over the software and particular requirements for engine features. Examples are MRS and Marilou with XNA and engine 'M' respectively. The concerning notification is that most of previously OSS simulators become proprietary. For example Webots 5, which a number of years ago was called Khepera, furthermore was OSS. During analysis of simulators for modular and mobile robotics several empirical observations can be devised. First is that when the simulator becomes proprietary all the documentation is limited to web pages. Second is that implementation of most simulators concentrate on generic simulator approach. Third is that newest tendency arises called towards engine independent simulators. This means the higher-level source code should be portable from one set of engines onto another set with minor modifications of the code. As for USSR, it adopts the similar engines and it seems to be on the close level to the leading simulators from the point of view on multi-robot support.

After looking into the main representatives of simulators for modular and mobile robots and their constituents, let us discuss the same simulators from interaction point of view.

[21] Microsoft XNA (XNA's Not Acronymed) is a set of tools with a managed runtime environment provided by Microsoft that facilitates computer game development and management.

[22] Engine 'M' is a graphics engine developed by AnyKode Company (developer of Marilou) and uses pixel/vertex shaders.

3.3 Interaction in Simulators

One of the major driving forces behind the research in HCI field is the fact that even unlimited computer power is of small use unless the user is able effectively communicate his/hers intentions to computer and receive feedback in efficient form. Historically this combination of user input and system output is called user interface. Through the history of computers (more than 50 years), humans interacted with computers in several major ways. To recall, the first were simple switches, wires, and punched cards. Next, CLI[23], which appeared around 1970 and is called Typewriter paradigm. Nowadays, HCI field is on the edge between two main paradigms: a) Desktop, where interaction is implemented by means of GUI or WIMP[24] based graphical interfaces, b) Natural interaction, where the emphasis is on the PUI (Perceptual User Interfaces). The GUI is significant improvement over the CLI, however in comparison to exponential increase in computation power the way we interact with computers do not changed significantly for around 30 years [72], [73].

3.3.1 Desktop Paradigm

Desktop paradigm appeared in period from 1970 to 1980 and since then is dominant. Here the GUIs are the core implementation technique and their source code constitute about 45-50% of all application source code. Through GUI, the user is able to interact with the underlying source code of software (”business logic”). Consequently, the typical application consists of a GUI, enabling the user to interact with source code and the source code implementing the “business logic”. Typical process of user interaction with GUI is based on use of input hardware (mouse and keyboard) as pointing devices and interaction with GUI widgets. The widgets include different GUI elements like windows, menus, buttons, and so on. By using pointing device user performs events like clicking the button, selecting the item in menu and so on. These events then have deterministic effect on the state of the software, which can be also reflected on the GUI. This is the traditional causality effect, when input causes deterministic output [72].

The reason why the GUIs became popular and still are the major technique to achieve interaction between the user and application is the fact that they are platform independent. Consequently, the developers can build the GUI over the event based architecture using common toolkit of widgets. The user is then able to interact with the GUI, by means of following the mental model of software functionality [74].

The fact that GUIs are used already more than 30 years does not mean that they are ideal, in fact, they posses several drawbacks. They limit the input of the user to only two DoFs and applications are bound to give the feedback to the user in the same format. The abilities of the users learned in 3D world should be mapped into 2D. Consider for example people, who have not been working a lot with GUIs previously and need to learn new software with completely different GUI. The curve of their learning process is not steep, because they need to learn new mental model of GUI functionality. GUIs also suffer from screen estate limitations: when users have, many different windows open and often repeat the same operations several times [73].

Probably the most significant drawback notable by GUI developers is a difficulty in testing it during software development process. Typical GUI testing process consists of: 1) test case generation, 2) output generation, 3) execution of test cases and verification with output and 4) coverage analysis. First, tester generates and numerates the sequence of GUI events, manually or using a model of GUI. The major difficulty here is the fact that there

[23] CLI (Command-Line Interface) is a mechanism for interacting with a computer operating system or software by typing commands to perform specific tasks.
[24] WIMP –Windows, Icons, Menus and Pointing devices.

are enormous amounts of permutations of interaction with GUI. Additionally, each GUI event can result in a number of GUI states. Consequently all that results into huge amount of test cases. Second, tester specifies the output for each GUI event. The form of the output can be screen shots, positions of widgets, titles and others. At this step, the major difficulty is that outputs should be generated for each event. Knowing that each event causes subsequent events the number of outputs increases with each event. This is resource intensive task. Third, tester executes test cases (sequence of GUI events) and compares the actual output to the one specified earlier. The mismatch is then reported as an error. The main difficulty here is the comparison of all test case outputs for each event of GUI. Finally, the coverage of test cases is evaluated. If test cases cover all the program statements and branches at least once, the coverage criterion is considered to be satisfied. As it is not enough, nearly all GUI are developed using agile software development processes. At each short iteration end the running GUI and underlying "business logic" are presented to the users, after that the feedback is received. Often, after receiving the feedback developers redesign and redevelop GUI. For testing, this means there are obsolete test cases and outputs, which should be updated and managed. Consequently, regression testing requires even more resources than single test [60], [72], [74], [75].

Usually during desktop paradigm, GUI in simulators for mobile robots was used to support simulation of 2D robots and environments, see Figure 3.2 for example.

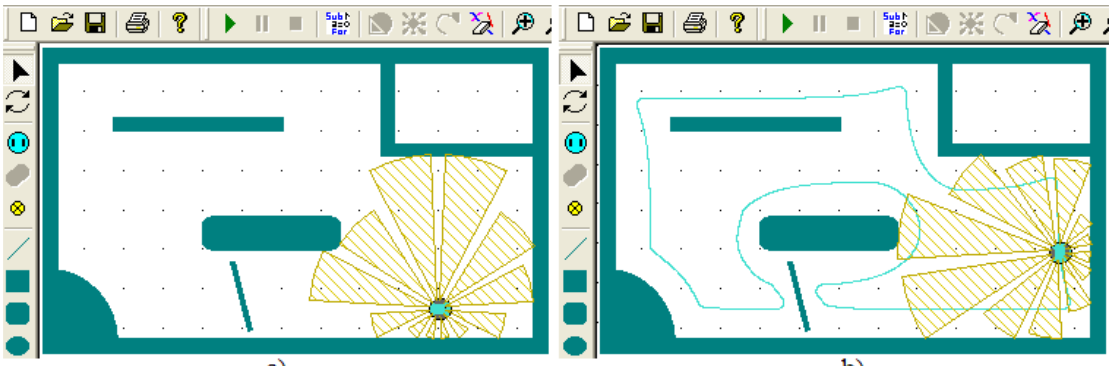


Figure 3.2 GUI of MobotSim simulator and simulation example of differential drive robot equipped with ultrasonic sensor: a) before simulation and b) at the end of simulation

The figure above depicts the control panels of the MobotSim [25] simulator (on the left and top) using which, user models the simulation environment from such elements like rectangles, lines, and circles (dark cyan color) by simply dragging and dropping them into the simulation environment. Later, using the same drag and drop technique, adds predefined structure of differential drive robot (cyan circle) equipped with ultrasonic sensors. Here dark yellow color indicates the rays of ultrasonic sensors. The model of the robot can be modified by simply double clicking on the robot in simulation environment and interacting with build in widgets. After that user implements the controller for the robot to avoid the walls and starts simulation. Figure 3.2 b, depicts the path followed by the robot (light cyan line) during simulation runtime.

3.3.2 Natural Interaction Paradigm

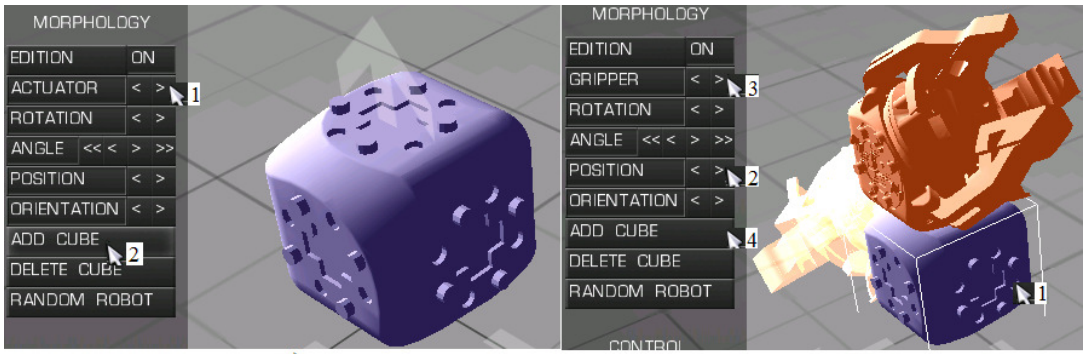
Desktop paradigm primary promotes functional interfaces, however new paradigm called natural interaction paradigm promotes direction towards natural, intuitive and adaptive interfaces. The paradigm emphasizes the use of natural human-to-human interaction modeling in computers. According to this paradigm, the ultimate interface of the future should make use of natural human-to-human interactions, human perceptions, and our tendency to interact with technology. As a result, such PUIs will take advantage of human

and machine capabilities as well as sensing, perceiving and reasoning.

There are several major advantages of PUIs over GUIs and they are the following. First, PUIs provide more natural and expressive model of dialog with computer. Second, PUIs reduce dependence on proximity to computer, which is dictated by use of such pointing devices like mouse and keyboard. Thirdly, encoding of natural interaction skills into the PUIs leads to decreased time to learn the PUI or in best-case scenario eliminates the need for learning it at all. PUIs will cover wider range of users and tasks, moreover PUIs are more user-oriented, rather than device oriented [72].

Nowadays, natural interaction paradigm is in the research stage. However, already now researches argue that interaction techniques used in virtual and augmented[25] realities, PUC (Pervasive and Ubiquitous Computing[26]), CAD and 3D graphics should play significant role. With respect to simulators, there are several interaction techniques and their varying implementations, which are relevant to the project. These are 2D GUI and direct 3D object manipulation.

2D GUI technique is widely spread and can be characterized as user interaction with 3D objects through 2D GUI embedded into or separate from simulation environment, see Figure 3.3 and Figure 3.4, respectively.



a) b)

Figure 3.3 2D GUI embedded into simulation environment of Cube Interface: a) after adding actuator module and b) after adding gripper module

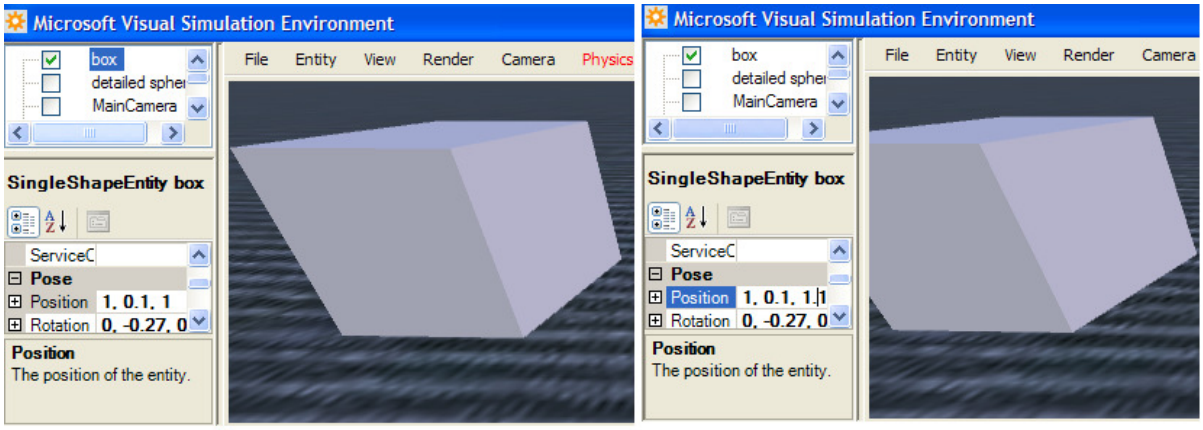
The above figure depicts user interaction with Molecube modules through the 2D GUI embedded into the simulation environment of a Cube Interface simulator. First user chooses the type of the module (here actuator), secondly adds it into simulation environment (see Figure 3.3 a, for sequence of interaction with 2D GUI). After that selects the added module, chooses the position (it is indicated by bright representation of a module) and the type of a new module (gripper), and clicks add cube button. As a result, the gripper module is added on top of actuator module. See Figure 3.3 b, for sequence of interaction with 2D GUI. In the figure above, only part of 2D GUI is shown for concision.

The main advantages of such 2D GUI interaction technique are ability to interact with simulated objects even during simulation runtime and ease in learning mental model of GUI functionality because it is intuitive. Of course assuming that user is familiar with Molecube modular robot. Moreover, this particular example of interaction with simulation environment and objects in it is probably the best from the point of view on QPSS in the field of modular robots, because it takes a short amount of time for the user to assemble even the most difficult morphology of the robot. This is most likely because the modules

[25] While the virtual reality systems work with purely computer-generated reality, the augmented reality combines these with the real world.

[26] PUC by Collins Cobuild: “Something that is ubiquitous is everywhere or seems to be everywhere at the same time” and “ Something, especially a quality, or smell, that is pervasive is present or felt throughout a place or thing”

are pre-modeled and interactive assembly is limited to modular robot design decisions.



a) b)

Figure 3.4 2D GUI separate from simulation environment of MRS: a) before changing box position and b) after

The above figure depicts user interaction with box object in simulation environment of MRS through 2D GUI separate from simulation environment. First user selects the box element in tree view (highlighted), because of this event the properties of the box object are displayed below (see Figure 3.4, a). Next user changes the value of box position in corresponding field in 2D GUI (highlighted) and clicks with the mouse somewhere in simulation environment. As a result box moves to the left (see Figure 3.4, b). In the figure, only part of 2D GUI is shown for concision.

The main advantages of the above usage of 2D GUI are extensive coverage of simulator functionalities and ability to set/get precise values of object's properties. On the other hand, there is minor disadvantage that all the functionalities are implemented in specific way. Thus, user should spend significant amount of time in order to master them.

Another notable implementation technique used in 2D GUI is a tree view. In general, it is parent-child abstract representation of objects in simulation environment. In simulators, it is used in two major ways: 1) for display and modification of simulation environment and objects in it at simulation runtime (example above) and 2) additionally supporting modeling of simulation environment and objects in it during simulation runtime . As a rule, in the first case the modeling of simulation environment and objects in it is achieved by means of general-purpose programming language (s), like C # in example above (See Code Snippet 3.1).

Code Snippet 3.1 Add 3D box in MRS, by means of C# [22]

```
void AddBox(Vector3 position){
    //Define the dimensions of the entity
    Vector3 dimensions = new Vector3(0.2f, 0.2f, 0.2f); // meters
    // create simple movable entity, with a single shape
    SingleShapeEntity box = new SingleShapeEntity(new BoxShape(
            new BoxShapeProperties(100, new Pose(),dimensions)),
            position);// mass in kilograms, relative pose, dimensions
    // Name the entity. All entities must have unique names
    box.State.Name = "box";
    // Insert entity in simulation.
    SimulationEngine.GlobalInstancePort.Insert(box);}
```

The source code above is an example of adding the box entity in simulation environment of MRS. This type of modeling of simulation environment and objects in it has two major disadvantages. First, user should have a good knowledge of general-purpose programming language. Second, know the inner workings of simulation framework. In this

case, the learning curve is far from being steep. The major advantages here are: a) ability to modify the model of simulation environment and objects in it at simulation runtime. Consequently, there is no need to restart simulation, whenever something should be modified only when added. b) Possibility to share the model of simulation environment and objects in it saved in particular format. In case of MRS it is XML[27].

2D GUI tree view can be also used for modeling of simulation environment and objects in it during simulation runtime. Here user defines the objects and environment not by means of the code, but through abstract tree view component, see Figure 3.5 for example.

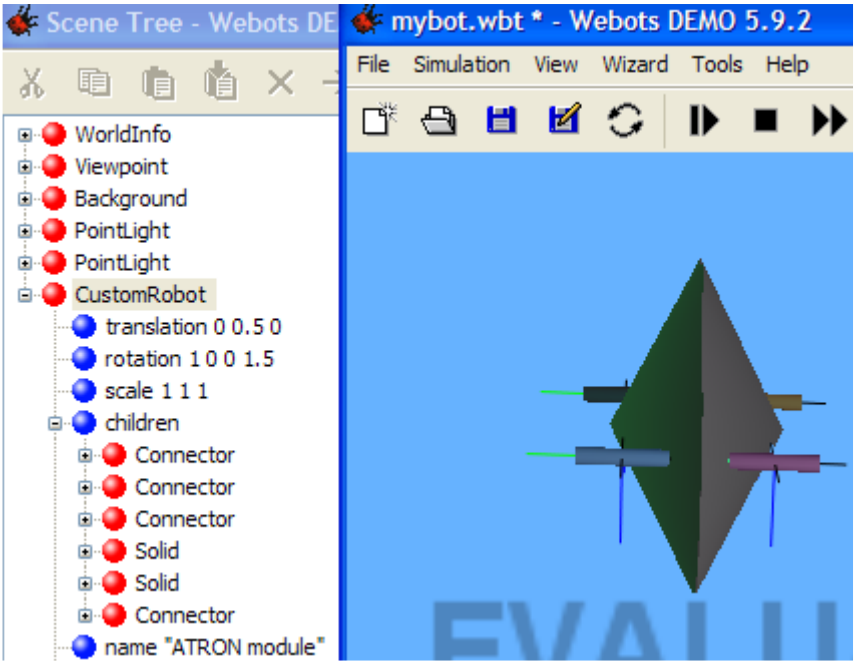


Figure 3.5 2D GUI tree view (left) in Webots 5 and rough 3D model of ATRON (right) in simulation environment

The above figure presents rough design of ATRON module and tree view in Webots 5, where user by means of modifying the pre-existing abstract template in a tree view is able to model both simulation environment and a robot at simulation runtime. Later write controller for the robot and associate the model with the controller. Only part of 2D GUI tree view is shown in the figure for concision. This kind of tree view utilization has two major advantages. First, user unfamiliar with inner workings of simulator can intuitively build models of robots at relatively short amount of time and during simulation runtime. Second, there is no need for user to know one of the particular general-purpose programming languages to model simulation environment and objects in it. Moreover, the models of simulation environment and objects in it can be shared between the users. Furthermore saved, imported, and exported, in such common standard like VRML. That leads to shorter learning curve of mental model of simulator functionality, sharing, and reuse of models.

Next type of implementation technique used in 2D GUI, is a widget support for CAD interface, see Figure 3.6. Typically, simulators supporting this kind of interaction technique become something in between the CAD software development kit (SDK[28]) and simulator. This means that user first models the simulation environment and objects in it

[27] XML (eXtensible Markup Language) is a general-purpose specification for creating custom markup languages.

[28] SDK is typically a set of development tools that allows a programmer to create applications for a certain software package, framework and so on.

using CAD SDK integrated in simulator, later implements controller, and runs simulation.

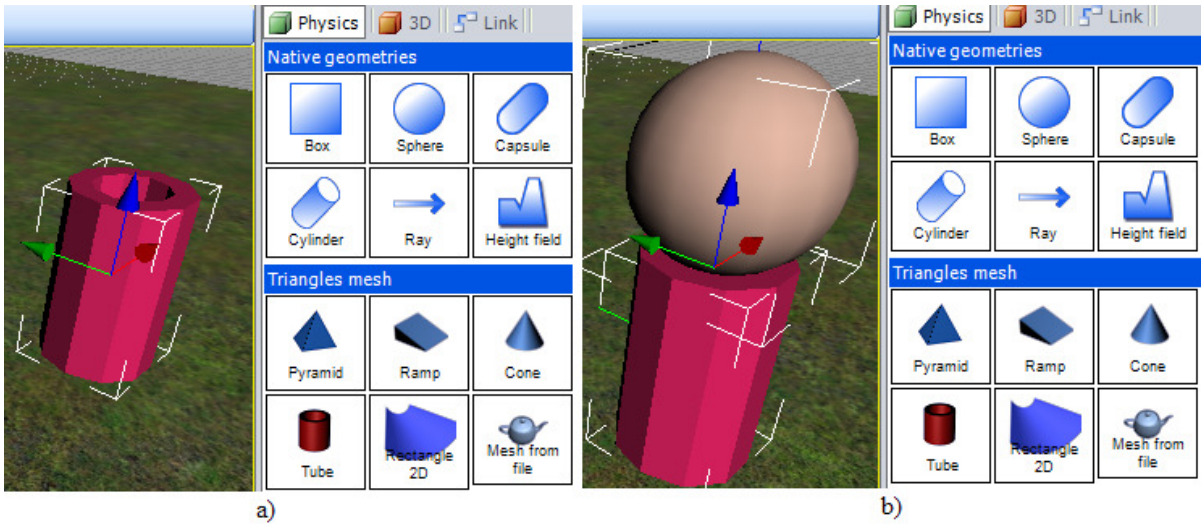


Figure 3.6 CAD interface (right) in Marilou and example of use: a) tube is added and b) sphere is added on top of the tube

The figure above depicts user interaction with objects (tube and sphere) in Marilou simulator, through the CAD interface, implemented by means of GUI widgets. Here the user just selects the tube icon and points where it should be and as a result, he/she is able to define the dimensions of the tune by means of moving the mouse along the chosen coordinate. Later the same procedure is repeated for the sphere object. The main advantages for using this kind of modeling of objects in simulation environment are the much faster learning curve of GUI mental model, because it is more intuitive, high speed of modeling and direct 3D object manipulation. On another hand, there are several disadvantages, like: a) the modeling can proceed only before the actual simulation runtime and b) it is difficult to design the GUI covering the wide range of user requirements.

While 2D user interfaces proved to be effective interaction technique for a long time, currently there is desire to overcome disadvantages of it. 3D user interfaces have three major advantages over 2D. First, people spend all their live around the 3D objects and learn manipulation, navigation, and interpretation of spatial relationships between them. These skills are not transferred well on 2D interfaces. Second, people should learn new skills for interacting with 2D interfaces. Moreover, 2D user interfaces constrain human skills to smaller set than 3D interfaces. Consequently, 3D interfaces should provide natural interaction with computer, used by humans during their lifetime. Third, 3D interfaces have wider design space rather than 2D interfaces. It is possible to design wider amount of 3D interfaces than 2D. Despite the benefits of 3D interaction, nowadays there are still several challenges to be addressed. One of them is the high cost of hardware necessary to support it, which is often limited to the laboratory or academia usage only. Next is to provide set of techniques necessary to interact with 3D interfaces [73].

Direct 3D object manipulation is interaction technique with 3D graphics objects. Furthermore, one of the techniques from the 3D interfaces. In simulators, main advantages of it are direct manipulation of graphics objects during simulation runtime, natural and intuitive interaction with 3D objects and it is leveraging functionality between 2D and 3D interfaces, consequently simplified design of 2D interfaces. Disadvantages are high coupling with the source code of simulator and complexity to design purely 3D object manipulation without support of 2D interfaces. Several examples of 3D object manipulation were already mentioned, like for example selection of Molecube module in Cube Interface during simulation runtime, see Figure 3.3 above and manipulation of 3D

objects in Marilou simulator, see Figure 3.6. Currently USSR supports only one direct manipulation technique, which is moving the objects in simulation environment during simulation runtime with the mouse. See Figure 3.7 for example.

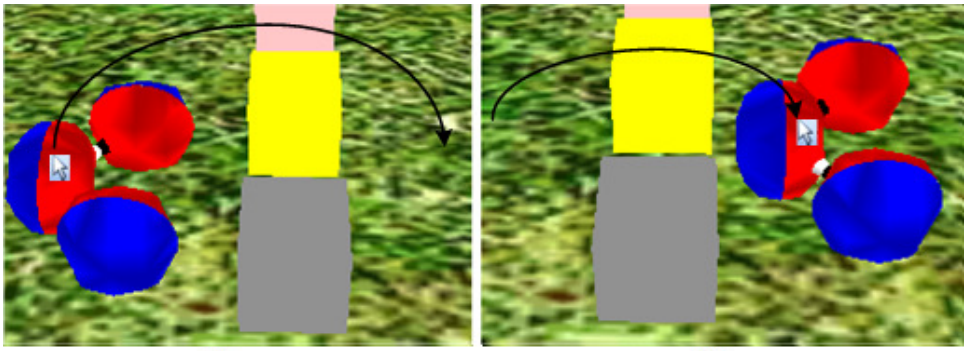


Figure 3.7 Example of direct 3D object manipulation in USSR: a) before moving simple ATRON vehicle with the mouse and b) after

The figure above presents the example of direct 3D object manipulation in USSR, where user selects the simple ATRON vehicle by clicking on it with the mouse and then moves it by means of mouse movement over the line of box obstacles.

After looking through interaction techniques used in simulators, let us discuss application of them in the simulators for modular and mobile robots in more detailed fashion.

3.3.3 Interaction in Simulators for Modular and Mobile Robots

Before going deeper into detailed discussion about interaction in simulators for modular and mobile robots, let us first clarify the emphasis of discussion. The emphasis can be presented by means of such terms like specification language and interaction technique. Interaction techniques were introduced in previous section. Specification language on the other hand should be explained more thoroughly. In this context, term specification language means the way user defines (models) simulation environment and objects in it. There are two major ways to achieve that and they are the following: by means of general-purpose programming language and modeling (mark up) language. The example for modeling simulation environment and object in it by means of general-purpose programming language was already discussed above (see Code Snippet 3.1). In most simulators, modeling languages (for example XML and VRML) are used for the same purpose. For instance, the VRML file describing the model of simulation environment and objects in it can be just imported or exported from the simulator, in some cases even modified. Probably the best example here is Webots 5 simulator. There is one more way to model the simulation environment and objects in it and it is by means of DSL[29] (Domain-Specific Language). Exploration of simulators for modular and mobile robots indicates that it is not a popular approach, most likely because it requires extensive knowledge of the problem domain, abstraction level in the problem domain, syntax and semantics of the language, and additional efforts during implementation. To be more precise only one simulator was discovered, which is using DSL for modeling of simulation environment and objects in it. It is called SimRobot [27] and uses DSL called RoSiML[30], which is

[29] DSL is a programming language or specification language implemented for a particular problem domain, a particular problem representation technique, and/or a particular solution technique.

[30] RoSiML (Robot Simulation Markup Language) is a DSL implemented in Fraunhofer Institute for Autonomous Intelligent Systems (Germany) for specification of simulation environment and objects in it.

implemented as XML schema[31].

All above three specification languages can be considered as special case of interaction, however they are at the lower level of interaction than the ones introduces in previous sections. This is because they require much wider knowledge from user before using simulator effectively and consequently they are on the lower level of abstraction from interaction point of view.

The result of case study and analysis of literature about simulators for modular robots from the interaction perspective see Table 3.3.

Table 3.3 Survey of simulators for modular robots in comparison to USSR, from interaction point of view. Here: 3D man. – direct 3D object manipulation, Implement. – implementation.

Name of simulator	Specification language (s)	Interaction technique (s)	Robot programming	Implemen. language	Usage
USSR	Java	3D man.	Java	Java	MMMI
New Unit Simulator	C++	2D GUI, separate	Macros[32]	C++	M-Tran developers
Galina	C++	None	C++	C++	SuperBot developers
Cube Interface	None	2D GUI embedded, 3D man.	Macros	C++	worldwide
Webots 5	VRML97, VRML2.0	2D GUI separate, tree view, 3D man.	C, C++, Java, Python[33]	C,C++	Over 450 universities
DPRsim	C,C++	None	C,C++	C,C++	Catom developers

From the table above it is possible to come to conclusion that simulators for modular robots follow two major implementation tendencies highly dependable on orientation towards specific user group. First orientation is towards user group, which can be called developers of modular robot (experts). That means the simulator is implemented with minimal requirements from interaction point of view, because developers of modular robot are not interested to have simulator with high-level interaction techniques at current state of simulator. Hypothetical reasons for that can be several. First of all, developers know the implementation language and inner workings of simulator so well that they are able to implement what is required fast enough using general-purpose programming language without implementing interaction techniques supporting it. Next, developers are not interested in widening the user group from experts to beginners in current version of simulator. Furthermore, development of simulator takes a lot of efforts and time, which is why developers are more interested in implementation of problem domain specifics first rather than concentrating on interaction techniques. Last can be the fact that simulators are still under development process and implementation techniques are still to come. As a result, such simulators are bound to be used by developers and most likely worldwide research colleagues and students, because of lack of interaction techniques, which widen the user group. Simulators following this tendency are USSR, Galina, and DPRsim.

[31] XML schema is a description of a type of XML document, typically expressed in terms of constraints on the structure and content of documents of that type.

[32] Macro is a rule or pattern that specifies how a certain input sequence should be mapped to an output sequence according to a defined rule or procedure.

[33] Python is a general-purpose, high-level programming language. Its design approach emphasizes programmer productivity and code readability.

Second orientation is towards wider group of users including experts and beginners. Here developers try to reach higher levels of interactivity and abstraction, by including such techniques like 2D GUI separate and embedded into simulation environment, tree view, and elements of 3D interfaces, like 3D object manipulation. Those simulators are New Unit Simulator, Webots 5, and Cube Interface. New Unit Simulator is somewhat special case, because it is not available for download worldwide, but it seems that its GUI is well designed to cover even requirements of beginners. Most likely, it is done to support the use of simulators by students internally. Webots 5 is interesting from the perspective of its support for modeling of simulation environment and objects in it during the simulation runtime, by means of interacting with 2D GUI and tree view. This solution seems to be effective to achieve QPSS in the mobile robotics field, because robots are modeled once and do not remodeled so often, moreover the controllers can be easily reassigned. The solution, which is more suitable for QPSS in simulators for modular robotics is something like in Cube Interface. Here the modules of the robot are pre-modeled and later are used to simply combine into whatever robot morphology necessary with the help of 2D GUI and one element of 3D object manipulation.

Interesting to note that unspoken agreement is to use C, C++ general purpose programming languages for implementing simulator. This is most likely because of support for interfacing with underlying engines. As a result, the programming of the robot in simulation environment (controller) and specification languages are the same as implementation languages in most cases.

In conclusion, it is obvious that the best representatives of simulators for modular robots from interaction perspective are Cube Interface and Webots 5. The main advantage of Cube Interface is the fact that it is intuitive and QPSS is achieved very fast. The only disadvantage is that the simulator is not generic, but robot specific. The main advantages of Webots 5 are ability to model robot at simulation runtime and specify the particular model of the robot as a definition. Consequently, later use it as kind of copy paste action, meaning that the same model can be used several times in different locations and with minor changes. It is difficult to find disadvantage as it seems it is covering most of requirements for QPSS during modeling of modular robots.

Next, let us discuss the simulators for mobile robots in comparison to USSR from the interaction perspective, see Table 3.4.

Table 3.4 Survey of simulators for mobile robots in comparison to USSR, from interaction point of view. Here: 3D man. – direct 3D object manipulation, Implement. – implementation.

Name of simulator	Specification language (s)	Interaction technique (s)	Robot programming	Implem. language	Usage
USSR	Java	3D man.	Java	Java	MMMI
Gazebo	XML	2D GUI separate,	C,C++	C,C++, Python	worldwide
Webots 5	VRML97, VRML2.0	2D GUI separate, tree view, 3D man.	C, C++, Java, Python	C,C++	Over 450 universities
MRS 1.5	XML	2D GUI separate, tree view	C++,C#, Python	C#	worldwide
Marilou	VRML	CAD interface	C,C++ ,VB# , C#, J#, CLI	C,C++	worldwide

From the table above it is clear that simulators for mobile robots are oriented towards wide

range of user group including experts and beginners. The major indicators proving that are support for specification languages, different interaction techniques, and multiple general-purpose programming languages for implementing robot's controller (robot programming). As a result, these simulators are used worldwide. USSR in comparison to them needs improvements in all considered areas in order to attract wider range of users.

It is worth to stress that most of the simulators are implemented by means of C, C++ general purpose programming languages. This is most likely dictated by historic context, performance or ease to interface underlying engines.

Natural following step in our discussion is let us attempt to construct the taxonomy of interactions techniques used in simulators for modular and mobile robotics and in this way organize our knowledge in more concise form. It is well-known fact that taxonomy is a practice and science of classification, originating from biological sciences. Classification, in turn, is a broad term with applications far beyond the biological sciences and refers to the arranging of concepts into categories according to specific criteria. Typically, classification reminds hierarchical structure with parent-child relationships. Some scientists (for example Immanuel Kant) argue that the human mind naturally organizes its knowledge about the world in such structures [26].

The hypothetical scientific method for classification can be broken into several basic steps, like the following: identification of the problem in interest, location and analysis of information, identification of categories and criteria, continuous update and iterative process. Let us walk thought each step briefly and finally construct the taxonomy of interaction techniques used in simulators for modular and mobile robotics.

Identification of the problem in interest often starts with identification of what is going to be classified. In our case, these are interaction techniques. Next, is to identify from which point of view they are going to be classified. In other words, on what the emphasis will be concentrated on. Due to the fact, that one of the goals for the project is to investigate the methods (ways) for modeling of simulation environment and objects in it, existent nowadays, that where the emphasis is on. Consequently, the emphasis or criteria for classification is interaction techniques used to model simulation environment and objects in it in simulators for modular and mobile robotics.

Usually, at the start of second step, all information about the problem of interest is scattered all over different sources of information (books, internet pages, scientific papers, and so on). Moreover, in most cases none existent for identified problem in complete form. That is why the sources of information are roughly filtered according to several factors like: availability, relevance, reliability, quality, amount of information, and others. After that, they are being read, compared, and analyzed, according to the same factors. At the same time, the most relevant and valuable information is filtered out. For example from the number of simulators read about during this project, only several of them where considered to be relevant and valuable to discuss deeper than others. All of them are already mentioned previously. The reason for choosing those was the fact, that they contain potential, hybrid, or inspiration for solutions to solve the problem with interaction in USSR. The list of other considered simulators can be found in Appendix B.

During the third step, the categories and criteria are identified. Moreover, the relationships between them are considered. Specification languages are the first criteria for classification by priority, because it is considered to be the bottleneck for achieving QPSS. That is due to observation that usually user spends more time for coding the simulation environment and objects in it, rather than coding the actual behaviors of a robot. According to the same criterion, interaction techniques in simulators for modular and mobile robots can be hypothetically split into three major categories. These are general-purpose programming language, modeling (markup) languages and DSLs (see Figure 3.8).

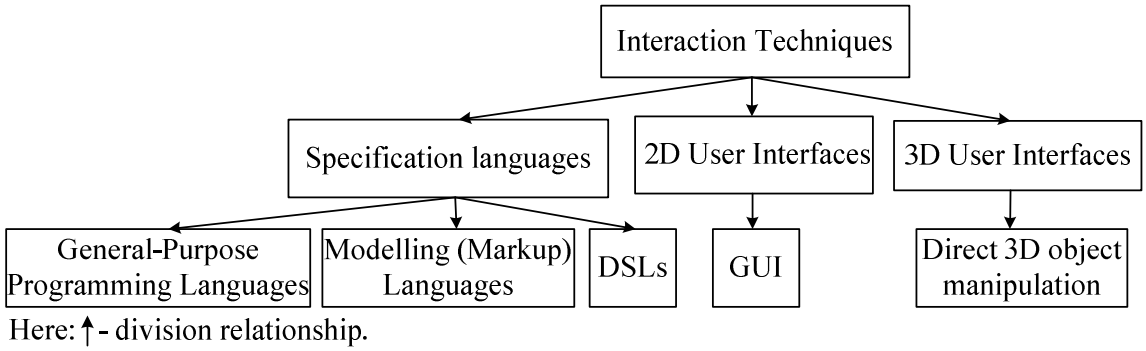


Figure 3.8 A taxonomy of interaction techniques used in simulators for modular and mobile robotics

In first case user models the simulation environment and objects in it using one of the general-purpose programming languages like C++, C# or Java. In second case user models objects in simulation environment using languages like XML, VRML. Finally, DSL is category where user uses specific languages describing problem domain.

Taking broad view on specification languages, the natural cause for diversity of solutions can be software evolution process or historical context. Usually software evolution of simulators goes through several stages. In early stages, the modeling of simulation environment and objects in it is achieved by means of general-purpose programming languages. Later, in order to attract more users and support interchange of models between different simulators, developers add support for modeling languages, or DSL. This adds abstraction layer and makes it easier and faster for users to achieve QPSS.

Second criteria for categorization is user interface, which is divided in two major categories called 2D and 3D user interfaces. In case of 2D user interfaces, there is GUI, which is dominating solution with different additional implementation techniques like tree embedded and separate GUI from simulation environment, tree view, and CAD interface. As for 3D user interfaces, there are unlimited ways to implemented them and they are still in research stage. So far, direct 3D object manipulation is the one already applied in simulators for modular and mobile robots.

Final and probably never-ending step in classification is continuous update of data in it. That is because time goes, some of the interaction techniques stop to be implemented and supported. Others appear and should be classified. The similar case is for categories and criteria. Some of them become irrelevant with the time. The iterative process on the other hand is useful at any time, but in the scope of classification. It can be characterized as localization of subcategories of the previous categories or deeper investigation of identified problem.

3.4 Summary

In this chapter, the discussion about simulators for modular robots we started from introduction of taxonomy of simulators and their main aspects (underlying engines). Notable tendency here is emphasis on dynamic simulation, involving simulation of laws of dynamics. Later we compared USSR to other simulators for modular and mobile robots from the perspective of underlying engines and support for robots. In general, it is possible to summarize that different simulators use similar underlying engines for rendering graphics and ensuring dynamics and collision detection (like OpenGL and ODE). It is kind of unspoken agreement to use OpenGL as graphics engine and ODE for dynamics and collision detection. This is most likely due the fact that it is a well-known and cross-language, cross-platform APIs. There are of course exceptions, which use different engines

because of several reasons like: necessity for hardware acceleration, complete control over the code and so on. The only two representatives of generic simulators for modular robots discovered during the analysis and discussion are USSR and Webots 5. In overall, it seems that these simulators are the only two existing simulators with generic support for simulation of modular robots. Currently USSR supports more modular robots like (ATRON, M-Tran and Odin) than Webots (supports YaMoR and M-Tran III). Moreover, USSR follows OSS paradigm, where Webots is proprietary.

In second part of the chapter we discussed two main paradigms of HCI field, they are the following: Desktop and Natural Interaction paradigms. Furthermore, we emphasized that HCI is now on the edge between the two. Currently the research proceeds towards supporting Natural Interaction paradigm by means of PUIs. Next we exemplified the interaction techniques of each paradigm. Following was comparison of USSR to simulators for modular and mobile robots from interaction perspective. Here we compared the simulators by means of support of interaction techniques, specification languages, robot behavior programming usage and so on. Comparing USSR to simulators for modular and mobile robots from point of view of interaction it is notable that USSR lacks support for higher level interaction. Consequently it is limited to be used only by expert users. Finally, closing the discussion about the interaction, the taxonomy of interaction in simulators for modular and mobile robots was devised.

The summary of aspects and interaction techniques of simulators for modular and mobile robots discussed during this chapter see in Table 3.5, on the next page.

Table 3.5 Aspects and interaction techniques of simulators for modular and mobile robots in comparison to USSR
Here: 3D man. – 3D object manipulation, Implement. – implementation.

Simulators for modular robots									
Name of simulator	Supported robot (s)	Graphics engine (s)	Physics engine	License	Download	Specification language (s)	Interaction technique (s)	Robot programming	Implemen. language	Usage
USSR	Multi-robot	OpenGL and jME	ODE	BSD	available	Java	3D man.	Java	Java	MMMI
New Unit Simulator	M-Tran	OpenGL, Vortex		proprietary		C++	2D GUI, separate	Macros	C++	M-Tran developers
Galina	Superbot	OpenGL				C++	None	C++		SuperBot developers
Cube Interface	Molecube	OpenGL, ORGE	PhysX	GNU	available	None	2D GUI embedded, 3D man.	Macros		worldwide
Webots 5	Multi-robot	OpenGL	ODE	proprietary		VRML97, VRML2.0	2D GUI separate, tree view, 3D man.	C, C++, Java, Python	C,C++	Over 450 universities
DPRsim	Catom	OpenGL	ODE	GNU	available	C,C++	None	C,C++	C,C++	Catom developers
Simulators for mobile robots										
Gazebo	Multi-robot	OGRE	ODE	GNU	available	XML	2D GUI separate,	C,C++	C,C++, Python	worldwide
Webots 5		OpenGL		proprietary		VRML97, VRML2.0	2D GUI separate, tree view, 3D man.	C, C++, Java, Python	C,C++	Over 450 universities
MRS 1.5		XNA	PhysX			XML	2D GUI separate, tree view	C++,C#, Python	C#	worldwide
Marilou		engine M	ODE			VRML	CAD interface	C,C++, VB#, C#, J#, CLI	C,C++	worldwide

Chapter 4

4 Problem Analysis

"Have you got a problem? Do what you can, where you are, with what you have got".
— Theodore Roosevelt

Chapter Objectives
1. Define and exemplify the problem with which author was tackled during Master Thesis.
2. Analyze and learn from currently existing approaches to it.

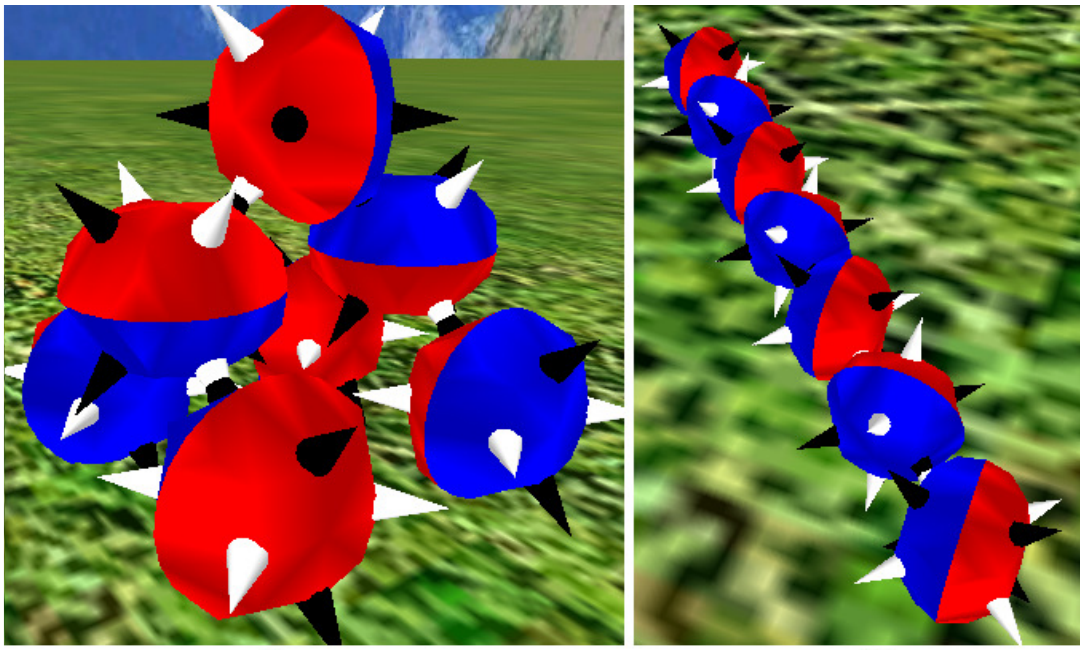
4.1 Introduction

In this chapter we will first present the problem, which is investigated and improved on in this Master Thesis named as: quick prototyping of simulation scenarios (QPSS). In general, the problem can be divided into two parts named as: construction of modular robot's morphology (shape) and assignment of behaviors (controllers) to modular robot's modules in simulators. The first part is concerned with assembling (constructing) the morphology of a modular robot from modules in a simulation environment. In a way, it is similar to CAD modeling of modular robot's morphology from elements that are modules (or their constituents). The main problematic issue here is how to optimize it in a sense of speed of efficiency in interaction, ease of use and diversity of modular robots. The second part is concerned with assignment of behaviors to modules in the morphology of a modular robot. The emphases here are also the speed of efficiency in interaction, ease of use, diversity of modular robots and additionally reuse of behaviors on similar morphologies. In order to investigate and exemplify both we will use USSR as our experimental platform. Moreover, we will sometimes reuse the examples already included in USSR for simplification of communication with the users familiar with them. After familiarization with the problem, we will analyze currently existing approaches to it in simulators for modular robots. To be more precise we will analyze two simulators which were discussed earlier and they are: Cube Interface and Webots 5. The main reasons for choosing these are: their relation to modular robots and ability to serve as inspiration, moreover the possibility to use them for comparison with USSR. By means of this analysis, we will identify the features of simulators which alleviate the problem under investigation in simulators for modular robots. As a result, at the end of the chapter will not only be able to identify the problem in the context of USSR, but also extract the features which we learned from other simulators and later on reuse and enhance them in context of our problem.

4.2 Problem Definition and Description

After an analysis of theoretical basis behind modular robots, their simulators addressed in previous chapters and discussions with experts of the field, we observe that currently physical experimentation with the modular robots is limited to a maximum of fifty modules in best-case scenario. Actually, the physical experimentation often involves significantly smaller numbers of modules. Fifty indicates the number of produced modules, but all of them are rarely used during a single physical experiment due to complexity and cost factors [55]. That is why scientists prefer to use simulators, where the number of modules involved in simulations is much greater. At the same time, scientists agree that

there is a proper basis of explored designs for modular robots and tend to focus their attention on experimenting with change of their morphology (shape). This means that a modular robot should be able to change its initial morphology into another. Moreover, a modular robot should be able to decide which morphology it should use according to environmental conditions. For instance, let us consider an ATRON car configuration (see Figure 4.1, a), which is a good enough morphology for moving on flat terrain. However, if it meets a narrow entrance the car is not able to fit into it. In this place the ATRON car should self-reconfigure into another type of morphology, for example snake (see Figure 4.1, b) and continue on its way.



a) b)

Figure 4.1 Examples of ATRON morphologies (configurations) in simulation in USSR: a) Car and b) Snake. Here: black and white cones are connectors.

Another point to stress is that the ATRON car should detect the narrow entrance and decide if it is able to fit into it. A natural choice here is to use simulators, because these kinds of simulations consist of experimentation with the robot's morphology (frequent change of it) as well as implementation of varying controllers (behaviors) for different modules. From the perspective of large numbers of modules with varying behaviors, simulators give significant advantages over physical experimentation because experiments can be carried out with larger numbers of modules, which is cheaper and faster. Furthermore, there is no need to physically or wirelessly connect and upload the controller to specific module in the morphology of modular robot, which is a time consuming task. On the other hand, new difficulties arise in simulations. Two major of them, which are emphasized here, are the following: 1) frequent construction of modular robot morphologies (programming them), consisting of varying amount of modules and 2) assigning several different behaviors for specific modules in the morphology of a modular robot. The first problem is considered not as significant as second. However, both problems complicate the process of QPSS and as result, discourage the users of using the simulators for modular robots. Probably the best way to understand each of these problems is to experience them on your own. However, it is not realistic in our case, so let us employ second known best way, which is description of the problem with the help of examples.

4.2.1 Construction of Morphology

For example, in order to construct ATRON car morphology (consists of seven modules) for use in simulation environment of USSR, Ulrik Pagh Schultz (expert of USSR) has spent about twenty minutes of programming during the first attempt, after generalization of the source code it took ten minutes and finally currently it takes five minutes. The main difficulties here are specifics of USSR implementation, 3D spatial definition of rotations and positions of the modules. Taking into account that Ulrik Pagh Schultz is an expert in USSR and that the ATRON car morphology consists of only seven modules, the time spend to construct it is too long. Just in case we are not yet completely convinced, let us take a deeper look what are the difficulties related to rotations, positions and specifics of USSR from programmer perspective in order to get the feeling of burden of information programmer is dealing with. For that, let us consider the source code for constructing ATRON car morphology consisting of seven modules. Refer to the code snippet beneath.

Code Snippet 4.1 The positions of the modules in ATRON car configuration

```
/*Container for storing global positions of the modules and their
annotations*/
ArrayList<ModulePosition> mPos = new ArrayList<ModulePosition>();
/* The distance between two physical ATRON modules */
float UNIT = 0.08f;
/* The position of the first module, where the format of the object is:
(the name of the module, position and rotation */
mPos.add(new ModulePosition("driver0", new
VectorDescription(2*0*ATRON.UNIT+Xoffset,0*ATRON.UNIT+Yoffset,0*ATRON.UNIT
), ATRON.ROTATION_EW));
/* The position of the second module(the same format, like above)*/
mPos.add(new ModulePosition("axleOne5", new
VectorDescription(1*ATRON.UNIT+Xoffset,1*ATRON.UNIT+Yoffset,0*ATRON.UNIT),
ATRON.ROTATION_UD));
/* The position of the third module(the same format, like above)*/
mPos.add(new ModulePosition("axleTwo6", new VectorDescription(-
1*ATRON.UNIT+Xoffset,-1*ATRON.UNIT+Yoffset,0*ATRON.UNIT),
ATRON.ROTATION_UD));
/* The position of the fourth module(the same format, like above)*/
mPos.add(new ModulePosition("wheel1left", new VectorDescription(-
1*ATRON.UNIT+Xoffset,-2*ATRON.UNIT+Yoffset,1*ATRON.UNIT),
ATRON.ROTATION_SN));
/* The position of the fifth module(the same format, like above)*/
mPos.add(new ModulePosition("wheel2right", new VectorDescription(-
1*ATRON.UNIT+Xoffset,-2*ATRON.UNIT+Yoffset,-1*ATRON.UNIT),
ATRON.ROTATION_NS));
/* The position of the sixth module(the same format, like above)*/
mPos.add(new ModulePosition("wheel3left", new
VectorDescription(1*ATRON.UNIT+Xoffset,-
2*ATRON.UNIT+Yoffset,1*ATRON.UNIT), ATRON.ROTATION_SN));
/* The position of the seventh module(the same format, like above)*/
mPos.add(newModulePosition("wheel4right",newVectorDescription(1*ATRON.UNIT
+Xoffset,-2*ATRON.UNIT+Yoffset,-1*ATRON.UNIT), ATRON.ROTATION_NS));
```

The source code above is an example of adding the ATRON car morphology to simulation environment of USSR. It is only the part of the code responsible for morphology, but already now it is obvious that programmer should be aware of such USSR specifics like: 1) `ModulePosition` class, which defines the global position of the module and its annotations in more abstract form; 2) naming the module with unique name, for example above: `"driver0"` is the name of the first module in the container. 3) `VectorDescription(x,y,z)` class, which defines the position of the module in Cartesian coordinate system; and 4) `RotationDescription(x,y,z)` class, which

defines the rotation of the module in the same coordinate system. In the example above it is represented as a constant, for instance `ATRON.ROTATION_EW`, meaning that it is east-west rotation. This is not yet all, next is the fact that programmer should calculate precise x, y, z values for the position and rotation of each module in order to get the right morphology. In the above example this process is simplified by means of introducing the constant (`UNIT`) and in a way abstracting the source code. However, still there is a need to calculate the end result. All these calculations are based on programmer's awareness of positive and negative directions of each coordinate in Cartesian coordinate system of USSR. Usually this process requires the programmer to be used working with coordinate system, furthermore ability to use them throughout writing the source code. Finally, we can agree that current way of implementing different morphologies of modular robots in USSR is at least not easy.

Consider now that another type of morphology of ATRON should be constructed. Let us assume the morphology consists of fourteen modules, this should take about ten minutes in best-case scenario and double the amount of code presented earlier. What if there is a need to construct the morphology consisting of one hundred modules, this should take about 1.2 hours and fourteen times more of the source code. Everyone should agree that the time spend to hard code the construction for each of above morphologies will not be used effectively even by expert (Ulrik Pagh Schultz). Moreover, this time will increase greatly for beginners, because they are not familiar with USSR specifics and require getting experience with build in examples. Furthermore, it will always take a lot of time to hardcode the construction of different morphology from the already existing ones. Another undesirable feature of above solution is the fact that it involves frequent use of so-called "loop development process", which is divided into two main parts: write the source code and test it. For construction of robot's morphology in simulator this is as follows: "write the source code – compile it – run the simulation – test if robot's morphology is meeting the requirements". This process repeats until desired morphology is reached. As a result, the simulation is restarted frequently. Moreover, the source code is prone to errors and the main feature of it is often "copy paste" (just imagine one hundred entries of module positions in Code Snippet 4.1 above). At this point we should definitely agree that the problem of construction of modular robot morphology is not manifesting itself significantly when the amount of modules is small, however with increase of them it becomes complex, tedious and repetitive task for expert and beginner to deal with. Furthermore, programmer is bound to Java as programming language, which makes it difficult for a less experienced user to work with USSR. Taking into account that already now experimentation with modular robot's morphologies requires easy and fast tool for constructing the morphologies of modular robots consisting of large number of modules (at least one hundred) current way of achieving this task in USSR is complex for both experts and beginners.

4.2.2 Assignment of Behaviors

Construction of a modular robot's morphology is only one side of the coin. The other side is assignment of behaviors. Here the problematic issue is that different modules in robot's morphology should behave differently, the same or both, depending on the choice of the user. Consequently, there should be a flexible way to experiment with implementation and reuse of behaviors for different modules. This also implies frequent re-assignment of behaviors, assignment of the same behavior only to a particular group of modules and so on. Lets us consider above example of ATRON car configuration in this context (see Figure 4.2, a).

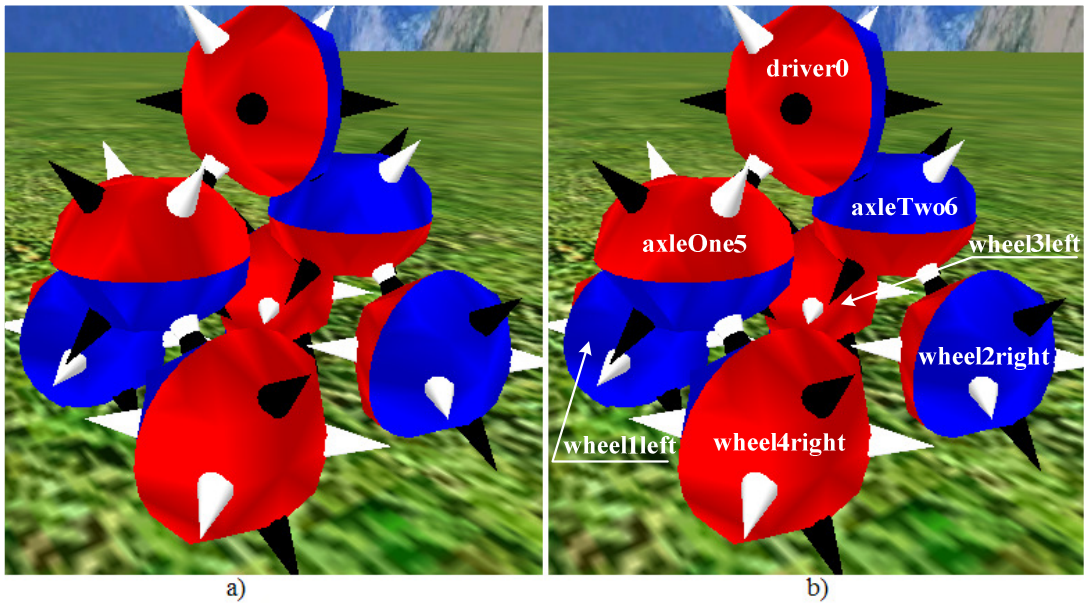


Figure 4.2 ATRON car configuration: a) before identification of the names for the modules and b) after.

At this point the user of a simulator is essentially facing a similar problem to construction of modular robot's morphology, meaning that the user by means of the source code should identify where in the morphology of a modular robot are situated modules such as wheels and then assign specific behavior to them. Next, indentify which wheel is "wheel1left", "wheel2right" and so on (see Figure 4.1 b). Here notice that the names of the modules are conforming to the Code Snippet 4.1 introduced earlier. After that, identify which module is for steering (able to rotate the front wheels to the left and right) and assign another type of behavior to them. In our case these are "axleOne5" and "axleTwo6". Usually there are only two ways to achieve identification of the modules and they are: 1) use global ID generated by simulator and 2) assign unique name for each module (in our case). Let us now consider the source code for ATRON car to drive forward in order to understand the problem more thoroughly.

Code Snippet 4.2 Example of ATRON car driving forward behavior using unique names

```
/* If the name of the module starts with "wheel 1-4" rotate one of the
module components continuously in appropriate direction  */
if(name.startsWith("wheel1")) rotateContinuous(dir); // dir-direction
if(name.startsWith("wheel2")) rotateContinuous(-dir);
if(name.startsWith("wheel3")) rotateContinuous(dir);
if(name.startsWith("wheel4")) rotateContinuous(-dir);
```

By means of the source code above, the user identifies the concrete module in the morphology of ATRON car and assigns to its wheel modules the behavior to rotate continuously. As a result ATRON car is driving forward in the simulation environment of USSR, see Figure 4.3. In this case it is obvious that before actually using the names of the modules the programmer should devise the unique name for each module in the morphology of the modular robot and after that refer to concrete modules and at the same time try to visualize the position of the module in the morphology. As a result, the current solution requires good knowledge of USSR specifics and keeping track of module names or ID. This problem is not manifesting itself significantly when the morphology of the modular robot consists of small number of modules, like in the example above. However, with a significant increase in the numbers of modules constituting the morphology of modular robot it becomes complicated and tedious to coordinate different implementations of behaviors for different modules, their names and position in the morphology of modular

robot, by means of the source code.

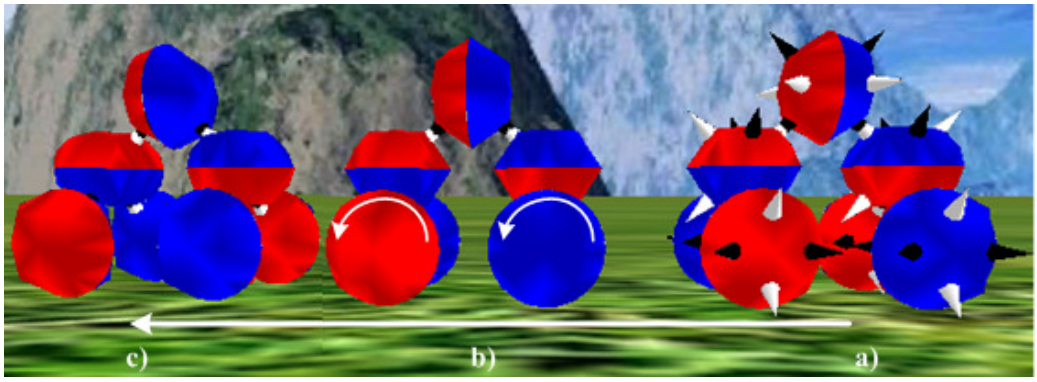


Figure 4.3 ATRON car driving forward: a) before running, b) in the middle and c) at the end of simulation. Here the white arrows indicate the path and the rotation of the modules.

Moreover, the same development process called "write the code – compile it – run the simulation – test if robot behaves according to the requirements" and "copy paste" of the source code are main undesirable features of such development.

We conclude that the two main obstacles in achieving fast and more abstract QPSS are: the slow and complicated construction of robot's morphology and assignment of behaviors. Consequently, these two directions are the main goals to improve on which are addressed here.

4.3 Analysis of Existing Approaches

Let us take the step back and try to investigate how the above problems were alleviated in the simulators for modular robots introduced in previous chapter. Probably the best examples worth considering are: Cube Interface and Webots 5. The main reason for that is the fact that both problems are solved on the higher level of programming and interaction in comparison to USSR and most of the other simulators discussed previously.

From construction of modular robot's morphology perspective in Cube Interface the first notable feature is interactive construction of modular robot morphology starting from the modules. Meaning, that there is no need for the user to pre-program or construct using a CAD interface the appearance of the module from the lower level components (spheres, boxes and so on). This is already done by default. As a result, the construction of modular robot morphology reminds somewhat higher level of CAD modeling from the basic objects being different Molecube modules. For achieving that, the user interacts with the GUI and simulation environment at the same time. Next feature is that construction of modular robot's morphology is restrained to the design limitations of the modular robot. For example, one Molecube module can connect to another module only on a six connection surfaces (each surface of the cube). In this way, the user is not able to exceed design limitations (construct wrong morphology) and in a way, is empirically learning the design limitations of the modular robot. Following is support with highly interactive, intuitive and integrated GUI. Moreover, the modules in simulation environment can be selected (chosen) by means of direct 3D object manipulation technique. As for behaviors, they are already pre-programmed for particular module and can be modified through GUI. For example, an actuator module with one DoF can be rotated around the rotation axis by means of one selection in the GUI. Last is the ability to save and load the morphology of modular robot into simulation from the file in a specific format. All these notable features enable the user to achieve high complexity of modular robot's morphology and assign close to desired behavior in a short amount of time. In comparison to USSR, it is notable that these features

would alleviate the problems with construction of modular robot's morphology and assignment of behaviors. The only negative feature to note is the fact that Cube Interface is modular robot specific and cannot be used to simulate other modular robots. At the same time, this feature was not initial design decision for simulator, so it is arguable if it can be considered as negative feature. It is more restricting requirement.

As for Webots 5 the construction of modular robot's morphology starts from CAD modeling of the module. This approach enables the user to model the module from low level geometrical components, like spheres, cylinders and so on. Later define this newly modeled module as definition and "copy paste" it in desired positions. The only problem is that the modules should be manually moved in those positions, which is tedious and imprecise for our case. On the other hand, in comparison to Cube Interface and USSR this feature gives more flexibility and abstract level of modeling to the user on the level of designing the module not by means of the code, but by means of interaction with GUI. Another disadvantage is that the user should be aware of the design limitations of a modular robot and control them him/herself during the construction of modular robot's morphology. Next, is the interactive GUI and direct 3D object manipulation technique (selection of objects). The only negative side of GUI in Webots 5 is the fact that there is a tree view displaying all construction components, which is expanding significantly with the increase in number of components, which makes it difficult to locate concrete component or the module in the morphology of modular robot. As for assignment of behaviors, the user is able to assign the behavior to each robot through a GUI (interactive assignment). To be more precise each robot can be assigned specific controller trough the tree view of GUI, which is similar to attachment. The last is saving and loading of robot's morphology from the file in VRML format, which promotes reusing and sharing of modular robot morphologies between the users. Finally, in order to organize the features discussed above in more compact way of representation, let us categorize them (see the table beneath).

Table 4.1 Summary of the features alleviating the problem under investigation

Feature	Category
Modeling on the level of components	Construction of Morphology
Modeling on the level of modules	
Accounting for design limitations	
Interactive GUI	Interaction
3D object manipulation technique	
Interactive assignment of behaviors	Behaviors (Controllers)
Saving and loading from the file	Sharing and Reuse

In the table above are summarized the features which are alleviating the problem with QPSS in simulators for the modular robots. Moreover, they are indicating that in order to achieve the higher level of programming during construction of modular robot's morphology and assignment of behaviors in USSR, there is a need to implement at least several of them in each category. The best case scenario is to support the USSR with all of them plus consider new innovative ideas.

4.4 Summary

During this chapter we investigated two problematic issues named as construction of morphology and assignment of behaviors in USSR. As a conclusion, we agreed that both have similar disadvantages, like: being complex in use on large numbers of modules in the morphology of modular robot, requiring good knowledge of USSR specifics, tedious, containing repetitive code and being inflexible for changes. Moreover, the usage of them is

often followed by frequent restarting of simulation. In attempt to learn from existing approaches which already solve these issues we analyzed the features of two simulators called Cube Interface and Webots 5 in comparison to USSR. Here we discovered that there are already solutions which can be reused in combination with new innovative ideas.

Chapter 5

5 Quick Prototyping of Simulation Scenarios

“Had he turned from politics to robotics, John Fitzgerald Kennedy might well have said, “Ask not what robotics can do for you, ask what you can do for robotics””.
— Michael Arbib

Chapter Objectives
1. Overview the main features of software implemented for USSR and called QPSS.
2. Emphasize the use of labels for reuse of controllers on similar morphologies of a modular robot.
3. Introduce the software architecture of QPSS and its place in the software architecture of USSR.

5.1 Introduction

The overall goal of this chapter is to present, from several perspectives, the software written for USSR. While keeping the discussion as general as possible with going into the details only for emphasizing the key features of it. The software was given the same name as a problem introduced in the previous chapter and it is QPSS (Quick Prototyping of Simulation Scenarios). For starters, we will overview the main features of QPSS with intention to get the whole scope of functionality. We will also identify the features adopted from other simulators and stress the innovative ideas. Next, with the help of specific example we will explore how the idea of labels [32], [77] can contribute to reuse of behaviors on similar morphologies of a modular robot. We will discuss it in details, because it is quite specific to the modular robots. Concluding, we will investigate how QPSS fits into the architecture of USSR, its major layers and modules of functionality. This should give the feeling of the source code on the high level and introduce the way it is structured.

5.2 Overview

At this point one would probably ask rhetorical question: “How is your approach different or innovative?”. Before actually answering this question, we will first need to get the feeling of overall solution to QPSS problem in USSR. The main reason for that is the fact that QPSS problem is influenced by a number of different features combined in coherent whole. Such as the ones discussed in previous chapter and new innovative ideas. Furthermore, they are intertwined together for achieving fast and easy QPSS. For concision, we will name the whole of these features as QPSS. The key features are the following:

- Interactive GUI and 3D simulation environment, which are responsible for interaction between the user, QPSS and USSR. Moreover, for supporting high level of programming during prototyping.
- Interactive construction of morphology, which is enabling the user to quickly and easily define the morphology of the modular robot, before the simulation is started.
- Extensible framework, which is an approach to implement QPSS as OO (Object-Oriented) framework supporting several modular robots and possibility to easier add the support for the new ones later on.

- Library of behaviors, which essentially is a number of pre-programmed behaviors for supported modular robots. A behavior can be interactively assigned to the module(s) in the morphology of a modular robot. The library supports addition of new behaviors, implemented by the user.
- Labels, which is a mechanism for identification of entities in the morphology of a modular robot, allowing reuse of behaviors on similar morphologies.

5.2.1 Interactive GUI and 3D Simulation Environment

First of all, the GUI of QPSS is designed so that it is based on the terminology from modular robotics field (connector, module, controller and so on), manipulation of entities in 3D environment, and GUI widgets common to wide range of applications (see Figure 5.1). In this way, the user is able to recognize the functionality by simply recalling the purpose of widgets from other applications.

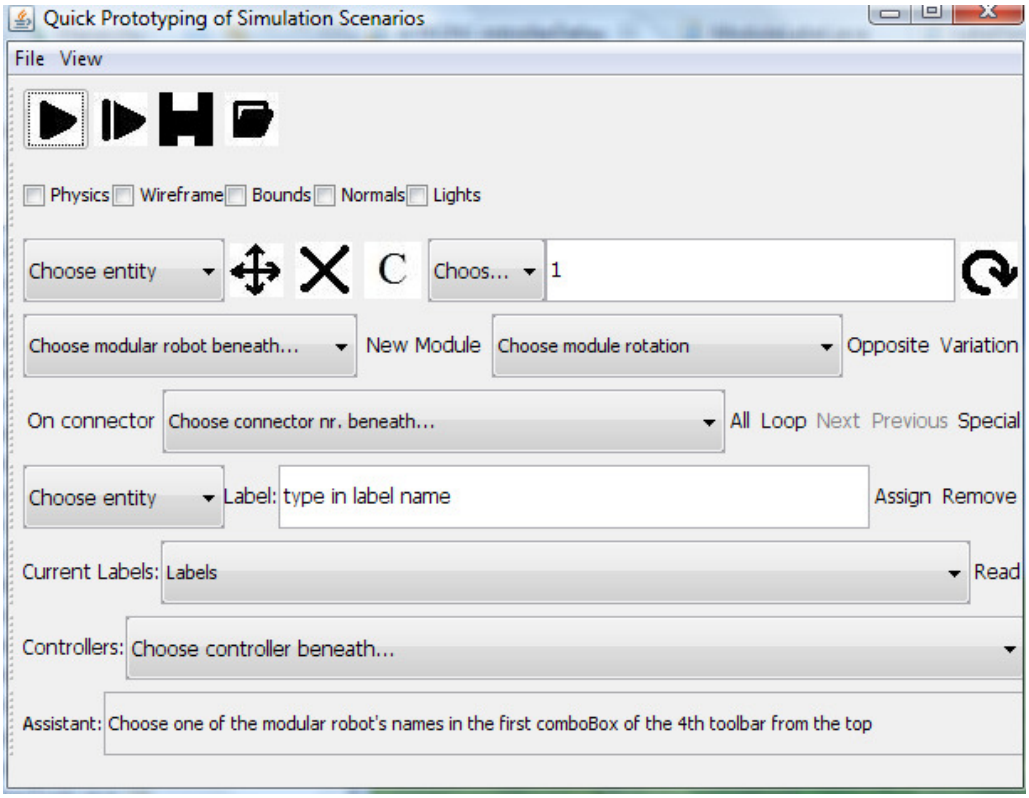


Figure 5.1 The GUI of QPSS

Moreover, the GUI includes the tools helping the user to learn its functionality and in this way reinforcing user's comprehension. Like for example toolbar named "Assistant" (last in the figure above), which gives hints to the user what to do next if one of the tools was chosen. Furthermore, the GUI is separate from simulation environment, which is a design decision based on requirement to do not bound to specific engines used by USSR. This ensures the portability of the source code, reuse and easier modification of GUI or replacement in the future. More detailed description of GUI can be found in Appendix C. As for simulation environment, the main input here is a 3D object manipulation technique, which is implemented as selection of entities (modules and connectors) with a mouse and moving entities (modules and components) from one place into another in 3D space.

5.2.2 Interactive Construction of Morphology

Interactive construction of morphology proceeds during static state of simulation. Meaning, that the user starts the simulation in paused state and by means of interaction with GUI and

3D simulation environment is able to construct the morphology of a modular robot. The process of modular robot's morphology construction is similar to CAD modeling on the level of modular robot's modules. Moreover, the design limitations of interconnection between the modules are accounted for. This means that basic construction elements are the modules of modular robots. This also implies that the module should be pre-programmed beforehand by programming the appearance of it from the lower level components like spheres, boxes and so on. During the construction of morphology the user is supported with a number of tools for interactive construction of modular robot's morphology. This gives to the user the possibility to choose the most suitable one for situation in question, preferable or familiar. At least three of these tools can be considered as innovative due to the fact that to the author's best knowledge they haven't been implemented in any other simulator for the modular robots before. Most of these tools are based on addition of modules in sequential fashion. Meaning that the user is selecting the connector or the module existing in the simulation environment and as a consequence new module is added next to existing one according to the design limitations. When the morphology of the modular robot is ready the user can save it in XML file, load it later on and modify it whenever it is necessary. After that the simulation can be started.

5.2.3 Extensible Framework

The design of QPPS is based on the OO (Object-Oriented) framework approach with three instantiations and possibility to support new ones. Meaning that, currently QPSS supports three modular robots (ATRON, M-Tran and Odin) and can be supported with others, like for example SuperBot, Catom, YaMoR mentioned previously and many others not yet mentioned. This implies continuous maintainability of QPSS. As a result, the QPSS is designed so that it is oriented towards reuse of the code (involving design patterns), abstraction of the code for all modular robots, continuous maintainability and support in the future. To be more precise the proportion of abstracted source code (generic) to modular robot specific source code is varying depending on the modular robot. However, in general it is observable that proportion is roughly fifty-fifty percent, taking as a measure the lines of code. The framework approach is also considered to be innovative, because during analysis of existing simulators for modular robots none of indications were discovered that the framework approach was used there.

5.2.4 Library of Behaviors

QPSS provides the user with a library of behaviors (controllers) which are pre-programmed for several modular robots. This enables the user to simply select the name of the behavior in the GUI and interactively assign it to the specific module(s) in the morphology of a modular robot. For instance, the user is able to interactively assign behaviors to ATRON car configuration in order for it to drive forward, which was discussed in previous chapter, see Figure 4.3. Moreover, the user is able to implement the behavior he/she is interested in to assign to the module(s) by using Java as programming language and specifics of USSR, later chose the name of this controller in the GUI and select the module in simulation environment to assign it to, by selecting it with the mouse click. The assignment of behaviors can be achieved both in paused and in running state of simulation. Furthermore, by means of assigning different behaviors to different modules in the morphology of a modular robot, overall behavior of the morphology can be observed and controlled interactively. As a consequence, the user is able to experiment with different implementations of behaviors on the different modules and combine the effect of each into complete behavior for a modular robot. This idea can be considered as partially

innovative, because similar case was discovered during the analysis in Webots 5 simulator. Here the behaviors are attached through GUI to the robots.

5.2.5 Labels

Labels are used for identification of the entities in the morphology of a modular robot and association of them to behaviors. By using labels the user assigns meaningful name to the specific part of the modular robot and later refers to this label during implementation of behavior. The main advantage of the labels is the fact that they can be used to abstract the morphologies and their behaviors. As a consequence, the labels can be seen as a link between the spatial configuration of a modular robot and implementation of behavior. As a result, by using labels it is feasible to implement behaviors, which are common to similar morphologies. In a way, it is reminding reuse of source code, however in case of modular robots it is reuse of behaviors (controllers). The idea of labels is also innovative and was inspired by [32], [77].

5.2.6 Comparison

Finally let us compare the features of QPSS (innovative and reused) in comparison to Cube Interface and Webots 5 for answering the question mentioned in the beginning of the section about innovations of QPSS. For that refer to the Table 5.1 beneath. Actually to the author's best knowledge this comparison can be considered to be expandable on all simulators for modular robot. Here the term "innovative" means that this feature was not used previously in the context of modular robots, "reused" means that it was inspired or adopted from previously mentioned simulators for modular robots (see Table 4.1 in previous chapter) and "-" indicates that the feature is not implemented for simulator in question.

Table 5.1 Comparison of features of QPSS for USSR to Cube Interface and Webots 5

<table>
<tr><th>Feature</th><th>QPSS for USSR</th><th>Cube Interface</th><th>Webots 5</th></tr>
<tr><td>Interactive GUI and 3D simulation environment</td><td colspan="3">Reused</td></tr>
<tr><td>Modeling on the level of components</td><td>-</td><td>-</td><td>Unique</td></tr>
<tr><td rowspan="2">Interactive construction of morphology (on the level of modules)</td><td rowspan="3" colspan="2">Reused</td><td>Reused</td></tr>
<tr><td>-</td></tr>
<tr><td>Saving and loading from the file</td><td>Reused</td></tr>
<tr><td>Extensible framework</td><td rowspan="3">Innovative</td><td>-</td><td>-</td></tr>
<tr><td>Library of behaviors</td><td>Pre-assigned</td><td>in GUI</td></tr>
<tr><td>Labels</td><td>-</td><td>-</td></tr>
</table>

From the table above it is obvious that uniqueness of current approach is extensible framework and the use of labels for identification of modules together with assignment of controllers. Moreover, the labels are used for reuse of controllers on similar morphologies of modular robots. Partially innovative can be also considered the idea of library of behaviors, because here the modules in the morphology of modular robot's morphology can be directly assigned the controller by means of selecting the module in the simulation environment. Similar case is supported in Webots 5, where the robot is assigned the controller by means of attaching it through GUI. Other features are adopted from similar simulators to ease the process of interaction with modules in simulation environment, promote sharing of morphologies between the users and in general enhance the process of QPSS.

After introducing the features of QPSS on the high level, let us emphasize the role of labels during implementation of behaviors for similar morphologies of a modular robot and at the same time get the feeling of QPSS usage. The main intention here is to present the labels more thoroughly, due to the fact that idea of labels is innovative and has a potential of being useful in the context of modular robots in the future.

5.3 Case Study: Reuse of Behaviors on Similar Morphologies

Typical workflow with QPSS can be divided into two major steps:

1. Interactive construction of modular robot's morphology;
2. Assignment of behaviors;
   2.1 Interactive assignment of behaviors using library of behaviors;
   2.2 Assignment of labels and implementation of controller using them;

The last step can be further split into two sub steps introduced above. Here each of sub steps can be considered as optional, because they are independent from each other. Assignment of behaviors can be also achieved by using only the support provided by USSR. Let us overview each of above steps on concrete example, from user perspective.

First of all, by means of interaction with GUI and 3D simulation environment, the user constructs three similar morphologies of ATRON car configuration, called: "simple vehicle", "four-wheeler" and "six-wheeler", see Figure 5.2. Later saves each of morphologies in XML file for future reuse. When desired the user loads the XML file describing one of morphologies, modifies it or runs simulation.

Second, the user experiments with interactive assignment of behaviors using library of behaviors on each of morphologies. In case of ATRON module these are different types of rotation of one hemisphere respectively to other one. For instance, the user chooses the behavior for continuous rotation in GUI and selects one of the modules in the "simple vehicle" morphology and as a result one of its hemispheres is rotating continuously. In this way, the user experiments with each morphology in order to familiarize with behaviors and their purpose. At the same time, observes how different behaviors influence overall behavior of modular robot's morphology.

Third, the user interactively assigns the labels to the modules and proximity sensors in each of morphologies. This is done in order to identify the modules and associate behavior of the module to its spatial position in the morphology of the modular robot. The labels are assigned semantically without visualization in 3D simulation environment. In the figure beneath, the labels were visualized to enhance exemplification. The labels are also saved in XML files.

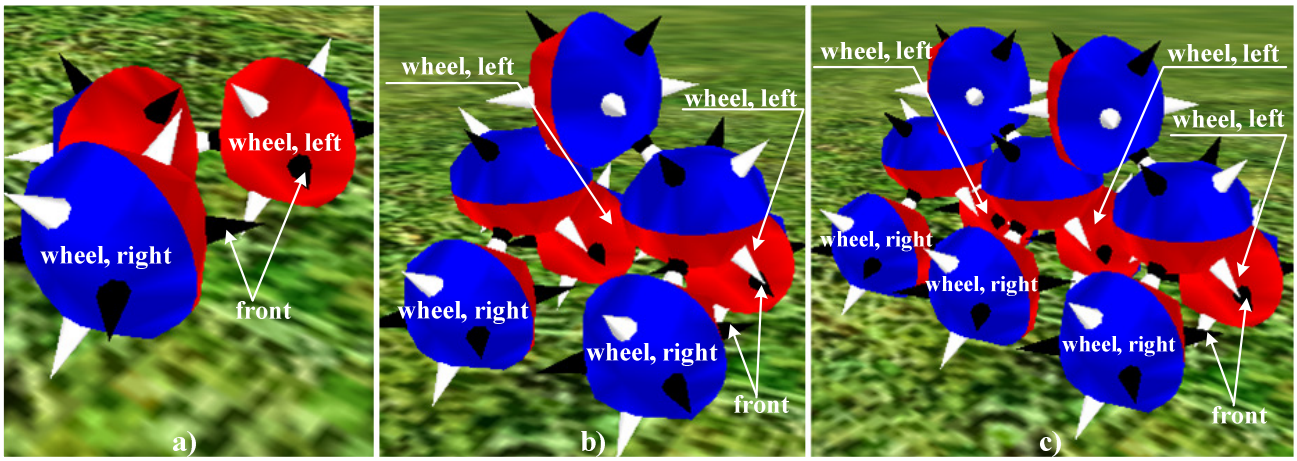


Figure 5.2 ATRON car morphologies with visualized labels: a) "simple vehicle", b) "four-wheeler" and c) "six-wheeler"

At this point, the user has three XML files, describing the three car morphologies, depicted in the figure above. Next, writes the source code of controller for “simple vehicle” to avoid the obstacles. Under avoiding is meant that the “simple vehicle” should drive forward to the box obstacles, then after detecting those with proximity sensors situated on the connectors of the front modules (in the figure the label “front”), turn to the left and continue on its way. After that, modifies the controller to support “four-wheeler” and “six-wheeler”, see Code Snippet 5.1 beneath.

Code Snippet 5.1 The ATRON controller for three different morphologies (“simple vehicle”, “four-wheeler” and “six-wheeler”) which are avoiding obstacles

```
/*Method activated for each module in separate thread*/
private void avoidObstacles(){
      /*Get labels of the module*/
      Labels moduleLabels = new ModuleLabel(module);
      while (true){
         /*rotate wheels slowly*/
         rotateWheels(moduleLabels, initialSpeed, 0);
         /*Get each proximity sensor on the module*/
         for(Sensor sensor: module.getSensors()) {
            /*Get labels of the sensor*/
            Labels sesorLabels = new SensorLabel(sensor);
            /*If sensor has label “front”*/
            if(sesorLabels.has("front")) {
               float sensorValue = sensor.readValue();
               /*if the modules are close to the obstacles*/
               if (sensorValue>distanceSensitivity) {
                  /*if it is left wheel module*/
                  if(moduleLabels.has("wheel") && moduleLabels.has("left"))
                     /*rotate in opposite direction and faster*/
                     rotateWheels(moduleLabels,-maximumSpeed,1000);
                  /*if it is right wheel module*/
               else if(moduleLabels.has("wheel")&& moduleLabels.has("right"))
                     /*rotate in the same direction and faster*/
                     rotateWheels(moduleLabels,maximumSpeed,1000);
}}}}}
/*rotates wheel modules continuously in appropriate direction */
private void rotateWheels (Labels labels, float dir, int time){
if(labels.has("wheel")&& labels.has("right")) rotateContinuous(dir);
if(labels.has("wheel")&& labels.has("left")) rotateContinuous(-dir);
/*needed for rotating in opposite directions*/
if(time>0) try { Thread.sleep(time); } catch(InterruptedException
            exn) { ; }
}
```

The implementation of controller above is based on running it in a separate thread for each module. First of all, the labels of the module are read in and according to them the wheel modules are indentified and initiated to rotate slowly. For that is used method called “rotateWheels”, which checks if current module is a wheel module and calls appropriate method for rotating it, depending if it left or right wheel. After that, the labels of each proximity sensor on the module are read in. If current proximity sensor has label named as “front”, then the proximity to the obstacles is checked. In case, the wheel module is too close to the obstacle, then left and right wheels are rotated in opposite directions and the speed of rotation is increased.

As a result of controller implementation presented above, all three morphologies of ATRON modular robot are able to exhibit simple behavior of avoiding obstacles by means of reusing the same controller. The implementation is based on identification of the front wheel modules in each of morphologies and when the obstacles detected, rotating them in

opposite directions. The only differences in controller implementation for different morphologies are the values of variables, like speed of rotation and distance to the obstacles. As an example, consider the behavior of ATRON "simple vehicle" configuration in the Figure 5.3 beneath.

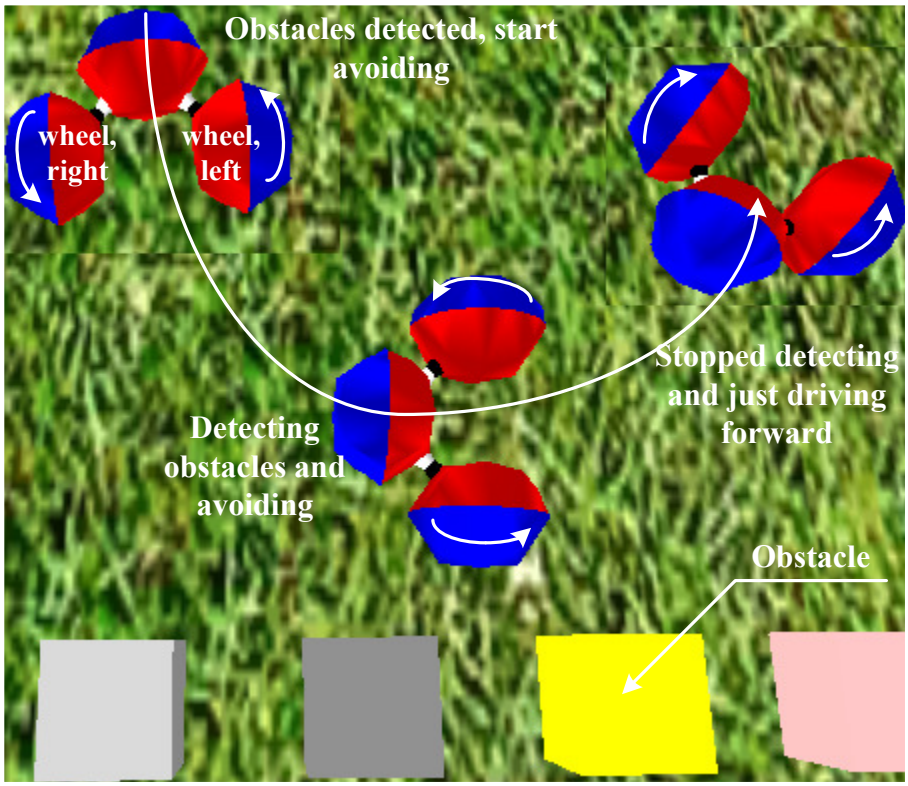


Figure 5.3 Example of ATRON "simple vehicle" configuration avoiding box obstacles

The figure above depicts ATRON "simple vehicle" configuration avoiding the box obstacles. Before the obstacles are detected, vehicle drives forward by means of rotating its wheel modules in the same direction (not shown). Nevertheless, when obstacles are detected, the rotation of front wheel modules changes so, that they speed up and one of the wheels ("wheel, left") changes the direction of rotation into opposite. As a result, the simple vehicle is turning to the left. It continues turning while detecting the obstacles. Finally, when the obstacles are no longer detected the vehicle is rotating its wheel modules in the same direction again and drives forward away from the obstacles.

5.4 Software Architecture

In general, the software architecture is the structure of the system, which comprises components and relationships between them. In order to imagine the place of QPSS in the architecture of USSR, let us begin our introduction with brief explanation of architecture of USSR and major decisions done during the design of it. The main goal during development of USSR is approach to implement it as generic OO framework for simulation of self-reconfigurable modular robots. Currently it is supporting ATRON, M-Tran and Odin. Consequently, the main design directions are the following: extendibility (maintainability), reuse of code, high level of abstraction, separation of concerns[34] and portability. Portability means that the code should be movable from current set of underlying engines onto others. As a final result, the current architecture of USSR consists of four layers upon which

[34] Separation of concerns is the process of breaking down the system into distinct features overlapping in functionality as little as possible. This includes information hiding, encapsulation and modularity.

concrete simulations are implemented, see Figure 5.7 the four layers from the bottom. Here the arrows indicate dependency and "..." indicate extendibility for new robot types. The "High" and "Low" indicate the level of dependency.

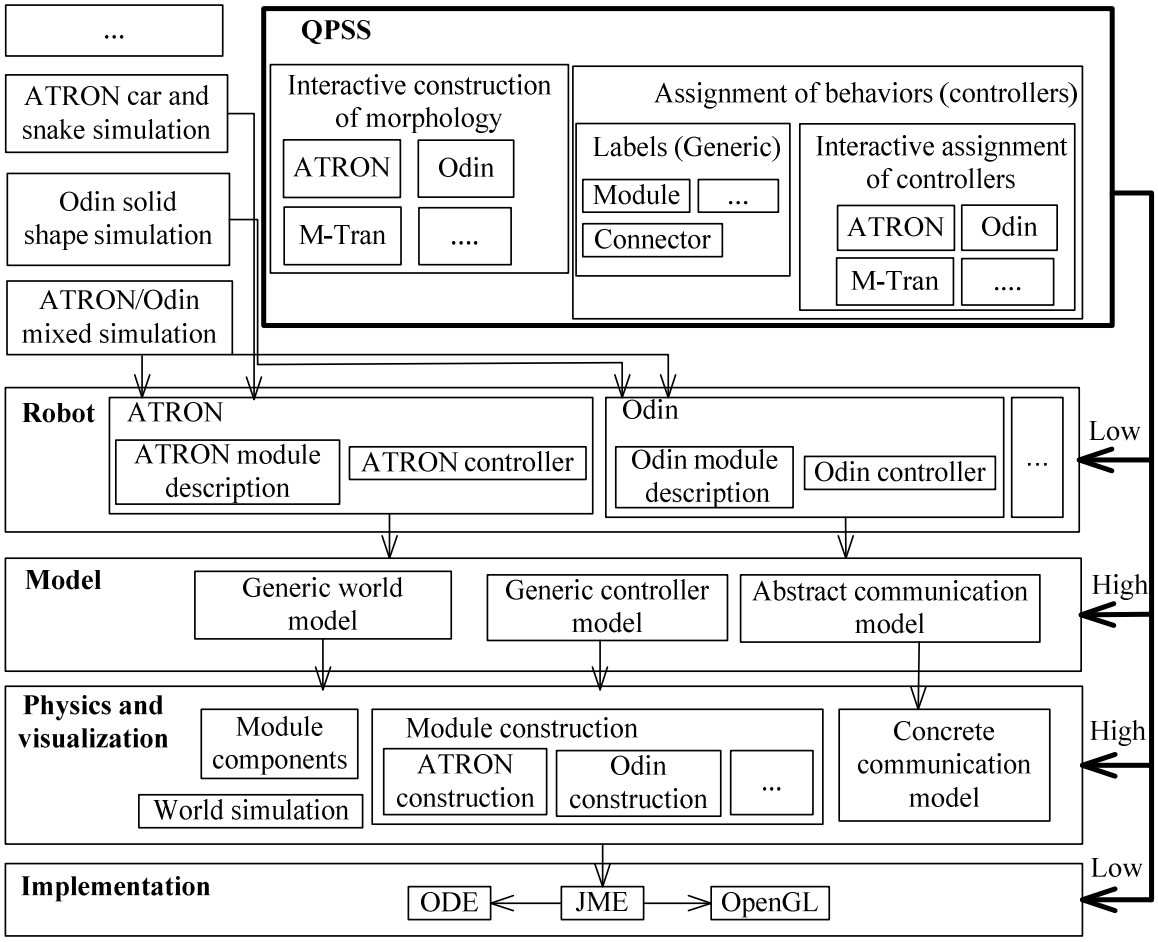


Figure 5.4 QPSS framework (**bold** lines) in the architecture of USSR, **bold** part was added and the rest was borrowed from [31]

In the figure above the first layer (from the bottom) is the "Implementation" layer, which is responsible for providing physics engine, visualization and programming environment. Here jME is the scenegraph independent of physics and graphics engines, however currently supports only ODE and OpenGL. The responsibility of the second layer is to provide simulation world including different entities, like: obstacles, ground, module components and concretization on communication model. The module components can be described as CAD models and low level shapes like spheres, cylinders and so on. On this level, the construction of the module from the components is programmed in Java, implemented as concrete construction classes for each modular robot. The third layer is an abstraction layer in order to provide physics and robot independence from underlying physics and visualization. Abstraction allows implementation of controllers to be independent from underlying engines and in the same fashion some parts of communication. Consequently, if the physics-based simulation engine is replaced with a transition-based, some of the generalized code can be reused. The fourth layer is dedicated to setting up and controlling simulations for a specific robot in a form of API. Some of the robot simulations are supported with more generic implementation of controller. Each controller is running in separate thread. On the top of four layers are implemented concrete simulations for supported modular robots [31].

A component named as QPSS is a result of this Master thesis. It is highlighted with bold and was implemented in terms of functionality from multiple lower layers. This was done in order to support higher level of programming and keeping in mind the extendibility of the framework. In general, QPSS depends on all four lower layers. The main disadvantage here is that is directly (not through interface) relying on the layer called "Implementation", which currently conflicts with the design decision of the code portability of USSR. However, the dependency is low and can be described as manipulation of data in terms of 3D environment. For example: matrixes, quaternion and so on. QPSS is highly dependent on the second and third layers (from the bottom). These two layers are necessity in order for QPSS to construct modules of different modular robots and manipulate their components on the higher level than underlying engines of USSR. QPSS is also dependent on the fourth layer called "Robot". However, most of this dependence is indirect or quite low. Like for example, dependence on controller of a modular robot in question, which is used during implementation of controllers for the feature library of behaviors discussed above.

It is probably obvious that the most of design decisions (mentioned previously) and overall idea of OO framework is also the main directions of QPSS. This is because it was in a way dictated by the design of USSR. So, it is observable that consistency was kept with the goals of USSR developers. The main responsibilities of the QPSS are: 1) interactive construction of modular robots morphologies and 2) assignment of behaviors (controllers). As for implementation the first was implemented so, that the abstraction of the code is enabling the user to reuse the functionality of it for enhancement of existing construction tools and ease the process of adding support for new modular robots. In short, the user is able to construct whatever morphology is desired by using current code for supported modular robots (ATRON, M-Tran and Odin). For assigning controllers there is a similar situation with abstraction of the code and extendibility. The only difference is the fact that assignment of labels is supported for modules and connectors in generic fashion. This means that there is no need to add new code in order to support new modular robots for these entities. It is required if there is a need to support other entities like: components, sensors and so on. The generality for labels was achieved by means of reusing the generic implementations of modules and connectors supported by USSR. There is also the support for interactive assignment of specific controllers (library of behaviors). Here the user implements the controller for the supported modular robot and later assigns it by means of interaction with GUI and simulation environment.

Finally let us emphasize separation of concerns in the design of the QPSS, which can be divided into modularity[35] and encapsulation[36], with the help of information hiding[37]. For that let us employ component diagram technique from UML standard [78], which is essentially introducing the software as composition of parts called components and their interconnection through interfaces or not. Each component can be a single class (in the figure beneath it is labeled as **C**), combination of classes according to the functionality they are responsible for (without **C**) and so on depending on the context. Combination of the classes is usually given general name according to functionality. Refer to the Figure 5.5

[35] Modularity is a design technique, the main goal of which is the software composed of parts called modules. Modules represent SoC and enforce logical boundaries between components. Typically modules are incorporated by means of using interfaces.

[36] Encapsulation is hiding of software component mechanism behind defined interface. The main purpose of it is to support potential change without affecting other components.

[37] Information hiding is a design principle based on hiding of design decisions which are most likely to change behind an interface. In this way other parts of the software are protected if design decision should be changed.

exemplifying the component diagram of QPSS for USSR. Minor annotations, which are not part of the UML standard, are used to communicate the topic more effectively.

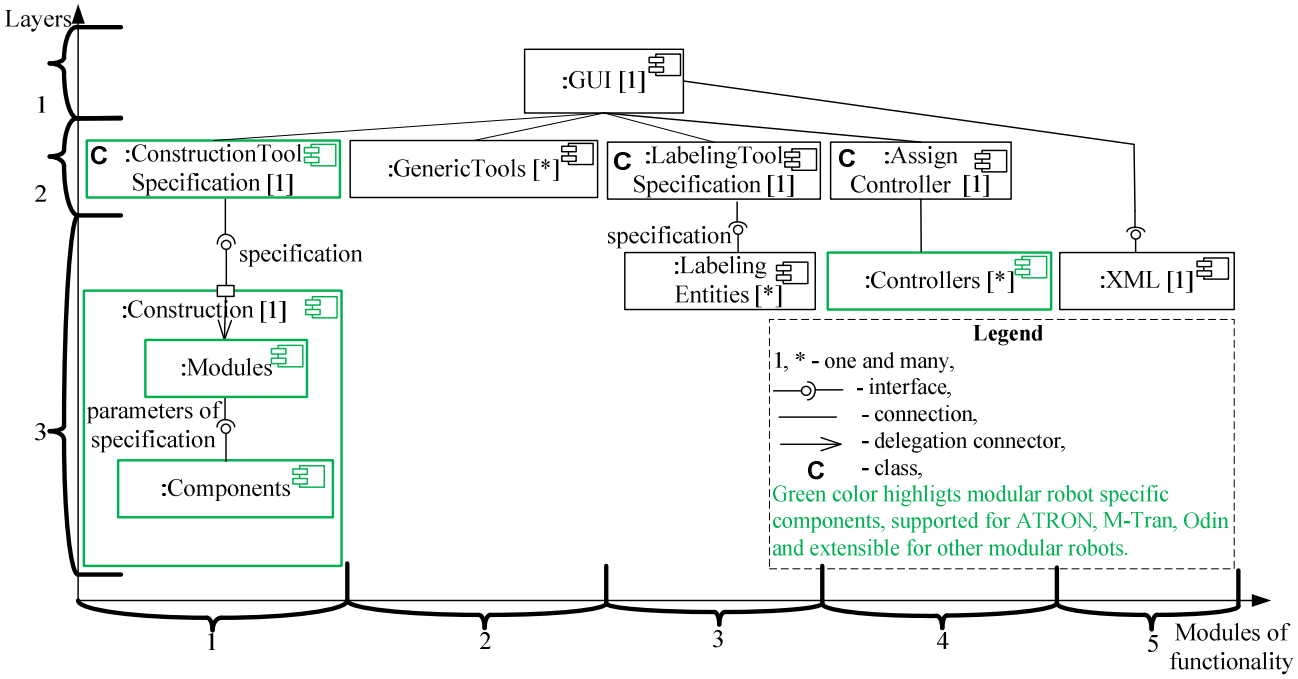


Figure 5.5 Component diagram of QPSS for USSR with minor annotations, which are not part of the UML standard

From the figure above it is notable that by responsibility the components of the QPSS can be divided into three major layers. The responsibility of the first layer (from the top) is GUI support. The second is responsible for combining GUI input and input from simulation environment. After that, it is calling appropriate functionality in the third layer. The main reasons why there is a direct connection between first and second layers (which is high coupling and not the best practice) are the following: a) it was decided to concentrate more on abstraction of the lower level functionality and b) during the project GUI and input from simulation environment were changing constantly and as a result it is time consuming to keep refactoring them at the same time with every change (meaning change of bigger architecture than existing one). The time was shifted towards supporting and enhancing essential third layer. The responsibility of the third layer is varying respectively to each module of functionality (modularity). In the figure above it is represented by five modules, which separate the functionality by enforcing logical boundaries according to their functionality. As a result, this should improve the maintainability of the framework. The first module (from the left) is modular robot specific, meaning that in order to support new modular robots this module will require maintainability. In general, it is responsible for creating new modules, processing them and rearranging the components of the modules during the process of interactive construction of morphology. It is separated into two subcomponents called “Modules” and “Components”. Both are equipped with interfaces for encapsulation. The main reasons for their separation were: 1) Separation of abstraction. The first handles functionality related to creating and moving modules. The second is responsible for handling the components of the modules created in the first; 2) To minimize the amount of the code included in each class and in this way ease the process of understanding the functionality supported by the classes. In general, this module is modular robot specific and requires extendibility in order to support new modular robots. The fact that there are two parts also contributes to easier maintainability. The second module is responsible for supporting with generic tools

enhancing the first module. Here generic means that these tools do not require additional implementation in order to use them on other modular robots. As a result, with additional support for other modular robots the minor changes or modifications should be expected here. The third module is responsible for labeling the entities in simulation environment. Here the component called "LabellingToolSpecification" is responsible for taking the inputs from GUI and simulation environment, after that through the interface calling concrete tool for labeling specific entity. As a result, encapsulation is also the main feature of design here. As for support, it is generic from the point of view that it is relying on the reuse of such abstract entities like modules and connectors supported by USSR. From the user perspective this means, it is possible to label modules and connectors of any modular robot currently supported by USSR and future ones. However, it is specific when there is a need to label other entities like for example components of the modules, sensors and so on. The main reason why there is no support for these now is the fact that there is no need for it at current state. At the same time, the design supports abstraction of the code, which should ease the future support of new entities. The responsibility of the fourth module is to support the user with interactive assignment of concrete controller to specific module in the morphology of the modular robot. It is relying on Java reflection package and is loading all controllers implemented by the user from the specified package into the GUI. These can be then directly called by means of combining inputs from the GUI and simulation environment. Currently there is no indication for a need of introducing an interface. Consequently, concrete controller is called directly from the "AssignController" component. Concrete controllers are specific to the modular robot they are implemented for. Fifth module is supporting the framework with ability to save the data about the morphology of a modular robot (including labels) in XML file and later load previously saved XML file. The functionality is accessed through the interface. Moreover, design enables the user to reuse the code for defining his/hers own format for XML generation and reading. For that there are a number of methods for reusability. The DCDs (Design Class Diagram) of all modules of functionality can be found in Appendix E.

Ideally the framework approach implies that the reuse of the implementation is based on black-box strategy. Here the programmer is reusing the implementation by means of relying purely on interfaces and specifications. So in this case there is no need to understand the implementation itself. However, the reality is that the majority of the frameworks support with mixture of black-box and white-box reuse. The last is when the programmer is reusing implementation through interfaces, specifications and by means of understanding the implementation itself [79-81]. QPSS is not an exception to this empirical observation. Meaning that, the programmer can reuse the functionality supported by QPSS in most places by means of black-box reuse. However, in some cases it is required to use white-box and especially when support for a new modular robot is implemented.

In conclusion, it is obvious that the QPSS is following the major OO design principles. However, as it is usually the case for the prototypes, future improvements should be expected.

5.5 Summary

In this chapter we discussed the software called QPSS (implemented for USSR) in the form of an overview, in order get the overall feeling of its functionality. We started with the presentation of its features and emphasized the innovative ones. These are the framework approach of implementation, interactive assignment of behaviors and the use of labels during assignment of behaviors. Special attention was dedicated to discuss the role of labels for reusing of controllers on similar morphologies of modular robot, due to the fact that this mechanism has an interesting potential and is quite specific to the modular

robots. Finally we presented the architecture of QPSS and concluded that the design of QPSS follows main OO best practices and conforms to the design decisions of USSR developers. However, we also agreed that QPSS is a prototype and as a consequence, there are still shortcomings, which should be improved on in the future.

Chapter 6

6 Interactive Prototyping of Modular Robots

"If the only tool you have is a hammer, you tend to see every problem as a nail."
— Abraham Maslow

Chapter Objectives

1. In details introduce and evaluate the features of QPSS for interactive construction of morphology, assignment of behaviors using library of behaviors and labels, during prototyping of modular robots.
2. Convince the reader that QPSS is supporting USSR with higher level of programming, quick and easy prototyping.

6.1 Introduction

In this chapter we will discuss the main features (tools) of QPSS from the user perspective. Moreover, we will address the features for interactive construction of modular robot's morphology, interactive assignment of behaviors using library of behaviors and the use of labels during assignment of behaviors. Furthermore, we will reason about their advantages and disadvantages in the context of the modular robots. The main intent of the chapter is to convince us that the QPSS is able to deliver on promise of supporting the user with high level of programming, quick and easy prototyping of simulation scenarios.

6.2 Interactive Construction of Morphology

For interactive construction of modular robot's morphology QPSS suggests to the user a set of tools which can be split into two main categories called generic and modular robot specific. Generic means that there is no need to add additional implementation in order to use them on the other modular robots than currently supported. Specific on the other hand require additional implementation, which is the maintainability of the framework. Refer to the Table 6.1 for complete set of features.

Table 6.1 Summary of features (tools) for interactive construction of morphology

Feature	Category	Role	State of simulation	Usability
Move module	Generic	Primary	Static and Dynamic	Active
Move component		Supplementary	Static	Passive
Delete module		Primary	Static and Dynamic	Active
Color connectors		Supplementary	Static	Active
Rotate module		Supplementary	Static	Passive
Rotate component		Supplementary	Static	Passive
Add default module	Specific	Primary	Before simulation	Active
Rotate module specifically		Supplementary		
Rotate module opposite				
Variation of modules				
On selected connector		Primary		
On chosen connector				
On all connectors				
Loop				

In the table above are presented all features describing the functionality of the QPSS for interactive construction of modular robot's morphology. Most of them fall into the category of being specific. This is also in a way an indicator that the framework approach is dominant. The features (tools) can be also divided into primary and supplementary according to their role during the process of construction. Primary means that these tools are the main tools and supplementary means that they are enhancing the functionality during the construction in combination with primary. By combining the usage of both the user is able to explore the flexibility of functionality. Most of the tools are designed for use in static state of simulation, meaning that the user is using the tools during paused state of simulation for construction of robot's morphology. Moreover, the user is able to start simulation, later pause it and continue using these tools again. The special case is the tools falling into the category named as "before simulation". These tools can only be used in static state of simulation and cannot be used when the simulation was started and later paused. For instance, if the user starts the simulation in static state and builds the ATRON "simple vehicle" configuration by means of tools from the category of "before simulation". Next starts simulation to observe the behavior and decides to change the morphology into ATRON car "four-wheeler". Then he/she stops (pauses) the simulation. At this point the user will not be able to use the tools from category of "before simulation". However, similar functionality can be achieved by means of saving the ATRON "simple vehicle" morphology in XML file before starting simulation, later restarting the simulation and loading the "simple vehicle" morphology from XML file, modifying current morphology into ATRON car "four-wheeler" and starting the simulation with new morphology. The last column of the table indicates if the tool is already actively used (active) or the tool is implemented for future tools (future ideas). Most of the features are assigned quite self-explanatory names. However, there are some with quite unique names, which require an example in order to get the feeling for their purpose and functionality. Taking this into account let us discuss these tools in context of constructing modular robot's morphology. Coming examples should also answer the rhetorical question: "Does the QPSS really enables the user quickly and easily construct modular robot's morphology?".

For each of three modular robots (ATRON, M-Tran and Odin) there are four major tools to interactively construct the morphology of modular robots and in the table above they are named as follows:

1. On selected connector, where the main emphasis is on selection of connectors on the modules. As a result, new module is added respectively to selected module and connector;
2. On chosen connector. Here the user chooses the connector number in GUI and later selects the module in simulation environment to which new module should be added;
3. On all connectors. This tool reminds the work of sculptor. Meaning, that user first selects the module in simulation environment and as a consequence new modules are added to all connectors of it. After that, the user is removing the modules which are excessive;
4. Loop. This tool requires from the user to select the module in the simulation environment and later move the newly added module from one connector on another by means of interaction with GUI elements.

It is necessary to stress, that all above tools fall into the category of "before simulation", which limits the user to construct desired morphology of a modular robot only before starting the simulation and does not allow modifying it during running simulation and pausing it later. Moreover, each of them has its advantages and disadvantages, which we

will discuss next and at the same time, exemplify the usage of each tool. For more detailed description of other features (tools) refer to Appendix C.

6.2.1 On Selected Connector

This tool is based on user selecting the connector on the module in simulation environment and as a result new module is added to this connector (see Figure 6.1 for exemplification). The process of new module addition is emphasized with the white arrow. The arrow tip with a number indicates the sequence of selection. For concision only part of GUI and simulation environment are shown.

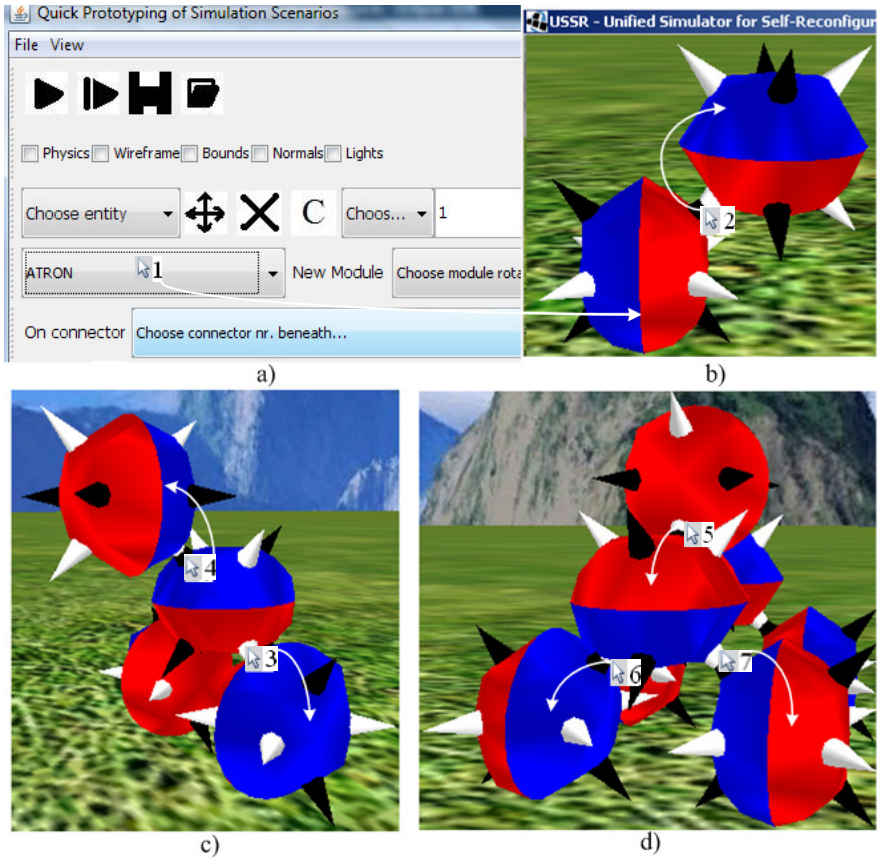


Figure 6.1 The usage of "On selected connector" construction tool on ATRON

The figure above exemplifies the usage of "On selected connector" construction tool during interactive construction of ATRON car "four-wheeler" configuration. At the beginning user starts the empty simulation and GUI of the QPSS, here user selects the type of the modular robot he/she would like to work with (in the combo box), which adds so-called default construction module (see Figure 6.1 a, b, and selections 1). After that selects desirable connector on previously added default construction module. This places the new ATRON module with respect to selected connector (black and white cones). Sequent selections of connectors on different modules result in ATRON car configuration (see Figure 6.1 b, c, d and selections 2-7). The process of ATRON car construction takes seven mouse clicks and less than fifteen seconds for the expert user (author of this report). Comparing the time to construct the same configuration by Ulrik Pagh Schultz using hard coding which is five minutes, using QPSS expert user can build at least twenty ATRON car configuration robots during the same five minutes. The advantage of the first solution over the hard coding is obvious.

There are three main advantages of this solution. First is decreased time to construct the morphology of a modular robot in comparison to hard coding. Second, there is no need for

the user to be aware which number of connector is selected, the only thing needed is some kind of mental model of robot's morphology to follow, when selecting the connectors. Third, the user does not need to be aware of how modules can be connected and how they cannot. In other words, there is no necessity to be aware of modular robot's design limitations. Consequently, the first solution can be also used for educational purposes because it is completely intuitive and can help to newcomers of the field to learn the limitations of connection and orientation of ATRON and other modular robot's modules (design decisions).

As usual, there are also several disadvantages to consider. First, it becomes quite difficult to construct robot's morphology in case when the connectors on the modules are of small dimensions, because the user should often zoom in closer to connector in order to successfully select it. For instance, in case of M-Tran and Odin simulations, interactive construction of morphology involves often zoom in and zoom out, which is uncomfortable. Second, this kind of solution becomes tedious in case when robot's morphology consists of large numbers of modules (in range of hundreds). Let say, that will be one hundred, then the user should select more than one hundred connectors and be aware of the result. For this case, solution that is more effective can be found.

Finally, we can agree that the first solution is effective if robot's morphology to construct consists of large numbers of modules (around one hundred) and connectors on the modules are big enough to select them successfully. As for user group, it is targeting the wide range of users from experts to beginners, because construction involves knowledge of such abstract terms like connectors and modules.

6.2.2 On Chosen Connector

The second tool is based on the user selecting the number of connector in GUI to which new module should be placed to, see Figure 6.2 for exemplification. Here the same notation is used as in the previous figure. For concision only part of GUI and simulation environment are shown.

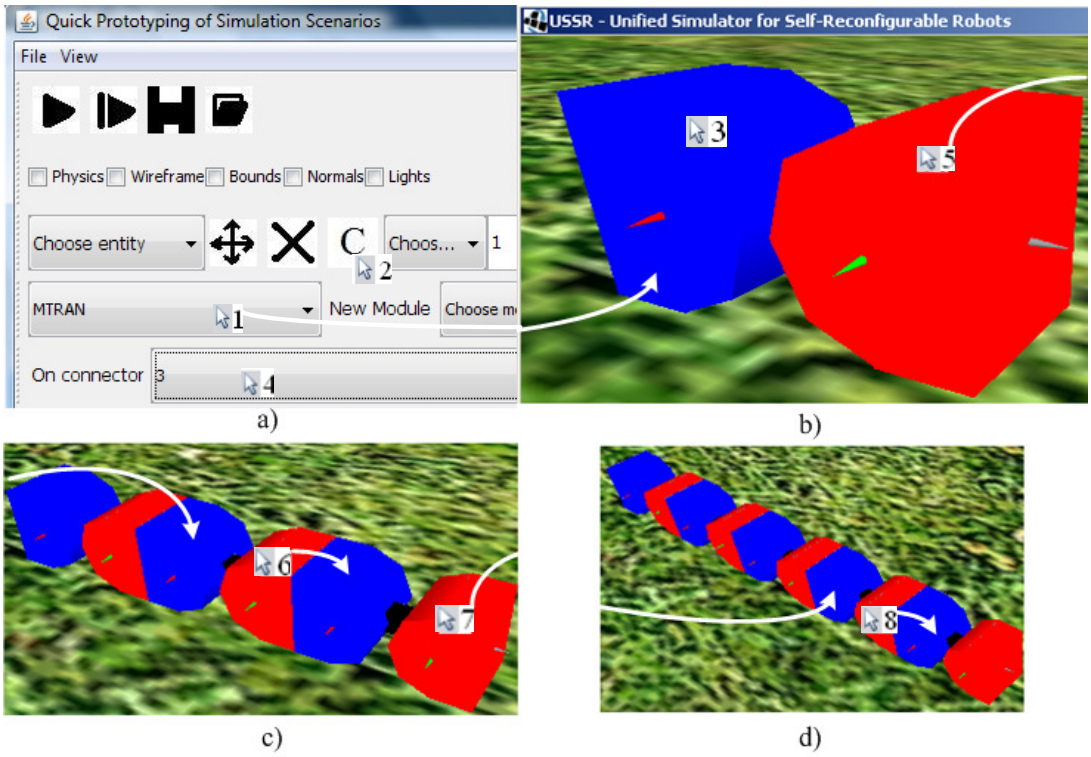


Figure 6.2 The usage of "On chosen connector" construction tool on MTRAN

The figure above exemplifies the usage of the second solution on M-Tran modules. At the beginning user starts the empty simulation and GUI of the QPSS, next the user selects the type of the modular robot he/she would like to work with (in the combo box), which adds so-called default construction module of MTRAN (see Figure 6.2 a, b, and selection 1). After that user selects helping tool for identification of connector numbers ("C"– Color connectors) and the module in simulation environment to apply it to (see Figure 6.2 a, b and selections 2, 3). As a result, the connectors of selected module change color and now it is possible to identify the number of connector which is associated with the color. As a result, the connectors of selected module change color and now it is possible to identify the number of connector which is associated with the color. In example above, grey color is number 3, green – 4 and red – 1. Next user selects the number of the connector (in the combo box) to add new module to (3 in this case) and after that selects the modules in simulation environment to apply it to. This adds new module in the front of the selected module on chosen connector with each selection of the module. In this fashion four modules are selected one by one and the final result is M-Tran snake configuration consisting of five modules (see Figure 6.2 a-d and selections 4-8).

The main advantage of the current solution is ability for expert user to place the module exactly where it is desired. Meaning that user should keep in mind the connector positioning, orientation and the number on the module. Moreover, this solution is quite useful when there is a need to add modules repetitively on the same connector, like in above example during construction of a snake configuration. From the beginner's perspective, this solution will give possibility to learn where the connectors are positioned and their numbering convention. Consequently, in this way familiarize the user with the design decisions for the robot. Next advantage is that there is no need for the user to select small connectors, which is problematic in previous solution. As a result, this solution is most useful when the numbers of connectors are important during interconnection of modules or when replicating precise robot morphology from the physical robot.

The major disadvantage of current solution when comparing it to the previous solution is the fact that it will take more time to construct the same robot's morphology with the current solution, because there are more transactions between interaction in GUI and simulation environment. Moreover, there is a need for a new connector numbering technique. Now these are the colors (color coding), but in the future the labeling (3D Text) of connectors in simulation environment is more attractive solution, because association of the color to the number of connector is not an optimal solution. Finally, current solution becomes tedious when there is a need to construct robot's morphology consisting of large numbers of modules. It is more suitable for morphologies consisting of moderate numbers of modules in the range of fifty.

In conclusion, second solution is best suitable for constructing the robot's morphology consisting of moderate number of modules and is flexible enough for both user groups from experts to beginners.

6.2.3 On All Connectors

The third tool relies on the user selecting desired module in simulation environment and as a result, new modules are placed to all connectors of selected module. Later, user is able to delete (remove) the excessive modules, see Figure 6.3 for exemplification. In this way, it is reminding sculptor's work process where excessive amount of stone is removed in order to cut out the sculpture. Here the same notation is used as in previous figure. For concision only part of the GUI and simulation environment are shown. Moreover, the steps for starting the empty simulation, GUI and identification of the modular robot to work with are skipped due to repeating nature of the functionality.

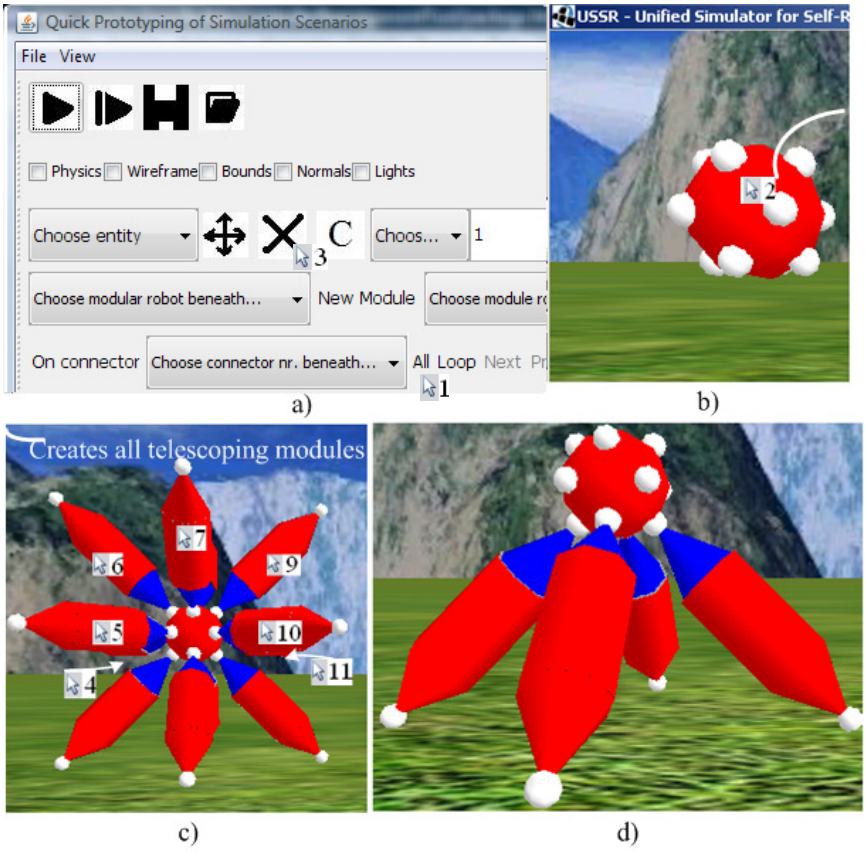


Figure 6.3 The usage of "On all connectors" construction tool on Odin

The figure above exemplifies the usage of third solution on Odin modules. First, the user chooses appropriate button for using third solution in GUI ("All") and selects the joint module of Odin in simulation environment of USSR (see Figure 6.3 a, b and selections 1, 2). As a result, twelve modules called telescoping links are added to the joint module. The reason why there are exactly twelve telescoping links is, because there are twelve connectors (white spheres) on joint module (red). Next, the user chooses to delete (remove) eight telescoping modules from simulation environment, see Figure 6.3 a, c, d and selections 3-11. Finally, selected telescoping modules are removed and the final morphology reminds upper part of a pyramid, see Figure 6.3 d.

The main advantages of third solution are ability to construct robot's morphology consisting of large numbers (hundreds) of modules within relatively short amount of time, the fact that user selects modules instead of small connectors and there is no need to keep in mind the numbers of connectors. Consequently, third solution should be useful for all targeted groups of the users. The disadvantage is excessive modules, which should be removed by means of additional tool. In case of constructing extra large (in scale of thousands) robot's morphologies this process can become tedious. That is why more optimal solution can be considered for robot morphologies consisting of thousands of modules.

6.2.4 Loop

The fourth solution (called "Loop from one connector to another") relies on user selecting the module in simulation environment and then looping through all possible connection options of newly added module. See Figure 6.4 for example. This solution was inspired by Cube Interface simulator, which is implemented for Molecubes modular robot. Here the same notation is used as in previous figure. For concision only part of the GUI

and simulation environment are shown. Moreover, the steps for starting the empty simulation, GUI and identification of the modular robot to work with are skipped due to repeating nature of functionality.

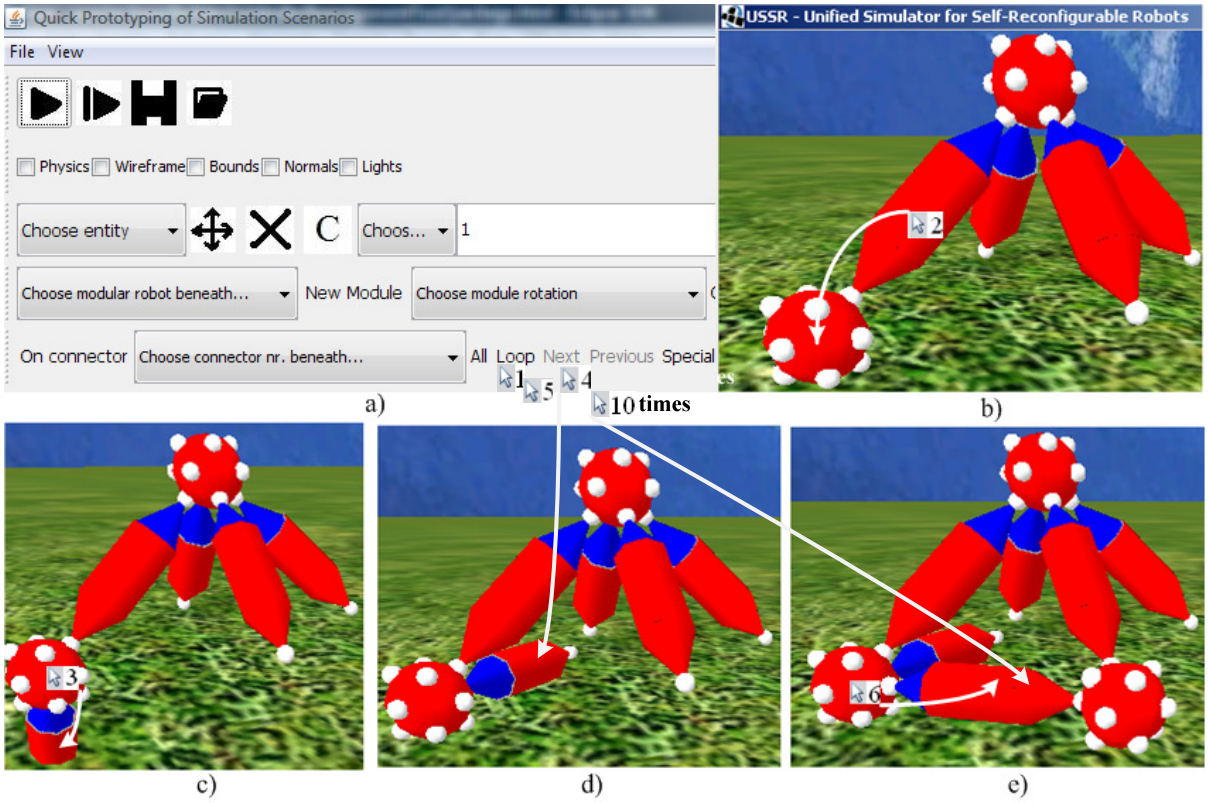


Figure 6.4 The usage of "Loop" construction tool on Odin

The figure above exemplifies the usage of fourth solution on Odin modules. Here we begin from the previous example, where we have built upper part of a pyramid. First of all, the user chooses appropriate button for using fourth solution in GUI ("Loop") and selects the telescoping link of Odin in simulation environment of USSR (see Figure 6.4 a, b and selections 1, 2). As a result, the Odin joint is added to default connector on the telescoping link. After that, the user selects Odin joint which results into addition of new telescoping link to the joint module (see Figure 6.4 a, c and selection 3). Next, user moves the telescoping link from the default connector to next ones by clicking "Next" button for increasing and "Previous" for decreasing number of connector. This moves the telescoping link from one connector on the joint module to another (see Figure 6.4 a, and selection 4). When desired position is achieved the user selects the same tool once again and selects the joint one more time. This adds new telescoping module, which after that is moved in desired position by pressing ten times "Next" (see Figure 6.4 a, e and selections 5, 6-10).

The advantages of fourth solution are the following: the fact that user selects modules instead of small connectors; there is no need to keep in mind the connector numbers and ability for the user to familiarize with possible connections of the modules and their orientations. The current solution targets robot morphologies consisting of moderate numbers of modules. Main disadvantage is that it becomes tedious to loop through all possible connections of the modules when there are many connectors on the module. Moreover, this process takes too much time comparing this solution to all previous.

In conclusion, we note that fourth solution is best suitable for construction of robot morphologies consisting of moderate numbers of modules. Furthermore, it is flexible enough to cover the needs of both expert and beginner user groups.

6.2.5 Evaluation

Finally let us summarize all discussed above in compact way of representation, see Table 6.2. Furthermore, in general evaluate the scope of the tools for interactive construction of modular robot's morphology.

Table 6.2 Construction tools for morphologies of modular robots and their properties

Tool	Supported MR	Numbers of modules	Targets user groups
On selected connector	ATRON, M-Tran, and Odin.	large (100)	from experts to beginners
On chosen connector		moderate (50)	
On all connectors		large (100)	
Loop		moderate (50)	

The table above indicates that all four construction tools are fully implemented for ATRON, M-Tran and Odin modular robots (MR). Looking from the perspective of numbers of modules constituting the robot's morphology it is notable that QPSS functionality covers the range of modules from moderate to large. However, new more optimal solution should be considered for extra large numbers of modules (in range of thousands). This is because current solutions become tedious when using them on tasks to construct robot morphologies consisting of thousands of modules. The main advantage of the current solutions is ability to quickly and interactively construct the robot's morphology consisting of a maximum one hundred modules by means of switching from one solution to another when it is most comfortable for a user. As for targeted user groups, the functionality of the QPSS covers the range from experts to beginners. Moreover, the user is presented with the choice from four available tools, which gives the freedom of choosing the most suitable tool for construction process. Consequently, the QPSS should attract more users and save valuable time to expert users.

6.3 Assignment of Behaviors

As it was previously discussed QPSS provides two approaches to assign behaviors to modular robots and they are the following: a) interactive assignment of behaviors using library of behaviors and b) using labels for identification of modules in the morphology of a modular robot, see Table 6.3. For both QPSS suggests to the user a set of features which can be split into two main categories called generic and modular robot specific. Here generic means that there is no need to add additional implementation in order to use them on the other modular robots than currently supported. At the same time, this also means that tools can be used on the modules and connectors of the modules of any modular robot. However, if there is a need to use them on other entities of the module (for example components, sensors and so on) additional maintainability for the QPSS is required. Specific means that there is a need for additional implementation in order to use the tool for other modular robot than supported ones. Here this essentially means that the tool called "Assign Controller" requires implementation of concrete controllers, but the process of assignment is generic.

Table 6.3 Summary of features (tools) for assignment of behaviors

Approach	Feature	Category	State of simulation
Library of behaviors	Assign controller	Specific	Dynamic and Static
Labels	Assign label	Generic	
	Remove label		
	Read labels		

In the table above are presented all features describing the functionality of the QPSS for assignment of behaviors. They are primary and active, meaning that they can be used without any restrictions. Moreover, they can be used in both states of simulation. Most of the features are assigned quite self-explanatory names. However, there are some with quite unique names, which require an example in order to get the feeling for their purpose and functionality. Taking this into account let us discuss these tools in context of assignment of behaviors. Coming examples should also answer the rhetorical question: "Does the QPSS really enables the user quickly and easily assign behaviors to a modular robot?"

There are two key features which will explore in detail and they are the following:

- Assign controller (from the library of behaviors approach), which was developed for interactive assignment of controllers to the modules in the morphology of a modular robot;
- Assign label (from labels approach), where the labels are used to identify the module(s) in the morphology of a modular robot.

Both above tools are major for their approach. Each of these tools has its advantages and disadvantages which we will discuss next and at the same time, exemplify the usage of each tool. For more detailed description of other features (tools) refer to the Appendix C.

6.3.1 Assign Controller

Next to introduce is the tool named as "Assign Controller". In general it is the tool for interactive assignment of behaviors to specific modules. See the Figure 6.5 for example. Here the white arrows indicate extension and contraction of OdinMuscle modules (telescoping links). Not all extension, contraction and selection arrows are shown to keep the figure as simple as possible.

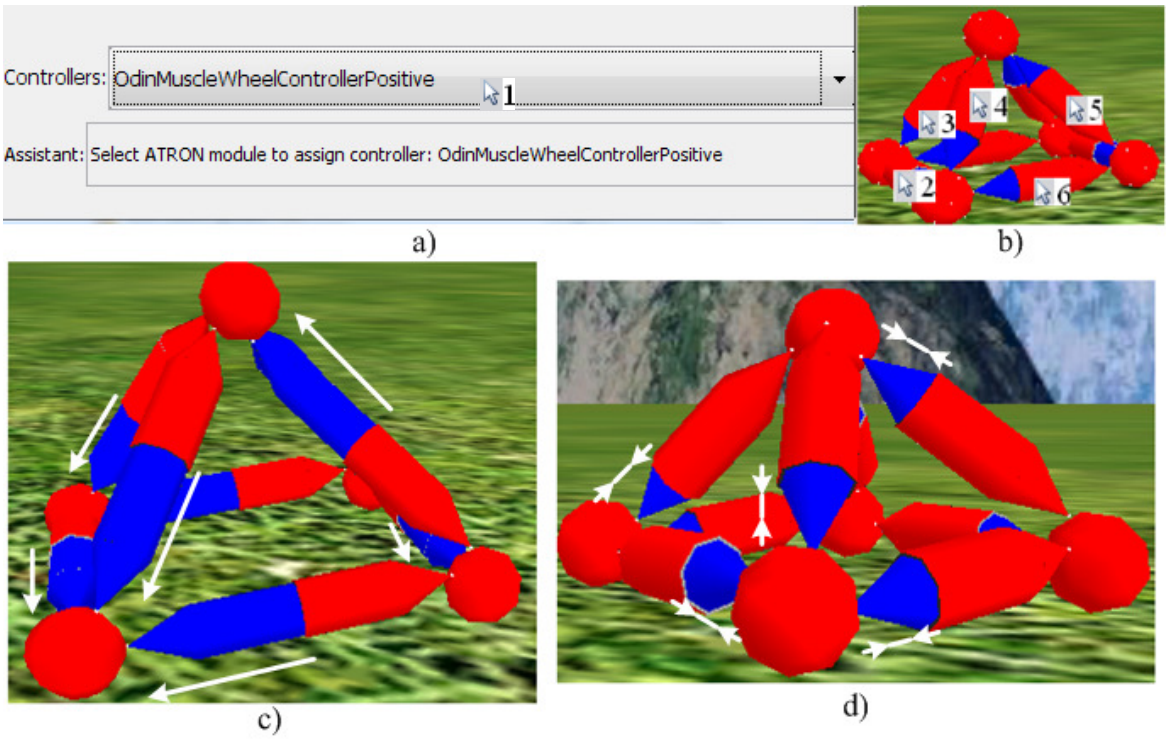


Figure 6.5 The usage of "Assign Controller" tool on Odin pyramid configuration

In the figure above is exemplified the usage of "Assign Controller" tool on Odin pyramid configuration. First, the user constructs the morphology of Odin pyramid by means of using construction tools introduced in previous section or loads it from XML file and starts the simulation. Second, chooses the controller he/she would like to assign and selects the

modules in simulation environment to apply it to (all telescoping links). In this case controller is "OdinMuscleWheelControllerPositive", which extends Odin telescoping links to their maximum length. As a result the magnitude of the Odin pyramid significantly increases (see Figure 6.5 a, b, c and selections 1-6). After that, the user chooses another controller for contraction of Odin telescoping links and selects each of telescoping links (not shown in the figure). Final result is initial pyramid configuration. All controllers are automatically loaded from the specific package by means of Java reflection into comboBox of GUI when it starts. This implies that the user implements the controller before starting the GUI. The main advantages of the current tool are: 1) the possibility for the user interactively assign concrete implementation of behavior to specific module(s) in the morphology of modular robot; 2) quickly and easily experiment with different versions of behaviors and 3) ability to combine the behaviors in more complex behavior of modular robot morphology by assigning different controllers to different modules.

6.3.2 Assign Label

This tool is based on user defining the label(s) in the GUI and later selecting the module or connector to assign it to, see Figure 6.6 for exemplification.

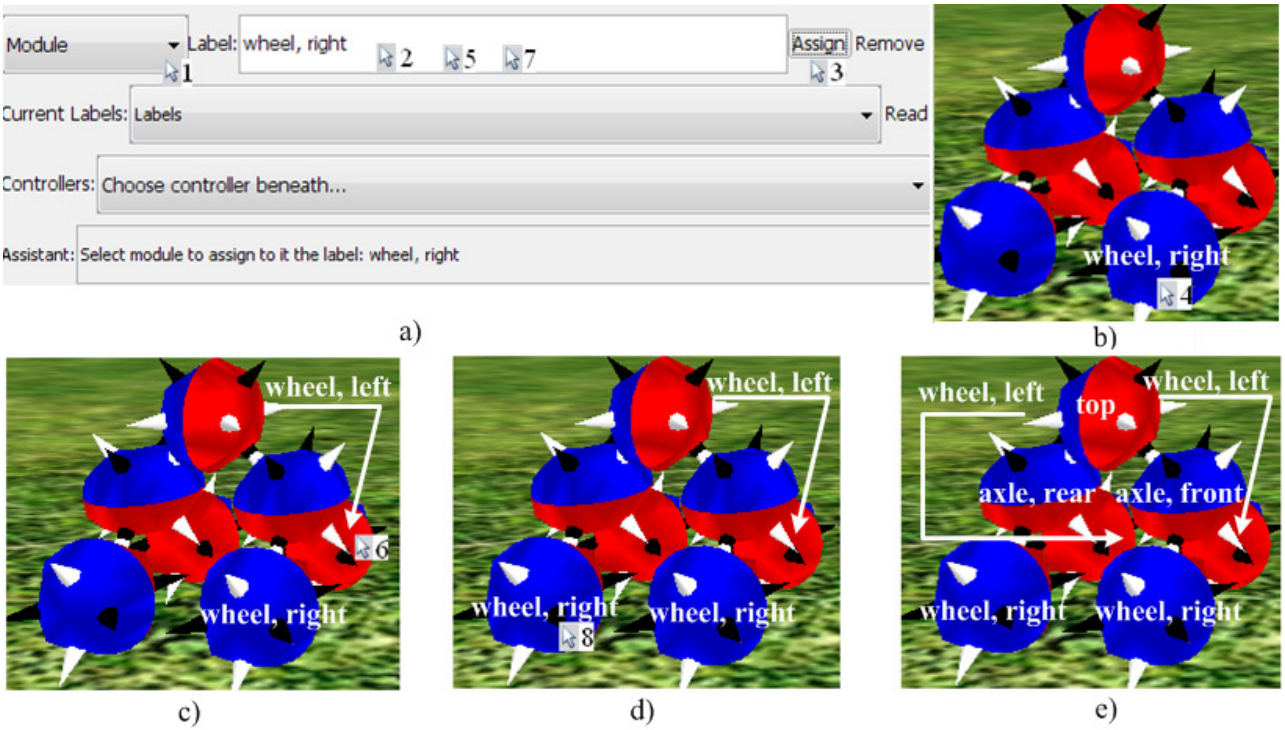


Figure 6.6 Assignment of labels to the modules in ATRON car configuration

The figure above exemplifies the usage of "Assign label" tool. First of all, the user constructs the morphology of ATRON car by means of the construction tools introduced in the previous sections or loads it from XML file. Next, the user chooses to which entity he/she would like to assign the label to (module in above case). Next, types in the names of the labels separating them by comma ("wheel, right") and presses "Assign" button. After that, selects the module in simulation environment to assign the label to. As a result, specific module is assigned the label typed in by the user in the GUI, see Figure 6.6 a, b and selections 1- 4. The process repeats until all modules in the morphology of ATRON car are assigned desired labels, see Figure 6.6 c, d, e and selections 5 - 8. Some of the selections were skipped in order to keep concision. Notice, that the labels are not graphically visualized in the simulation environment. Here this was done in order to

enhance exemplification. However, it is necessary to mention that experiments were carried out for that also. The decision was to do not include this functionality for two reasons: 1) examples supported by the jME contained unfinished implementation with a number of exceptions and 2) it is quite difficult to read the 3D text on the modules.

Now that there is the ATRON car configuration with assigned labels let us investigate how labels affect the process of assigning behaviors. See the code snippet beneath.

Code Snippet 6.1 Example of ATRON car driving forward behavior using labels

```
/* Instantiate the object for receiving the labels of the current module.
If current module is wheel and right rotate in positive direction. If
current module is wheel and left rotate in negative direction*/
Labels labels =  new ModuleLabel(module);
float speed = 0.9f; //speed of rotation
if(labels.has("wheel")&& labels.has("right")) rotateContinuous(speed);
if(labels.has("wheel")&& labels.has("left")) rotateContinuous(-speed);
```

The result of the source code above is ATRON car driving forward, which was introduced in Figure 4.3 in chapter four. Let us now compare the above source code and the source code in Code Snippet 4.2 from chapter four, where the same behavior of ATRON car was implemented by means of giving unique names to the modules. It is obvious that by amount of the source code both solutions are similar. Naturally rising question is: “So why current solution is better?”. The main advantages of current solution are: 1) The fact that the user no longer needs to keep track for uniqueness of the names of the modules; 2) Does not need to write the code for creating the modules; 3) There is a possibility to assign the same label to a group of modules and assign the same behavior to all modules in the group; 4) There are can be as many as needed labels assigned to the modules; 5) Labels can be also assigned to connectors and with additional implementation to other entities, later used to assign controllers to them; 6) The same implementation of controller can be reused on similar morphologies. For example, above example of controller implementation can be used on morphologies of ATRON, like “simple vehicle” and “six-wheeler” and more.

The main disadvantage of the tool for assignment of labels (Assign Label) is the fact that it will be tedious to assign label to each module in the morphology consisting of large number of modules. Just imagine the morphology of one hundred modules and user is assigning to each module at least one label. This process will take significant amount of time and will discourage the user using it. As a result, for these morphologies more effective way should be considered.

In general, we note that assignment of labels gives more flexibility to the user during assignment of behaviors than unique names and global IDs. Next, it eases the process of identification of modules in the morphology of modular robot, because this is done in simulation environment on a concrete morphology rather than through the source code. Moreover, it is possible to reuse the behaviors on similar morphologies of modular robots. Furthermore, we note that that the mechanism of using labels should be investigated in more detail in order to utilize whole potential of it in the future. The idea of labels was inspired by [32], [77].

6.4 Summary

During this chapter we familiarized with a number of features (tools) for both: interactive construction of modular robot’s morphology and assignment of behaviors, implemented in QPSS. First of all, we presented the features for interactive construction of morphology and were convinced that the user is supported with a number of tools, which significantly ease and speed-up the process of construction. Moreover, the user is not limited to a single tool, quite an opposite the user is presented with a possibility to choose the most suitable

for a situation in question. On the other hand, we also discussed the limitations of these tools and discovered that they are effective in the range maximum one hundred modules constituting the morphology of the modular robot. This is because these tools become tedious when constructing the morphology of the modular robot consisting of extra large (thousands) number of modules. Concluding our discussion, we came to an agreement that the new more effective tools should be implemented for this case. Next, we were presented the features for assignment of behaviors, which enable the user to interactively assign behaviors to the modules in the morphology of a modular robot and to implement the behaviors by means of using labels. Both give to the user the freedom of choice and enable to prototype the behaviors significantly faster. However, we also came to conclusion that the idea of labels should be explored in much more detail in the future, in order to utilize the whole potential of it. But in general, it is clear that QPSS enables the user to quickly prototype simulation scenarios involving supported modular robots.

Finally relying on the experiences introduced in previous sections we can agree that QPSS speeds-up and eases the process of prototyping in USSR. Moreover, to best author's knowledge QPSS exhibits features, which are not exhibited by any other simulator for modular robots. To emphasize innovative ideas are the following: different tools to interactively construct the morphology of modular robots, interactive assignment of behaviors involving 3D object manipulation and use of labels for identification together with assignment of behaviors to modules.

Chapter 7

7 Usability Tests

"Usability is like oxygen – you never notice it, until it is missing."
— Unknown

Chapter Objectives

1. Explain the motivation behind carrying out the usability tests for QPSS.
2. Introduce the main steps in the test plan, which was created to keep focus and comprehensive work style.
3. Discuss results of the usability tests and their influence on QPSS. Moreover, present overall opinion of participants about QPSS.

7.1 Introduction

The intent of this chapter is to overview the usability tests, which were in general carried out for improving QPSS and receiving overall opinion about it. We will begin our discussion with presenting the motivation for involving usability test in the development of QPSS. Next, present the key features of the test plan, which was devised for keeping focus and comprehensive work style before and during usability tests. Finally, we will discuss how usability tests affected the QPSS and evaluate the feedback received from the participants of them.

7.2 Motivation

It is well-known wisdom that theory is quite different from the practice. In our case this means that we can argue about usefulness of the QPSS theoretically, carry out experiments on our own and attempt to convince the society of scientists working with modular robots by means of introducing documented results. However, often this is only a half of the way in order to promote the users in actually using it. Meaning, that it is not enough for software to be implemented and well documented, it should be also tested for usability with concrete users. In general, usability testing is a technique used to evaluate a product by testing it on users [28]. The results of the tests give new perspective on requirements (called feedback), the feeling of completion for some of requirements and involve the users into development process which diminishes the feeling of "rebellion". In software development, the term "rebellion" describes the situation when the users are presented and should be working on their daily basis, with software they have not been working before. Usually, in such situation the majority of them neglect using it, because it is difficult and time-consuming to learn it, they prefer to use the methods and applications they are comfortable with and proven to be useful in the past. As an example, just recall the appearance of Windows Vista and the opinions of its users about it. Knowing all that, the development process of QPSS was followed by the small usability tests involving Ulrik Pagh Schultz as the main user. Received feedback was analyzed and improved on during the project. At the end of the project, it was decided to widen the group of users by means of introducing bigger usability tests and promote using the QPSS. At the same time, receive the feedback and overall opinion about it.

Before the testing it was decided to create the test plan, which helped to keep focus and comprehensive work style.

7.3 Test Plan

Test plan consisted of several key points, which we will discuss next and were inspired by [29], [30]. They were the following:

- Participants were chosen to be the persons in one or another way familiar with modular robots and willing to try out QPSS.
- Pre-test questions were oriented towards identifying and clarifying the interests of participants and how QPSS could help to achieve their goals.
- Scope and purpose were defined for focusing on what should be tested during the test.
- Data collection is the process of observing the user during the test and capturing the difficulties he/she is facing.
- Questionnaire was used to receive the feedback about tested features of QPSS and overall opinion about it.
- Schedule and location were devised so that it was possible to improve QPSS after usability tests according to the feedback and before the testing, allocate the most suitable time for participants.

7.3.1 Participants

First and probably foremost important are the participants. Taking into account that QPSS is quite specific to modular robots, tests were designed with orientation towards two major user groups called experts and beginners. Experts were the users, who in one or another way were and are participating in the development of USSR, familiar with modular robots and continuously working with them. The group of experts was further split into two subgroups: users who are mainly working with ATRON and the ones working with Odin. For each group, the tests were designed to work with their relevant modular robot. As for group of users called beginners, they were presented with simple form of documentation with the help of which, they could explore the functionality of the QPSS in step-by-step fashion, see Appendix C for used example. In total, there were four experts (including moderate) and one beginner, who participated in the tests.

7.3.2 Pre-test Questions

Pre-test questions helped to define the scenarios for testing. Here the expert users were asked two simple questions: 1) with which morphology are you working most often? 2) With which simulations are you using most often and what would you like to achieve there? As a result, during the first part of the test the user was constructing the morphology of the modular robot with which he was familiar with and during the second experimenting with behaviors he worked before or was interested in by means of using QPSS. In this way, the users were able to compare their old style of work and advantages of QPSS.

7.3.3 Scope and Purpose

In the case of the user group called experts, it was interesting to test if QPSS is actually simplifying the process of construction of modular robot's morphology and assignment of behaviors. Due to the fact, that the users were the experts, none documentation was delivered before the test. The strategy during the test was simple, but straightforward and consisted from two steps. First, they were shown how to interactively construct the morphology of a modular robot and later interactively assign behaviors using QPSS. At the same time, relevant functionality was explained in the GUI. Second, they were trying to repeat the same process on their own. This strategy was a form of challenge. Moreover, they were also guided through the process with the hints and explanations. Furthermore,

they were also promoted to follow the hints given by the toolbar named "Assistant" for evaluation of its usefulness. In this way, it was tested how easy is it to use the functionality of QPSS and understand the GUI by means of presenting examples before the first use. As for beginners, they were following step-by-step documentation describing simple example and were observed by the author if it is easy to guess GUI functionality, to navigate in the GUI and simulation environment in order to achieve presented example.

It was decided to do not include the testing of labels, because of several reasons: a) it is quite time-consuming task, which requires from the participant to invest significant effort to understand and experiment with. The whole idea behind usability tests was to keep them as short as possible without taking too much time from the participants; b) experimentation with labels is still in early stage of research. More time is required to explore and improve QPSS so that it will reflect the whole potential of labels.

7.3.4 Data Collection

The main goal here was not to test the features which were included in the test, but discover new unpredictable difficulties not accounted for, before the test. As a result, during the second step of the test (when user was working with QPSS) author was identifying the elements of the GUI which were difficult to guess and understand for the user. Usually it is easy to identify these, because then the user is confused and cannot continue without the hints and explanations. See Appendix D for noted difficulties.

7.3.5 Questionnaire

After the test each participant was asked to fill in a short questionnaire about their experiences with QPSS. The questionnaire was intentionally written in a form of multiple - choice, because the main goal of it was to get specific feedback on features which were tested. Only several questions allowed broader answers for getting overall feeling about QPSS and future possibilities. Mainly, questions were oriented towards getting answers on main goals of the tests relating to the GUI simplicity, usefulness of the toolbar called "Assistant" and overall opinion comparing the QPSS and participant's previous experiences with USSR in the context of constructing modular robot's morphology and assignment of behaviors. For the questionnaire refer to Appendix D.

7.3.6 Schedule and Location

Before the tests each participant was asked about the best time and place to carry out the tests for keeping informal and as comfortable as possible test environment. As a consequence, most of the tests took place in computer rooms using author's PC.

It was decided to carry out two usability tests, in order to receive the feedback after the first and improve on it for the second. Second usability test was used for checking if improvements solved the problems discovered during the first usability test. At the same time, collect new data. In the first usability test were involved three moderate level users of USSR, second – one expert and one beginner.

7.4 Results and Improvements

During the first usability test it was noticed that the users were confused in two cases in relation to GUI design, which are the following: 1) when the functionality of the feature (tool) consisted of several components in the GUI. To be more precise it was difficult for the user to do not forget to press several components in a sequence in order to define the tool. For instance in order to start interactively constructing the morphology of a modular robot, the user had to press three components in a sequence. 2) Sometimes it was difficult

to associate the purpose of the tool to its name or different values in the combo boxes. See Appendix D for more detailed description of above cases.

After the first usability test, author came to conclusion which can be expressed in one sentence as: There is a lesson to be learned here. During development of QPSS the design of GUI was more oriented towards testing (debugging) the functionality of QPSS, rather than being user oriented. As a result, some of the functionality of several tools was separated and assigned to different components in the GUI. This resulted into problematic issues with GUI described above. In an attempt to improve on these problems the GUI design was changed by means of several modifications: 1) Unification of functionality from several components into one, where it was possible, 2) Using textual hints in the combo boxes, which describe the purpose of them, 3) Introducing default sequence of selection, meaning that if user selects one of the components, then the most probable functionality after it is assigned automatically. This enables the user to skip several selections in the GUI. In case different functionality is desired the user is able to override it by means of selecting other tools.

Even with problematic issues mentioned above, after the first usability test the participants found QPSS as being useful for both interactive construction of morphology and assignment of behaviors. During interaction with GUI they found tool bar called "Assistant" being helpful in guiding to how to use the tools. Moreover, the users expressed interest in using QPSS after the test and during it. Furthermore, each of them had interesting ideas how to improve QPSS and new functionality to add. Due to the time limitation for the project, only the most urgent of them were implemented in QPSS, other left as a future work. All suggestions proposed by participants can be found in Appendix D.

During the second usability test, significant improvement was noted. Participants were no longer as confused as in previous usability test. The expert participant achieved the goal and learned functionality quite fast. Nevertheless, beginner was quite confused in understanding the sequence of workflow. It seems that current design of GUI reflects the grouping of functionality into toolbars. This does not work well. Possible solution here is to redesign the GUI so that it will reflect the prototyping with modular robots in more step by step fashion. Here the first step would be the construction of morphology and second assignment of behaviors. In this way, during each step the user will face smaller set of functionality and GUI will be more intuitive. This is quite well-known design strategy for developing internet forms, where information to fill in comes is presented to the user in step by step forms. In one sentence this could be summarized as: "Divide and conquer".

Finally, basing our conclusion on the results of the questionnaire and experiences with QPSS we note that QPSS enables the user to achieve quick and easy prototyping of simulation scenarios. The fact is that users expressed interest in QPSS and evaluated it as useful comparing it to their previous style of work. At the same time, the GUI of QPSS was improved for usability test and is not yet completely intuitive. Taking into the account that the design of GUI was not of highest priority current design is acceptable. However, future redesign and new usability test could evolve it to be more intuitive.

7.5 Summary

In this chapter we first discussed the motivation for carrying out usability test during development of QPSS. Here we came to conclusion that there are a number of reasons which promote us using usability tests, like: promoting, improving QPSS, receiving feedback about it and so on. After that we introduced the test plan, which was used during the work and was found to be helpful in organizing the usability tests. Next, we overviewed the results of the usability test, their influence on QPSS and general opinion of usability tests participants about QPSS. Basing our conclusion on experiences in the

previous chapter and experiences of usability tests participants we agreed that QPSS is significantly alleviating the process of modular robot morphology construction and assignment of behaviors. This results into quick and easy prototyping of simulation scenarios in USSR. We also agree that the GUI could more intuitive.

Chapter 8

8 Implementation and Design

"In the beginner's mind there are many possibilities, but in the expert's mind there are few."
— Suzuki, Shunryu

Chapter Objectives

1. Describe the main principles and design decisions behind the module of functionality of OPSS for interactive construction of modular robot's morphology.
2. Discuss the implementation and design of the modules of functionality responsible for assignment of behaviors.
3. Present the design and implementation details of additional functionality provided by QPSS.

8.1 Introduction

The main goal of this chapter is to explain the implementation and design details of each module of functionality provided by QPSS. With this intent we begin discussing the module responsible for interactive construction of modular robot's morphology. Here are addressed main principles of operation and exemplification of design. Next, in the same fashion we introduce two modules for assignment of behaviors, module for generic construction tools and the module responsible for saving-loading of data, about the modules in static state of simulation environment, from XML files.

8.2 Interactive Construction of Morphology

In order to communicate the main principles of operation lying in the module of functionality of the QPSS, responsible for interactive construction of modular robot's morphology let us investigate the flow of the logic common to the four major tools for interactive construction of morphology exemplified in chapter six. For that we will involve the usage of activity diagram, which is essentially visualization of the logic flow [78]. Refer to the Figure 8.1. The figure exemplifies the flow of the logic which is generic to the four construction tools of modular robots morphology. In this abstract form it is also generic to any modular robot supported by the framework (currently these are ATRON, M-Tran and Odin). However the details of implementation differ from one modular robot to another. That is on what, the idea of the framework is based on. First action in the flow is the receiving of input specified by the user from GUI. In general, it is the specification (set-up) of the tool the user would like to use on a modular robot. At this point the concrete tool is not yet completely set up. The functionality of the tool is called after receiving the input from simulation environment (selection of the module, connector and so on) and checking if correct tool is chosen in GUI for the modular robot selected in simulation environment. If it is matching, the chosen tool can be used in unlimited amount of selections on the modules in simulation environment (<< unlimited>> amount of selections in the diagram). Chosen tool will be active until new tool be chosen in the GUI. At this point after selecting the concrete entity (module or connector) in the simulation environment the matching tool for matching modular robot is called. As a result new

module of the same type of modular robot is added into simulation environment taking the selected module as raw template. This is a copy of selected module (the same rotation, position and appearance). After that the description of selected module (components) is checked against the array of objects keeping the information for mapping selected module to newly added module. The format for storing information is: (connectorNr,selectedComponentRotation,newComponentRotation, newComponentPosition). The first two constants represent the information about the module (its components) selected in simulation environment and next two the information about the rotation and the position of newly added module (its components).

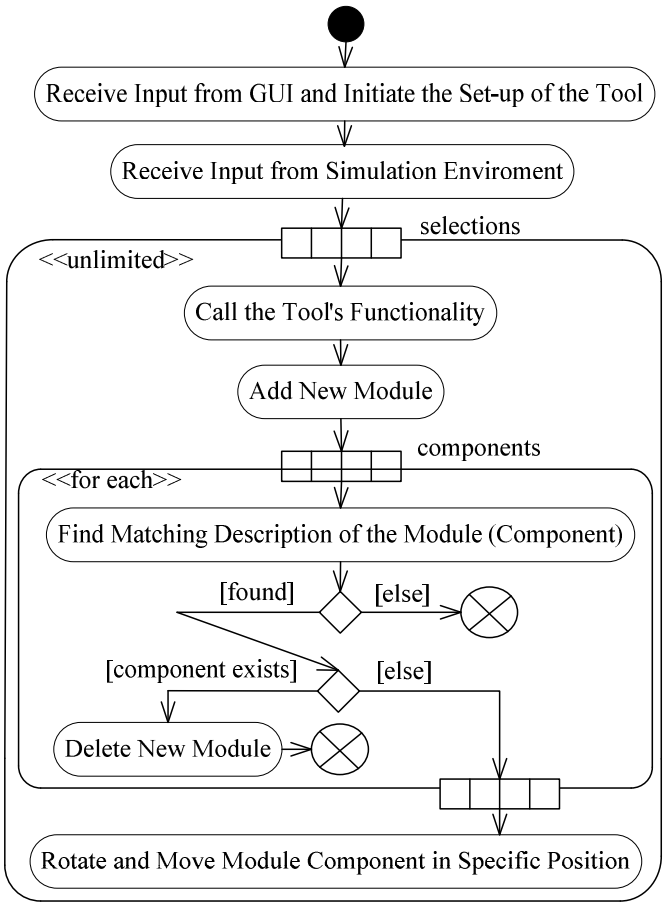


Figure 8.1 The flow of the logic for the four morphology construction tools exemplified in the chapter six (for example: “On selected connector”)

In other words, how new module should be rotated and where placed, respectively to selected module. The format of storing the information is a bit different for each modular robot, because of the information variation for different types of modular robots. The search for match is executed for each component of the module (<<for each>> in above figure). If the match is found then the position of the newly added module component is checked against all the positions of the modules in simulation environment. In case there is already such position the newly added module is deleted. This is done to avoid overlapping of modules, which is critical for correctly interconnecting the modules later on. The check for the existence of the same module (component) with the same position is executed in modular robot specific search interval for each Cartesian coordinate. This ensures the precision of the match. If there is no repeating module with the same position newly added component is rotated and positioned according to the last two entries in the object keeping the information for mapping selected module to newly added module. These are newComponentRotation and newComponentPosition.

After introducing general flow of control, let us examine the main principles on which current solution is based on in details in order to analyze their advantages and disadvantages. These are the principles of placing newly added module components respectively to the selected one and the search for existing components at interval in space.

8.2.1 Placement of New Modules

As it was previously mentioned the construction of modular robot's morphology requires the mapping of information about selected module to newly added module. For this each construction tool is using the array of objects describing the information about the selected module (all connector numbers and standard rotations) and newly added module, which should be placed respectively to selected one (all rotations and calculated positions). Here all values are constants except the calculation of position. The position is calculated respectively to the current position of selected module. For visualization consider example beneath.

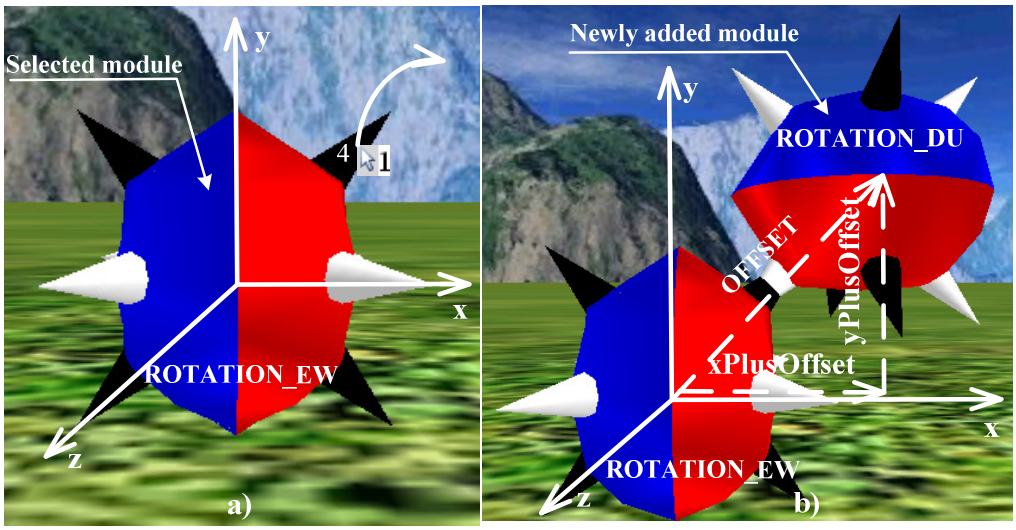


Figure 8.2 Placement and rotation of new ATRON module respectively to selected module: a) selected module and b) selected with newly added modules

The figure above shows the module selected in simulation environment of USSR. To be more precise the connector number four is selected and rotation of selected module is "ROTATION_EW", see Figure 8.2 a. As a result of this selection the QPSS scans through the array of objects containing the mapping of modules and locates the entry describing current situation. Refer to the code snippet beneath, last two LOCs.

Code Snippet 8.1 Example of object mapping selected module to newly added module

```
/* The physical distance between the two connected ATRON modules */
final static float OFFSET = 0.08f;
/* The x and y are current values of selected module position*/
float xPlusOffset = x + OFFSET;
float yPlusOffset = y + OFFSET;
 /* Mapping of selected module to newly added module*/
new ModuleMapEntryHelper(CONNECTORnr4,ATRON.ROTATION_EW,ATRON.ROTATION_UD,
new Vector3f(xPlusOffset,yPlusOffset,z)),
```

The entry was located by two search criteria named as connectorNr4 and "ROTATION_EW". The first two constants in ModuleMapEntryHelper are matching our current situation. The last two values are describing what should be done in order to place new module correctly on the connector number four. In our case it is to rotate it with "ROTATION_UD" and place it in position (`xPlusOffset,yPlusOffset,z`). The

variables with Offset annotation indicate that these variables should be calculated dynamically by extracting current value of x and y position of selected module and adding to them the physical OFFSET between two ATRON modules. The z value for newly added module is the same as for selected module. Refer to the Figure 8.2 b above for geometrical explanation of calculation. Notice the triangle called equilateral with equal length of sides. The same procedure is repeated for each selection of other connectors on any modules.

On the same principle is based construction of M-Tran and Odin modular robot morphologies. The key concept here is of course OFFSET value, which is varying from one modular robot to another and depends on physical dimensions of it.

The main advantages of current approach are simplicity and robustness. Simplicity means that the user with low level of experience in programming with Java and knowledge of geometry should be able to quickly understand the code for creating the array of objects describing his/hers desired new modular robot's rotations and respective positions of the modules. Consequently the maintainability of the QPSS should not be a difficult task. In case, if the underlying principle would be much more complex, there could be the risk that users would avoid using it. Furthermore, current approach is highly relevant to modular robots, because most of them have so-called grid positioning of modules. Grid positioning means that the modules are arranged into dense, regular and often symmetric structures. Under the term robustness is meant the fact that the solution works with one hundred percent precision. During experimentation stage of the project there were no cases of failures for all three modular robots.

The first disadvantage of current solution lies in its lack of flexibility. It is not flexible enough to support so-called reconstruction of morphology. Meaning that there is no way to further construct the morphology of modular robot after starting simulation and later stopping it to modify the morphology. However, this was not the requirement for the project, but could be considered in future. Here the array of objects describing the mapping of modules can be replaced by more generic mathematical solution by means of introducing right Java interface. Moreover, current approach will be difficult to reuse for irregular, highly heterogeneous non-grid modules. The second disadvantage is the fact that it requires quite a lot of LOCs to implement. For example only to encode the array describing the mapping of selected module to newly added module requires for ATRON – 54, for M-Tran – 174 and for Odin – 36 LOCs. The general tendency is: if the module consists of more than two components the amount increases significantly. In order to improve on both disadvantages more generic solution should be devised. During the project several attempts were taken to find it. For ATRON modular robot such solution was close to usable. The main reasons why this solution was not included were, the following: 1) it is not strict requirement for the project; 2) time limitations of the project indicated that the time should be used for assignment of behaviors.

8.2.2 Search for Existing Modules (Components)

Before placing newly added module respectively to selected module QPSS is checking if the position in interest is already occupied by another module. For that the "candidate" position for newly added module component is checked against all positions of the module components currently in simulation environment. The search is executed not just comparing the precise values of x, y, z, but in modular robot specific search interval for each coordinate. Let us reuse the example above in order to explain the geometry behind it, see the figure Figure 8.3. In the figure is depicted selected module and to the right of it the "candidate" position for newly added module. Before actually placing the newly added module in this position, the "candidate" position is checked against all positions of components existing in simulation environment. Moreover, it is checked in search interval

for x, y, z. The search interval is always the same for different axles (see highlighted with orange color). However, it is specific for different modular robots. The values for search interval were found empirically and depend on the physical dimensions of the modules and distances between their interconnection. In fact, they are considered as search tolerances. Currently they are not precise enough and should be improved in the future. If during the search is found the component with position falling into the search interval, newly added module is deleted and as a result there is no overlapping of modules (components).

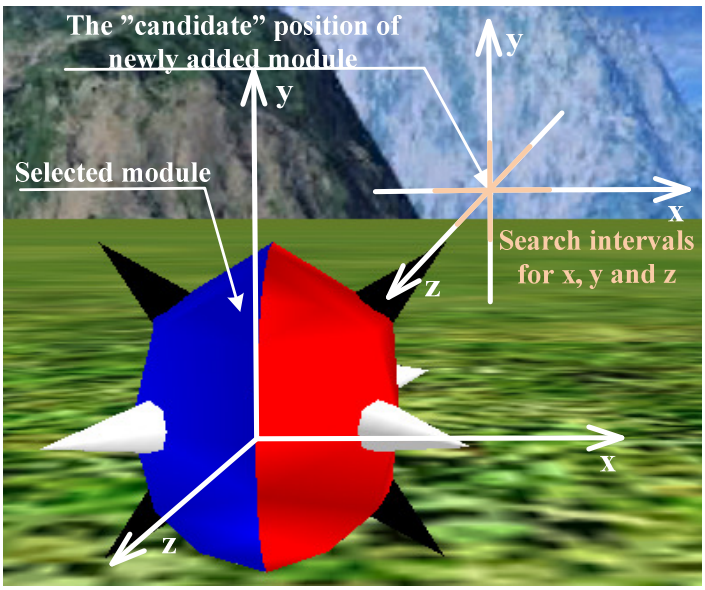


Figure 8.3 Search for existing modules

Next let us discuss the design of QPSS module of functionality responsible for interactive construction of morphology, which includes both principles explained above.

8.2.3 Design

For the DCD (Design Class Diagram) of the functionality for interactive construction of morphology see Figure 8.4 , which is only partial DCD covering the four construction tools explained previously. More detailed DCD can be found in Appendix E.

First of all, there are two entry points of QPSS in the design of USSR. These are: for starting the GUI of QPSS (class "QuickPrototyping") and activating selection tools (class "ConstructionToolSpecification"). As a result, only slight modifications were made in order to integrate QPSS. Consequently, with co-evolution of QPSS and USSR, the small ripple effect[38] should be expected when introducing changes. The main responsibility of both classes is to take in the user's input and call appropriate construction tool's functionality. Essentially, this means that the class highlighted with cyan color is responsible for instantiation of objects together with the class highlighted with orange color and calling appropriate methods associated with the tool. For instantiation of objects is used design pattern called Abstract Factory [82], [83], by means of which are instantiated objects of interfaces highlighted with blue and brown color. The reason why there are two parts is separation of concerns and abstraction. The part highlighted with brown color is responsible for creation of new modules and part highlighted with blue is responsible for rotation and positioning of new module with respect to the selected module for different modular robots. Both parts of implementation follow the design pattern called

[38] Ripple effect is "effect caused by making a small change to a system which impacts many other parts of a system"

Template Method, the unique features of which are utilization of template methods executed on the objects (indicated by the red letter T in the figure) and primitive operations executing different functions (red letter P). At the same time, template methods are combination of primitive methods. Initially the design pattern called Strategy was used. However, the implementation contained a lot of repeating code and it was changed to Template Method pattern. As a consequence, the abstract class of each part is now containing the template methods with the source code which is similar for supported modular robots and primitive operations are housing distinct source code. This also means that the source code is abstracted for construction of all three supported modular robots.

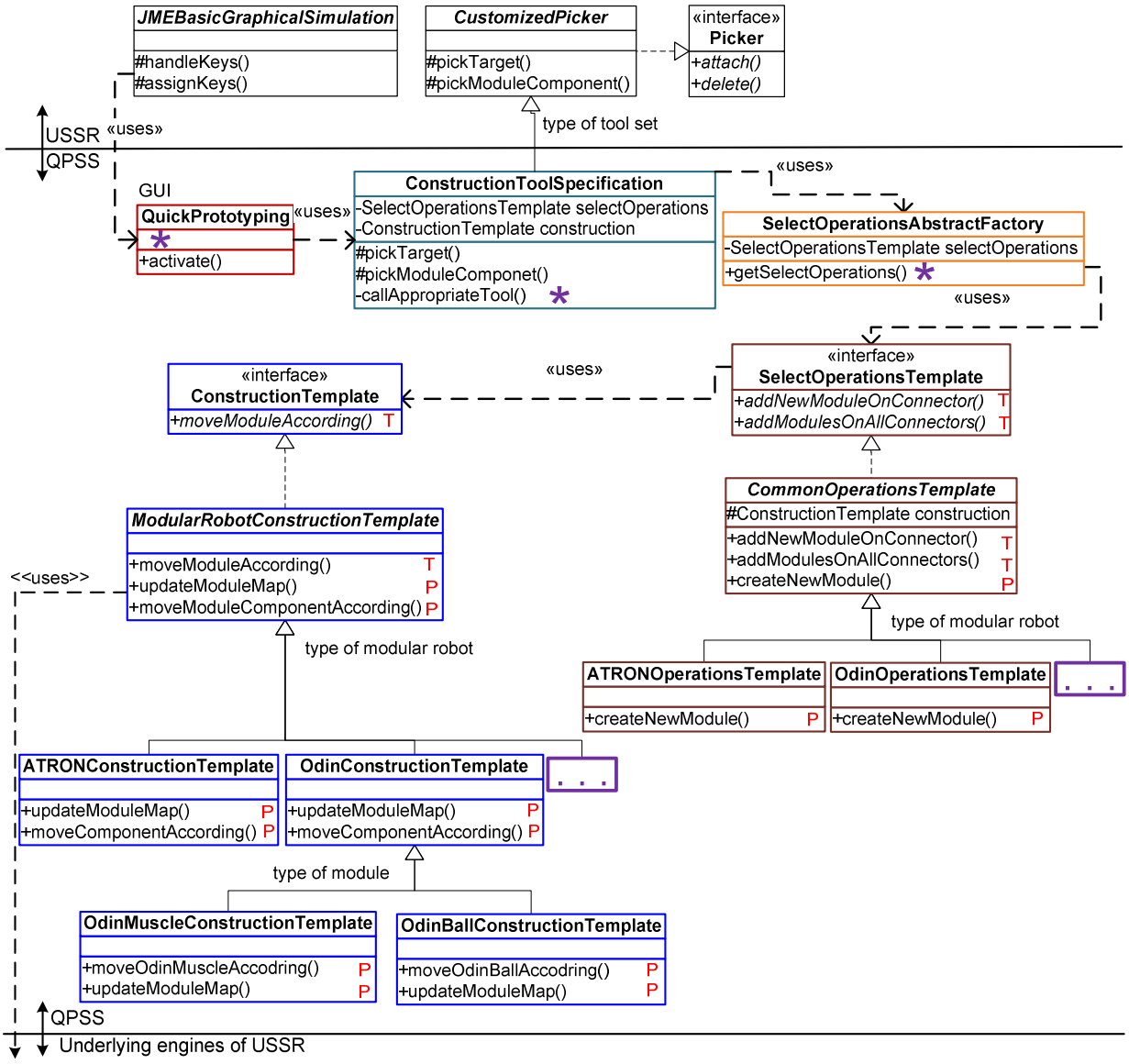


Figure 8.4 Partial DCD covering the four interactive construction tools

The only notable disadvantage in the current design is direct usage of jME by class called "ModularRobotConstructionTemplate", which conflicts with the design decision of USSR developers to implement USSR as engine independent. This can be improved by means of introducing interface in the future.

The main design feature is of course extendibility because that what the frameworks are all about. In other words, it is possibility to add support for a new modular robot reusing existing source code. In order to support new modular robot the user (programmer)

should implement the classes highlighted with purple color (rectangles in the diagrams above). Here the main implementation should be oriented towards programming primitive operations and reuse of the source code existing in template methods. The source code of currently supported modular robots should help in achieving that. It is also highly recommendable to scan for missing implementation in the methods labeled with purple asterisk. Like for instance, addition of object instantiation in the class called "SelectOperationsAbstractFactory". In general, for supporting new modular robot the user should implement three methods and modify several others. As result, the four different tools for interactive construction of modular robot's morphology will be available to use.

Finally we can agree that the design is following major OO principles and is prepared for extendibility. Next let us in similar way discuss the modules of functionality for assignment of behaviors.

8.3 Assignment of Behaviors

There are two modules of functionality responsible for two different ways to assign behaviors. They are the following: interactive assignment of behaviors using the library of behaviors and using labels for identification of modules in the morphology of a modular robot. From the last we will discuss the tool called "Assign Label". Let us investigate the flow of the logic for both of them, see Figure 8.5.

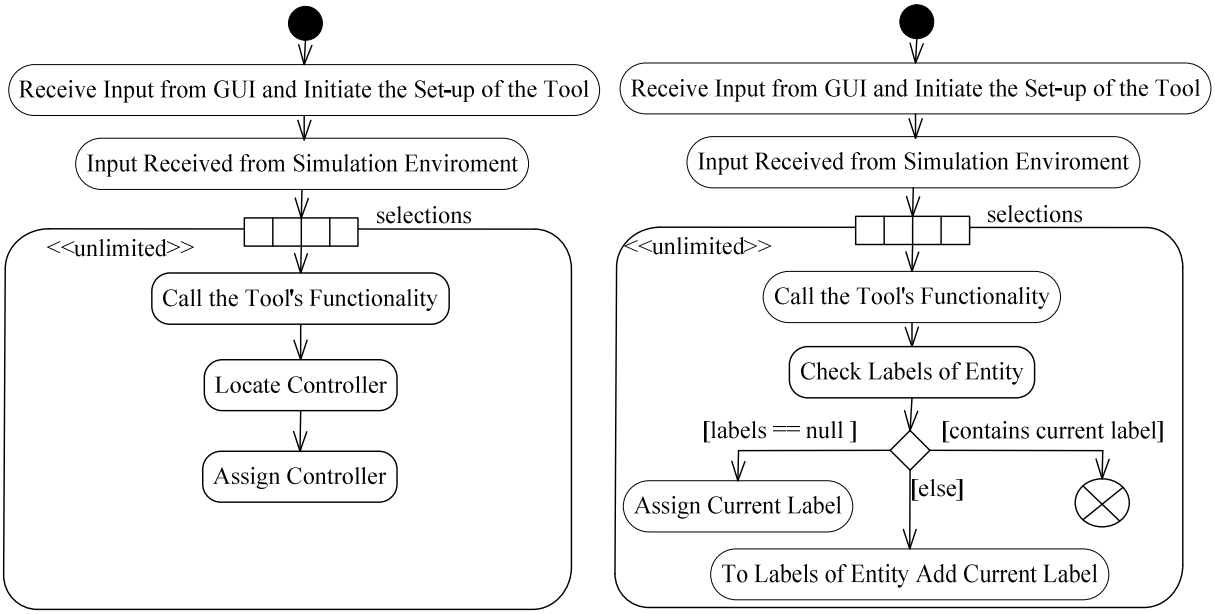


Figure 8.5 The flow of the logic for assignment of behaviors: a) interactive assignment of behaviors using library of behaviors and b) Assign Label tool

From the figure above it is observable that both approaches for assignment of behaviors share the same logic flow for calling tools functionality as for interactive construction of morphology. Here the user chooses the tool in the GUI and it is activated when the entity (module or connector) is chosen in the simulation environment. The tool can be used unlimited amount of times on entities in simulation environment until another tool is chosen in the GUI. The tool for interactive assignment is based on Java reflection package. Here the names of controller classes located in the default package are loaded into the GUI component (combo box). In order to add new controller the user should implement new class and place it into the default package. As a result, the name of the controller class chosen in the GUI combo box can be assigned by means of selecting the entity in the

simulation environment. The controller is located by its name chosen in the GUI and assigned by means of calling predefined method called “activate”. As for assignment of labels the initial action is to check if the entity already has the label typed in the GUI. If the entity already has the same label then it is not assigned. If there are no labels assigned to the entity or there is no the same label, it is simply added to the labels previously assigned to the entity. The labels are stored as strings separated by comas.

The main advantage of integrative assignment of behaviors using library of behaviors is the fact the user can easily learn how to implement his/hers controller by simply analyzing existing behaviors and experimenting with them. Essentially, the user should familiarize with only one package and will be able to quickly and easily assign controller to specific module in the morphology of supported modular robots.

As for disadvantages, the first is that the user should have at least minimal skills in programming with Java. Second is the fact that when Java reflection package is used it is difficult to locate where in the controller exceptions or bugs appear, because the highest detectable exception is the class name in which it appeared, but not precise place.

The main advantage of assignment of labels is its significant flexibility. The user can assign whatever label is desired and whenever it is most comfortable (in static state and during running simulation).

The disadvantage is the fact that it is time consuming to assign several labels to all modules in the morphology of a modular robot consisting of significant number of modules. In the future, more efficient way to assign the label should be considered. For example possibility to load the labels from the text or XML files into GUI at runtime.

8.3.1 Design

As it was previously mentioned the implementation of interactive assignment of behaviors is based on Java reflection package. Here the main responsibility for locating the class specified by the user (in the GUI) is assigned to the class called “AssignControllerTool”. Then the class is instantiated and the method named as “activate” is called, see Figure 8.6. More detailed DCD can be found in the Appendix E.

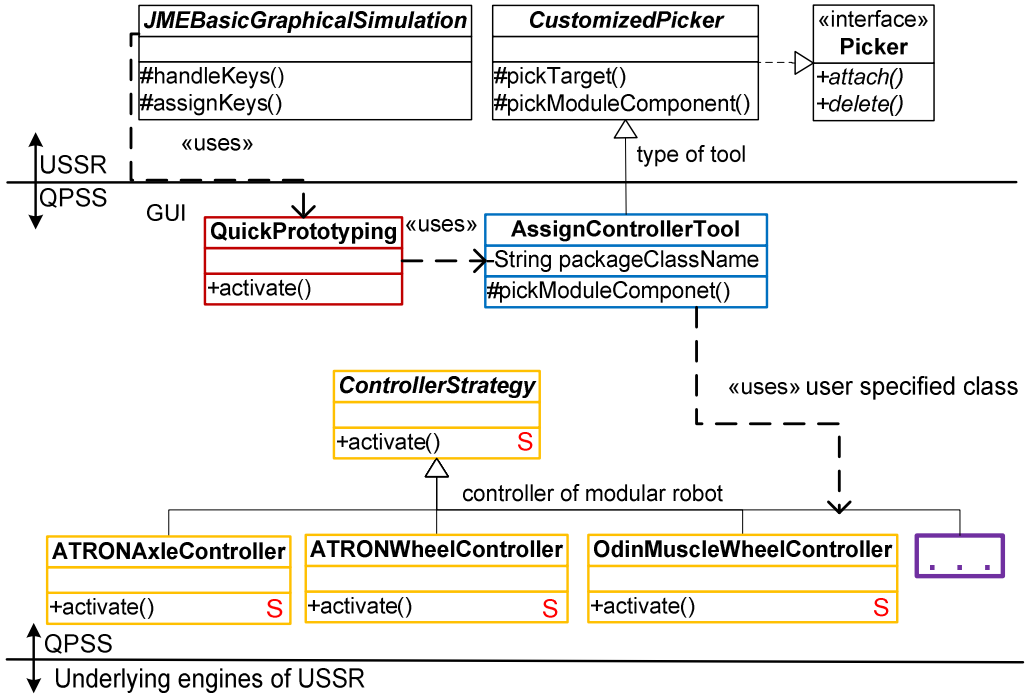


Figure 8.6 Partial DCD covering main elements of interactive assignment of behaviors

The implementation of specific controllers is based on Strategy pattern (highlighted with orange color). Here the method called “activate” (highlighted with the red letter S) is the method from Strategy pattern and is the main method for implementing controller logic. In order to add new implementation of controller the user should implement new class (highlighted with purple rectangle) and inherit it from the class “ControllerStrategy”. After that implement the method “activate” and as a result the user will be able to choose the name of newly implemented controller class in the GUI and interactively assign it to the module in the simulation environment of USSR. The main reason why the Strategy pattern was chosen for the design is the fact that each controller implementation is completely different. Nevertheless with evolution of this functionality Template Method could be more suitable choice.

As for design of Assign Label tool, it is based on Factory and Template Method patterns see Figure 8.7. More detailed DCD can be found in Appendix E.

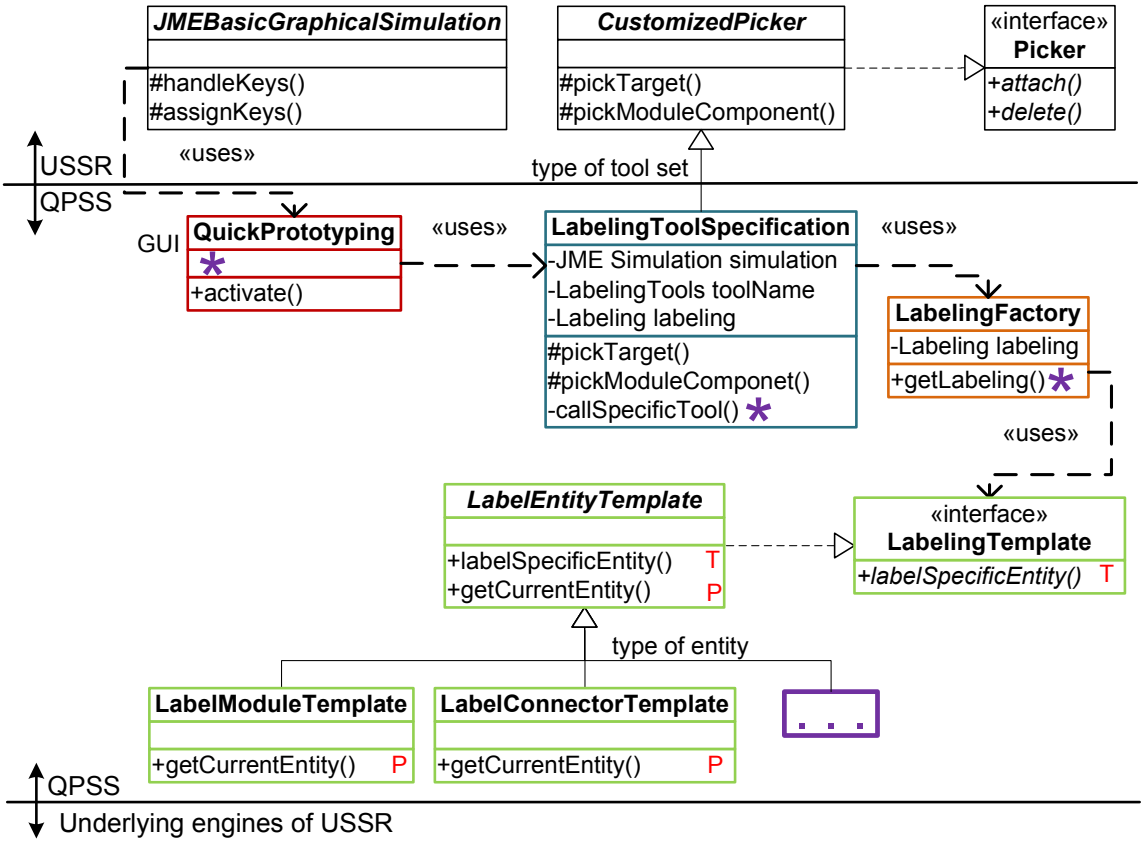


Figure 8.7 Partial DCD covering the main elements of Assign Label tool

The Factory pattern (highlighted with brown color) is used to instantiate the classes for labeling different entities. So far are supported module and connector. In order to support the labeling of different entities Template Method is used (highlighted with green color). Here the template method (highlighted with the red letter T) is the main method to activate labeling of entity. Primitive operations (highlighted with the red letter P) are used to receive the entity the user is interested to label. For supporting new entity to label (highlighted with purple rectangle), for example component of the module or sensors, the user should implement new class, inherit it from the class named “LabelEntityTemplate” and implement primitive operation. Also scan the classes for missing implementation (highlighted with purple asterisk). The main reasons why Template Method pattern was used are: because there is about ninthly percent of source code repeating for each entity

and easier addition of new supported entities.

Now that we discussed the implementation and design on the main functionality let us take a glance at the additional functionality supporting previous.

8.4 Additional Functionality

In QPSS as additional functionality are considered generic tools enhancing interactive construction of modular robot's morphology and saving-loading the data about simulation from XML files.

The design of the module of functionality responsible for generic tools is based on usage of Strategy pattern, Figure 8.8. Initially it was implemented in USSR and currently modified for QPSS. To be more precise the method "pickModuleComponent(…)" responsible for selection of modules and their components was implemented before. For QPSS the method pictkTarget(…) was added, which is responsible for selection of connectors on the modules. Both methods are major for getting user's input from simulation environment and calling further functionality in any of tools supported by QPSS.

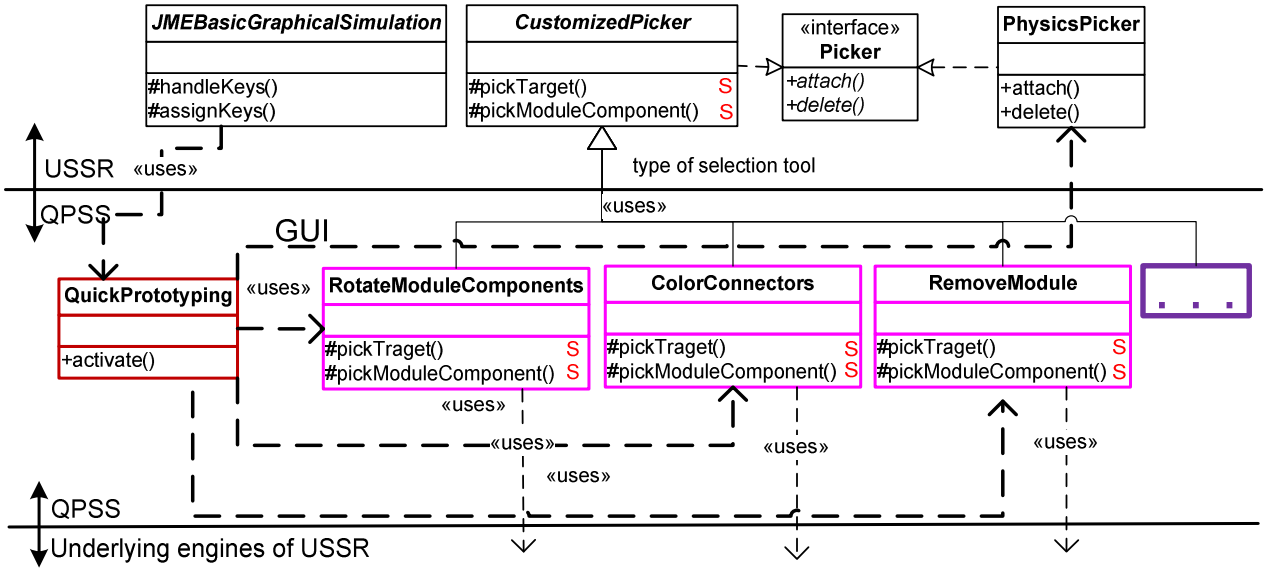


Figure 8.8 Partial DCD covering the main elements of generic construction tools

In the figure above, generic tools are highlighted with pink color and red letter "S" indicates the methods following Strategy pattern. Each class is assigned self-explanatory name. Purple rectangle indicates that in order to add new generic tool, the new class should be inherited from the class named as "CustomizedPicker". Currently all generic tools directly depend on the underlying engine of USSR, which is contradicting to design decision of USSR developers for USSR to be engine independent. Nevertheless, the dependency is low. As a consequence, the future task is to improve on this disadvantage. One more thing to mention is the class called "PhysicsPicker", which initially was responsible for moving of modules in the simulation environment with the mouse. It was limited to moving modules in dynamic state. For QPSS it was modified to support moving modules and their components in the static state of simulation.

The module of functionality responsible for saving-loading of XML files is based on usage of SAX (Simple API for XML) and DOM (Document Object Model) application programming interfaces. SAX is a serial access parser API for XML and DOM is cross-platform and engine-independent convention for representing and interaction with XML, HTML and so on. For saving the data about the modules in simulation environment is used SAX and for loading – DOM. It was previously known that DOM is slower than SAX,

however it was decided to use both. The main reason is to support the programmers, which are familiar with at least one of them.

The design relies on the usage of Template Method pattern, see Figure 8.9. Here red letters ("T" and "P") indicate template and primate methods, respectively.

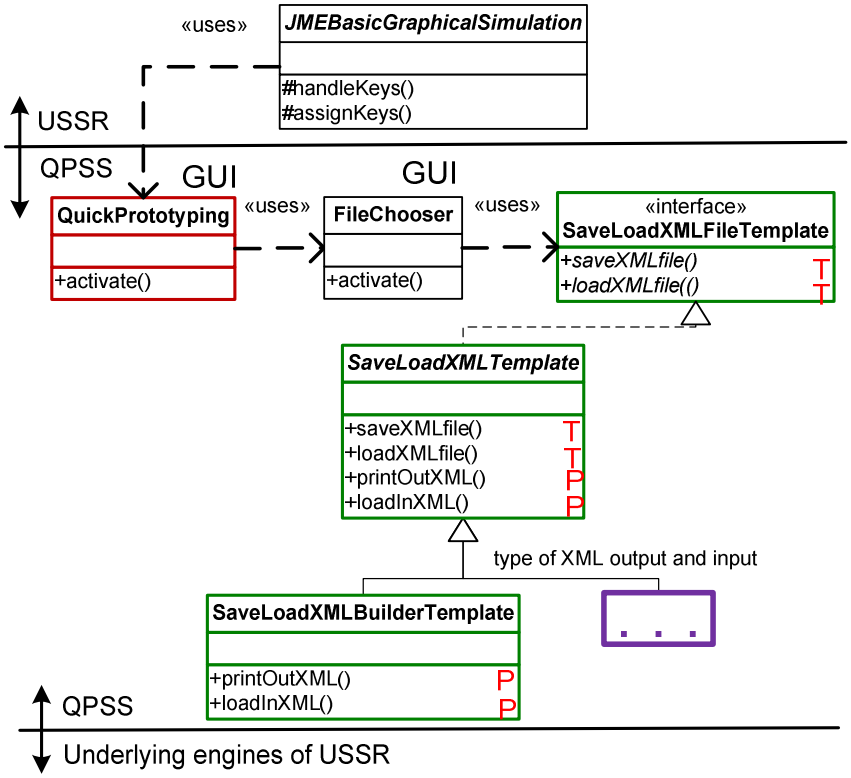


Figure 8.9 Partial DCD covering the main elements of saving-loading of XML files

In the figure above the classes and interface highlighted with green color belong to Template Method pattern. The main reason why this pattern was used is reuse of the source code and support the programmer with possibility to implement his/hers format of saving the data in the XML file (highlighted with purple rectangle). The proportion of reused to specific source code is approximately forty-sixty.

In the XML files is saved the data about the modules in the simulation environment. To be more precise, the following: position, rotation, type, name, connectors and physical appearance. The only currently missing data to save is the name of the controller assigned to the module. Because of that the user should reassign the controller when XML file is loaded. In the future the support for saving controllers should improve reusability using XML files.

8.5 Summary

In this chapter we discussed each module of functionality provided by QPSS. First we introduced the module responsible for interactive construction of morphology. Here we came to conclusion that underlying principles are quite simple and based on geometry of modular robots (mechanical design). However, simplicity can be also considered as the major advantage, because QPSS is implemented as OO framework. So, as simpler the solution as easier it will be to understand it and as a consequence maintain it in the future. We also agreed that more generic solution can substitute current solution and as a result, improve the functionality of QPSS. The major disadvantages of the module of functionality responsible for interactive assignment of behaviors using the library of

behavior are: the user should have at least minimal skills in programming with Java and the highest detectable exception of bug is the name of the class where it appeared, but not precise line of code (because java reflection package is used). Major problematic issue with the module of functionality responsible for assignment of labels is the fact that it becomes tedious to assign the labels to the morphology of modular robot consisting of large number of modules. As a result, more effective way to get user's input should be devised in the future. The main problem of design in the module of functionality responsible for generic construction tools is direct usage of underlying engines of USSR, however the dependency is low. Saving-loading of XML files is missing the saving of the names of controllers.

In general, it is observable that the design of each module of functionality is following the best OO practices, nevertheless improvements should be considered in future. The main design patterns used in QPSS are: Abstract Factory, Factory, Strategy and Template Method. This enables to support each module of extendibility and as a result easier maintainability in future.

Chapter 9

9 Perspectives

"Lack of money is no obstacle. Lack of an idea is an obstacle."
— Ken Hakuta

Chapter Objectives
1. Briefly describe the future work for QPSS.
2. Present several future ideas, which could support QPSS and in this way ease the process of prototyping with modular robots.

9.1 Introduction

The intent of this chapter is to discuss the most urgent tasks, which should be addressed in QPSS in order for it to be ready for use and the future ideas, which could evolve QPSS to be more mature. The last is discussed from the several perspectives like: interactive construction of morphology, extendible framework and assignment of behaviors.

9.2 Future Work

In general, we observe that QPSS supports the user with quick prototyping of simulation scenarios. Nevertheless, we also agree that QPSS is a prototype. As every prototype, QPSS requires more time for further refining of the source code and functionality. The most important tasks to address in the near future should be:

- Testing and improvement of the source code responsible for search of existing modules. Sometimes modules overlap, because this functionality is failing. The main problem here is the values of search interval, which were found empirically. Instead, geometrical values from the design of the modular robots should be devised.
- Saving-loading the names of controllers assigned to the modules in the morphology of a modular robot in XML file.
- Modifying QPSS to be engine independent, like USSR is. For that all classes using jME engine, should be using it not directly, but through the pattern like for example Proxy.
- Improving the source code for supporting easier replacement of current solution for the tools of interactive construction of morphology with more generic solution.
- Redesign of GUI. The GUI could be more user friendly, by means of designing it according to the workflow during prototyping. For instance, the first tab of GUI represents the construction of modular robot's morphology and second assignment of behaviors. Each tab is a step and the user works with GUI in step by step fashion. When done the user starts the simulation and the tabs or elements unavailable during simulation are disabled.

9.3 Future Ideas

It is interesting observation that at the end of the project one realizes how much he/she learned and experienced. However, the most intriguing is that one gains new perspectives on the potential of the problem. Here new directions, ideas and possibilities arise. QPSS is

not an exception to this observation. In relation to that, lets us analyze what could be the next directions for QPSS. Of course, lets us keep the discussion as realistic as possible and within potentially useful solutions for modular robots. These are the following for each of the key features of QPSS:

- Interactive construction of modular robot's morphology, where are presented several ideas, which could improve QPSS for constructing modular robot's morphologies consisting of large number of modules (thousands).
- Extensible framework. Here is addressed the problem of identifying the parts of the framework to be modified and added during the process of supporting new modular robot.
- Assignment of behaviors. Here is presented a wide range of ideas for improving assignment of behaviors, because it seems that control is the most researchable issue in modular robots, nowadays. Consequently, requires new ideas and flexible tools to experiment with it.

9.3.1 Interactive Construction of Morphology

In the retrospect on QPSS we note that the design of the source code for the interactive construction of morphology is able to support implementation of new features (tools) simplifying the process of construction. Now these are the four major tools, however this is far away from being the limit. At the same time, we agree that these tools effectively cover the construction of modular robot's morphologies consisting of a maximum one hundred modules. Taking into the account that there is a need to simulate modular robot's morphologies consisting of much greater numbers of modules (thousands) new tools could be implemented for QPSS in the future.

First idea is the tool which will be able to replicate the morphologies, which were constructed previously. For instance, the user constructs ATRON car morphology, then uses the tool for replication and as result the copy of ATRON car is added. Knowing that replication of modular robots is one of the directions currently investigated by scientists. It would be interesting to support QPSS with possibility for replication in dynamic (running) state of simulation as well as in static. In case of dynamic state the modules would be relying on the sensory feedback and self-reconfiguration. In static state, it would remind simple "copy→ paste" functionality. The same idea could be supported with reading of physical robot's morphology in real life and recreating it in the simulation environment.

Second promising idea could be to support QPSS with possibility to define the macro of the tool. Meaning, that first the user defines the connectors on the module he/she would like to connect new modules to, next saves it as a macro and as a consequence new tool is available for use, which follows the sequence of module addition described in the macro, see Figure 9.1 for exemplification.

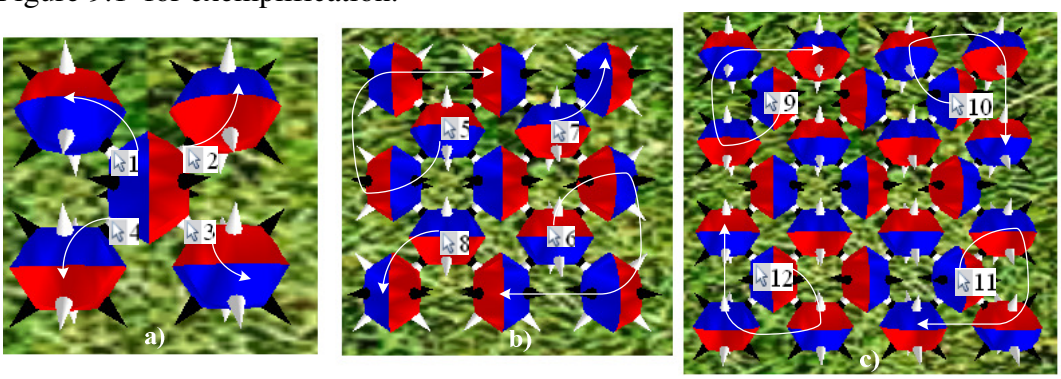


Figure 9.1 Example of macro definition and use on ATRON morphology: a) macro definition and b, c) usage of macro on morphology of ATRON

The figure above depicts the process of macro definition and usage in order to construct rectangular shape of ATRON morphology. First of all the user selects four connectors on ATRON module and defines it as a macro, see Figure 9.1 a, selections 1- 4. Next uses this macro to construct desired morphology, by simply selecting the modules to which four new modules should be added, see Figure 9.1 b, c, selections 5-12. Overlapping modules are not added. Here the user constructs the morphology consisting of twenty five modules and it takes only eight selections, not including the macro definition. It is necessary to mention that none of the currently supported tools in QPSS is able to achieve that. This is only one advantage of macros, another one is the fact that it will be very flexible to support a large number of tools in which are preferable for the user. Moreover, the user will be programming on higher level, the tools which are specific and most useful for the modular robot in interest.

Third is to support QPSS with more generic implementation of the four construction tools, which will allow modifying the morphology of a modular robot during the simulation and in static state of it. Moreover, add the tools which will enable the user to modify the appearance of modules in the sense of rotation of their components with respect to each other. This is already supported in QPSS, but they are in the experimental stage and require additional implementation in the USSR. Furthermore, enhance construction of morphology with the tools for grouping of modules, moving the morphology into desired position, addition of new modules by dragging from one of the connectors in different modular robot specific shapes (lines, circles and so on) and so on. The combination of these tools will enable the user to explore the flexibility of QPSS and construct morphologies which will be very close to the real life.

9.3.2 Extensible Framework

As it was previously mentioned the implementation of the four interactive construction tools is quite simple, heavily relies on the basic geometry and design of the modular robots. Some will probably argue that simplicity is also a limitation in a sense that the morphology of a modular robot cannot be changed after running the simulation. Author will agree with them, but on the other hand will argument why this implementation was preferable over more generic. There were three major reasons: 1) during managing the time of the project it was decided to approach the problem by means of simpler solution, so that there will be some time left for assignment of behaviors. As a result, both problems would be investigated and experimented with. 2) Missing implementation in USSR for storing different rotations of each component of the module. Now the rotation of each component is the same as rotation of the module itself. So there is no way for storing different rotations of each component. 3) Simpler solution will be easier to understand and as a consequence the extensibility of the framework should be easier to understand and implement too.

Actually the extendibility of the framework seems to be the major problem in OO framework design approach [79-81]. To be more precise, scientists investigate how to ease the process of identifying the parts of the source code which should be modified and added, during supporting new instances. This implies that the programmer is not familiar with the framework beforehand. Taking from here, the first idea would be to carry out the usability test for testing how difficult is it to support new modular robot in QPSS. Moreover, improve the design according to the received feedback. Next implement generic support for four interactive construction tools, carry out the usability test on this solution and improve on it according to the feedback. Then compare generic and current solutions, from the perspective of understandability and ease of adding support for new modular robots. Maybe even leave both implementations, so that the user could choose the one

he/she finds easiest to understand and utilize. On the other hand, we can completely forget about quality of the source code and involve auto-generation of the source code for supporting new modular robots. Here in a number of different ways we could involve GUI, and simulation environment, as input devices for identifying the modular robot specifics. After that auto-generate the source code supporting new modular robot.

9.3.3 Assignment of Behaviors

QPSS supports two approaches for assignment of behaviors and they are: interactive assignment using library of behaviors and using the labels for identification of modules in the morphology of modular robot.

First could be improved by sequencing behaviors in a form of queue. Meaning that the user implements the controllers he/she would like to assign to the modular robot. Later using QPSS test them on the modules in the morphology of a modular robot. Finally, for each module drugs and drops the behaviors into the queue of behaviors. Here the main parameter for transition from ending one behavior and activating another one is time line. Meaning, that the user defines when behavior should start and stop by means of varying the time. In this way, smooth transition from one behavior into another should be achieved. After that starts simulation and maybe correct the behavior of modular robot by dynamically adding the controller into the time line. This way it will be easier to coordinate execution of different controllers on different modules as well as observe their influence on morphology of a modular robot. This idea reminds the video which is decomposed into the frames and user is able to modify the frames, change their sequence timing and so on. For instance, in the case of ATRON car, the wheels can be assigned to rotate forward for five minutes and after that for five minutes back. As a result, ATRON car will be driving forward and then returning into initial position.

The labels could be supported with more efficient way for loading them into GUI. Currently the user types in each label one by one in the GUI in order to assign it later on. More flexible way could be to load them from the XML or any other type of the file into GUI. This could be done at runtime of QPSS. As a consequence, it would be easier for a user to choose appropriate label and assign it faster.

From programming point of view, the labels could be supported with a small DSL representing specific entities of a modular robot. This should be quite useful for expert users of USSR, because DSL would be representing abstract terms from the domain of modular robots. This should simplify association of entities from real-life to the programming language and consequently easier programming.

The last but not least, is the support for different control strategies discussed in chapter two. For instance, support QPSS with the easy to use role control. In this way, the users familiar with this type of control could be experimenting with it faster. Moreover, it can be supported with runtime sketching of controller. Here the user could stop simulation, modify controller and after that continue running simulation.

9.4 Summary

In this chapter we first presented several tasks, which should be fixed in near future in order for QPSS to be deliverable for use by the end users. Next we discussed different ideas which could improve QPSS and provide better support for its user. We agreed that the most promising of these are: introduction of macro support for defining different interactive construction tools in need for the user, simplifying the support of new modular robot, implementing DSL for working with labels and support of runtime sketching of controller.

10 References

10.1 Online References

[1] USSR homepage.
http://modular.mmmi.sdu.dk/wiki/USSR, nr. of references: none.
[2] Webots homepage. Last updated on 20 September 2008, at 05:52.
http://www.cyberbotics.com/products/webots/, nr. of references: 1.
[3] Molecube homepage. Last updated 20:33, 19 April 2008
http://128.253.249.235/cubes/index.php?title=Molecubes_For_Everyone, nr. of references: 8.
[4] Hints for writing the search paper.
http://owl.english.purdue.edu/workshops/hypertext/ResearchW/index.html
[5] Morphing production lines and ATRON homepage. Website was last modified on 10/4-2008. http://ecosoc.sdu.dk/coe/Morphing_production_lines, nr. of references: 8.
[6] M-Tran (I-II-III) homepage. Last updated 5 August 2008
http://unit.aist.go.jp/is/dsysd/mtran3/,
[7] SuperBot homepage. Last updated January 2008
http://www.isi.edu/robots/superbot.htm
[8] Catom homepage.
http://www.cs.cmu.edu/~claytronics/
[9] Robotics in general. Last updated on 29 August 2008, at 15:20,
http://en.wikipedia.org/wiki/Robotics#cite_note-2, nr. of references: 58.
[10] Description of Three Laws of Robotics. Last updated on 27 August 2008, at 18:03,
http://en.wikipedia.org/wiki/Three_Laws_of_Robotics, nr. of references: 35.
[11] Industrial robot in general. Last updated on 5 September 2008, at 17:58, http://en.wikipedia.org/wiki/Industrial_robot, nr. of references: 1.
[12] Mobile robot in general. Last updated on 3 September 2008, at 18:43,
http://en.wikipedia.org/wiki/Mobile_robot, nr. of references: 3.
[13] Simulation in general. Last updated on 22 September 2008, at 13:06
http://en.wikipedia.org/wiki/Simulator#cite_note-environment-3, nr. of references: 14.
[14] Introduction into OpenGL. Last updated on 20 September 2008, at 15:00
http://en.wikipedia.org/wiki/OpenGL, nr. of references: 13.
[15] jMonkey Engine in general. Last updated on 15 September 2008, at 04:13.
http://en.wikipedia.org/wiki/JMonkey_Engine, nr. of references: 3.
[16] Introduction into ODE. Last updated on 11 August 2008, at 18:28
http://en.wikipedia.org/wiki/Open_Dynamics_Engine, nr. of references: none.
[17] OpenAL in general. Last updated on 16 September 2008, at 09:43.
http://en.wikipedia.org/wiki/OpenAL, nr. of references: none.
[18] Webots on wikipedia. Last updated on 27 August 2008, at 17:23
http://en.wikipedia.org/wiki/Webots, nr. of references: 7.
[19] DPRsim homepage.
http://www.pittsburgh.intel-research.net/dprweb/index.html
[20] Gazebo homepage. Last updated on 02 November 2006, 21:08:26
http://playerstage.sourceforge.net/index.php?src=gazebo, nr. of references: none.
[21] Gazebo on wikipedia. Last updated 14:53, 5 August 2008.
http://playerstage.sourceforge.net/wiki/Gazebo, nr. of references: 2.
[22] Microsoft Robotics Studio homepage. Last updated on 2008.
http://msdn.microsoft.com/en-us/robotics/default.aspx, nr. of references: none.
[23] Marilou homepage. Last updated on April 10 2008.

http://www.anykode.com/index.php, nr. of references: 7.
[24] Marilou on wikipedia. Last updated on 4 August 2008, at 20:18. http://en.wikipedia.org/wiki/AnyKode_Marilou, nr. of references: none.
[25] MobotSim homepage. http://www.mobotsoft.com/index.php?option=com_content&view=article&id=44:mobotsim&catid=34:mobotsim&Itemid=54
[26] Explanation of term taxonomy. Last updated on 17 October 2008, at 15:33. http://en.wikipedia.org/wiki/Taxonomy, nr. of references: 2.
[27] SimRobot homepage. Last updated: August 5, 2005. http://www.informatik.uni-bremen.de/simrobot/index_e.htm#12006
[28] Introduction into usability testing. Last updated: on 14 April 2009, at 20:55. http://en.wikipedia.org/wiki/Usability_test
[29] Learn about Usability Testing. http://www.usability.gov/refine/learnusa.html
[30] Step-by-Step Usability Guide. http://www.usability.gov/index.html

10.2 Published and Draft References

[31] David Christensen, Ulrik Pagh Schultz, David Brandt, and Kasper Stoy. *A Unified Simulator for Self-Reconfigurable Robots*. Modular Robots @ MMMI, Maersk Institute, University of Southern Denmark, 2008, 6 pages
[32] M. Bordignon, K. Støy, D. J. Christensen and U. P. Schultz, *Towards Interactive Programming of Modular Robots*, in Proceedings of IROS 2008 Workshop on Self-Reconfigurable Robots/Systems and Applications. September 22 nd, 2008. 6 pages.
[33] Kasper Støy, David Brandt, David J.Christensen. Book title: *An Introduction to Self-Reconfigurable Robots* (Draft Version). Chapters: 1-3. Maersk Mc-Kinney Moller Institute for Production Technology, University of Southern Denmark, 2008, 314 pages.
[34] John Hallam, *AI Architectures: module notes*. September 6 2005, 33 pages, State: Not published.
[35] R. A. Brooks. *Intelligence without reason.* In Proceedings of 12th International Joint Conference on Artificial Intelligence, pages 569-595, August 1991. Cited by: 1526.
[36] Ronald C. Arkin. Book title: *Behaviour-Based Robotics*. Chapters:1-2. Third printing, 2000, © 1998 Massachusetts Institute of Techology, 491 pages, ISBN: 0-262-01165-4.
[37] T.Fukuda and S.Nakagawa. *Dynamically Reconfigurable Robotic System.* In Proc., 1988 the IEEE Int. Conf. on Robotics & Automation, volume 3, pages 1581–1586, Philadelphia, PA, USA, 1988. Cited by: 96.
[38] Esben H. Østergaard. Ph.D thesis title: *Distributed Control of the ATRON Self-Reconfigurable Robot.* Maersk Mc-Kinney Moller Institute for Production Technology, University of Southern Denmark, 21st October 2004, 240 pages.
[39] Kasper Støy, A. Lyder, R.M.F. Garcia and D. Christensen. *Hierarchical Robots.* MMMI, SDU. 4 pages, Date. State: Not published.
[40] David Johan Christensen, Ph.D thesis title: *Elements of Autonomous Self-Reconfigurable Robots.* Chapters:1-10. MMMI, University of Southern Denmark, June 2008, 224 pages. State: Not published.
[41] Morten W. Jorgensen, Esben H. Ostergaard, Henrik H. Lund. *Modular ATRON: Modules for a self-reconfigurable robot.* In Proceedings of IEEE/RSJ International Conference on Robots and Systems, (IROS), Sendai, Japan, Pages 2068-2073, Sep. 30-Oct.2, 2004. Cited by: 67.
[42] Henrik H. Lund, H.H, Rasmus L. Larsen, Esben H. Østergaard. *Distributed Control in*

Self-reconfigurable Robots. In Proceedings of ICES'03, The 5th International Conference on Evolvable Systems: From Biology to Hardware, Trondheim, Norway, Pages 296-307, March 17-20, 2003. Cited by: 8.

[43] Esben H. Østergaard and Henrik H. Lund. *Evolving Control for Modular Robotic Units.* In Proceedings of CIRA'03, IEEE International Symposium on Computational Intelligence in Robotics and Automation, Kobe, Japan, Pages 886-892, July 16-20, 2003. Cited by: 8.

[44] Esben H. Østergaard and Henrik H. Lund. *Distributed Cluster Walk for the ATRON Self-reconfigurable robot.* In Proceedings of the 8th Conference on Intelligent Autonomous Systems (IAS-8), Amsterdam, Pages 291-298, March 10-13, 2004. Cited by: 16.

[45] David J. Christensen, Esben H. Østergaard, Henrik H. Lund. *Metamodule Control for the ATRON Self-Reconfigurable Robotic System.* In Proceedings of the 8th Conference on Intelligent Autonomous Systems (IAS-8), Amsterdam, Pages 291-298, March 10-13, 2004. Cited by: 8.

[46] David Brandt and Esben H. Østergaard. *Behaviour Subdivision and Generalization of Rules in Rule Based Control of the ATRON Self-Reconfigurable Robot*. In Proceedings of the International Symposium on Robotics and Automation (ISRA), Queretaro, Mexico, Pages 67-74, Sep. 2004. Cited by: 4.

[47] Haruhisa Kurokawa, Kohji Tomita, Akiya Kamimura, Shigeru Kokaji, Takashi Hasuo, Satoshi Murata. *Distributed Self-Reconfiguration of M-Tran III Modular Robotic System.* International Journal of Robotics Research**,** Volume 27, Issue 3-4 (March 2008), Pages 373-386.

[48] Esben H. Østergaard, Kohji Tomita and Haruhisa Kurokawa. *Distributed Metamorphosis of Regular M-TRAN Structures.* In Proceedings of the 7th International Symposium on Distributed Autonomous Robotic Systems (DARS), Toulose, France, Pages 161-170, June 22-25, 2004. Cited by: 3.

[49] Wei-Min Shen, Harris C.H. Chiu, Mike Rubenstein, Behnam Salemi. *Rolling and Climbing by the Multi-functional SuperBot Reconfigurable Robotic System.* In Proc. STAIF 2008, Space Technology Intl. Forum, Albuquerque, New Mexico, February 2008, 10 pages.

[50] Wein-Min Shen, Maks Krivokon, Harris Chiu, Jacob Everist, Michael Rubestein, Jagadesh Venkatesh, *Multimode Locomotion via SuperBot Robot*. Proceedings of the 2006 IEEE International Conference on Robotics and Automation, Orlando, Florida-May 2006. pages 2552-2557. Cited by: 8.

[51] Preethi Bhat, James Kuffner, Seth Goldstein, Siddhartha Srinivasa. *Hierarchical Motion Planning for Self-reconfigurable Modular Robots. In 2006 IEEE/RSJ Interna*tional Confernce on Intelligent Robots and Systems (IROS), October, 2006

[52] Victor Zykov, Phelps Williams, Nicolas Lassabe and Hod Lipson, *Molecubes Extended: Diversifying Capabilities of Open-Source Modular Robotics*. In Proceedings of IROS 2008 Workshop on Self-Reconfigurable Robots/Systems and Applications. September 22 nd, 2008. 12 pages.

[53] A. Lyder, R.F.M. Garcia, K. Stoy, *Mechanical Design of Odin, an Extendable Heterogeneous Deformable Modular Robot*, in Proceedings of the IEEE/RJS 2008, IROS2008, 6 pages.

[54] R.F.M. Garcia, A. Lyder, David J. Christensen and K. Stoy. *Reusable Electronics and Adaptable Communication as Implemented in the Odin Modular Robot (Draft).* 2008, 7 pages.

[55] M. Yim, W. M. Shen, B. Salemi, D. L. Rus, M. Moll, H. Lipson, E. Klavins, G. S. Chirikjian. *Modular Self-Reconfigurable Robot Systems [Grand Challenges of*

Robotics] Robotics & Automation Magazine, IEEE, Volume 14, Issue 1, March 2007 Page(s):43 –52. Cited by: 22.

[56] Murata, S. Kurokawa, H. *Self-Reconfigurable Robot: Shape-Changing Cellular Robots Can Exceed Conventional Robot Flexibility*. Robotics & Automation Magazine, IEEE Volume 14, Issue 1, March 2007 Page(s):71 – 78.

[57] Akiya Kamimura, Haruhisa Kurokawa, EiichiYoshida, Satoshi Murata, Kohji Tomita and Shigeru Kokaji. *Automatic Locomotion Design and Experiments for a Modular Robotic System.* IEEE/ASME Transactions on Mechatronics, Vol. 10. No. 3, pages 314- 324 June 2005. Cited by: 33, according to Google Scholar.

[58] G. C. Pettinaro, I. W. Kwee, L.M. Gambardella. *Definition, Implementation, and Calibration of the Swarmbot3D Simulator.* Technical Report No. IDSIA-21-03. December 2003, 79 pages. Cited by: 1.

[59] Selim Temizer. *The State of the Art and the Future of Modeling and Simulation Systems***.** Journal of Aeronautics and Space Technologies. January 2007, Volume 3, Number 1, Pages 41-50.

[60] Bjørn Grønbæk & Brain Horn. Preliminary Investigation of the Thesis Problem Domain. Title: *Investigation of Elements for an Automated Test Tool for Virtual Environments.* MMMI, SDU. June 6, 2008, 137 pages.

[61] Stanislaw Raczynski. Book title: *Modelling and Simulation. The computer science of illusion.* Chapters: 1-2. Universidad Pan Americana, Mexico. RSP Series in Computer Simulation and Modeling. 2006. 222 pages. ISBN-13 978-0-470-03017-2, ISBN-10 0-470-03017-8.

[62] Satoshi Murata, Eiichi Yoshida, Akiya Kamimura, Haruhisa Kurokawa, Kohji Tomita and Shigeru Kokaji. *M-TRAN: Self-Reconfigurable Modular.* IEEE/ASME transactions on mechatronics, VOL.7, NO.4, DECEMBER 2002. Cited by: 140.

[63] Akiya Kamimura , Eiichi Yoshida , Satoshi Murata , Haruhisa Kurokawa, Kohji Tomita and Shigeru Kokaji. *A Self-Reconfigurable Modular Robot (MTRAN) – Hardware and Motion Generation Software –*. DARS 2002, 10 pages Cited by: 1.

[64] Mark Moll, Peter Will, Maks Krivokon, and Wei-Min Shen. *Distributed Control of the Center of Mass of a Modular Robot.* Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on Volume , Issue , Oct. 2006 Page(s): 4710 – 4715. Cited by: 2.

[65] Olivier Michel. *Webots: Professional Mobile Robot Simulation*. International Journal of Advanced Robotic Systems, Volume 1 Number 1 (2004), ISSN 1729-8806, pp. 39-42. Cited by: 28.

[66] Laszlo Vajta and Tamas Juhasz. *3D simulation in the advanced robotics design, test and control.* Budapest Univestity of Technology and Economics. Faculty of Electrical Eng. And Informatics. April 2005, 13 pages.

[67] A. Sproewitz, M. Asadpour, Y. Bourquin and A. J. Ijspeert. *An active connection mechanism for modular self-reconfigurable robotic systems based on physical latching.* School of Computer and Communication Sciences at EPFL, Swiss Federal Institute of Technology, CH1015 Lausanne, Switzerland. 2008. 6 pages.

[68] Cyberbotics. Webots Reference Manual. Chapters: 1-2. August 4, 2008, 208 pages.

[69] Nathan Koenig, Andrew Howard. *Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator.* Proceedings 01 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems. September 28 - Octoberr 2, 2004, Sendai, Japan. Pages 2149-2154. Cited by: 48.

[70] Brian P. Gerkey, Richard T. Vaughan, Andrew Howard. *The Player/StageProject: Tools for Multi-Robot and Distributed Sensor Systems.* In Proceedings of the International Conference on Advanced Robotics (ICAR 2003). Pages 317-323,

Coimbra, Portugal, June 30 - July 3, 2003. Cited by: 303.
[71] Gaurav S. Sukhatmd, Kasper Stoy, Andrew Howard, Maja J. Matarid. *Most Valuable Player: A Robot Device Server for Distributed Control.* In Proceedings of the International Conference on Intelligent Robots and Systems, 2001, IEEE/RSJ, Vol. 3 (2001), pp. 1226-1231 vol.3. Cited by: 215.
[72] Matthew Turk. *Moving from GUIs to PUIs*. Microsoft Research, Microsoft Corporation, One Microsoft Way, Redmond, WA 98052, Symposium on Intelligent Information Media, Tokyo, Japan, December 1998. 7 pages. Cited by: 4.
[73] Jeffrey S. Pierce. Dissertation title: *Expanding the Interaction Lexicon for 3D Graphics*. Chapters: 1-2. School of Comsputer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213, November 2001, 136 pages. Cited by: 5.
[74] Qing Xie. Dissertation title: *Developing Cost-Effective Model-Based Techniques for GUI Testing.* Chapters: 1-2. Graduate School of the University of Maryland, College Park, 2006, 153 pages. Cited by: 5.
[75] Farid Ben Hajji, Erik Dybner. *3D Graphical User Interfaces.* Department of Computer and Systems Sciences Stockholm University and The Royal Institute of Technology. July 1999, 50 pages. Cited by: 3.
[76] Edited by: Andrew Sears and Julie A. Jacko. *The Human-Computer Interaction Handbook. Fundamentals, Evolving Technologies, and Emerging Applications*. Second Edition. Chapters: 1-2. 2008, 1386 pages. ISBN-13: 978-0-8058-5870-9 (Hardcover), ISBN: 0-8058- 5870-9.
[77] U.P. Schultz, M. Bordignon, D. Christensen, and K. Støy. *Spatial Computing with Labels.* Position Paper- Spatial Computing Workshop at IEEE SASO. 2008, 6 pages
[78] Martin Fowler. *UML Distilled. Third Edition. A Brief Guide to the Standard Object Modeling Language*. ISBN 0-321-19368-7. 175 pages.
[79] Jan Bosch, Peter Molin, Michael Mattsson, Perolof Bengtsson. *Object-Oriented Frameworks – Problems & Experiences.* University of Karlskrona/Ronneby. Department of Computer Science and Business Administration. S-372 25 Ronneby, Sweden, 2001. 18 pages.
[80] H. Ben-Abdallah, N. Bouassida, F. Gargouri and A. Ben-Hamadou. *A UML based Framework Design Method.* Published by ETH Zurich, Chair of Software Engineering, 8 pages, 2004.
[81] J. Van Gurp and J. Bosch. *Design, Implementation and Evolution of Object Oriented Frameworks: Concepts and Guidelines.* Software Practice and Experience. 2000. 25 pages.
[82] Erich Gamma, Richard Helm, Ralph Joghnson and John Vlissides. *Design Patterns. Elements of Reusable Object-Oriented Software.* Addison-Wesley. ISBN 0-201-63361-2. September 2004, 395 pages.
[83] James W. Cooper. *The Design Patterns. Java Companion.* Addison-Wesley Design Patterns Companion. October 2, 1998, 218 pages.

Appendixes

Appendix A: Additional List of Literature

During the project several other sources of information were read and analyzed, but were not referred directly in the report. These are:

1. Introduction into PhysX. Last updated on 19 September 2008, at 20:17. http://en.wikipedia.org/wiki/PhysX, nr. of references: 17.
2. Description of OGRE. Last updated on 16 September 2008, at 21:08. http://en.wikipedia.org/wiki/OGRE, nr. of references: 2.
3. Introduction into GLUT. Last updated on 20 September 2008, at 05:52. http://en.wikipedia.org/wiki/OpenGL_Utility_Toolkit, Nr. of references: none.
4. Introduction into Artificial Intelligence. Website was last modified on 6 September http://en.wikipedia.org/wiki/Artificial_Intelligence#General_intelligence, nr. of references: 158.
5. D.Brandt, J.C.Larsen, D.J.Christensen, R.F.M.Garcia, D.Shaikh, U.P.Schultz, and K.Stoy, *Flexible, FPGA-Based Electronics for Modular Robots,* in Proceedings of IROS 2008 Workshop on Self-Reconfigurable Robots/Systems and Applications. September 22 nd, 2008. 5 pages.
6. Anders Lyhne Christensen, Rehan O'Grady, Marco Dorigo, *Towards Adaptive Morphogenesis in Self-Assembling Robots,* in Proceedings of IROS 2008 Workshop on Self-Reconfigurable Robots/Systems and Applications. September 22 nd, 2008. 4 pages.
7. Serge Kernbach, Leonardo Ricotti, Jens Liedke, Paolo Corradi, Mathias Rothermel, *Study of Macroscopic Morphological Features of Symbiotic Robotic Organisms*, in Proceedings of IROS 2008 Workshop on Self-Reconfigurable Robots/Systems and Applications. September 22 nd, 2008. 8 pages.
8. Maja J.Mataric, Gaurav S. Sukhatme, Esben H. Østergaard. *Multi-Robot Task Allocation in the Uncertain Environments*. Autonomous Robots vol. 14, Pages 255-263, Kluwer Academic Publishers, 2003. Cited by: 4.
9. Khatib, O. Brock, K. S. Chang, F. Conti, D. Ruspini, L. Sentis. *Robotics and Interactive Simulation.* Robots: intelligence, versatility, adaptivity. 2002, Pages: 46 – 51 . Cited by: 35.

Appendix B: Other Simulators for Mobile Robots

A number of other simulators for mobile robots were considered during the project, however where found to be not documented well enough, see Table B.1.

Table B.1 Other simulators for mobile robots

Name of simulator	Abbreviation explained	2D or 3D	References
MobotSim	Mobile Robot Simulator	2D	[84]
SimRobot	-	3D	[85], [86]
USARSim	Unified System for Automation and Robot Simulation	3D	[87-90]
EyeSim	EyeBot SIMulator	2D&3D	[91 [93]
Swarmbot3D	-	3D	[94], [95]
AMORsim	Autonomous MObile Robots SIMulator	2D	[96]
Ubersim	-	3D	[97], [98]
COROSIM	-	3D	[99]

In the table above is summarized general information about other simulators for mobile robotics considered during the project. This information can be used as inspiration for future researchers of the same topic. Additionally the list of literature referred in the table is presented below.

[84] MobotSim homepage. http://www.mobotsoft.com/index.php?option=com_content&view=article&id=44:mobotsim&catid=34:mobotsim&Itemid=54

[85] SimRobot homepage. Last updated: August 5, 2005. http://www.informatik.uni-bremen.de/simrobot/index_e.htm#12006

[86] T. Laue, K. Spiess, T. Röfer (2006). *SimRobot - A General Physical Robot Simulator and Its Application in RoboCup.* In A. Bredenfeld, A. Jacoff, I. Noda, Y. Takahashi (Eds.), RoboCup 2005: Robot Soccer World Cup IX, No. 4020, pp. 173–183, Lecture Notes in Artificial Intelligence. Cited by: 25.

[87] USARsim homepage. Last updated 11/06/2007. http://usarsim.sourceforge.net/

[88] S. Carpin, M. Lewis, J. Wang, S. Balakirsky, C. Scrapper (2007). *USARSim: a robot simulator for research and education.* Proceedings of the 2007 IEEE Conference on Robotics and Automation. 6 pages. Cited by: 12.

[89] S. Carpin, M. Lewis, J. Wang, S. Balakirsky, C. Scrapper (2007). *Bridging the gap between simulation and reality in urban search and rescue.* Robocup 2006: Robot Soccer World Cup X, LNAI Vol. 4434 Springer, pp. 1–12.

[90] M. Zaratti, M. Fratarcangeli, L. Iocchi (2007). *A 3D Simulator of Multiple Legged Robots based on USARSim*. Robocup 2006: Robot Soccer World Cup X, Springer, LNAI Vol. 4434 Springer, pp. 13–24.

[91] EyeSim homepage. http://robotics.ee.uwa.edu.au/eyebot/doc/sim/sim.html

[92] Thomas Bräunl. *The EyeSim Mobile Robot Simulator*. CITR, The University of Auckland, New Zealand. 2000, 9 pages. Cited by: 7.

[93] Andreas Koestler, Thomas Bräunl. *Mobile Robot Simulation with Realistic Error Models.* International Conference on Autonomous Robots and Agents, ICARA 2004, Dec. 2004, Palmerston North. New Zealand, pp. 46-51 (6). Cited by: 7.

[94] G. C. Pettinaro, I. W. Kwee, L.M. Gambardella. *Acceleration of 3D Dynamics Simulation of S-Bot Mobile Robots using Multi-Level Model Switching*. Technical Report No. IDSIA-20-03 November 28, 2003. 12 pages. Cited by: 3.

[95] G. C. Pettinaro, I. W. Kwee, L.M. Gambardella. *Definition, Implementation, and Calibration of the Swarmbot3D Simulator.* Technical Report No. IDSIA-21-03. December 2003, 79 pages. Cited by: 1.

[96] Toni Petrinic, Edouard Ivanjko, Ivan Petrovic. *AMORsim - A Mobile Robot Simulator for Matlab.* Proceedings of 15th International Workshop on Robotics in Alpe-Adria-Danube Region, June 15-17, Balatonfüred, Hungary, 2006. 6 pages.

[97] UberSim homepage. http://www.cs.cmu.edu/~robosoccer/ubersim/

[98] Jared Go, Brett Browning, Manuela Veloso *Accurate and Flexible Simulation for Dynamic, Vision-Centric Robots.* Autonomous Agents and Multiagent Systems, 2004. AAMAS 2004. Proceedings of the Third International Joint Conference. Pages 1388-1389. Cited by: 12.

[99] Eric Colon, Kristel Verbiest. *3D mission oriented simulation*. Royal Military School. 2007, 15 pages. Cited by: 12.

Appendix C: User Guide

In order to begin working with QPSS (in USSR) in the Eclipse, do the following:

1. Locate simulation file called "BuilderMultiRobotSimulation.java in the directory "ussr\src\ussr.builder\BuilderMultiRobotSimulation.java". (QPSS can be also used with any other simulation class for any supported modular robots, like ATRON, M-Tran and Odin. However, then the bugs should be expected). But this is not likely.
2. Start simulation for above file. The simulation should be in static state (paused state), meaning that you should not press "P" button on keyboard. At least not yet.
3. Press "Q" button on keyboard and wait a bit. On slow machines the "Quick Prototyping of Simulation Scenarios" window is quite slow to respond. If the window does not appear press "Q" and hold it down for a bit again.
4. In the appeared QPSS window (see Figure C.1) choose one of the buttons (pickers, also called selection tools) in the toolbars of GUI. You can identify the name of the toolbar by going to "View→ Toolbars→…"

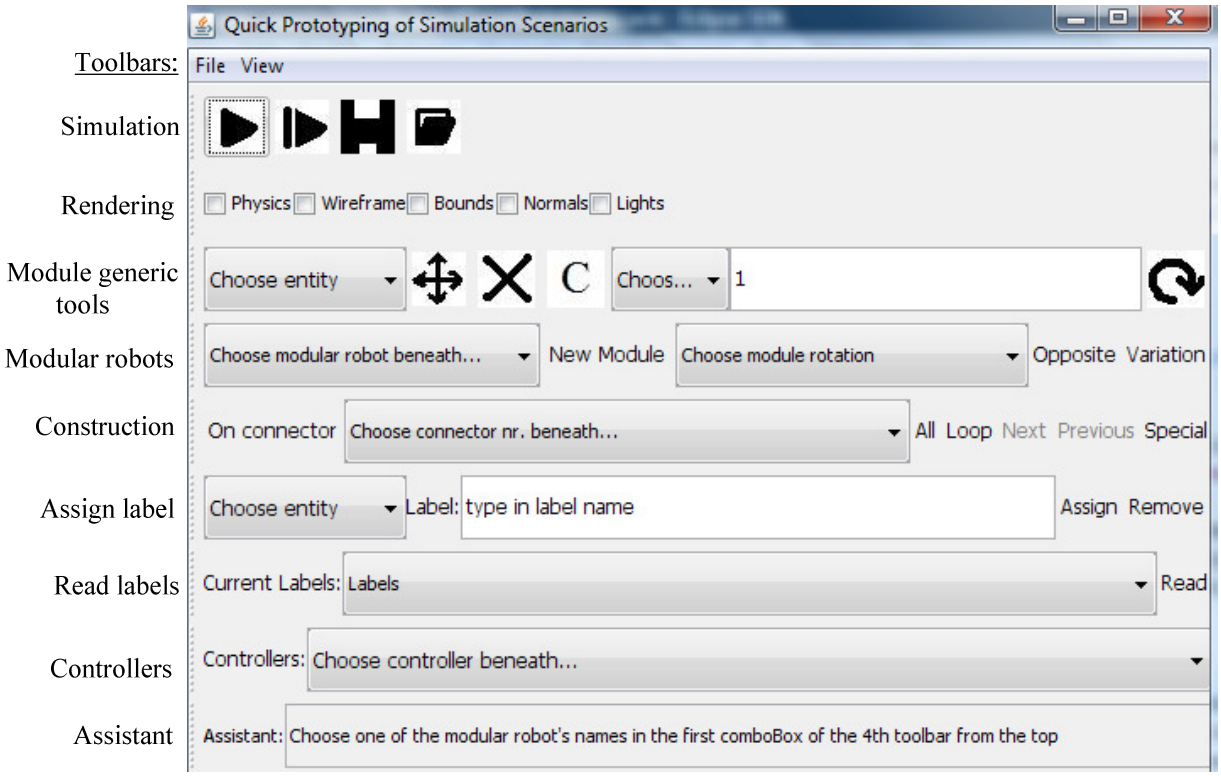


Figure C.1 The GUI of QPSS

The Table C.1 beneath contains description of each toolbar from the top in the figure above. Moreover, each component (button, text field and so on) of the toolbar is explained in details.

Table C.1 Description of toolbars and their components

ToolBar	Button name	Explanation
Simulation	Pause/Play	Starts or pauses simulation.
	StebByStep	Runs simulation with single execution step.
	Save	Saves the data about modular robot in XML file (slow to respond, wait for it). Should be used in static state of simulation.

	Open	Loads the data about modular robot from XML file into simulation in static state (slow to respond, wait for it).
Rendering	Physics	Starts or stops rendering physics.
	Wireframe	Starts or stops rendering physics.
	Bounds	Starts or stops rendering bounds.
	Normals	Starts or stops rendering normals.
	Lights	Starts or stops rendering lights.
Module generic tools	ComboBox entities to choose	For choosing the entities to manipulate (rotate or move).
	Move	It is a selection tool, where after selecting the module with left side of the mouse it is moved with movement of the mouse to desired location; Is added as additional support and its main purpose is to move default construction modules. The icon on the button reminds the cross.
	Delete	It is a selection tool, where after selecting the module with the left side of the mouse it is deleted (removed) from simulation environment; The icon on the button reminds the cross rotated 45 degrees.
	C	It is a selection tool, where after selecting the module with left side of the mouse its connectors are colored with color coding. The format is Connector-Color: 0-Black, 1-Red, 2-Cyan, 3-Grey, 4-Green, 5-Magenta, 6-Orange, 7-Pink, 8-Blue, 9-White, 10-Yellow, 11-Light Grey.
	Cartesian axes (in ComboBox (x,y and z))	Are used together to rotate components of the modules or module with specific angle. First choose one of the Cartesian axes in ComboBox (for example: "x"), after that enter the angle in degrees in the TextField (let say 90) and at last select "Rotate" icon. After all that select one of the components of the module in simulation environment. This tool is experimental and can be used for small test adornments.
	TextField for angle of rotation	
	Rotate	
Modular robots	ComboBox with Modular robots names	Choose the modular robot you would like to work with from the list of supported ones.
	Default	Adds default construction module of modular robot. If in previous ComboBox you chosen "ATRON", then ATRON module will be added into simulation at default position.
	ComboBox with the names of rotations	Rotates the default module with standard rotations of the modular robot chosen the first comboBox.

	Opposite	Rotates the module selected in simulation environment with opposite rotation to current rotation of the module.
	Variation	It is a selection tool for additional properties of modular robot modules. For example: in case of Odin replaces OdinMuscle with other types of modules. In case of MTRAN rotates module 90 degrees around axes.
Construction	On Connnector	It is a selection tool, where you can just click on connector of the module with the left side of the mouse and next module will be added to selected connector.
	ComboBox with numbers of connectors	It is a selection tool, where after choosing the connector number (in ComboBox) where new module will be added, after that click on one of the modules in simulation environment with the left side of the mouse. As a result, new module will be added to the chosen connector on selected module.
	All	It is a selection tool, where you just click on the module with the left side of the mouse and all possible modules will be added to the selected module connectors, except the ones which are already occupied.
	Loop	"Loop"- is a selection tool, where you just click on the module with the left side of the mouse and later (wait a bit) press on buttons "Next" or "Previous". As a result, new module will be moved from one connector to another with increasing number of connector ("Next") and decreasing number of connector ("Previous").
	Next	
	Back	
	Special	It is an experimental tool (do not use it).
Assign Label	ComboBox with the names of entities	It is for choosing the entity to assign the label to. Currently are supported: Module and Connector.
	Text field with the title :"type in label name"	It is for typing in the name of the label. The format for entering several labels is for example: label1,label2, label3 . Notice comma is used as separation sign. In this way it is possible to assign several labels at a time. The single label is simply: label1
	Assign	It is for actually assigning the label to the desired entity. Execute two actions above, after that press this button and then entity in simulation environment.
	Remove	It is for removing specific label from the labels assigned to the entity. First type in the label name in the text field (mentioned above) and then press "remove", after that select the entity

		in simulation environment for which you would like to remove specified label. NOTICE: There should not be any comma for one label.
Read Labels	comboBox with title "Current Labels"	It is for displaying the labels assigned to the entity. Just expand the "comboBox" and see which labels are assigned. You can also choose one of them and then it will be displayed in the textField (in the toolbar above)). After that you can reassign it to any other entity. In order to fill in this combo box first select “Read” button (this is explained next).
	Read	It is for reading the labels assigned to the entity. Just press this button and then select the entity in simulation environment. As a result the labels assigned to this entity will be displayed in both comboBox and textField (toolBar above).
Controllers	ComboBox with the names of the classes(controllers)	It is for choosing the controller you would like assign to the module. Remember the controllers you created should be placed in the package named as "ussr.builder.controllerReassignmentTool" inherit from "ControllerStrategy. java" and have activate() method implemented. Then your new controller will be working and displayed in the current comboBox.
Assistant	This toolbar simply gives the hints what to do next. In a way it is simple version of help. Most of the time it is reliable. However, do not trust it blindly.	

5. Using the tools introduced in the table above construct desired morphology (shape) of a modular robot. Easiest way, is to start from toolBar called "Modular robots" by choosing modular robot name in the comboBox, after that choosing appropriate rotation in rotations comboBox, later shift to the toolbar called "Construction". By using one of the four tools construct the morphology of a modular robot. There is also toolBar called Assistant, which displays the hints describing what to do next. Follow it, but do not trust it one hundred percent. Next you can assign labels to modules and so on, depending on the preference. When done with morphology you can save the data about your modular robot in XML file (toolbar called “Simulation”). After that, press "Play" in the top-left corner of GUI. Now you can interactively assign the controllers to the modules by using the toolbar called “Controllers”. Just select the controller name and after that select the module in the simulation environment. Assignment of controllers can be also done before running the simulation (in static state).

C1 Step by Step Example for ATRON Car Configuration

1. Locate the simulation file called "BuilderMultiRobotSimulation.java in the directory “ussr\src\ussr.builder\BuilderMultiRobotSimulation.java”.

2. Start simulation for above file. The simulation should be in static state (paused state), meaning that you should not press "P" button on keyboard. At least not yet.
3. Press "Q" button on the keyboard and wait a bit. On slow machines the "Quick Prototyping of Simulation Scenarios" window is quite slow to respond. If the window does not appear press “Q” and hold it down for a bit again.
4. In the appeared GUI window locate the combo box with modular robot names in the toolbar called “Modular robots” (annotated with “choose modular robot”). Choose ATRON in this combo box. This will add so-called default construction module (see Figure C.2 a, b, and selection 1). In the figure the process of new module addition is emphasized with the white arrow. The arrow tip with a number indicates the sequence of selection. For concision, only part of the GUI and simulation environment are shown.

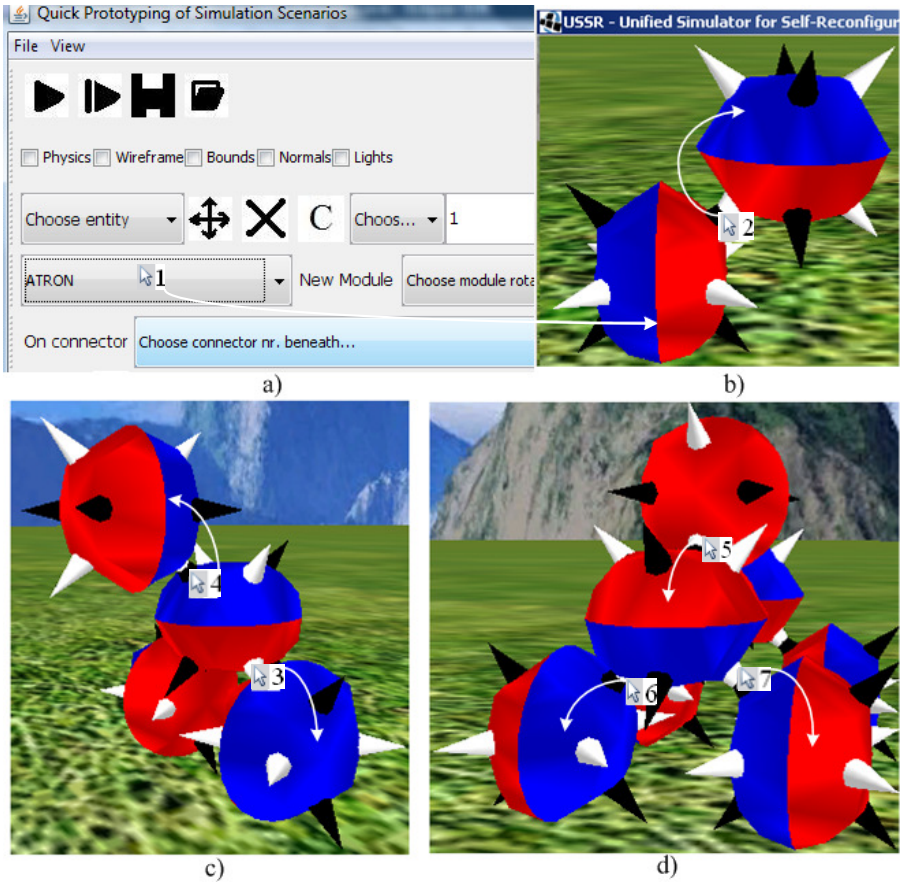


Figure C.2 Step by step example for interactive construction of ATRON car configuration

5. After that select desirable connector on the previously added default construction module in the simulation environment. This will place the new ATRON module with respect to selected connector (black and white cones). Sequent selections of connectors on different modules should result in ATRON car configuration (see Figure C.2 b, c, d and selections 2-7).
6. When ATRON car configuration is ready (see Figure C.3 b) press button called “Run” in the left-top corner of GUI, see Figure C.3 a and selection 1). Here grey arrows indicate transition from static state of simulation into dynamic. In other words, from modeling of modular robot into running state of simulation.

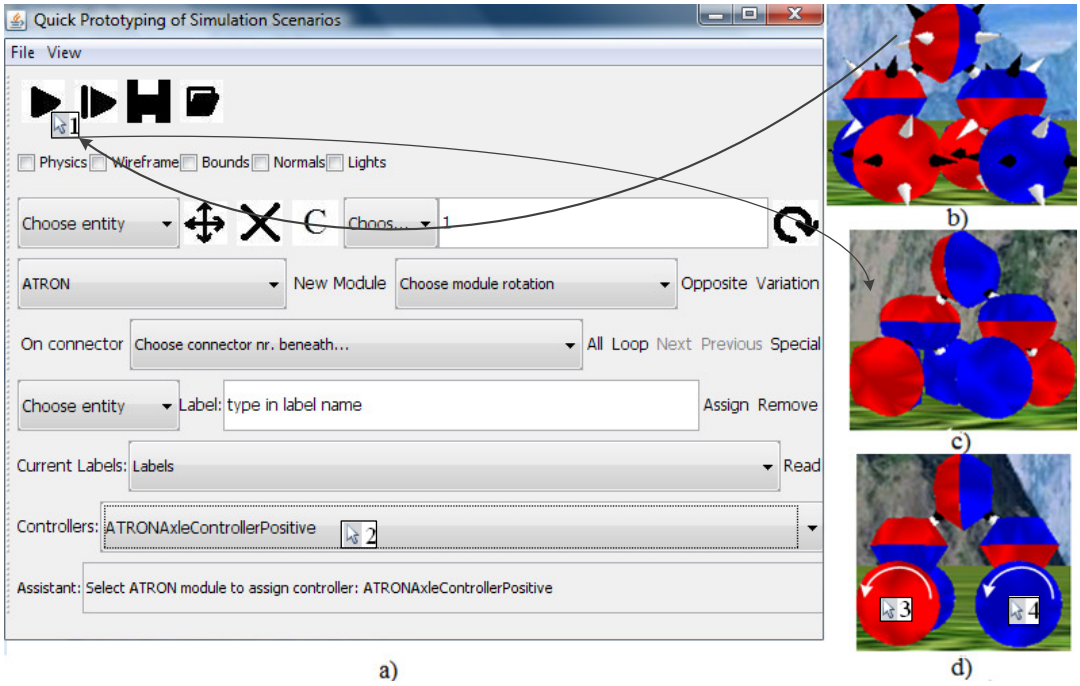


Figure C.3 Step by step example for assignment of behaviors to wheel modules

7. Next choose the controller called "ATRONWheelControllerPositive" in the toolbar named as "Controllers", see Figure C.3 and selection 2.
8. Now shift to simulation environment and select the wheels of ATRON car, see Figure C.3 selections 3, 4.
9. Repeat selections 3 and 4 above, but choose the controller named as "ATRONWheelControllerNegative" and select the wheels on the opposite side of ATRON car configuration.
10. The result should be ATRON car configuration driving forward or back (depends from where to look at it).

Appendix D: Questionnaire for Usability Tests and Results

Participants: 4 experts (including moderates) and 1 beginner

1. Is toolbar called "Assistant" helpful during the first use of QPSS?
   a) Yes
   b) No
   c) Not sure yet
   Statistics (experts and moderates): 3 answered Yes, 1 – Not sure yet.
   Statistics (beginner): Yes.
2. How difficult was it to understand (guess) the purpose of tool named as "On Connector" and use it during construction of modular robot's morphology in your case?
   a) Easy
   b) Moderate
   c) Difficult
   d) Impossible

Statistics (experts and moderates): 1 answered Easy, 2 – Moderate, 1 – Difficult.
Statistics (beginner): answered Difficult.

3. Was it easy to understand how to use the tool for interactive assignment of behaviors ("Assign Controller")? And use it with the modular robot in your case?
   a) Easy
   b) Moderate
   c) Difficult
   d) Impossible
   Statistics (experts and moderates): 3 answered Easy, 1 – Moderate.
   Statistics (beginner): Easy.
4. What is your opinion about the GUI of QPSS? In the context of how it reflects the functionality of QPSS, but not the visual quality of the design. In other words, is it easy to understand the functionality behind different button, toolbars and so on.
   a) Easy
   b) Moderate
   c) Difficult
   d) Impossible
   Statistics (experts and moderates): 3 answered Moderate, 1 – Easy.
   Statistics (beginner): Moderate.
5. Is it confusing that first you should construct the morphology of modular robot in the static simulation environment and later start the simulation?
   a) Yes
   b) No
   c) Not sure
   Statistics (experts and moderates): 2 answered Yes, 2 – No.
   Statistics (beginner): No.
6. In general, what is your opinion about QPSS, when you compare it to your previous style of work with USSR?
   a) Useful
   b) Useless
   c) New use (Unusual use)
   d) Not sure yet
   Statistics (experts and moderates): 4 answered Useful.
   Statistics (beginner): New use (Unusual use).
7. Will you explore other functionality of QPSS after this usability test?
   a) Yes
   b) No
   c) Not yet.
   Statistics (experts and moderates): 4 answered Yes.
   Statistics (beginner): Yes.
8. Have you had an experience in working with other simulator for modular robots except USSR?
   a) Yes
   b) No
   Statistics (experts and moderates: 3 answered No, 1 – Yes.
   Statistics (beginner): No.
9. How could you evaluate your experience with USSR?

a) I am expert
b) I am moderate
c) I am beginner

Statistics (experts and moderates): 3 answered "I am moderate", 1 – "I am expert".
Statistics (beginner): "I am beginner".

10. In your opinion, what could be better in QPSS? From the perspective of functionality. In other words, what tools would you like to be supported in QPSS? Be concise.
General suggestions written for the above question see Table D1. Here future work indicates the suggestion which is realistic to implement in the near future, done – was implemented after usability test and to decide – suggestion which should be first analyzed and decided if it will be useful for a wider audience.

Table D1 Suggestions for QPSS in USSR from participants of usability test

Participant's name	Suggestions	State
Ulrik Pagh Schultz (expert)	1. Ensure marking of active buttons, because now it is difficult to be aware which one is selected and is in use. 2. Changes in the GUI	1. Future work 2. Done
David Christensen (expert)	1. Reset (restart) of simulation. 2. Integration of GUIs of several tools used with USSR, for example Modular Commander, QPSS, USSR and so on. 3. Choose new modular robot while running.	1. Future work 2. To decide 3. Future work
Andreas Lyder (moderate)	1. Ensure marking of active button, because now it is difficult to be aware which one is selected and is in use. 2. QPSS should be the main window and there the simulations can be opened and closed. 3. QPSS can be integrated in the simulation environment of USSR.	1. Future work 2. To decide 3. To decide
Mirko Bordignon (moderate)	1. The toolbar called "Assistant" could be more evident, so the user is focusing on it at once when QPSS is started. 2. It would be useful to assign the behaviors (controllers) during static state of simulation. 3. Ensure marking of active and passive buttons, because now it is difficult to be aware which one is selected and is in use. Moreover, the difference where is the text and where is the button. 4. Runtime sketching of source code for behaviors (Coding in GUI).	1. Future work 2. Done 3. Future work 4. Future work
Franko Mendoza Garcia (moderate)	1. Indicate when the "Test before simulation" (also button) process is finished. 2. Allow for changing the default position of the robot being assembled 3. Implement controller for OdinMusle, which is	1. Done 2. Done 3. Done

	expanding and contracting it.	
Danish Shaikh (beginer)	1. Auto saving during simulation runtime. 2. The GUI could reflect the workflow (like for example steps for construction of morphology, assignment of behaviors and so on). Maybe in the form of templates. 3. Use 3D text for labeling of connectors or something similar. So that it would be possible to identify number of connector much easier. Display the connectivity of two modules. Which connector is connected to which (number of connector). 4. Display of current action (active tool), possible physics and constraints. 5. Dynamic addition of new modules and connection of them during simulation runtime.	1. To decide 2. To decide 3. Future work 4. Future work 5. Future work

During the test each participant was observed, moreover the difficulties participant was facing were noted. These are the following (including solution):

- It is difficult for the user to quickly learn how to choose modular robot ("Modular robots" toolbar → first combo box from the left), add default module (" Default" button in the same toolbar) and after that choose "On connector" tool for construing modular robot's morphology. Here the confusion was noticed. This is problematic place, because it is essential for beginning construction of modular robot's morphology. Taking this into account it is crucial to improve these GUI components. The solution was to merge the functionality of combo box, "Default" button and "On Connector" button into only one combo box. In this way, the user will be choosing the modular robot name in the combo box and as a result, the default (first) construction module will be added into simulation environment. Furthermore, the default construction tool will be activated automatically. In this way, by only one selection in the combo box, user will be able quickly move to construction of modular robot's morphology.
- Difficulties with controller assignment in the tool bar called "Controllers". To be more precise, transition from choosing the controller name in the combo box to "Assign" button (users often forget to press "Assign" button). This problem is nearly identical to the one above. As a consequence, the solution was to merge functionality of both into combo box.
- Next is the transition from static state of simulation into dynamic. Here when the morphology of modular robot is ready the user should press button called "Test before simulation" (is essentially connecting the connectors on the modules) and after that select the button for starting simulation. The problem is that the users often forget to press "Test before simulation". The solution was to merge connection of connectors with button "Play".

Appendix E: DCDs of QPSS for Each Module of Functionality

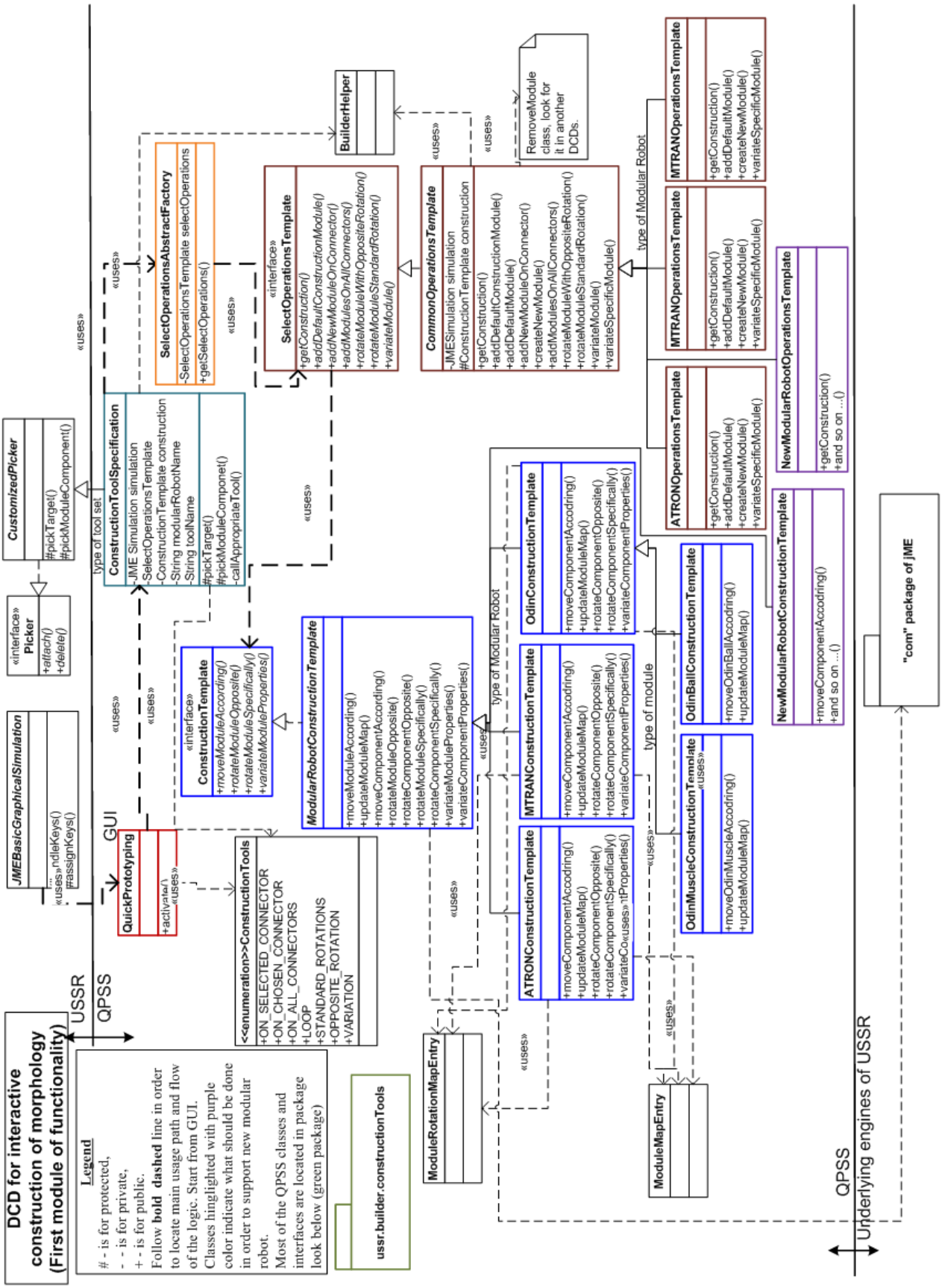


Figure E.1 DCD for interactive construction of morphology

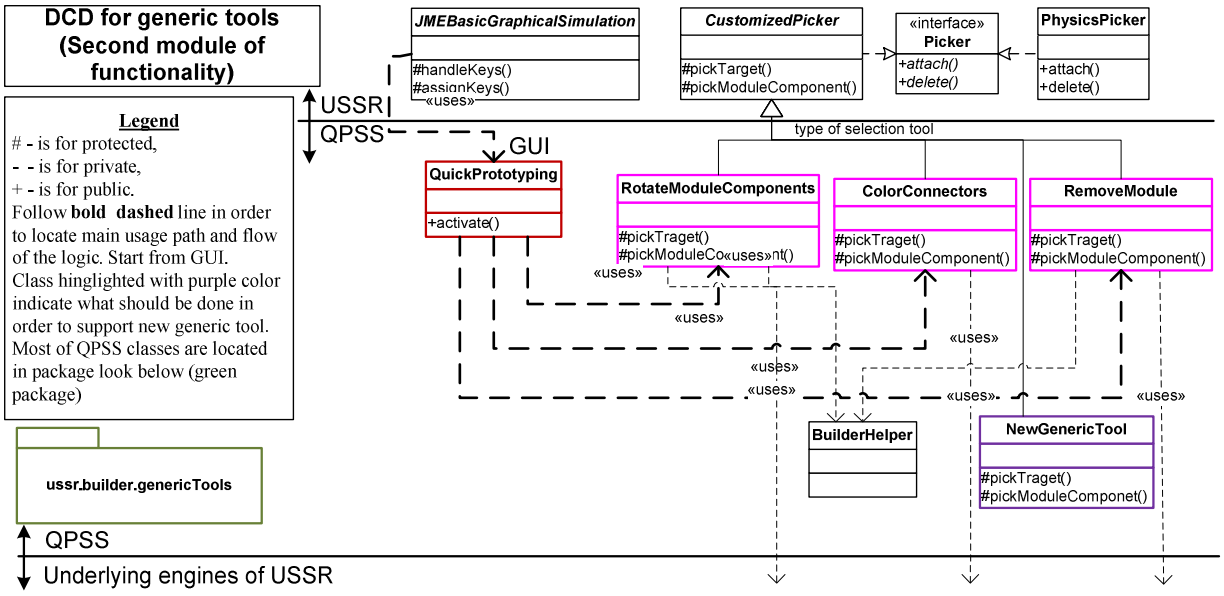


Figure E.2 DCD for generic construction tools

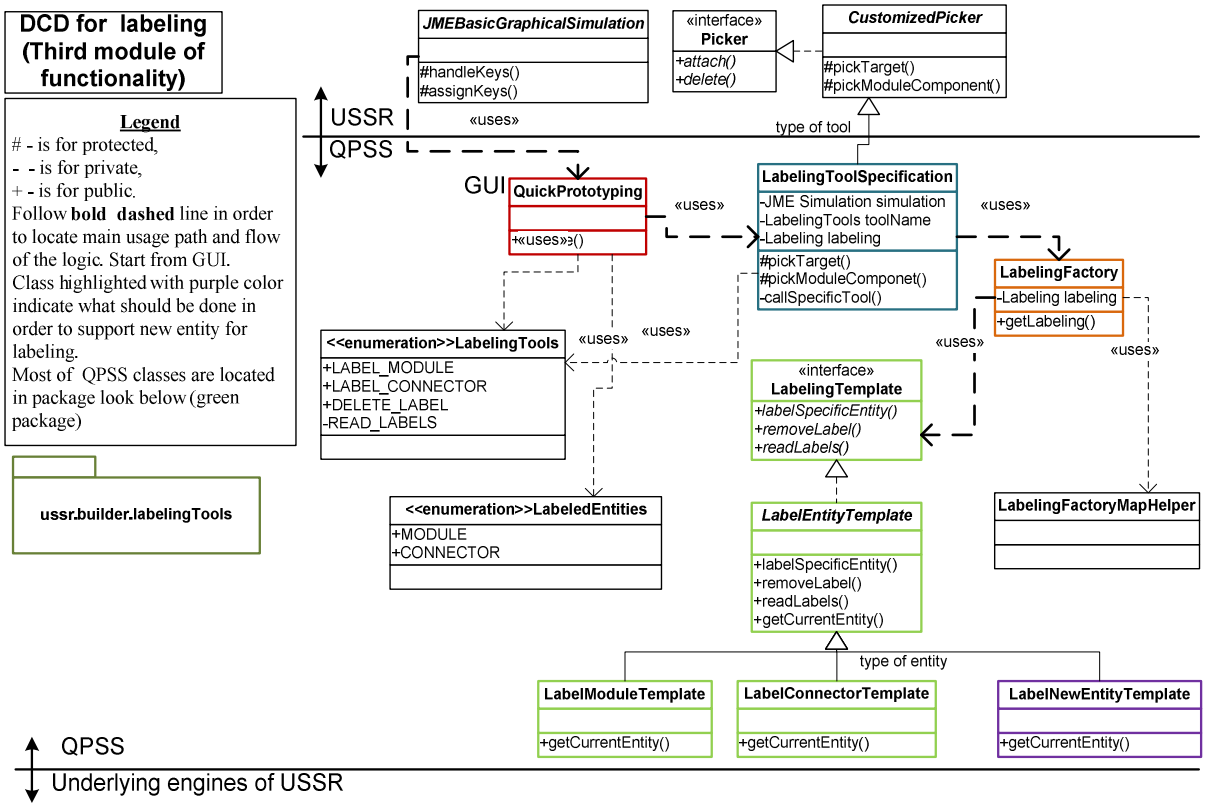


Figure E.3 DCD for labeling

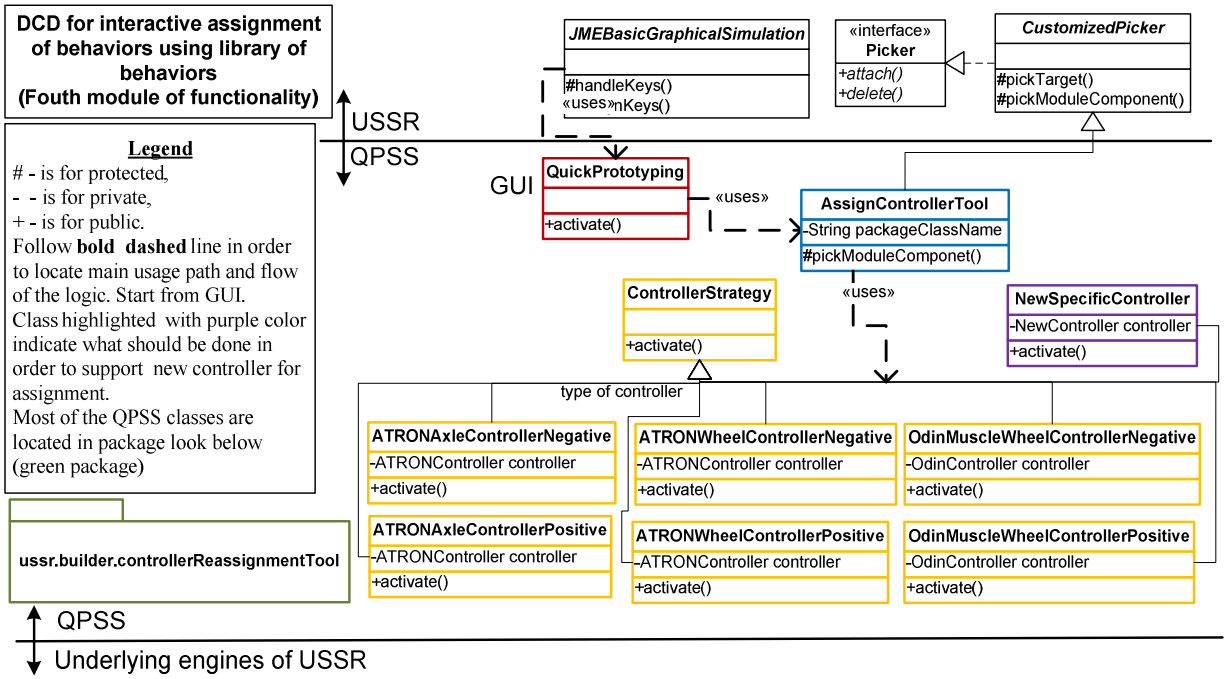


Figure E.4 DCD for interactive assignment of behaviors

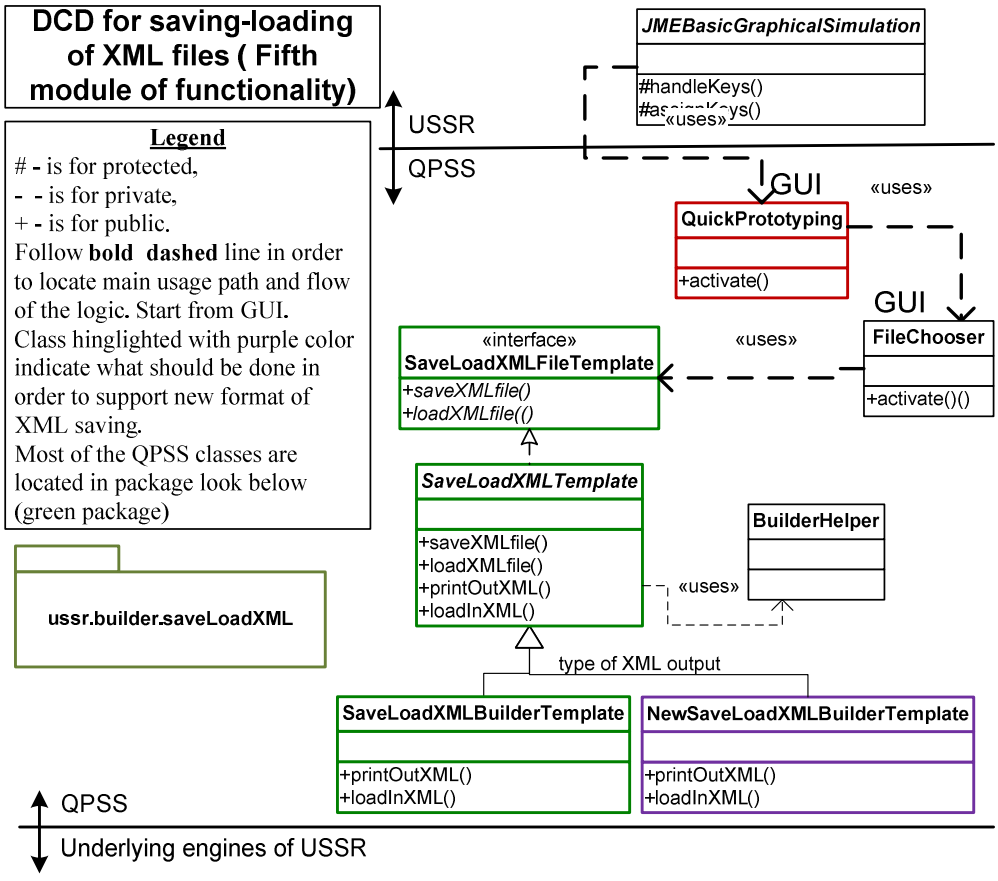


Figure E.5 DCD for saving-loading of XML files