Graphical User Interface for Roboscoop Applications

Bachelor Thesis
Jonas Stulz
November 10, 2014

Advisors: Andrey Rusakov, Dr. Bertrand Meyer
Department of Software Engineering, ETH Zürich
Abstract

Software written for robotics, where the user interacts with the robot, often consists of two major parts: The Graphical User Interface (GUI), which is used to get inputs from the user, while the controller makes calculations and navigates the robot. The two parts are logically separated and don’t interact much with each other. Although the controller mainly has to stay on the hardware of the robot, the GUIs are more free and could also be located on a remote device.

The two parts are modular: Two different applications could have identical GUIs and on one application could have multiple very different GUIs. The work needed to provide a stable network communication between the parts is very similar in distinct applications.

This thesis aims to provide a protocol to simplify the work to connect GUIs and Controller in such applications. The focus of this solution is on the simplicity of the usage and to provide enough freedom for the developer in the design choices he is making, such as programming language or number of GUIs.
# Contents

## Contents

1. **Introduction**  
   1.1 Motivation ........................................ 1  
   1.2 Overview of the Thesis .......................... 2

2. **Requirements** ........................................ 3

3. **Architecture** .......................................... 5  
   3.1 Language and Framework ........................... 5  
   3.2 Basic Structure ..................................... 5  
   3.3 Communication .................................... 6

4. **Evaluation** ............................................ 9  
   4.1 Examples ........................................... 9  
   4.2 Analysis ............................................ 11

5. **Limitations and Future work** ......................... 13

6. **Results and Conclusion** ............................... 15

A. **Interfaces** ........................................... 17  
   A.1 Application ......................................... 17  
   A.2 Controller .......................................... 17  
       A.2.1 Deferred Features .............................. 17  
       A.2.2 Helper Classes for Communication .......... 18  
   A.3 GUI .................................................. 19  
       A.3.1 Datagram Socket ................................ 19  
       A.3.2 Stream Socket .................................. 19

B. **Documentation** ........................................ 21
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>B.1</td>
<td>Overall Structure</td>
<td>21</td>
</tr>
<tr>
<td>B.2</td>
<td>MAIN and Roboscoop_Application</td>
<td>22</td>
</tr>
<tr>
<td>B.3</td>
<td>SERVER</td>
<td>23</td>
</tr>
<tr>
<td>B.4</td>
<td>CONNECTION</td>
<td>24</td>
</tr>
<tr>
<td>B.5</td>
<td>MESSAGE</td>
<td>25</td>
</tr>
<tr>
<td>B.6</td>
<td>MESSAGE_PROCESSOR</td>
<td>26</td>
</tr>
<tr>
<td>B.7</td>
<td>CONTROLLER</td>
<td>28</td>
</tr>
<tr>
<td>B.8</td>
<td>BUFFER</td>
<td>29</td>
</tr>
<tr>
<td>B.9</td>
<td>STATUS and STATUS_INTERN</td>
<td>29</td>
</tr>
<tr>
<td>B.10</td>
<td>COMMAND</td>
<td>30</td>
</tr>
<tr>
<td>B.11</td>
<td>COMMAND_INSTRUCTION</td>
<td>30</td>
</tr>
<tr>
<td>B.12</td>
<td>CONSTANTS</td>
<td>30</td>
</tr>
<tr>
<td>C</td>
<td>Code</td>
<td>33</td>
</tr>
<tr>
<td>C.1</td>
<td>RS_DISTRIB_GUI</td>
<td>33</td>
</tr>
<tr>
<td>C.1.1</td>
<td>ROBOSCOOP_APPLICATION</td>
<td>33</td>
</tr>
<tr>
<td>C.1.2</td>
<td>RS_DISTRIB_GUI_BUFFER</td>
<td>33</td>
</tr>
<tr>
<td>C.1.3</td>
<td>RS_DISTRIB_GUI_COMMAND</td>
<td>35</td>
</tr>
<tr>
<td>C.1.4</td>
<td>RS_DISTRIB_GUI_COMMAND_INSTRUCTION</td>
<td>36</td>
</tr>
<tr>
<td>C.1.5</td>
<td>RS_DISTRIB_GUI_CONNECTION</td>
<td>40</td>
</tr>
<tr>
<td>C.1.6</td>
<td>RS_DISTRIB_GUI_CONSOLE_LISTENER</td>
<td>46</td>
</tr>
<tr>
<td>C.1.7</td>
<td>RS_DISTRIB_GUI_CONSTANTS</td>
<td>47</td>
</tr>
<tr>
<td>C.1.8</td>
<td>RS_DISTRIB_GUI_CONTROLLER</td>
<td>49</td>
</tr>
<tr>
<td>C.1.9</td>
<td>RS_DISTRIB_GUI_MAIN</td>
<td>52</td>
</tr>
<tr>
<td>C.1.10</td>
<td>RS_DISTRIB_GUI_MESSAGE</td>
<td>53</td>
</tr>
<tr>
<td>C.1.11</td>
<td>RS_DISTRIB_GUI_MESSAGE_PROCESSOR</td>
<td>60</td>
</tr>
<tr>
<td>C.1.12</td>
<td>RS_DISTRIB_GUI_SERVER</td>
<td>71</td>
</tr>
<tr>
<td>C.1.13</td>
<td>RS_DISTRIB_GUI_STATUS</td>
<td>73</td>
</tr>
<tr>
<td>C.1.14</td>
<td>RS_DISTRIB_GUI_STATUS_INTERN</td>
<td>74</td>
</tr>
<tr>
<td>C.2</td>
<td>RS_DISTRIB_GUI_JAVA_GUI</td>
<td>76</td>
</tr>
<tr>
<td>C.2.1</td>
<td>CONNECTION</td>
<td>76</td>
</tr>
<tr>
<td>C.2.2</td>
<td>GRAPHICS</td>
<td>80</td>
</tr>
<tr>
<td>C.3</td>
<td>RS_DISTRIB_GUI_SIMPLE_GUI</td>
<td>87</td>
</tr>
<tr>
<td>C.3.1</td>
<td>APPLICATION</td>
<td>87</td>
</tr>
<tr>
<td>C.3.2</td>
<td>BUFFER</td>
<td>88</td>
</tr>
<tr>
<td>C.3.3</td>
<td>CONNECTION</td>
<td>89</td>
</tr>
<tr>
<td>C.3.4</td>
<td>INTERFACE_NAMES</td>
<td>90</td>
</tr>
<tr>
<td>C.3.5</td>
<td>MAIN_WINDOW</td>
<td>91</td>
</tr>
<tr>
<td>C.4</td>
<td>RS_DISTRIB_GUI_SIMULATOR</td>
<td>94</td>
</tr>
<tr>
<td>C.5</td>
<td>RS_DISTRIB_GUI_SIMPLE_SIMULATOR</td>
<td>98</td>
</tr>
</tbody>
</table>

## Bibliography

101
Chapter 1

Introduction

1.1 Motivation

The Roboscoop framework [2] is aimed at resolving coordination, synchronization, and other concurrency issues for robotics. It is based on SCOOP (Simple Concurrent Object-Oriented Programming) [1], a programming model for concurrency that excludes data races by construction. However, it lacks the convenient and functional graphical interfaces needed for better interaction with robots.

To simply write GUIs for those applications would not be very far-sighted. A flexible approach, where the work of the Roboscoop developer who writes a GUI can be simplified, would be more beneficial. Furthermore it is likely that the developer of the controller and the GUI are not the same person and a restriction of the programming language to be the same for both parts would be an unnecessary constraint.

Although the protocol is meant for Roboscoop applications, there is no need to make it dependent on Roboscoop. A standalone protocol could be used for other applications in and outside of robotics which follow the same structure, a separation of GUI and the rest of the software.
1. Introduction

1.2 Overview of the Thesis

The contents of the different chapters of the thesis are briefly described.

- Requirements
  This section describes what the protocol needs to provide in order to be useful.

- Architecture
  This chapter explains the basic structure of the protocol and how the different parts communicate with each other.

- Evaluation
  The capabilities of the protocol are discussed in this section with help of two examples.

- Limitations and Future Work
  This section lists the limitations of the protocol and what could be done to improve it in the future.

- Results and Conclusion
  This chapter sums up the findings of the thesis and comes to a conclusion.

- Appendix A: Interfaces
  The technical details of the Interfaces for the GUI and controller to communicate with the server are listed here. This section is important when using the protocol.

- Appendix B: Documentation
  This appendix describes the architecture of the protocol in depth.

- Appendix C: Code
  The last appendix contains all code of the thesis. This includes the Protocol, two sample GUIs and two sample controllers.
Chapter 2

Requirements

For the protocol to be useful, the cost for the developer to get to know and implement it must be as low as possible. This has to be achieved while keeping restrictions to the functionality at a minimum.

It is certainly possible to use the protocol for a controller and a GUI on the same hardware, but to tap its full potential the two components are separated by a network. The protocol has to be robust and account for unwanted behavior on this channel, like duplication or loss of data.

Although the protocol can be used on its own, the main focus lies on writing GUIs for Roboscoop applications. As Eiffel is the language of Roboscoop, it has to be possible to work with the protocol in Eiffel and with SCOOP.

Generally the communication between the GUI and controller is very asynchronous. In this protocol the communication in direction GUI to controller is called COMMAND. Such a COMMAND has a name and attributes and is almost exclusively used to forward user input. Messages in the other direction are called STATUS. A STATUS can be a state of the robot, such as a position or speed, or a state of the server, for example a parse error. The protocol has to work optimally for this asynchrony.

To summarize the requirements:

- Low overhead
- Few restrictions
- Robustness
- SCOOP/Eiffel
- COMMAND/STATUS
Chapter 3

Architecture

3.1 Language and Framework

The possibility to extend the Roboscoop framework was discarded because having a Roboscoop application is not necessary to run the framework. Instead it is implemented as an Eiffel-Library. With this design choice the protocol works as a standalone and can be used to develop applications for robotics from scratch. Both mentioned possibilities restrict the programming language of the controller to Eiffel. Lifting this restriction and still being able to work with Roboscoop applications would increase the complexity of the protocol too much.

Since it’s a main goal of the protocol to separate the GUI physically from the controller, the restriction of using Eiffel as programming language for the GUIs can be lifted with only a slight increase of complexity. With the use of sockets and a JSON subset a GUI written in any programming language can communicate with the protocol. This separation makes the simultaneous support of multiple GUIs easier.

3.2 Basic Structure

Figure 1 shows a complete class diagram of the protocol. The protocol works in a server-client structure where the server is a part of the protocol whilst the clients are the GUIs. It consists of three main parts; the controller-interface, the buffer and the server. The server uses the buffer to communicate with the controller-interface/controller and uses sockets to communicate with the GUIs.

In an attempt to simplify the connection between GUI and server/controller the server listens to a broadcast on a predefined port. It replies with a broadcast containing the IP-address and port of the socket to connect to.
3. Architecture

Those values are defined at creation of the server. With this method, the GUI can be programmed to work in a changing network environment.

Because of the controller, buffer and server running on the same application, concurrent computing is an important topic. The protocol uses separate objects from SCOOP as an alternative to EiffelThreads because Roboscoop also works with these objects. This means that Roboscoop applications are already configured to use separate objects and there is no additional configuration work to do. The keyword separate is used to describe an object which can be accessed from different processors. The SCOOP framework handles concurrency of such objects and excludes data races. This additional layer of abstraction makes the framework easy to develop with.

The controller, which the user of the protocol has to write, can be written without using Scoop because of the controller-interface handling the concurrent computing. This can be more useful when the protocol is used as a standalone without Roboscoop.

Another part of the library is the console listener which listens to the “stop” string in the console and halts the execution if it detects it.

3.3 Communication

There are three important types of communication between protocol and GUI. Figure 2 shows a simplified runtime-diagram of them.

- The COMMAND is the GUI’s tool to communicate with the server and controller. Mostly, it is used to forward user input from the GUI through the server to the controller in order to control the robot, but it is also used to send instructions directly to the server. To request all possible COMMANDs and to register for a STATUS are the most important instructions.

- The STATUS is sent from the controller to the server while a GUI can register to it. They represent states of the robot which are of interest to the GUI. If a GUI has registered to such a STATUS, it is notified every time when it changes. Different GUIs can register to different (overlapping) sets of STATUSes.

- The Command Instruction is sent from the controller to the server and determines the commands and arguments which the server forwards from the GUI to the controller. If a command does not comply with the guidelines, the server sends an error message back to the GUI and does not forward a message to the controller. The main goal of this tool is to simplify the writing of a GUI to a controller written by another developer.
3.3. Communication

Figure 1. A complete class diagram. For detailed information about all the classes consult the documentation chapter of the thesis.

Figure 2. Runtime diagram of the most important communication types.
In this chapter the capabilities of the protocol are discussed with help of two examples. The first one is a minimal example to indicate the complexity of the protocol, the second is a larger one which should simulate a real usage. Both controllers are only simulators and do not interact with any real robot.

4.1 Examples

To use the library the developer has to write two classes; the CONTROLLER and a GUI. Finally he has to create an instance of the RS_DISTRIB_GUI_MAIN class in his Roboscoop application. If the framework is used without a Roboscoop application, this instance is the only thing needed in the root class of the application. The generic parameter of the class determines the CONTROLLER used, while the arguments of the creation procedure define the ports and IP-address of the sockets.

The minimal example is discussed first.

The following class is an example for a root class when using the protocol without Roboscoop.

```java
class EXAMPLE_APPLICATION
create
    make
feature
    make
do
        create main.run ("84.75.135.127", 2000, 8888)
end
main: RS_DISTRIB_GUI_MAIN [RS_DISTRIB_GUI_SIMPLE_SIMULATOR]
end
```

The controller needs to inherit from the RS_DISTRIB_GUI_CONTROLLER interface and implement the five deferred features.
4. Evaluation

The next class is a very simple controller. It simulates a robot which can support two commands, `start` and `stop`, with one status, `running`.

```eiffel
class RS_DISTRIB_GUI_SIMPLE_SIMULATOR
inherit RS_DISTRIB_GUI_CONTROLLER
create make feature {NONE}
at_create
  do -- initialize the global variable, no server needed.
      create running.make ("running", "no")
  end
initialize local
  do -- initialize status and command instruction at the server
      command_instruction: RS_DISTRIB_GUI_COMMAND_INSTRUCTION
      send_status (running)
      create command_instruction.make ("start")
      command_instruction.allow_none
      send_command_instruction (command_instruction)
      create command_instruction.make ("stop")
      command_instruction.allow_none
      send_command_instruction (command_instruction)
  end
execute
  do -- There is no need to calculate something in this very simple Simulator.
  end
execute_every_time_interval
  do -- Send the status to the server
      send_status (running)
  end
handle_command (a_command: RS_DISTRIB_GUI_COMMAND)
  do
      if a_command.name.is_equal ("start") then
        running.set_value ("yes")
      elseif a_command.name.is_equal ("stop") then
        running.set_value ("no")
      end
  end
running: RS_DISTRIB_GUI_STATUS
end
```

The last part needed is a GUI. The `SIMPLE_GUI` is based on a “Graphics application, multi-platform, with EiffelVision 2” project from Eiffelstudio. It consists of a single button with which the simulated robot can be started or stopped. The state (running or not) of the robot is displayed in the middle...
of the button. The code for this GUI is located in the code section under
RS_DISTRIB_GUI_SIMPLE_GUI

The other GUI-controller pair is capable of more.

The CONTROLLER simulates a robot which can accelerate, break and steer. Depending on the COMMANDS received from the GUI, the controller calculates the state of the robot and sends STATUSES to the server containing its x/y coordinates, speed, rotation speed and rotation angle. The code for this CONTROLLER is located in the code section under RS_DISTRIB_GUI_SIMULATOR.

The GUI (figure 3) is written in Java and consists of a map where the simulated position of the robot is visible and buttons to control this robot. The controller responds to the commands issued by pressing the buttons whereafter the GUI shows the movements of the robot. The GUI was written with the help of the Standard Widget Toolkit of Eclipse and can be found in the Code Section under RS_DISTRIB_GUI_JAVA_GUI.

4.2 Analysis

The first example should show that the use of the protocol was not overly complicated. The size and structure of the SIMPLE_SIMULATOR indicate this. The SIMPLE_GUI is bigger and more complicated, but this lies in the nature of GUIs and the template. Little of its complexity comes from the interaction with the protocol.

The second example should simulate a real usage of the protocol. Of course the classes are bigger than in the minimalistic example, but the complexity has not increased excessively. The simulated robot could be controlled in real-time, both on the same computer as well as in a local network.
4. Evaluation

Figure 3. A picture of the example GUI. The red dot represents the robot while the small blue dot indicates the direction in which the robot is facing.
Chapter 5

Limitations and Future work

All planned features could be implemented with one exception: To make the usage of the protocol easier it was planned to automatically update the IP-address of the server. The retrieval of this information was not successful in Eiffel and the IP-address and port have to be assigned manually as a parameter at creation of the server.

All the testing of the protocol was done with simulated robots and with a single developer. To really capture the benefits and flaws of the protocol, testing with real robots and different developers should be done.

To be able to stop the execution of the server, a CONSOLE_LISTENER was implemented. Its only purpose is to listen to the "stop" command, but its structure could be used for much more, for example to configure the server at run-time which is not possible at this moment.

If a GUI only wants to request a STATUS once, it has to register for it and unregister immediately afterwards so as not to get further updates for this status. If this functionality is needed a lot, a “once” command could be added to the set of usable commands for the GUIs to automate this procedure.
Chapter 6

Results and Conclusion

The project resulted in a protocol to use with Roboscoop applications or as a standalone. A minimal example controller and GUI were created which show a not overly complicated setup of the protocol. A realistic controller and bigger GUI were also created to demonstrate a possible real use of the protocol. Testing on one computer and on a local network showed real time control of the simulated robot with the GUI and without crashes.

Almost all wanted features could be realized with the exception of an automated IP-address recognition. Some network changes might require a recompilation of the framework.

The tests with the examples went well but further evaluations with real robots and different developers will show more benefits and flaws of the protocol.
A.1 Application

To use the protocol, the (Roboscoop) application has to have an instance of the MAIN class with the desired CONTROLLER as generic parameter. The creation procedure run of the MAIN class has three arguments. The IP-address and port on which the stream socket is expected to connect and the port on which the datagram socket sends and listens for broadcasts.

A.2 Controller

A.2.1 Deferred Features

The controller has to be an Eiffel class which inherits from the deferred class RS_DISTRIBUT.GUI_Controller. The class has five deferred features, the rest of the features are frozen. A brief view over the deferred methods:

- at create is executed in the creation procedure of the class. It has to be used because the creation procedure itself is frozen.

- initialize is executed before all following but after the server setup has finished. Initializations where the server is needed can be made in this feature.

- execute is executed every iteration of the loop as fast as specified in the CONSTANTS class.

- execute every timeinterval is executed every time interval which can be set at runtime. This can be used in order to prevent flooding of the network by only sending periodic updates to the GUI through the SERVER.
• *handle_command* is called when a COMMAND arrives from the GUI. In this feature the further process of the COMMANDs should be specified.

### A.2.2 Helper Classes for Communication

To communicate with the GUIs through the SERVER the CONTROLLER uses three helper classes:

- The *STATUS* class is a simple one, which is used to send states of the robot to the SERVER, an example being the position or speed. It consists of a name string and a value string and has to be sent with the feature "send_status" from the deferred class. A sent STATUS with the same name as an old one is understood as update of the old STATUS.  

- The *COMMAND* class is the equivalent to the status class but for the other direction. If a button is pressed on a GUI, the latter sends a command to the controller through the server. It consists of the name and argument string while also containing the id of the originating GUI if this should be needed.

- *COMMAND INSTRUCTIONs* are used to restrict the kind of COMMANDs the SERVER accepts from the GUIs. The name string of the class determines the exact name a COMMAND can poses. If there is no COMMAND INSTRUCTION with a matching name, the server sends back an error message to the GUI and does not forward the COMMAND to the CONTROLLER.

For the argument of the COMMAND there are many possibilities. The CONTROLLER can allow strings, integers, doubles or none. It can further restrict the argument by reducing the integer argument to a range or the string argument to a set of predefined strings. As a last option multiple arguments can be allowed. The same holds for the arguments. If there is a command sent with non-matching arguments, it is not forwarded and an error message is sent back.
A.3 GUI

The GUI communicates with a subset of JSON through sockets to the SERVER. Since the protocol only handles the JSON type "string", the quotation marks can be omitted. This makes working with JSON strings for the GUI easier. The only JSON the server accepts are command_name and argument pairs. Examples of accepted JSON packets: 'name:argument', 'name:"argument"', 'name:argument1,"argument2"', 'name1:argument1, name2:argument2'

This is what allows the GUI to be written in any language which enables socket communication.

A.3.1 Datagram Socket

The datagram socket is only used to gain connection information from the SERVER to know where to connect to the stream socket. The SERVER will listen to a "request" command on the port specified at creation of the server with the arguments "address" or "port" or both. The response will be a JSON object with "address" as its name and the IP-address as argument or the equivalent with port or both.

A.3.2 Stream Socket

The stream socket is the main channel where the GUI communicates with the SERVER. The SERVER accepts command_name and value pairs. The help COMMAND can be sent to the socket to request all possible commands the server accepts. If the help COMMAND has an argument, only information about the COMMAND with the name specified in the argument is sent back. A set of COMMANDs which is specified in the CONTROLLER exists as well as a set of static COMMANDs which are used by the SERVER. These are:

- help
  
The SERVER sends back information about all COMMANDs which are accepted. If the optional argument is present, only information about the COMMAND specified in the argument is sent.

- register
  
The STATUSES specified in the argument are registered on the SERVER to receive updates if one of them changes.

- unregister
  
A previously registered STATUS specified in the argument is unregistered again.
A. Interfaces

- **safe**
  Saves the set of registered *STATUSes* to the slot named after the argument string on the *SERVER*.

- **load**
  Loads the set of registered *STATUSes* from the slot specified in the argument.

- **get_user_id**
  The *SERVER* sends back the user id of the GUI. This could be useful when debugging a system with multiple GUIs in order to know from which GUI the *CONTROLLER* receives *MESSAGES*. 
Appendix B

Documentation

The design of the framework is described in this chapter in detail. All classes and the most important features are listed by name.

B.1 Overall Structure

![Diagram showing the overall structure of the framework with classes and their relationships.]
The framework is called RS_DISTRIBUT_GUI and in this documentation all classes, which are named RS_DISTRIBUT_GUI_EXAMPLE, are referred to as EXAMPLE.

The MAIN is the heart of the framework and consists of separate instances of the three parts, the SERVER, the CONTROLLER and the CONSOLE_LISTENER. To use the framework the Application has to have an instance of the MAIN class. The three parts communicate via a separate BUFFER using helper classes (COMMAND_INSTRUCTION, COMMAND, STATUS, STATUS_INTERN) which get serialized to become separate. The CONSOLE_LISTENER merely listens for the "stop" string from the console and thus needs no helper classes.

The SERVER consists of two parts, the CONNECTION which is responsible for socket communication with the GUIs) and the MESSAGE_PROCESSOR which analyses the received MESSAGES from the sockets and decides what to do with them, like sending them to the CONTROLLER or processing them. The CONNECTION can handle multiple different GUIs via the sockets.

The GUIs can be written in any language which allows socket communication.

The CONTROLLER is a deferred class to help the concrete CONTROLLER with communication and parallel execution. The concrete CONTROLLER is used to control the robot and send STATUS updates to the SERVER. The concrete CONTROLLER has to be written in Eiffel since it needs to inherit from an Eiffel class.

The last part of the framework are the static CONSTANTS with configurable values for performance, communication, parsing and default sizes.

B.2 MAIN and Roboscoop_Application

To run the framework, the program needs to have an instance of MAIN with a concrete CONTROLLER as generic parameter and execute the create procedure run with three arguments: The IP-address and port of the stream socket and the port of the datagram socket. If no code is needed around the framework, Roboscoop_Application’s make should be used as root.
The **SERVER** is responsible for the communication between the **GUIs** through sockets and the **CONTROLLER** via **BUFFER**. It also processes and triggers **MESSAGES** from and to **GUI** and **CONTROLLER**.

To help fulfill these tasks the class has two helper classes, the **CONNECTION** and the **MESSAGE_PROCESSOR**.

Important features:

- **run**
  
  This is the heart-feature of the **SERVER**. It consists of the basic loop of which cycles through its other features of the **SERVER**

- **listen_to_udp**
  
  The **SERVER** asks the **CONNECTION** if a broadcast has been detected with the datagram socket and if so, lets this **MESSAGE** be processed by the **MESSAGE_PROCESSOR** which sends an answer back to the **CONNECTION** to broadcast through said socket.

- **listen_to_connection_changes**
  
  The **SERVER** asks the **CONNECTION** if there are any **GUIs** wanting to connect to a stream socket and if so, lets the **CONNECTION** connect the **GUIs** with one.

- **listen_to_tcp**
The SERVER asks the CONNECTION if there are any MESSAGES pending on the Stream-Sockets which were connected to GUIs and if so, lets this MESSAGES be processed by the MESSAGE_PROCESSOR which sends an answer back to the CONNECTION to send back to the Stream-Socket where the MESSAGES originated from.

Broken stream socket connections are cleaned up here.

• listen_to_buffer

The SERVER asks the BUFFER if there are any changed STATUSes the GUIs are registered for and sends those to the CONNECTION to send to the GUIs through stream socket.

### B.4 CONNECTION

The CONNECTION is used by the SERVER to handle the communication with the sockets. The class reads broadcasted MESSAGES and broadcasts MESSAGES from the MESSAGE_PROCESSOR through the datagram socket.

Furthermore the CONNECTION also accepts stream sockets from GUIs and sends/receives MESSAGES through them.

Because of race-conditions when a GUI is crashing, some features have empty rescue clause.

Important features:

• read_from_udp

This feature reads the broadcasted MESSAGE from the datagram socket and parses it to a JSON object.

• send_to_udp

The argument, a JSON object, gets serialized to a string and then broadcasted through the datagram socket.
B.5 MESSAGE

- **accept tcp socket**

  This feature tries to accept waiting stream sockets from GUIs. If there are more GUIs waiting and accepted than specified in the CONSTANTS the waiting socket gets connected to an overflow socket and an error message gets sent back.

- **read from tcp**

  In comparison to read from udp this feature takes a user id as parameter and only reads the MESSAGE associated to the id since there is one socket per GUI.

- **send to tcp**

  This feature tries to send a MESSAGE to a specific user. To enable this feature the user has to be connected to the stream socket.

B.5 MESSAGE

The MESSAGE is used to parse, process and answer requests from the sockets. It has a JSON object with own parser and also carries the id of the user associated with the MESSAGE.

Important features:

- **parse**

  A raw string gets parsed to a JSON object. Since the framework only accepts very specific MESSAGES, the parser is very small. If a parse error occurs, an error notice gets written to the resulting JSON object of the MESSAGE.

- **put, add, get, remove**

  These are the interaction features with the JSON object respectively JSON key-value pairs. **Put** will overwrite a value while **add** appends the new value to the old one. **Get** will return the value for a key and if no pair with requested key is found an empty string. **Remove** deletes a key and all its values from the object.

- **to_string**

  Beside the use to print JSON objects, this feature is also used to serialize the objects so that they can be sent back to the sockets.
The MESSAGE_PROCESSOR is processing MESSAGEs received from the CONNECTION. It decides if the MESSAGE is aimed at the SERVER or at the CONTROLLER and writes answers to the MESSAGEs which get sent back to the GUIs.

It also listens to MESSAGEs from the CONTROLLER via the Buffer and handles accordingly.

Important features:

- **process_tcp, process_udp**
  
  This is the entry point for a MESSAGE from the CONNECTION. First the MESSAGE gets checked for errors (verify) and if there are none, decided what to do with it (handle_MESSAGE) and last there is an answer written which gets send back to the CONNECTION via the socket (answer_message).

  Since the capability of the udp part in this framework is very limited. It’s not possible to alter the state of the SERVER via udp-Socket and the handle_message part is omitted.

- **verify**

  This feature is used by the process features to check the satisfaction of the constraints (COMMAND_INSTRUCTION) the CONTROLLER or SERVER has defined. It will write an error message to the MESSAGE if something is wrong and else write an "ok" string.

  The feature is split up into multiple smaller ones for better maintenance.

- **handle_message_tcp**

  The process feature uses this feature to decide what to do with the MESSAGE. If the MESSAGE contains a COMMAND aimed for the
B.6. MESSAGE_PROCESSOR

SERVER, it gets executed and deleted from the MESSAGE. If it contains MESSAGEs aimed for the CONTROLLER, they get sent to the CONTROLLER via the BUFFER.

- **answer_message.tcp, answer_message_udp**

The answer to the GUI is formed here. If there was an error the answer contains an error message. If there was a help COMMAND the answer will consist of a help MESSAGE with all the possible COMMANDs and their arguments. In the normal use case the answer MESSAGE is just an "ok"-String.

- **check_buffer**

Other than with MESSAGEs from the sockets, the class has to actively look for MESSAGEs from the CONTROLLER. The MESSAGE from the BUFFER can either be a STATUS or a COMMAND_INSTRUCTION which will be added to the SERVER using add_command and add_status.

- **add_command_instruction**

Adds a COMMAND_INSTRUCTION to the set of usable COMMANDs. Meta COMMAND names are not allowed (unregister, register, load, safe, get_user_id) and if there already is another COMMAND_INSTRUCTION with the same name, the old one gets overwritten.

- **add_status**

Adds a STATUS to the registrable STATUSes of each user. If the STATUS already existed, the old value gets overwritten.

- **register_status, unregister_status**

Each user who is connected to a stream socket can register and unregister STATUSes which values will get sent to the GUIs whenever they change. Different GUIs can have different STATUSes registered. Registering a registered STATUS or unregistering an unregistered STATUS will result in an error notice in the answer MESSAGE.

- **safe_state, load_state**

The set of registered STATUSes can be saved to be restored later for example because of a disconnection or multiple GUIs which want to synchronize.
B.7 CONTROLLER

The CONTROLLER is a deferred class to be implemented by the user when he wants to write an CONTROLLER running on a robot using this framework. The class handles communication with the framework and parallel execution of various methods.

Important features:

- **send_status, send_command_instruction**
  Send a STATUS or COMMAND_INSTRUCTION to the SERVER through the BUFFER and allows the user to be able to only work with not separate objects.

- **run**
  This feature runs the basic loop of the class which cycles through the other important features of the class. How fast the loop runs can be configured in the CONSTANTS.

- **check_buffer**
  Is executed every loop iteration and checks the BUFFER for MESSAGE from the SERVER. If there is a MESSAGE, handle_command is executed with the MESSAGE as argument.

- **initialize**
  This deferred method is executed once before starting the loop but after the setup of the structure of the framework and is used to provide a pseudo creation procedure where the user already can communicate with the SERVER.

- **execute, execute_every_time_interval, set_time_interval**
  These two deferred execute features get executed periodically to allow for calculations to be made or STATUS updates. Execute_every_time_interval is only executed at most the time_interval specified in set_time_interval with a default of 200 ms. This can, for example, be used to prevent flooding of a network.
• handle_command

Every time a COMMAND is sent from the SERVER this deferred feature is executed with the COMMAND as argument. The user can decide here what to do with the COMMAND.

B.8 BUFFER

The BUFFER is used for the communication between SERVER and CONTROLLER. All MESSAGEs get serialized to a string and put on an Arrayed Queue. Each direction has an own queue, getter and setter.

Important features:

• application_values, server_values
  The queues to store the serialized objects.
• set_server, get_server, has_server
  The access features for MESSAGEs sent from SERVER to CONTROLLER.
• set_application, get_application, has_application
  The access features for MESSAGEs sent from CONTROLLER to SERVER.

B.9 STATUS and STATUS INTERN

The classes used to send data from CONTROLLER through the SERVER to the GUIs. The STATUS class is only used to for the CONTROLLER to send data to the SERVER while the STATUS INTERN class is then used by the SERVER to store the data and decide when to send it to the GUIs (when it has changed). Two classes are used to hide complexity from the user.
Important features:

- **name, value**
  The identifier and content of the data.

- **changed**
  The boolean the STATUS INTERN uses to see if data has changed. Is set when set_value, set_changed or set_unchanged is called.

### B.10 COMMAND

A COMMAND is used when a GUI sends a MESSAGE to the CONTROLLER. What kind of COMMANDs and arguments are allowed is restricted by the CONTROLLER with COMMAND_INSTRUCTIONs.

Important features:

- **name, argument**
  The data of the command.

### B.11 COMMAND_INSTRUCTION

A COMMAND_INSTRUCTION is used to restrict the possible COMMANDs that are accepted by the SERVER from the GUI. The name of a COMMAND has to be exactly the name of an existing COMMAND_INSTRUCTION and the arguments have to follow the specified guideline.

Important features:

- **allow_***
  Used by the CONTROLLER to say what kind of argument is allowed. This can be String, Integer, Double, Range or None paired with allow_multiple to allow more than one argument.

- **_*allowed**
  Request what kind of arguments are allowed.

### B.12 CONSTANTS

This static class contains all CONSTANTS used by the framework. Some of them have to be adapted to secure proper running, others can be changed if needed.

- The JSON constants are used by the GUIs to talk to the SERVER. Changing them can cause problems with existing GUIs.
• The command constants appear in error and help *MESSAGEs* from the *SERVER* to the *GUI* and are display only.

• The performance constants specify the wait times between each iteration of the three MAIN loops. They can be changed if there are performance issues.
C.1  RS_DISTRIB_GUI

C.1.1  ROBOSCOOP_APPLICATION

```ruby
note
description : “Entry Point of the Framework”

class
  ROBOSCOOP_APPLICATION
create
  make

feature
  make
    do
      create main.run ("84.75.135.127", 2000, 8888)
    end
  main: RS_DISTRIB_GUI_MAIN [RS_DISTRIB_GUI_SIMPLE_SIMULATOR]
end
```

C.1.2  RS_DISTRIB_GUI_BUFFER

```ruby
note
description :
  “A buffer used to communicate between Server, Application and Console Listener.”

class
  RS_DISTRIB_GUI_BUFFER
create
  make

feature  --  Access

  make
    --  create with default sizes (overflow is handled by datastructure).
  do
```
create server_values.make ("RS_DISTRIB_GUI_CONSTANTS", default_size)
create application_values.make ("RS_DISTRIB_GUI_CONSTANTS", default_size)
stopped := false
end

stop
-- stop the execution. Is propagated to Server and Application.
do
stopped := true
end

is_stopped : BOOLEAN
-- is the execution stopped?
do
Result := stopped
end

feature -- Access Server

set_server (a_value : separate STRING)
-- insert a value into the server buffer.
do
server_values.extend (create (STRING).make_from_separate (a_value))
end

get_server : STRING
-- read a value from the server buffer.
require
has_server_element
do
Result := server_values.item
server_values.remove
end

has_server_element : BOOLEAN
-- has the server buffer some elements?
do
Result := server_values.count > 0
end

feature -- Access Application

set_application (a_value : separate STRING)
-- insert a value into the application buffer.
do
application_values.extend (create (STRING).make_from_separate (a_value))
end

get_application : STRING
-- read a value from the application buffer.
require
has_application_element
do
Result := application_values.item
application_values.remove
end
C.1. RS_DISTRIBUT_GUI

end

has_application_element : BOOLEAN
-- has the application buffer some elements?
do
  Result := application_values . count > 0
end

feature {NONE} -- Global Variables

application_values : ARRAYED_QUEUE[STRING]
-- values sent from application to server.

server_values : ARRAYED_QUEUE[STRING]
-- values sent from server to application.

stopped : BOOLEAN
-- is the execution stopped?
end

C.1.3 RS_DISTRIBUT_GUI_COMMAND

note

description :
"A command which is sent from the server to the application mostly triggered by a
message from the tcp−socket."

class
RS_DISTRIBUT_GUI_COMMAND

create
make, make_from_serialisation

feature -- Access

make (a_name: STRING; a_argument: STRING; a_user_id: INTEGER)
-- create.
do
  name := a_name
  argument := a_argument
  user_id := a_user_id
end

make_from_serialisation ( a_serialisation : STRING)
-- create from serialisation, used to handle separate objects.
do
  name := a_serialisation . split (‘:’) . at (1)
  argument := a_serialisation . split (‘:’) . at (2)
  user_id := a_serialisation . split (‘:’) . at (3) . to_integer
end

serialize : STRING
-- serialize object to make a separate into a non separate.
do
C. Code

Result := name + ";" + argument + ";" + user_id .out
end

name: STRING
   -- name of the command.

argument: STRING
   -- argument of the command.

user_id: INTEGER
   -- id of user who sent the command. This id might change if the user reconnects.
end

C.1.4 RS_DISTRIBUT_GUI_COMMAND_INSTRUCTION

note
description :
"A set of constraints the Application can set for Commands. The Server will check these constraints and only send valid commands to the application."

class
RS_DISTRIBUT_GUI_COMMAND_INSTRUCTION

create
make, make_from_serialisation

feature  -- Access

make (a_name: STRING)
   -- create.
do
   name := a_name .as_lower
   create custom.make
   argument_allowed := \{RS_DISTRIBUT_GUI_CONSTANTS\}.none_allowed
ensure
   no_forbidden_characters : not name .has (',') and not name .has (':')
end

make_from_serialisation (a_string: STRING)
   -- create from serialisation, used to handle separate objects.
require
   a_string .starts_with ("c")
do
   create custom.make
   name := a_string .split (':') .at (2)
   argument_allowed := a_string .split (':') .at (3)
   lower := a_string .split (':') .at (4) .split (',') .at (1) .to_integer
   upper := a_string .split (':') .at (4) .split (',') .at (2) .to_integer
   make_custom_from_serialisation (a_string .split (':') .at (5))
   multiple_allowed := a_string .split (':') .at (6) .to_boolean
end

serialize : STRING
C.1. RS_DISTRIBUT_GUI

--- serialize object to make a separate into a non separate.

do
  Result := "c:" + name.out + ";" + argument_allowed + ":" + lower.out + "," + upper.out + ":" + serialize_custom + ":" + multiple_allowed.out
end

serialize_custom : STRING
--- serialize the custom values.
local
  l_string : STRING
do
    l_string := ""
    across custom as element loop
      if l_string.count > 0 then
        l_string.append (",")
      end
      l_string.append (element.item)
    end
  Result := l_string
end

make_custom_from_serialisation (a_string : STRING)
--- create custom strings from a serialisation.
do
  if a_string.count > 0 then
    across a_string.split (',') as element loop
      add_custom_string (element.item)
    end
  end
end

to_string : STRING
--- make command instruction to string to print.
do
  if to_string_arguments.is.equal ("") then
    Result := name.out
  else
    Result := name.out + ";" + to_string_arguments.out.as_upper
  end
end

to_string_arguments : STRING
--- make arguments to string to print.
local
  l_result : STRING
do
    l_result := argument_allowed.out
    if argument_allowed.is.equal ({RS_DISTRIBUT.GUI,CONSTANTS}.range.allowed) then
      l_result.append ("[" + lower.out + "]" + upper.out + "]")
    elseif argument_allowed.is.equal ({RS_DISTRIBUT.GUI,CONSTANTS}.custom.allowed) then
      l_result.append ("{" + serialize_custom + "}")
    end
    if multiple_allowed then
      else
        Result := result
    end
end
C. Code

```plaintext
l_result.append("\n")
end
Result := l_result
end

has_custom_string ( a_string : STRING): BOOLEAN
   -- is a specific string allowed as argument?
do
   Result := across custom as l_custom some l_custom.item.is_equal (a_string) end
end

name: STRING
   -- name of the command.

multiple_allowed : BOOLEAN
   -- are multiple arguments allowed?

lower,upper: INTEGER
   -- boundaries (including) if a range of integers is allowed as arguments.

custom: LINKED_LIST[STRING]
   -- custom strings if custom strings are allowed as values.

argument_allowed: STRING
   -- what kind of argument is allowed.

feature   -- Basic Features

allow_string
   -- allow strings as arguments.
do
   argument_allowed := {RS_DISTRIB_GUI_CONSTANTS}.string_allowed
end

allow_integer
   -- allow integers as arguments.
do
   argument_allowed := {RS_DISTRIB_GUI_CONSTANTS}.integer_allowed
end

allow_double
   -- allow doubles as arguments.
do
   argument_allowed := {RS_DISTRIB_GUI_CONSTANTS}.double_allowed
end

allow_range (a_lower: INTEGER; a_upper: INTEGER)
   -- allow a range of integers as arguments.
do
   argument_allowed := {RS_DISTRIB_GUI_CONSTANTS}.range_allowed
   lower := a_lower
   upper := a_upper
end
```
allow_none
   -- allow no arguments.
   do
      argument_allowed := \{ Rs_Distrib_Gui_CONSTANTS\}.none_allowed
   end

allow_custom_strings (a_custom_strings : LINKED_LIST[STRING])
   -- allow custom strings as arguments
   do
      argument_allowed := \{ Rs_Distrib_Gui_CONSTANTS\}.custom_allowed
      across a_custom_strings as one_string loop
         one_string.item.to_lower
      end
      custom := a_custom_strings
   end

allow_multiple
   -- allow multiple arguments. This is used in combination with another argument
   restriction and false by default
   do
      multiple_allowed := true
   end

integer_allowed : BOOLEAN
   -- are only integer arguments allowed?
   do
      Result := argument_allowed.is_equal (\{ Rs_Distrib_Gui_CONSTANTS\}.integer_allowed)
   end

range_allowed : BOOLEAN
   -- are only integer arguments in a specific range allowed?
   do
      Result := argument_allowed.is_equal (\{ Rs_Distrib_Gui_CONSTANTS\}.range_allowed)
   end

none_allowed : BOOLEAN
   -- are no arguments allowed?
   do
      Result := argument_allowed.is_equal (\{ Rs_Distrib_Gui_CONSTANTS\}.none_allowed)
   end

custom_allowed : BOOLEAN
   -- are custom strings arguments allowed?
   do
      Result := argument_allowed.is_equal (\{ Rs_Distrib_Gui_CONSTANTS\}.custom_allowed)
   end

string_allowed : BOOLEAN
   -- are string arguments allowed?
   do
      Result := argument_allowed.is_equal (\{ Rs_Distrib_Gui_CONSTANTS\}.string_allowed)
   end
C. Code

```
end

add_custom_string (a_string : STRING)  
    -- add a string to the custom stings allowed as arguments.
require
    string_does_already_exist : not has_custom_string (a_string)
    custom_allowed: argument_allowed. is_equal ({RS_DISTRIB_GUI_CONSTANTS}. custom_allowed)
do
    custom.extend (a_string)
end

remove_custom_string (a_string : STRING)  
    -- remove a string from the custom strings allowed as arguments.
require
    string_does_not_exist : has_custom_string (a_string)
    custom_allowed: argument_allowed. is_equal ({RS_DISTRIB_GUI_CONSTANTS}. custom_allowed)
local
    i: INTEGER
do
    from
        i := 1
    until
        not custom.valid_index (i)
    loop
        if custom.at (i). is_equal (a_string) then
            custom.go_lth (i)
            custom.remove
        end
        i := i + 1
    end
end
```

C.1.5 RS_DISTRIB_GUI_CONNECTION

note
description :
"A class to handle udp broadcasts and connect users to tcp sockets."

class
RS_DISTRIB_GUI_CONNECTION

create
make

feature {ANY}  -- Access
make (a_address : STRING; a_stream_socket_port: INTEGER; a_datagram_socket_port: INTEGER)
    -- create.
C.1. RS_DISTRIB_GUI

```
host_ip_address := a_address
stream_socket_port := a_stream_socket_port
datagram_socket_port := a_datagram_socket_port
create tcp_message . make_filled (create {RS_DISTRIB_GUI_MESSAGE}.make(−1), 1, {RS_DISTRIB_GUI_CONSTANTS}.max_number_of_connections)
create accepted_sockets . make_filled (create {NETWORK_STREAM_SOCKET}.make, 1, {RS_DISTRIB_GUI_CONSTANTS}.max_number_of_connections)
create message_already_read . make_filled (true, 1, {RS_DISTRIB_GUI_CONSTANTS}.max_number_of_connections)
create user_connected_tcp . make_filled (false, 1, {RS_DISTRIB_GUI_CONSTANTS}.max_number_of_connections)
create overflow_socket . make
end

get_address : STRING
     −− should return address of machine but does only return localhost right now.
do  Result := host_ip_address
end

get_port : STRING
     −− return port of tcp socket.
do  Result := get_tcp_socket . port . out
end

cleanup
     −− cleanup socket connection after execution.
do  get_tcp_socket . cleanup
get_udp_socket . cleanup
rescue
end

message_already_read : ARRAY[BOOLEAN]

feature {ANY} −− UDP Features

is_ready_to_read_udp : BOOLEAN
     −− is there data to read on the udp socket?
do  Result := get_udp_socket . is_readable
end

read_from_udp: RS_DISTRIB_GUI_MESSAGE
     −− read data from udp socket and parse to JSON Object.
local
l_message : RS_DISTRIB_GUI_MESSAGE
l_raw_message: PACKET
do  l_raw_message := get_udp_socket . received ({RS_DISTRIB_GUI_CONSTANTS}.max_packet_size, 0)
create l_message . make_from_packet (l_raw_message, 0)
Result := l_message
```

41
C. Code

```
end

send_to_udp (a_message: STRING)
-- broadcast a message via the udp socket.
local
  l_raw_response: PACKET
  i: INTEGER
do
ddebug ("connection")
  print ("udp send:%N" + a_message + "%N%N")
end
a_message.append (",eof")
create l_raw_response . make (a_message . count)
From
  i := 0
until
  i >= a_message . count
loop
  l_raw_response . at (i) := a_message . at (i+1)
  i := i + 1
end
get_udp_socket . send (l_raw_response , 0)
end

feature { ANY } -- TCP Features

accept_tcp_socket
-- listen for accepting clients .
local
  l_user_id: INTEGER
do
get_tcp_socket . accept
if attached get_tcp_socket . accepted as accepted_socket then
  l_user_id := get_unused_user_id
  accepted_socket . set_blocking
  if l_user_id = -1 then
    overflow_socket := accepted_socket
    overflow_socket . putstring ("Maximum connections reached")
    debug ("connection")
    print ("Tried to connect user but maximum connections reached.")
  end
else
  accepted_sockets [l_user_id] := accepted_socket
  user_connected_tcp [l_user_id] := true
  debug ("connection")
  print ("Connected user:" + l_user_id . out + "%N")
end
end
get_unused_user_id: INTEGER
-- get the lowest user id with no open connection.
```

42
local
  i: INTEGER
  l_found: BOOLEAN

do
  Result := -1
from
  i := 1
  l_found := false
until
  l_found or i > \{RS_DISTRIB_GUI_CONSTANTS\}.max_number_of_connections
loop
  if not user_connected_tcp[i] then
    l_found := true
    Result := i
  else
    i := i + 1
  end
end
end

get_user_ids: LINKED_LIST[INTEGER]
  -- get all user ids with open connections.
local
  l_user_ids: LINKED_LIST[INTEGER]
i: INTEGER

do
  create l_user_ids.make
from
  i := 1
until
  i > \{RS_DISTRIB_GUI_CONSTANTS\}.max_number_of_connections
loop
  if user_connected_tcp[i] then
    l_user_ids.extend(i)
  end
  i := i + 1
end
Result := l_user_ids
end

is_ready_to_read_tcp (a_user_id: INTEGER): BOOLEAN
  -- has socket from user id data to read, if yes write it to message buffer.
do
  Result := false
if get_accepted_tcp_socket(a_user_id).is_open_read and get_accepted_tcp_socket(a_user_id).is_readable then
  get_accepted_tcp_socket(a_user_id).read_line
if attached get_accepted_tcp_socket(a_user_id).last_string as l_raw_message then
  if l_raw_message.count > 0 then
    message_already_read [a_user_id] := false
    tcp_message[a_user_id] := create \{RS_DISTRIB_GUI_MESSAGE\}.
      make_from_string(l_raw_message, a_user_id)
  Result := true
end
read_from_tcp(a_user_id: INTEGER): RS_DISTRIBUT.GUI.MESSAGE
-- read message from buffer.
require
message_already_read [a_user_id] = false
do
debug("connection")
    print("Message from user: "+a_user_id.out+"%N"+tcp_message [a_user_id].to_string +"%N")
end
message_already_read [a_user_id] := true
Result := tcp_message [a_user_id]
end

send_to_tcp(a_message: STRING; a_user_id: INTEGER)
-- send message to user with specific user id.
do
if get_accepted_tcp_socket(a_user_id).isReadable then
    get_accepted_tcp_socket(a_user_id).putstring(a_message)
    get_accepted_tcp_socket(a_user_id).put newline
    debug("connection")
    print("sent to user: "+a_user_id.out+"%N"+a_message+"%N")
else
    debug("connection")
    print("sending failed: %N"+a_message+"%N")
end
end
rescue

disconnect_user_tcp(a_user_id: INTEGER)
-- open slot of a disconnected user for new connection.
do
debug("connection")
    print("disconnect user: "+a_user_id.out+"%N")
end
accepted_sockets [a_user_id] := create {NETWORK_STREAM_SOCKET}.make
user_connected_tcp [a_user_id] := false
rescue
end

is_connection_broken(a_user_id: INTEGER): BOOLEAN
-- is the user at a specific position still there?
do
Result := get_accepted_tcp_socket(a_user_id.item).isReadable
end

feature {NONE} -- Sockets
get_accepted_tcp_socket (a_user_id: INTEGER): NETWORK_STREAM_SOCKET
--- get a socket of a connected user.
require
  user_connected_tcp [a_user_id] = true
do
  Result := accepted_sockets [a_user_id]
end

get_udp_socket: NETWORK_DATAGRAM_SOCKET
--- get the udp socket.
local
  l_udp_socket: NETWORK_DATAGRAM_SOCKET
once
  create l_udp_socket .make_bound (datagram_socket_port)
  l_udp_socket .enable_broadcast
  Result := l_udp_socket
end

get_tcp_socket: NETWORK_STREAM_SOCKET
--- get the tcp socket used to connect to new users.
local
  l_tcp_socket: NETWORK_STREAM_SOCKET
once
  create l_tcp_socket .make_server_by_port (stream_socket_port)
  l_tcp_socket .listen (1)
  l_tcp_socket .set_non_blocking
  l_tcp_socket .set_out_of_band_inline
  Result := l_tcp_socket
end

feature {NONE} -- Global Variables

host_ip_address: STRING
--- the ip address where the server is located. It is used to connect to the stream socket.

stream_socket_port: INTEGER
--- The port on which the stream socket gets connected.

datagram_socket_port: INTEGER
--- The port on which the datagram socket sends and receives messages.

user_connected_tcp: ARRAY[BOOLEAN]
--- is a user connected at a specific user id?

accepted_sockets: ARRAY[NETWORK_STREAM_SOCKET]
--- all sockets to which users can connect.

overflow_socket: NETWORK_STREAM_SOCKET
--- socket to signal user that there are too many connections open on the server.
tcp_message: ARRAY[RS_DISTRIB_GUI_MESSAGE]
--- message buffer for tcp messages.
C. Code

end

C.1.6 RS_DISTRIB_GUI_CONSOLE_LISTENER

note
description :
"A listener to handle console input."

class
RS_DISTRIB_GUI_CONSOLE_LISTENER

inherit
EXECUTION_ENVIRONMENT

create
make

feature -- Access

make (a_Buffer : separate RS_DISTRIB_GUI_BUFFER)
  -- creation
  do
    buffer := a_Buffer
    execution_stopped := false
  end

run
  -- listens to stop command in console.
  do
    io.put_new_line  -- used to flush console to prevent the immediate termination of
      -- the program if the last console entry was stop
    from io.read_line
    until io.last_string.is_equal({RS_DISTRIB_GUI_CONSTANTS}.stop) or
      execution_stopped
      execution_stopped := true
    loop
      io.read_line
      sleep({RS_DISTRIB_GUI_CONSTANTS}.wait_console_listener ∗ 1_000_000)
    end
    stop_execution (buffer)
  rescue
  end

stop
  -- stop the execution of server, application and self
  do
    execution_stopped := true
    stop_execution (buffer)
  end

feature {NONE} -- Basic Features

stop_execution (a_buffer : separate RS_DISTRIB_GUI_BUFFER)
  -- sends stop command to buffer to stop server and application
  do

C.1. RS_DISTRIB_GUI

buffer: separate RS_DISTRIB_GUI_BUFFER
-- separate buffer

execution_stopped: BOOLEAN
-- has execution been stopped from another place than console

C.1.7 RS_DISTRIB_GUI_CONSTANTS

Error: STRING = "error"
-- used to signal an error in JSON.

Parse_error: STRING = "parse_error"
-- used to signal a parsing error in JSON.

Request: STRING = "request"
-- used to signal a request for the udp socket in JSON.

Register: STRING = "register"
-- used to signal a request to register a status in JSON.

Unregister: STRING = "unregister"
-- used to signal a request to unregister a status in JSON.

Help: STRING = "help"
-- used to signal a request to the help message in JSON.

Ok: STRING = "ok"
-- used to signal no problems with message in JSON.

Safe: STRING = "safe"
-- used to signal safe current state in JSON.

Load: STRING = "load"
-- used to signal load state in JSON.

get_user_id: STRING = "get_user_id"
-- used to request user id in JSON.

Address: STRING = "address"
C. Code

- -- used to signal address request for udp socket in JSON.

Port: STRING = "port"
- -- used to signal port request for udp socket in JSON.

feature -- -- Command Constants

None_allowed: STRING = "none"
- -- used to signal no arguments allowed in command instruction.

Integer_allowed: STRING = "integer"
- -- used to signal integer arguments allowed in command instruction.

String_allowed: STRING = "string"
- -- used to signal string arguments allowed in command instruction.

Double_allowed: STRING = "double"
- -- used to signal double arguments allowed in command instruction.

Custom_allowed: STRING = "custom"
- -- used to signal custom string arguments allowed in command instruction.

Range_allowed: STRING = "range"
- -- used to signal range arguments allowed in command instruction.

feature -- -- Connection Constants

Max_packet_size: INTEGER = 1024
- -- maximal number of characters the udp socket can read.

Max_number_of_connections: INTEGER = 10
- -- maximal number of clients able to connect to the server. Setting this too big (>100) can result in poor performance.

feature -- -- Performance Constants

Wait_Server: INTEGER = 0
- -- How long does the Server wait between each loop executions in milliseconds. Setting this too high (>100) can result in poor performance.

Wait_Application: INTEGER = 0
- -- How long does the Application wait between each loop executions in milliseconds.

Wait_Console_listener: INTEGER = 500
- -- How long does the Console Listener wait between each loop executions in milliseconds.

feature -- -- Other Constants

Stop: STRING = "stop"
- -- string that terminates the execution if entered in the console

default_size : INTEGER = 10
C.1. RS_DISTRIBUT.GUI

--- Size of all hashtables, queues and arrays when initialised. All grow when needed.

end

C.1.8 RS_DISTRIBUT_GUI_CONTROLLER

note

description :
"A Basis for a ROBOSCOOP application which wants to use the distributed GUI. This class handles communication and timing."

defered class
RS_DISTRIBUT_GUI_CONTROLLER

inherit
EXECUTION_ENVIRONMENT

feature -- Access

frozen make (a_buffer : separate RS_DISTRIBUT_GUI_BUFFER)
-- create.
do
buffer := a_buffer
stopped := false
time_interval := 200
at_create
create last_time .make_now
end

frozen run
-- initialize and run loop.
do
initialize
run_loop
end

feature {NONE} -- Basic Features

frozen send_command_instruction (a_command_instruction:
RS_DISTRIBUT_GUI_COMMAND_INSTRUCTION)
-- send a command instruction to the buffer/server to update or create.
do
sent_separate (buffer , a_command_instruction . serialize )
end

frozen send_status (a_status : RS_DISTRIBUT_GUI_STATUS)
-- send a status to the server/buffer to update or create.
do
sent_separate (buffer , a_status . serialize )
end

frozen stop (a_buffer : separate RS_DISTRIBUT_GUI_BUFFER)
C. Code

```plaintext
-- stop application, server and console listener.
do
  a_buffer : stop
  stopped := true
end

frozen set_time_interval ( a_miliseconds : INTEGER )
  -- set the lower bound of the interval between the executions of 'execute_every_timeinterval'
do
  time_interval := a_miliseconds
end

frozen get_time_interval :
  INTEGER
  -- get the lower bound of the interval between the executions of 'execute_every_timeinterval'
do
  Result := time_interval
end

feature {NONE} -- Deferred Features

at_create
  -- is executed at creation of the class before the server is running.
defered end

initialize
  -- is executed at the start of run when the server is already running.
defered end

execute
  -- is executed every loop iteration of run.
defered end

handle_command ( a_command: RS_DISTRIB_GUI_COMMAND )
  -- is executed when a command is sent from the buffer/server.
defered end

execute_every_time_interval
  -- is executed at most every specified time interval, default is 200ms.
defered end

feature {NONE} -- Intern Methods

frozen sent_separate ( a_buffer : separate RS_DISTRIB_GUI_BUFFER; message: STRING)
  -- send to separate buffer.
do
  a_buffer . set_server ( message )
end

frozen run_loop
  -- loop over methods until stop is called.
local
  l_time, l_last_time_every, l_last_time_exe : TIME
```
do
  from initialize
  create l_time.make_now
  create l_last_time_every.make_now
  create l_last_time_exe.make_now
until stopped
loop
  l_time.make_now
  l_time.fine_second_add (− time_interval / 1000)
  if l_time > l_last_time_every then
    execute_every_time_interval
    l_last_time_every.make_now
  end
  check_buffer ( buffer )
  l_time.make_now
  dt := l_time.fine_second − l_last_time_exe.fine_second
  if dt < 0 then
    dt := dt + 60
  end
  l_last_time_exe.make_now
  execute
  sleep ( {RS_DISTRIB_GUI_CONSTANTS}.wait_application * 1_000_000 )
end
end

frozen check_buffer ( a_buffer : separate RS_DISTRIB_GUI_BUFFER)
  -- check if buffer has elements and handle them.
local
  l_command: STRING
do
  if a_buffer.has_application_element then
    create l_command.make_from_separate ( a_buffer.get_application )
    handle_command ( create {RS_DISTRIB_GUI_COMMAND}.make_from_serialisation ( l_command ) )
  end
  if a_buffer.is_stopped then
    stop ( buffer )
  end
end

feature {NONE}  -- Global Variables

  frozen buffer: separate RS_DISTRIB_GUI_BUFFER
  -- separate buffer for communication between application and server.
  frozen stopped: BOOLEAN
  -- is server running?
  frozen time_interval: INTEGER
  -- amount of miliseconds between execution of "execute_every_time_interval"
C. Code

```plaintext
frozen last_time : TIME
   -- when was the last time the "execute_every_time_interval" was calculated?

frozen dt : DOUBLE
   -- How much time passed since the last execution of "execute"

end

C.1.9 RS_DISTRIB_GUI_MAIN

note
description :
   "The main class. Server, Application and Console Listener are created and started here."

class
RS_DISTRIB_GUI_MAIN [G -> RS_DISTRIB_GUI_CONTROLLER create make end]

create
run

feature -- Access

run ( a_host_ip_address : STRING; a_stream_socket_port : INTEGER; a_datagram_socket_port : INTEGER)
   -- run and create server, application and console_listener
local
   l_buffer : separate RS_DISTRIB_GUI_BUFFER
   l_application : separate G
   l_address : separate STRING

   do
      l_address := a_host_ip_address
      create host_ip_address . make_from_separate ( l_address )
      create l_buffer . make
      create server . make ( l_buffer, host_ip_address, a_stream_socket_port, a_datagram_socket_port )
      create console_listener . make ( l_buffer )
      create l_application . make ( l_buffer )
      application := l_application
      run_separate ( server, application, console_listener )
   end

feature {NONE} -- Basic Features

run_separate ( a_server : separate RS_DISTRIB_GUI_SERVER; a_application: separate RS_DISTRIB_GUI_CONTROLLER; a_console_listener: separate RS_DISTRIB_GUI_CONSOLE_LISTENER)
   -- run separate objects

   do
      a_server . run
      a_application . run
      a_console_listener . run
   end
```
C.1. RS_DISTRIBUT_GUI

feature {NONE}  -- Global Variables

    server : separate RS_DISTRIBUT_GUI_SERVER
             -- separate server

    application : separate RS_DISTRIBUT_GUI_CONTROLLER
                -- separate application

    console_listener : separate RS_DISTRIBUT_GUI_CONSOLE_LISTENER
                     -- separate console listener

    host_ip_address : separate STRING

end

C.1.10 RS_DISTRIBUT_GUI_MESSAGE

note
description :
"A JSON object with a light parser."

class
RS_DISTRIBUT_GUI_MESSAGE

create
    make, make_from_packet, make_from_string

feature  -- Access
    make (a_user_id : INTEGER)
        -- create.
        do
            create hash_table.make (10)
            hash_table.wipe_out
            user_id := a_user_id
        end

    make_from_string (a_raw_message: STRING; a_user_id: INTEGER)
        -- create and parse string.
        do
            make(a_user_id)
            parse (a_raw_message)
        end

    make_from_packet (a_raw_message: PACKET; a_user_id: INTEGER)
        -- create and parse packet.

    local
        l_string_builder : STRING
        i : INTEGER
        do
            from
                i := 0
            l_string_builder := ""
            until
not a raw message. valid position (i)

loop
  l_string_builder .append (a_raw_message .at (i).out)
i := i + 1
end
make_from_string ( l_string_builder , a_user_id )
end

user_id : INTEGER
  -- the user id from the creator of the message
to_string : STRING
  -- string representation
local
  l_string_builder : STRING
  l_first : BOOLEAN

  do
    l_string_builder := ""
    l_first := true
    across hash_table . current_keys as key loop
      if not l_first then
        l_string_builder .append (","
      end
    l_first := false
    l_string_builder .append ( item_to_string (key.item) )
  end
Result := l_string_builder
end
to_sending_string : STRING
  -- string representation
local
  l_string_builder : STRING
  l_first : BOOLEAN

  do
    l_string_builder := ""
    l_first := true
    across hash_table . current_keys as key loop
      if not l_first then
        l_string_builder .append (","
      end
    l_first := false
    l_string_builder .append ( item_to_sending_string (key.item) )
  end
Result := l_string_builder
end
to_string_only (a_argument: STRING): STRING
  -- string representation of one JSON Identifier and value.

  do
    if hash_table .has (a_argument) then
      Result := item_to_string (a_argument)
    else
      not a raw message. valid_position (i)
      loop
        l_string_builder .append (a_raw_message .at (i).out)
        i := i + 1
      end
      make_from_string (l_string_builder , a_user_id )
      end

      user_id : INTEGER
        -- the user id from the creator of the message
to_string : STRING
        -- string representation
      local
        l_string_builder : STRING
        l_first : BOOLEAN

        do
          l_string_builder := ""
          l_first := true
          across hash_table .current_keys as key loop
            if not l_first then
              l_string_builder .append (","
            end
          l_first := false
          l_string_builder .append (item_to_string (key.item) )
        end
        Result := l_string_builder
      end

to_sending_string : STRING
  -- string representation
local
  l_string_builder : STRING
  l_first : BOOLEAN

  do
    l_string_builder := ""
    l_first := true
    across hash_table .current_keys as key loop
      if not l_first then
        l_string_builder .append (","
      end
    l_first := false
    l_string_builder .append (item_to_sending_string (key.item) )
  end
  Result := l_string_builder
end

to_string_only (a_argument: STRING): STRING
  -- string representation of one JSON Identifier and value.

  do
    if hash_table .has (a_argument) then
      Result := item_to_string (a_argument)
    else
    not a raw message. valid_position (i)
    loop
      l_string_builder .append (a_raw_message .at (i).out)
      i := i + 1
    end
    make_from_string (l_string_builder , a_user_id )
  end

  user_id : INTEGER
    -- the user id from the creator of the message
to_string : STRING
    -- string representation
  local
    l_string_builder : STRING
    l_first : BOOLEAN

    do
      l_string_builder := ""
      l_first := true
      across hash_table . current_keys as key loop
        if not l_first then
          l_string_builder .append (","
        end
      l_first := false
      l_string_builder .append (item_to_string (key.item) )
    end
    Result := l_string_builder
  end
  to_sending_string : STRING
    -- string representation
local
  l_string_builder : STRING
  l_first : BOOLEAN

  do
    l_string_builder := ""
    l_first := true
    across hash_table .current_keys as key loop
      if not l_first then
        l_string_builder .append (","
      end
    l_first := false
    l_string_builder .append (item_to_sending_string (key.item) )
  end
  Result := l_string_builder
end
  to_string_only (a_argument: STRING): STRING
    -- string representation of one JSON Identifier and value.

    do
      if hash_table .has (a_argument) then
        Result := item_to_string (a_argument)
      else
    not a raw message. valid_position (i)
    loop
      l_string_builder .append (a_raw_message .at (i).out)
      i := i + 1
    end
    make_from_string (l_string_builder , a_user_id )
  end

  user_id : INTEGER
    -- the user id from the creator of the message
to_string : STRING
    -- string representation
  local
    l_string_builder : STRING
    l_first : BOOLEAN

    do
      l_string_builder := ""
      l_first := true
      across hash_table . current_keys as key loop
        if not l_first then
          l_string_builder .append (","
        end
      l_first := false
      l_string_builder .append (item_to_string (key.item) )
    end
    Result := l_string_builder
  end

to_sending_string : STRING
    -- string representation
local
  l_string_builder : STRING
  l_first : BOOLEAN

  do
    l_string_builder := ""
    l_first := true
    across hash_table . current_keys as key loop
      if not l_first then
        l_string_builder .append (","
      end
    l_first := false
    l_string_builder .append (item_to_sending_string (key.item) )
  end
  Result := l_string_builder
end

to_string_only (a_argument: STRING): STRING
    -- string representation of one JSON Identifier and value.

    do
      if hash_table .has (a_argument) then
        Result := item_to_string (a_argument)
      else

Result := ""
end
end

is_empty:: BOOLEAN
-- has the JSON set elements?
do
Result := hash_table.count = 0
end

feature -- Basic Features

remove (a_key:: STRING)
-- remove identifier and value from JSON set.
do
hash_table.remove (a_key.as_lower)
end

put (a_key:: STRING; a_item:: STRING)
-- put identifier and value to JSON set.
do
if hash_table.has (a_key.as_lower) then
hash_table.replace (a_item, a_key.as_lower)
else
hash_table.put (a_item, a_key.as_lower)
end
end

add (a_key:: STRING; a_item:: STRING)
-- add value to an identifier in JSON set.
do
if attached hash_table.at (a_key.as_lower) as l_old_value then
  if l_old_value.ends_with ("\"\") then
    hash_table.replace (l_old_value + \"\", a_item, a_key.as_lower)
  else
    hash_table.replace (\"\" + l_old_value + \"\", a_item, a_key.as_lower)
  end
else
  hash_table.put (a_item, a_key.as_lower)
end
end

get (a_key:: STRING): STRING
-- returns value of identifier in JSON set
do
if attached hash_table.item (a_key.as_lower) as l_item then
  Result := l_item
else
  Result := ""
end
has (key: STRING): BOOLEAN
   -- has the JSON set this identifier?
do
   Result := hash_table . has (key, as_lower)
end

get_keys : ARRAY[STRING]
   -- get all identifiers.
do
   Result := hash_table . current_keys
end

consists_only_of  (a_key: STRING; a_arguments: LINKED_LIST[STRING]): BOOLEAN
   -- do the values at this identifier consist only of values passed in list?
do
   if attached hash_table . at (a_key) as l_item then
      l_item . prune_all_leading ( '{' )
      l_item . prune_all_trailing ( '}' )
      Result := across l_item . split ( ',' ) as l_element all
               across a_arguments as l_argument some l_argument.item . as_lower . is_equal ( l_element . item . as_lower )
      end
   else
      Result := false
      end
end

feature {NONE} -- Parsing

parse (a_string: STRING)
   -- parse a string to JSON, only subset of JSON is allowed.
local
   i: INTEGER
do
   from
      i := 0
   until
      i > a_string . count
   loop
      i := parse_item (a_string, i + 1)
   end
end

parse_item (a_string: STRING; a_position: INTEGER): INTEGER
   -- parse an item.
local
   l_key, l_position : INTEGER
   l_quote : CHARACTER
do
   l_quote := "!
   l_quote := l_quote . next
   a_string . prune_all (l_quote)
from
l_position := a_position
l_key := 0
until
a_string . at ( l_position ) . is_equal ( ', ' ) or
l_position > a_string . count
loop
if a_string . at ( l_position ) . is_equal ( ' ' ) then
l_position := parse_value ( a_string , l_position + 1)
if l_position <= a_string . count and not a_string . at ( l_position ) . is_equal ( ', ' )
then
add ( ( RS_DISTRIB_GUI_CONSTANTS ) . parse_error, "Commands have to be separated by ','")
elseif a_string . at ( l_position ) . is_equal ( '{ ' ) or a_string . at ( l_position ) . is_equal ( ' } ' )
or a_string . at ( l_position ) . is_equal ( ': ' ) or a_string . at ( l_position ) . is_equal ( ' } ' )
then
add ( ( RS_DISTRIB_GUI_CONSTANTS ) . parse_error, "Unexpected Bracket")
l_key := l_position
l_position := a_string . count + 1
else
l_key := l_position
l_position := l_position + 1
end
end
if l_key = 0 then
add ( ( RS_DISTRIB_GUI_CONSTANTS ) . parse_error, "Keys can not be empty")
end
add ( a_string . substring ( a_position , l_key ) , a_string . substring ( l_key + 2, l_position - 1 ) )
Result := l_position
end

parse_value ( a_string : STRING ; a_position : INTEGER ) : INTEGER
−− parse an argument.
local
l_position : INTEGER
l_in_brackets : BOOLEAN
do from
l_position := a_position
if a_string . at ( l_position ) . is_equal ( '{ ' ) then
l_in_brackets := true
l_position := l_position + 1
else
l_in_brackets := false
end
until
( a_string . at ( l_position ) . is_equal ( ',' ) and not l_in_brackets ) or
a_string . at ( l_position ) . is_equal ( ' ' ) or
l_position > a_string . count
loop
if a_string . at ( l_position ) . is_equal ( '{ ' ) and l_in_brackets then
add ([RS_DISTRIB_GUI_CONSTANTS].parse_error, "Nested Brackets are not allowed")
    l_position := a_string.count + 1
else if a_string.at (l_position).is_equal (']') or a_string.at (l_position).is_equal (')') or a_string.at (l_position).is_equal ('}') then
    add ([RS_DISTRIB_GUI_CONSTANTS].parse_error, "Unexpected character” + a_string.at (l_position).out + “"
    l_position := a_string.count + 1
else
    l_position := l_position + 1
end end

if a_string.at (l_position).is_equal ('}') and not l_in_brackets then
    add ([RS_DISTRIB_GUI_CONSTANTS].parse_error, "Bracket was not opened")
    l_position := a_string.count + 1
end
if l_position <= a_string.count and not a_string.at (l_position).is_equal (',') then
    l_position := l_position + 1
end
Result := l_position
end

feature {NONE} -- Basic private Features

item_to_string (a_key: STRING): STRING
    -- string representation of an element.
    do
        if attached hash_table.item (a_key) as l_item then
            if l_item.is_equal (""") then
                Result := a_key
            else
                Result := a_key + ";" + l_item
            end
        else
            Result := ""
        end
    end

item_to_sending_string (a_key: STRING): STRING
    -- JSON representation of an element.
    local
        l_quote: CHARACTER_8
    do
        create l_quote
        l_quote := ";"
        l_quote := l_quote.next
        if attached hash_table.item (a_key) as l_item then
            if l_item.is_equal (""") then
                Result := l_quote.out + a_key + l_quote.out
            else
                Result := l_quote.out + a_key + l_quote.out + ";" + value_to_sending_string (l_item.out)
end
else
  Result := ""
end
end

value_to_sending_string (a_value: STRING): STRING
  -- JSON representation of a value.
local
l_quote: CHARACTER 8
i: INTEGER
do
  l_quote := "\\"
  if a_value.starts_with ("{"") then
    from
      i := 2
    until
      i >= a_value.count
    loop
      if a_value.at (i).is_equal (',') or a_value.at (i).is_equal ('}') then
        if a_value.at (i + 1).is_equal ('{') then
          a_value.insert_character (l_quote, i+2)
        else
          a_value.insert_character (l_quote, i+1)
        end
      else
        a_value.insert_character (l_quote, i)
      end
      i := i + 1
    end
    a_value.insert_character (l_quote, 2)
    a_value.insert_character (l_quote, a_value.count)
  Result := a_value
  else
    Result := l_quote.out + a_value + l_quote.out
  end
end

get_key (a_message: STRING): STRING
  -- get the identifier of a parsed object.
do
  Result := a_message.split (':') .at (1).as_lower
end

get_value (a_message: STRING): STRING
  -- get the value of a parsed object.
do
  if a_message.split (':').count > 1 then
    Result := a_message.split (':').at (2)
  else
C. Code

```pascal
Result := ""
end
end

feature {NONE} -- Global Variables

hash_table: HASH_TABLE[STRING,STRING]
-- JSON set
end

C.1.11 RS_DISTRIBUT_GUI_MESSAGE_PROCESSOR

note
description :
"Messages get processed and verified here. This contains sending feedback or commands to application or sockets when needed."

class
RS_DISTRIBUT_GUI_MESSAGE_PROCESSOR

create
make

feature -- -- Access

make (a_address : STRING; a_port: STRING; a_buffer: separate RS_DISTRIBUT_GUI_BUFFER)
-- create and initialize.
do
  buffer := a_buffer
  initialize_udp_commands
  initialize_tcp_statuses
  initialize_tcp_commands
  address_value := a_address
  port_value := a_port
end

add_command_instruction (a_command_instruction:
RS_DISTRIBUT_GUI_COMMAND_INSTRUCTION)
-- add command to useable commands.
require
no_meta_command: (not a_command_instruction.name.is_equal (RS_DISTRIBUT_GUI_CONSTANTS).unregister)) and
  (not a_command_instruction.name.is_equal (RS_DISTRIBUT_GUI_CONSTANTS).register)) and
  (not a_command_instruction.name.is_equal (RS_DISTRIBUT_GUI_CONSTANTS).load)) and
  (not a_command_instruction.name.is_equal (RS_DISTRIBUT_GUI_CONSTANTS).safe)) and
  (not a_command_instruction.name.is_equal (RS_DISTRIBUT_GUI_CONSTANTS).get_user_id))
do
  if tcp_commands.has (a_command_instruction.name) then
    tcp_commands.replace(a_command_instruction, a_command_instruction.name)
```
else
    tcp_commands.put(a_command_instruction, a_command_instruction.name)
end

end

add_status (a_status : RS_DISTRIB_GUI_STATUS_INTERN)
-- add status to registerable statuses.
local
    INTEGER
    i
do
    if attached tcp_statuses . at (a_status . name) as l_status then
        l_status . set_value (a_status . get_value)
    else
        tcp_statuses . put (a_status, a_status . name)
    from
    i := 1
until
    i > {RS_DISTRIB_GUI_CONSTANTS}.max_number_of_connections
loop
    unregistered_tcp_statuses . at (i) . put ("", a_status . name)
    i := i + 1
end
end

executeStopped : BOOLEAN
\begin{center}
\begin{tabular}{|l|l|}
\hline
\textbf{executeStopped} & \textbf{BOOLEAN} \\
\hline
\end{tabular}
\end{center}
do
    Result := executeStopped_separate (buffer)
end

checkBuffer
-- check separate buffer
do
    checkBuffer_separate (buffer)
end

listenForStatusChanges (a_user_id : INTEGER): RS_DISTRIB_GUI_MESSAGE
-- check if a registered status has changed since last iteration.
local
    l_answer: RS_DISTRIB_GUI_MESSAGE
do
    create l_answer . make (a_user_id)
    across tcp_statuses as l_status loop
        if l_status . item . changed and registered_tcp_statuses [a_user_id] . has (l_status . item . name) then
            l_answer . add (l_status . item . name, l_status . item . get_value)
        end
    end
    Result := l_answer
end

resetChanged
-- sets all changed flags to false
C. Code

```
# C# code

query across tcp_statuses as l_status loop
l_status .item.set_unchanged
end

process_tcp (a_message: RS_DISTRIB_GUI_MESSAGE): RS_DISTRIB_GUI_MESSAGE
-- process parsed message from tcp socket

do
verify (a_message, tcp_commands)
if not (a_message.has ((RS_DISTRIB_GUI_CONSTANTS).error) or
a_message.has ((RS_DISTRIB_GUI_CONSTANTS).parse_error) or
a_message.has ((RS_DISTRIB_GUI_CONSTANTS).help))
then
handle_message_tcp (a_message)
end
Result := answer_message_tcp(a_message)
end

process_udp (a_message: RS_DISTRIB_GUI_MESSAGE): RS_DISTRIB_GUI_MESSAGE
-- process parsed message from udp socket.

do
verify (a_message, udp_commands)
Result := answer_message_udp (a_message)
end

clean_registered_commands (a_user_id: INTEGER)
-- reset the state of registered commands for an user id.

local
i: INTEGER

do
from i := 1 until registered_tcp_statuses [ a_user_id ].current_keys .count < i
loop
unregister_status ( registered_tcp_statuses [ a_user_id ].current_keys .at(i), a_user_id )
i := i + 1
end
end

feature {NONE} -- separate acess

check_buffer_separate (a_buffer: separate RS_DISTRIB_GUI_BUFFER)
-- check if buffer has a message and if yes process it.

local
l_message: STRING

do
if a_buffer .has_server_element then
create l_message .make_from_separate ( a_buffer .get_server )
if l_message .starts_with ("s") then
add_status (create (RS_DISTRIB_GUI_STATUS_INTERNAL).make_from_serialisation ( l_message))
elseif l_message .starts_with ("c") then

62
add_command_instruction (create {RS_DISTRIB_GUI_COMMAND_INSTRUCTION}. make_from_serialisation (l_message))
else
debug ("message_processor")
print ("ERROR wrong serialisation")
end
end
end
end

execution_stopped_separate (a_buffer : separate RS_DISTRIB_GUI_BUFFER): BOOLEAN
do
Result := a_buffer . is_stopped
end

feature {NONE} -- Initialisation Features

initialize_udp_commands
-- initialize request commands for udp.
local
l_argument_list : LINKED_LIST [STRING]
l_request_command: RS_DISTRIB_GUI_COMMAND_INSTRUCTION
do
create udp.commands.make ({RS_DISTRIB_GUI_CONSTANTS}. default_size)
create l_request_command.make ({RS_DISTRIB_GUI_CONSTANTS}. request)
create l_argument_list . make
l_argument_list . wipe_out
l_argument_list . extend ({RS_DISTRIB_GUI_CONSTANTS}. address)
l_argument_list . extend ({RS_DISTRIB_GUI_CONSTANTS}. port)
l_request_command. allow_custom_strings (l_argument_list )
l_request_command. allow_multiple
udp.commands.put (l_request_command, l_request_command.name)
end

initialize_tcp_commands
-- initialize commands that are always useable.
local
l_command: RS_DISTRIB_GUI_COMMAND_INSTRUCTION
do
create tcp.commands.make ({RS_DISTRIB_GUI_CONSTANTS}. default_size)
create l_command.make ({RS_DISTRIB_GUI_CONSTANTS}. register)
l_command.allow_multiple
tcp.commands.put (l_command, l_command.name)
create l_command.make ({RS_DISTRIB_GUI_CONSTANTS}. unregister)
l_command.allow_multiple
tcp.commands.put (l_command, l_command.name)
create l_command.make ({RS_DISTRIB_GUI_CONSTANTS}. load)
l_command.allow_multiple
tcp.commands.put (l_command, l_command.name)
create l_command.make ({RS_DISTRIB_GUI_CONSTANTS}. safe)
C. Code

```plaintext
l_command.allow_string
TCP_COMMANDS.put(l_command, l_command.name)

create l_command.make ({RS_DISTRIB_GUI_CONSTANTS}.safe)
l_command.allow_string
TCP_COMMANDS.put(l_command, l_command.name)

create l_command.make ({RS_DISTRIB_GUI_CONSTANTS}.get_user_id)
l_command.allow_none
TCP_COMMANDS.put(l_command, l_command.name)
end

initialize_tcp_statuses
--initialize empty registered statuses tables for each possible user.
local i: INTEGER
do
  create tcp_statuses . make ({RS_DISTRIB_GUI_CONSTANTS}.default_size)
  create registered_tcp_statuses . make_filled (create [HASH_TABLE[string, string]].make ({RS_DISTRIB_GUI_CONSTANTS}.default_size), 1, {RS_DISTRIB_GUI_CONSTANTS}.max_number_of_connections)
  create unregistered_tcp_statuses . make_filled (create [HASH_TABLE[string, string]].make ({RS_DISTRIB_GUI_CONSTANTS}.default_size), 1, {RS_DISTRIB_GUI_CONSTANTS}.max_number_of_connections)
  create saved_statuses . make ({RS_DISTRIB_GUI_CONSTANTS}.default_size)
from
  i := 1
until
  i > {RS_DISTRIB_GUI_CONSTANTS}.max_number_of_connections
loop
  registered_tcp_statuses . at (i) := create [HASH_TABLE[string, string]].make ({RS_DISTRIB_GUI_CONSTANTS}.default_size)
  unregistered_tcp_statuses . at (i) := create [HASH_TABLE[string, string]].make ({RS_DISTRIB_GUI_CONSTANTS}.default_size)
  i := i + 1
end
end

feature {NONE} -- Communication Features

answer_message_tcp (a_message: RS_DISTRIB_GUI_MESSAGE):
  {RS_DISTRIB_GUI_MESSAGE}
  --create the message that is sent back via tcp.
local l_answer: RS_DISTRIB_GUI_MESSAGE
do
  create l_answer.make (a_message . user_id)
  if a_message . get ({RS_DISTRIB_GUI_CONSTANTS}.error).is_equal (""") and
    a_message . get ({RS_DISTRIB_GUI_CONSTANTS}.parse_error).is_equal (""") and
    not a_message . has ({RS_DISTRIB_GUI_CONSTANTS}.help) then
    l_answer . put ({RS_DISTRIB_GUI_CONSTANTS}.ok, """)
  elseif not a_message . get ({RS_DISTRIB_GUI_CONSTANTS}.parse_error).is_equal ("")
    then
```

64
C.1. RS_DISTRIB_GUI

```lisp
answer.put ({RS_DISTRIB_GUI_CONSTANTS}.parse_error, a_message.get ({
    RS_DISTRIB_GUI_CONSTANTS}.parse_error))
else if not a_message.get ({RS_DISTRIB_GUI_CONSTANTS}.error).is_equal ("") then
    answer.put ({RS_DISTRIB_GUI_CONSTANTS}.error, a_message.get ({
        RS_DISTRIB_GUI_CONSTANTS}.error))
end

if a_message.has ({RS_DISTRIB_GUI_CONSTANTS}.get_user_id) and answer.has ({
    RS_DISTRIB_GUI_CONSTANTS}.ok) then
    answer.put ({RS_DISTRIB_GUI_CONSTANTS}.get_user_id, answer.user_id.out)
end

if a_message.has ({RS_DISTRIB_GUI_CONSTANTS}.help) then
    if attached tcp_commands.at (a_message.get ({RS_DISTRIB_GUI_CONSTANTS}.help)) as l_command then
        answer.put ({RS_DISTRIB_GUI_CONSTANTS}.help, "{" + one_command_to_string (l_command, answer.user_id) + "}")
    else
        answer.put ({RS_DISTRIB_GUI_CONSTANTS}.help, help_to_string (tcp_commands, answer.user_id))
    end
end

Result := answer

answer_message_udp (a_message: RS_DISTRIB_GUI_MESSAGE):
    -- create the message that is sent back via udp.
local
    l_answer: RS_DISTRIB_GUI_MESSAGE
do
    create l_answer.make (a_message.user_id)
    if a_message.get ({RS_DISTRIB_GUI_CONSTANTS}.request).has_substring ({
        RS_DISTRIB_GUI_CONSTANTS}.port) then
        l_answer.put ({RS_DISTRIB_GUI_CONSTANTS}.port, port.value)
    end
    if a_message.get ({RS_DISTRIB_GUI_CONSTANTS}.request).has_substring ({
        RS_DISTRIB_GUI_CONSTANTS}.address) then
        l_answer.put ({RS_DISTRIB_GUI_CONSTANTS}.address, address.value)
    end
Result := l_answer
end

handle_message_tcp (a_message: RS_DISTRIB_GUI_MESSAGE)
    -- send verified message to application or/and execute commands in message on server
do
    if a_message.has ({RS_DISTRIB_GUI_CONSTANTS}.register) then
        across a_message.get ({RS_DISTRIB_GUI_CONSTANTS}.register).split (",") as l_status
            loop
                register_status (l_status.item, a_message.user_id)
            end
        a_message.remove({RS_DISTRIB_GUI_CONSTANTS}.register)
    end
end
```
end
if a_message has (RS_DISTRIB_GUI_CONSTANTS).unregister then
  across a_message.get (RS_DISTRIB_GUI_CONSTANTS).unregister.split (',') as
    l_status
    loop
      unregister_status (l_status.item, a_message.user_id)
    end
  a_message.remove (RS_DISTRIB_GUI_CONSTANTS).unregister
end
if a_message has (RS_DISTRIB_GUI_CONSTANTS).safe then
  safe_state (a_message.get (RS_DISTRIB_GUI_CONSTANTS).safe), a_message.user_id
  a_message.remove (RS_DISTRIB_GUI_CONSTANTS).safe
end
if a_message has (RS_DISTRIB_GUI_CONSTANTS).load then
  load_state (a_message.get (RS_DISTRIB_GUI_CONSTANTS).load), a_message.user_id
  a_message.remove (RS_DISTRIB_GUI_CONSTANTS).load
end
across a_message.get keys as l_key loop
  if not l_key.item.is_equal (RS_DISTRIB_GUI_CONSTANTS).get_user_id then
    send_command (create (RS_DISTRIB_GUI_COMMAND).make (l_key.item, a_message.get (l_key.item), a_message.user_id), buffer)
  end
end
end

feature {NONE} -- Command Features

send_command (a_command: RS_DISTRIB_GUI_COMMAND; a_buffer: separate RS_DISTRIB_GUI_BUFFER)
  -- send a command to the buffer/application.
do
  a_buffer.set_application (a_command.serialize)
end

register_status (a_status_name: STRING; a_user_id: INTEGER)
  -- register status for a specific user id. Changes to this command will be sent to the user.
do
  if unregistered_tcp_statuses.at(a_user_id).has(a_status_name) then
    if attached tcp_statuses.at(a_status_name) as l_status then
      l_status.set_changed
    end
  end
  unregister_tcp_statuses.at(a_user_id).remove(a_status_name)
  registered_tcp_statuses.at(a_user_id).put("", a_status_name)
end

unregister_status (a_status_name: STRING; a_user_id: INTEGER)
  -- unregister status for a specific user id.
do
  if registered_tcp_statuses.at(a_user_id).has(a_status_name) then
    registered_tcp_statuses.at(a_user_id).remove(a_status_name)
end
end
unregistered_tcp_statuses .at (a_user_id) .put ("", a_status_name)
end
end

safe_state  ( a_identifier : STRING; a_user_id: INTEGER)
local
  l_hash_table : HASH_TABLE[STRING,STRING]
do
  create l_hash_table.make ((RS_DISTRIB_GUI_CONSTANTS).default_size)
  across
    registered_tcp_statuses .at (a_user_id) .current_keys as key
  loop
    l_hash_table.put ("", key.item)
  end
  saved_statuses .put (l_hash_table, a_identifier)
end

load_state  ( a_identifier : STRING; a_user_id: INTEGER)
do
  debug ("message_processor")
  print ("try to load %N")
end

clean_registered_commands (a_user_id)
if attached saved_statuses .at (a_identifier) as l_saved_statuses then
  across
    l_saved_statuses .current_keys as key
  loop
    debug ("message_processor")
    print ("I register " + key.item.out +"%N")
  end
end
end

feature {NONE} -- Verification Features

verify (a_message: RS_DISTRIB_GUI.MESSAGE; a_commands: HASH_TABLE[RS_DISTRIB_GUI.COMMAND_INSTRUCTION,STRING])
-- verify if the message meets all requirements. Generate error messages if not.
local
  l_items : STRING
do
  across a_message.get_keys as l_key loop
    if l_key.item.is_equal ((RS_DISTRIB_GUI_CONSTANTS).help) or
      l_key.item.is_equal ((RS_DISTRIB_GUI_CONSTANTS).parse_error) or
      l_key.item.is_equal ((RS_DISTRIB_GUI_CONSTANTS).safe) then
      -- Do Nothing
    elseif attached a_commands.at(l_key.item) as l_command then
      l_items := a_message.get (l_key.item)
      l_items.prune_all_leading (""
      l_items.prune_all_trailing ("")
      if not l_command.multiple_allowed and l_items.split (",") .count > 1 then

a_message.add({RS_DISTRIB_GUI_CONSTANTS}.error, l_key.item + " does not allow multiple arguments")
else
   across l_items . split (',') as l_item loop
      verify_argument (a_message, l_command, l_item.item)
   end
end
else
   a_message.add({RS_DISTRIB_GUI_CONSTANTS}.error, "Unknown Identifier(" + l_key.item + ")")
end
end

verify_register_argument (a_message: RS_DISTRIB_GUI_MESSAGE; a_argument: STRING)
-- verify if register argument exists.
do
   if not across unregistered_tcp_statuses .at (a_message.user_id).current_keys as key some key.item is equal (a_argument) end then
      add_bad_argument (a_message, a_argument, hash_table_to_string (unregistered_tcp_statuses .at (a_message.user_id)))
   end
end

verify_unregister_argument (a_message: RS_DISTRIB_GUI_MESSAGE; a_argument: STRING)
-- verify if unregister argument exists.
do
   if not across registered_tcp_statuses .at (a_message.user_id).current_keys as key some key.item is equal (a_argument) end then
      add_bad_argument (a_message, a_argument, hash_table_to_string (registered_tcp_statuses .at (a_message.user_id)))
   end
end

verify_load_argument (a_message: RS_DISTRIB_GUI_MESSAGE; a_argument: STRING)
-- verify if load argument exists.
do
   if not across saved_statuses .current_keys as key some key.item is equal (a_argument) end then
      add_bad_argument (a_message, a_argument, hash_table_to_string (saved_statuses ))
   end
end

verify_argument (a_message: RS_DISTRIB_GUI_MESSAGE; a_command: RS_DISTRIB_GUI_COMMAND_INSTRUCTION; a_argument: STRING)
-- verify if argument meets requirements.
do
   if a_command.name.is_equal({RS_DISTRIB_GUI_CONSTANTS}.register) then
      verify_register_argument (a_message, a_argument)
   elseif a_command.name.is_equal({RS_DISTRIB_GUI_CONSTANTS}.unregister) then
      verify_unregister_argument (a_message, a_argument)
   elseif a_command.name.is_equal({RS_DISTRIB_GUI_CONSTANTS}.load) then
      verify_load_argument (a_message, a_argument)
   end
end
C.1. RS_DISTRI_B_GUI

```plaintext
elseif a_command.string.allowed
    or (a_command.integer.allowed and a_argument.is_integer)
then
elseif a_command.range.allowed and a_argument.is_integer
if a_argument.to_integer >= a_command.lower
    and a_argument.to_integer <= a_command.upper
else
    a_message.add([{RS_DISTRI_B_GUI_CONSTANTS}.error,"Argument not in range("
        + a_argument + ";", + a_command.to_string_arguments + ")"
    )
end
elseif a_command.has_custom_string(a_argument)
else
    a_message.add([{RS_DISTRI_B_GUI_CONSTANTS}.error,"No argument is allowed"
    )
end
add_bad_argument(a_message, a_argument, a_command.to_string_arguments)
end
end

add_bad_argument(a_message: RS_DISTRI_B_GUI_MESSAGE; a.received: STRING; a.expected: STRING)
-- adds a "bad_argument" error to the message.
do
    a_message.add([{RS_DISTRI_B_GUI_CONSTANTS}.error,"Bad Argument(argument
        received: " + a.received + "; argument expected: " + a.expected + ")"
    )
end

feature {NONE} -- Serialisation Features

help_to_string (a.commands: HASH_TABLE[
    RS_DISTRI_B_GUI_COMMAND_INSTRUCTION,STRING]; a.user_id: INTEGER):
    STRING
    -- generate help message.
local
    l_result_string : STRING
    do
        l_result_string := "{help:STRING"
        across a.commands as l_command
            l_result_string.append("", + one_command_to_string(l_command.item, a.user_id))
        end
        Result := l_result_string + "}"
end

one_command_to_string (a.command: RS_DISTRI_B_GUI_COMMAND_INSTRUCTION;
    a.user_id: INTEGER): STRING
    do
        if a.command.name.is_equal([{RS_DISTRI_B_GUI_CONSTANTS}.register)
            Result := [{RS_DISTRI_B_GUI_CONSTANTS}.register + "" + hash_table.to_string(has_unregistered_tcp_statuses .at (a.user_id))
        elseif a.command.name.is_equal([{RS_DISTRI_B_GUI_CONSTANTS}.unregister)
            Result := [{RS_DISTRI_B_GUI_CONSTANTS}.unregister + "" + hash_table.to_string(has_registered_tcp_statuses .at (a.user_id))
        elseif a.command.name.is_equal([{RS_DISTRI_B_GUI_CONSTANTS}.load)
```

69
\textbf{Result} := \{RS\_DISTRIB\_GUI\_CONSTANTS\}.load + ":" + hash_table_to_string (saved_statuses)

\textbf{else}
\begin{itemize}
  \item \textbf{Result} := a\_command\_to\_string
\end{itemize}
\textbf{end}
\textbf{end}

\textbf{hash\_table\_to\_string} \ (a\_hash\_table : HASH\_TABLE[ANY,STRING]): STRING
\textit{-- string representation of a hashtable.}
\begin{itemize}
  \item \textbf{local}
  \begin{itemize}
    \item \textbf{l\_string} : STRING
    \item \textbf{key\_set} : ARRAY[STRING]
  \end{itemize}
  \item \textbf{do}
  \begin{itemize}
    \item \textbf{key\_set} := a\_hash\_table . current\_keys
    \item \textbf{l\_string} := ""
    \item if \textbf{key\_set . count} = 0 then
      \item \textbf{l\_string} := "NONE"
    \item elseif \textbf{key\_set . count} = 1 then
      \item \textbf{l\_string} := key\_set . at (1).out
    \item elseif \textbf{key\_set . count} > 1 then
      \item \textbf{l\_string} . append ("(")
      \item across key\_set as key \textbf{loop}
      \item if key . item . is\_equal ("") then
        \item \textbf{l\_string} . append ("",")
      \item else
        \item \textbf{l\_string} . append (key . item + ",")
      \item end
    \item end
    \item \textbf{l\_string} . prune\_all\_trailing (',')
    \item \textbf{l\_string} . append (")")
  \item end
  \item \textbf{Result} := l\_string
\end{itemize}
\end{itemize}

\textbf{array\_to\_string} \ (a\_array : ARRAY[ANY]): STRING
\textit{-- string representation of an array.}
\begin{itemize}
  \item \textbf{local}
  \begin{itemize}
    \item \textbf{l\_string} : STRING
  \end{itemize}
  \item \textbf{do}
  \begin{itemize}
    \item \textbf{l\_string} := ""
    \item across a\_array as item \textbf{loop}
    \item \textbf{l\_string} . append (item . item . out + ",")
  \item end
  \item \textbf{l\_string} . prune\_all\_trailing (',')
  \item \textbf{Result} := l\_string
\end{itemize}
\end{itemize}

\textbf{feature \{NONE\}} \textit{-- Global Variables}
\begin{itemize}
  \item \textbf{address\_value} : STRING
  \textit{-- ip address received from ROBOSCOOP\_CONNECTION for tcp socket.}
  \item \textbf{port\_value} : STRING
  \textit{-- port number received from ROBOSCOOP\_CONNECTION for tcp socket.}
\end{itemize}
C.1. RS_DISTRIBUT_GUI

**tcp commands:** HASH_TABLE[RS_DISTRIBUT_GUI_COMMAND_INSTRUCTION,STRING]

-- table of all useable commands via tcp socket.

**udp commands:** HASH_TABLE[RS_DISTRIBUT_GUI_COMMAND_INSTRUCTION,STRING]

-- table of all useable commands via udp socket.

**tcp statuses:** HASH_TABLE[RS_DISTRIBUT_GUI_STATUS_INTERN,STRING]

-- table of all useable statuses via tcp socket.

**saved statuses:** HASH_TABLE[HASH_TABLE[STRING,STRING],STRING]

-- table of all safed status sets.

**registered tcp statuses:** ARRAY[HASH_TABLE[STRING,STRING]]

-- table of all registered tcp statuses.

**unregistered tcp statuses:** ARRAY[HASH_TABLE[STRING,STRING]]

-- table of all unregistered tcp statuses.

**buffer:** separate RS_DISTRIBUT_GUI_BUFFER

-- separate buffer

---

C.1.12 RS_DISTRIBUT_GUI_SERVER

**note**

description:

"The Server connects the Connection and the Message Processor classes. It listens to sockets and sends the messages to the Message Processor."

class

RS_DISTRIBUT_GUI_SERVER

inherit

EXECUTION_ENVIRONMENT

create

make

feature -- -- Access

make ( a_buffer : separate RS_DISTRIBUT_GUI_BUFFER; a_address: separate STRING;
   a_stream_socket_port: INTEGER; a_datagram_socket_port: INTEGER)

-- create.

local

l_address : STRING

do

l_address := create {STRING}.make_from_separate (a_address)

create connection.make (l_address, a_stream_socket_port, a_datagram_socket_port)

create message_processor.make (connection.get_address, connection.get_port, a_buffer)

end

run
run server.
do
running := true
debug ("server")
print ("Server startet!\n")
end
from until not running loop
listen_to_udp
listen_to_connection_changes
listen_to_tcp
listen_to_buffer
sleep (\{RS_DISTRIB_GUI_CONSTANTS\}.wait_server * 1,000,000)
end
connection_cleanup
debug ("server")
print ("Server stopped\n")
end
rescue
connection_cleanup
end
stop_server
--- stop the loop, the application will terminate
do
running := false
end
feature \{NONE\} --- Basic Features

listen_to_udp
--- listen to udp socket and process possible messages.
local
l_message: RS_DISTRIB_GUI_MESSAGE
do
if connection.is_ready_to_read_udp then
l_message := connection.read_from_udp
l_message := message_processor.process_udp (l_message)
connection.send_to_udp (l_message.to_sending_string)
end
end

listen_to_connection_changes
--- listen for new tcp connections.
do
connection.accept_tcp_socket
end

listen_to_tcp
--- listen to tcp socket and process possible messages.
local
l_message: RS_DISTRIB_GUI_MESSAGE
do
across connection.get_user_ids as l_user_id loop
if connection.is_ready_to_read_tcp (l_user_id.item) then
C.1. RS_DISTRIB_GUI

C.1.13 RS_DISTRIB_GUI_STATUS

Class RS_DISTRIB_GUI_STATUS

Create make

Feature -- Access

note
description:
"A status which is used to send Updates from Application to Server."

make (a_name: STRING; a_value: STRING)
   -- create.
   do
       name := a_name
       value := a_value
   ensure
       no_forbidden_characters_in_name : not name.has (',' ) and not name.has (';')
       no_forbidden_characters_in_value : not value.has (',' ) and not value.has (';')
   end

name: STRING
   -- name of status.

value: STRING
   -- value of status.

serialize : STRING
   -- serialize object to make a separate into a non separate.
   do
       Result := "s:" + name + "r:" + value
   end

set_value (a_value: STRING)
   do
       value := a_value
   ensure
       no_forbidden_characters_in_value : not a_value.has (',' ) and not a_value.has (';')
   end

C.1.14 RS_DISTRIB_GUI_STATUS_INTERN

note
description : "A status which is used to know when to send updates from the server to the GUIs."

class
RS_DISTRIB_GUI_STATUS_INTERN

create
   make_from_name_and_value, make_from_serialisation

feature -- -- Access

make_from_name_and_value (a_name: STRING; a_value: STRING)
   -- create.
   do
       name := a_name
       value := a_value
       changed := true
   end

make_from_serialisation (a_string : STRING)
C.1. RS_DISTRIBUT_GUI

--- create from serialisation, used to handle separate objects.

```haskell
require
  a_string . starts_with ("s")
do
  name := a_string . split (':') . at (2)
  value := a_string . split (':') . at (3)
  changed := true
end

name: STRING
  --- name of status.

dchanged: BOOLEAN
  --- has status changed?

feature  --- Basic Features

set_value  (a_value: STRING)
  --- set a value, if not equal to old one set changed to true.
do
  if not value . is_equal (a_value) then
    value := a_value
    changed := true
  end
end

get_value: STRING
  --- get value and set changed to false.
do
  Result := value
end

serialize: STRING
  --- serialize object to make a separate into a non separate.
do
  Result := "s:" + name + ":" + value + ":" + changed . out
end

set_unchanged
  --- sets fvalue of changed flag to false
do
  changed := false
end

set_changed
  --- sets fvalue of changed flag to true
do
  changed := true
end

feature {NONE}  --- Global Variables

value: STRING
  --- value of status.
C. Code

C.2 RS_DISTRIBUT_GUI_JAVA_GUI

C.2.1 CONNECTION

```java
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.Socket;
import java.net.SocketException;
import org.eclipse.swt.widgets.Display;

public class Connection {
    Graphics graphics;
    Socket socket;
    DatagramSocket datagramSocket;
    Boolean connected;
    Display display;
    Boolean running;
    Long broadcastInterval = 5000L;
    Integer rep = 0;
    String address;
    Integer port;

    public Connection(Graphics graphics, Display display) {
        this.graphics = graphics;
        this.display = display;
        running = false;
        graphics.setStatus(0);
        try {
            datagramSocket = new DatagramSocket();
            datagramSocket.setBroadcast(true);
            datagramSocket.setSoTimeout(1);
        } catch (SocketException e) {
            graphics.sendRedirectToConsole("ERROR: Failed to initialize Socket.");
        }
    }

    public void run() {
        running = true;
        connected = false;
        graphics.setStatus(1);
        Thread thread = new Thread() {
            @Override
```
public void run () {
    while (running) {
        display.syncExec (new Runnable () {
            @Override
            public void run () {
                if (!connected && running) {
                    cleanSocket () ;
                    sendBroadcast () ;
                    listenForBroadcast () ;
                } else if (connected && running){
                    listenForMessage () ;
                }
                cleanSocket () ;
            }
        });
        cleanSocket () ;
    }
    thread.start ();
}
public void stop () {
    running = false;
    connected = false;
    graphics.setStatus (0);
    graphics.sendToConsole("INFO: Server stopped.");
}
public void cleanSocket () {
    if (socket != null) {
        try {
            socket.close ();
        } catch (IOException e) {
            graphics.sendToConsole("ERROR: Could not close Socket.");
        }
    }
}
private void listenForBroadcast() {
    try {
        DatagramPacket receivePacket = new DatagramPacket(new byte[75], 75);
        try {
            datagramSocket.receive(receivePacket);
            byte[] recData = receivePacket.getData();
            String result = new String(recData);
            result = result.substring(0, result.lastIndexOf(“,eof”));
            System.out.println(result);
            int i = 0;
            String stringBuilder = “”;
            while (i < result.length()) {
                if (result.charAt(i) != ‘’)
                    stringBuilder += result.charAt(i);
                i++;
            }
            result = stringBuilder;
            graphics.sendToConsole(“UDP:” + result + “———”);
            for (String element: result.split(“,”)) {
                if (element.startsWith(“port”))
                    port = Integer.parseInt(element.split(“:”)[1]);
                else if (element.startsWith(“address”))
                    address = element.split(“:”)[1];
            }
            try {
                socket = new Socket(address, port);
                connected = true;
                graphics.setStatus(2);
                sendMessage(“register: {x,y,speed,angular_speed,angular_rotation}”);
            } catch (IOException e) {
                e.printStackTrace();
                graphics.sendToConsole(“ERROR: Failed to generate socket”);
            }
        } catch (IOException e1) {
            e.printStackTrace();
            graphics.sendToConsole(“ERROR: Failed to generate socket”);
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}

private void listenForMessage() {
    try {
        if (socket.getInputStream().available() > 0) {
            BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
            String inputLine;
            inputLine = in.readLine();
            parseMessages(inputLine);
        }
    } catch (IOException e1) {
    }
}
public void sendMessage(String message) {
    try {
        if (connected) {
            PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
            out.print(message);
            out.println();
        }
    } catch (IOException e) {
        graphics.sendToConsole("ERROR: Failed to generate output writer.");
    }
}

public void parseMessages(String message) {
    int i = 0;
    int begin = 0;
    int inBrackets = 0;
    while (i < message.length()) {
        if (message.charAt(i) == '{')
            inBrackets++;
        else if (message.charAt(i) == '}')
            inBrackets--;
        else if (message.charAt(i) == ',' && inBrackets == 0) {
            parseMessage(message.substring(begin, i));
            begin = i + 1;
        }
        i++;
    }
    parseMessage(message.substring(begin, i));
}

public void parseMessage(String message) {
    String stringbuilder = "";
    int i = 0;
    while (i < message.length()) {
        if (message.charAt(i) != '\n')
            stringbuilder += message.charAt(i);
        i++;
    }
    executeMessage(stringbuilder);
}

public void executeMessage(String message) {
    if (message.startsWith("x:"))
        graphics.setX(Integer.parseInt(message.substring(2)));
    else if (message.startsWith("y:"))
        graphics.setY(Integer.parseInt(message.substring(2)));
    else if (message.startsWith("z:"))
        graphics.setZ(Integer.parseInt(message.substring(2)));
    else if (message.startsWith("scale:"))
        graphics.setScale(message.substring(7));
}

graphics.sendToConsole("ERROR: Failed to generate input writer.");
graphics.setY(Integer.parseInt(message.substring(2)));  
else if (message.startsWith("speed:"))  
    graphics.setSpeed(Integer.parseInt(message.substring(6)));  
else if (message.startsWith("angular speed:"))  
    graphics.setAngularSpeed(Integer.parseInt(message.substring(14)));  
else if (message.startsWith("angular rotation:"))  
    graphics.setAngularRotation(Double.parseDouble(message.substring(17)));  
else if (message.startsWith("ok"));  
else  
    graphics.sendToConsole("unknown:" + message);

C.2.2 GRAPHICS

import java.util.LinkedList;
import org.eclipse.swt.SWT;
import org.eclipse.swt.events.KeyAdapter;
import org.eclipse.swt.events.KeyEvent;
import org.eclipse.swt.events.MouseAdapter;
import org.eclipse.swt.events.MouseEvent;
import org.eclipse.swt.events.SelectionAdapter;
import org.eclipse.swt.events.SelectionEvent;
import org.eclipse.swt.graphics.Point;
import org.eclipse.swt.layout.FormAttachment;
import org.eclipse.swt.layoutFormData;
import org.eclipse.swt.layout.FormLayout;
import org.eclipse.swt.widgets.Button;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Label;
import org.eclipse.swt.widgets.Scale;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.swt.widgets.Text;
import org.eclipse.ui.forms.widgets.FormToolkit;
import org.eclipse.wb.swt.SWTResourceManager;

public class Graphics {

    protected Shell shell;
    private Text debugConsole;
    private Text inputTextField;
    private Scale scaleRotationSpeed;
    private final FormToolkit formToolkit = new FormToolkit(Display.getDefault());
    private Connection connection;
    private Scale scaleSpeed;
    private Composite rotation;
    private Composite robot;
    private Composite map;

    /**
     * Launch the application.
     */

}
public static void main(String[] args) {
    try {
        Graphics window = new Graphics();
        window.open();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

/**
 * Open the window.
 */
public void open() {
    final Display display = Display.getDefault();
    createContents();
    shell.open();
    shell.layout();
    connection = new Connection(this, display);
    while (!shell.isDisposed()) {
        if (!display.readAndDispatch()) {
            display.sleep();
        }
    }
}

/**
 * Create contents of the window.
 */
protected void createContents() {
    shell = new Shell();
    shell.setMinimunSize(new Point(600, 900));
    shell.setSize(450, 300);
    shell.setText("SWT Application");
    shell.setLayout(new FormLayout());
    debugConsole = new Text(shell, SWT.BORDER | SWT.READ_ONLY | SWT.MULTI);
    FormData fd_debugConsole = new FormData();
    fd_debugConsole.bottom = new FormAttachment(100, -10);
    fd_debugConsole.right = new FormAttachment(0, 588);
    debugConsole.setLayoutData(fd_debugConsole);
    debugConsole.setBackground(SWTResourceManager.getColor(SWT.COLOR_INFO_FOREGROUND));
    scaleSpeed = new Scale(shell, SWT.NONE);
    FormData fd_scaleSpeed = new FormData();
    fd_scaleSpeed.right = new FormAttachment(100, -421);
    fd_scaleSpeed.left = new FormAttachment(0, 36);
    scaleSpeed.setLayoutData(fd_scaleSpeed);
C. Code

```java
scaleSpeed.setEnabled(false);

inputTextField = new Text(shell, SWT.BORDER);
inputTextField.addListener(new KeyAdapter {
    @Override
    public void keyPressed(KeyEvent e) {
        if (e.character == 13) {
            connection.sendMessage(inputTextField.getText());
        }
    }
});
FormData fd_inputTextField = new FormData();
fd_inputTextField.right = new FormAttachment(0, 549);
fd_inputTextField.top = new FormAttachment(0, 516);
inputTextField.setLayoutData(fd_inputTextField);

scaleRotationSpeed = new Scale(shell, SWT.NONE);
scaleRotationSpeed.addMouseListener(new MouseAdapter {
    @Override
    public void mouseUp(MouseEvent e) {
        connection.sendMessage("set angular speed:" + scaleRotationSpeed.getSelection());
    }
});
FormData fd_scaleRotationSpeed = new FormData();
fd_scaleRotationSpeed.bottom = new FormAttachment(100, -192);
fd_scaleRotationSpeed.left = new FormAttachment(0, 36);
scaleRotationSpeed.setLayoutData(fd_scaleRotationSpeed);

Label lblSpeed = new Label(shell, SWT.NONE);
FormData fd_lblSpeed = new FormData();
fd_lblSpeed.bottom = new FormAttachment(scaleSpeed, -6);
fd_lblSpeed.left = new FormAttachment(0, 92);
fd_lblSpeed.right = new FormAttachment(0, 141);
lblSpeed.setLayoutData(fd_lblSpeed);
lblSpeed.setText("Speed");

Label lblRotationspeed = new Label(shell, SWT.NONE);
fd_scaleRotationSpeed.top = new FormAttachment(lblRotationspeed, 1);
FormData fd_lblRotationspeed = new FormData();
fd_lblRotationspeed.left = new FormAttachment(0, 60);
fd_lblRotationspeed.bottom = new FormAttachment(100, -220);
lblRotationspeed.setLayoutData(fd_lblRotationspeed);
lblRotationspeed.setText("Rotation Speed");

map = formToolkit.createComposite(shell, SWT.NO_REDRAW_RESIZE);
map.setBackground(SWTResourceManager.getColor(SWT.COLOR_WHITE));
FormData fd_map = new FormData();
fd_map.bottom = new FormAttachment(0, 510);
fd_map.right = new FormAttachment(0, 549);
fd_map.top = new FormAttachment(0, 10);
fd_map.left = new FormAttachment(0, 49);
map.setLayoutData(fd_map);
```
```java
formToolkit.paintBordersFor(map);

robot = formToolkit.createComposite(map, SWT.NONE);
robot.setBackground(SWTResourceManager.getColor(SWT.COLOR_RED));
robot.setBounds(243, 243, 20, 20);
formToolkit.paintBordersFor(robot);
robot.setLayout(null);

rotation = formToolkit.createComposite(robot, SWT.NONE);
rotation.setBackground(SWTResourceManager.getColor(SWT.COLOR_BLACK));
rotation.setBounds(0, 0, 5, 5);
formToolkit.paintBordersFor(rotation);
rotation.setLayout(null);

fd_scaleRotationSpeed.right = new FormAttachment(100, -421);

Button btnAccelerate = new Button(shell, SWT.NONE);
btnAccelerate.addMouseListener(new MouseAdapter() {
    @Override
    public void mouseDown(MouseEvent e) {
        connection.sendMessage("start_accelerate");
    }
    @Override
    public void mouseUp(MouseEvent e) {
        connection.sendMessage("stop_changing_speed");
    }
});

FormData fd_btnAccelerate = new FormData();
fd_btnAccelerate.left = new FormAttachment(0, 303);
fd_btnAccelerate.right = new FormAttachment(inputTextField, 6);
btnAccelerate.setLayoutData(fd_btnAccelerate);
btnAccelerate.addSelectionListener(new SelectionAdapter() {
    @Override
    public void widgetSelected(SelectionEvent e) {
    }
});
btnAccelerate.setText("Accelerate");

Button btnBrake = new Button(shell, SWT.NONE);
btnBrake.addMouseListener(new MouseAdapter() {
    @Override
    public void mouseDown(MouseEvent e) {
        connection.sendMessage("start_brake");
    }
    @Override
    public void mouseUp(MouseEvent e) {
        connection.sendMessage("stop_changing_speed");
    }
});
```
FormData fd_btnBrake = new FormData();
fd_btnBrake.right = new FormAttachment(btnAccelerate, 0, SWT.RIGHT);
fd_btnBrake.left = new FormAttachment(0, 303);
btnBrake.setLayoutData(fd_btnBrake);
btnBrake.addSelectionListener (new SelectionAdapter () {
    @Override
    public void widgetSelected (SelectionEvent e) {

    }
});
btnBrake.setText("Brake");

Button btnLeft = new Button(shell , SWT.NONE);
fd_btnBrake.bottom = new FormAttachment(btnLeft, 56, SWT.BOTTOM);
fd_btnBrake.top = new FormAttachment(btnLeft, 6);
fd_btnAccelerate.bottom = new FormAttachment(btnLeft, -6);
fd_lblRotationspeed.right = new FormAttachment(btnLeft, -44);
FormData fd_btnLeft = new FormData();
fd_btnLeft.bottom = new FormAttachment(lblRotationspeed, 0, SWT.BOTTOM);
fd_btnLeft.left = new FormAttachment(0, 245);
fd_btnLeft.top = new FormAttachment(0, 605);
btnLeft.setLayoutData(fd_btnLeft);
btnLeft.addMouseListener(new MouseAdapter() {
    @Override
    public void mouseDown(MouseEvent e) {
        connection.sendMessage("start_turn_left");
    }
    @Override
    public void mouseUp(MouseEvent e) {
        connection.sendMessage("stop_turn");
    }
});
btnLeft.setText("Left");

Button btnStop = new Button(shell , SWT.NONE);
btnStop.addMouseListener(new MouseAdapter() {
    @Override
    public void mouseDown(MouseEvent e) {
        connection.sendMessage("stop");
    }
});
fd_btnLeft.right = new FormAttachment(btnStop, -6);
FormData fd_btnStop = new FormData();
fd_btnStop.left = new FormAttachment(0, 351);
fd_btnStop.top = new FormAttachment(btnAccelerate, 6);
fd_btnStop.bottom = new FormAttachment(lblRotationspeed, 0, SWT.BOTTOM);
btnStop.setLayoutData(fd_btnStop);
btnStop.setText("Stop");

Button btnRight = new Button(shell , SWT.NONE);
btnRight.addMouseListener(new MouseAdapter() {
    @Override
    public void mouseDown(MouseEvent e) {

    }
});

84
connection.sendMessage("start_turn_right");
}
@override
public void mouseUp(MouseEvent e) {
    connection.sendMessage("stop_turn");
}
}};
fd_btnStop.right = new FormAttachment(btnRight, −6);
FormData fd_btnRight = new FormData();
fd_btnRight.bottom = new FormAttachment(debugConsole, −72);
fd_btnRight.top = new FormAttachment(btnAccelerate, 6);
fd_btnRight.right = new FormAttachment(inputTextField, 0, SWT.RIGHT);
btnRight.setLayoutData(fd_btnRight);
btnRight.addSelectionListener (new SelectionAdapter () {
    @override
    public void widgetSelected (SelectionEvent e) {
    }
});
btnRight.setText("Right");
connectionInfo = new Text (shell, SWT.BORDER);
fd_debugConsole.top = new FormAttachment(0, 727);
connectionInfo.setEnabled (false);
connectionInfo.setEditable (false);
connectionInfo.setBackground(SWTResourceManager.getColor (SWT.COLOR_GREEN));
FormData fd_connectionInfo = new FormData();
fd_connectionInfo.right = new FormAttachment(scaleRotationSpeed, 35, SWT.RIGHT);
fd_connectionInfo.left = new FormAttachment(scaleRotationSpeed, −75);
fd_connectionInfo.top = new FormAttachment(0, 689);
connectionInfo.setLayoutData(fd_connectionInfo);
formToolkit.adapt(connectionInfo, true, true);
Button btnRunServer = new Button(shell, SWT.NONE);
btnRunServer.addMouseListener(new MouseAdapter () {
    @override
    public void mouseDown(MouseEvent e) {
        if (!connection.running) {
            connection.run();
        } else {
            connection.stop();
        }
    }
});
FormData fd_btnRunServer = new FormData();
fd_btnRunServer.right = new FormAttachment(connectionInfo, −6);
fd_btnRunServer.top = new FormAttachment(scaleRotationSpeed, 6);
btnRunServer.setLayoutData(fd_btnRunServer);
formToolkit.adapt (btnRunServer, true, true);
btnRunServer.setText("connect");
FormData fd_canvas = new FormData();
fd_canvas.top = new FormAttachment (lblSpeed, 0, SWT.TOP);
fd_canvas.right = new FormAttachment (btnAccelerate, −68);
FormData fd_canvas_1 = new FormData();
fd_canvas_1.bottom = new FormAttachment (lblSpeed, 0, SWT.BOTTOM);
$$fd_{canvas}1.right = new FormAttachment(btnAccelerate, -60);$$
$$fd_{canvas}1.top = new FormAttachment(0, 549);$$
$$fd_{canvas}1.left = new FormAttachment(0, 208);$$

```java
    public void setX (Integer x) {
        x = (((x + map.getSize() .x/2 + robot .getSize() .x/2 + map.getSize() .x) + map.getSize() .x) % map.getSize() .x ;
        robot .setBounds(x, robot .getLocation () .y, robot .getSize () .x, robot .getSize () .y);)
    }

    public void setY (Integer y) {
        y = ((((y + map.getSize() .y/2 + robot .getSize() .y/2 + map.getSize() .y) + map.getSize() .y) % map.getSize() .y ;
        robot .setBounds(robot .getLocation () .x, y, robot .getSize () .x, robot .getSize () .y) ;
    }
}

public void setStatus (int status) {
    if (status == 0)
        connectionInfo .setText ("not running");
    else if (status == 1)
        connectionInfo .setText ("connecting");
    else if (status == 2)
        connectionInfo .setText ("connected");
    else
        connectionInfo .setText ("unknown");
}

public void setSpeed (Integer speed) {
    scaleSpeed .setSelection (speed);
}

public void setAngularSpeed (Integer speed) {
    scaleRotationSpeed .setSelection (speed);
}

public void setAngularRotation (Double angle) {
    double x,y;
    int intx,inty;
    x = robot .getSize () .x/2 + Math.sin(angle)*robot .getSize () .x/2;
    y = robot .getSize () .y/2 + Math.cos(angle)*robot .getSize () .y/2;
    intx = Math.min(intx, robot .getSize () .x - rotation .getSize () .x);
    inty = Math.min(inty, robot .getSize () .y - rotation .getSize () .y);
    rotation .setLocation (intx,inty);
}

LinkedList <String> consoleContent;
private Text connectionInfo;

public void sendToConsole (String message) {
    if (message == null || !message.isEmpty() ) {
        if (message.length() > 100)
            message = message.substring (0, 100);
        if (consoleContent == null)
```
```java
consoleContent = new LinkedList<String>();
if (consoleContent.size() > 7)
    consoleContent.remove();
consoleContent.add(message);
String finalstring = "";
for (String line : consoleContent){
    finalstring += line + "\n";
}
finalstring = finalstring.substring(0, finalstring.length() - 1);
d debugConsole.setText(finalstring);
d debugConsole.redraw();
}
}

C.3 RS_DISTRIB_GUI_SIMPLE_GUI

C.3.1 APPLICATION

note
description : "Root class for this application."
author : "Generated by the New Vision2 Application Wizard."

class
APPLICATION

inherit
EV_APPLICATION

create
make_and_launch

feature {NONE} -- Initialization

make_and_launch
-- Initialize and launch application
do
default_create
prepare
launch
end

prepare
-- Prepare the first window to be displayed.
-- Perform one call to first window in order to
-- avoid to violate the invariant of class EV_APPLICATION.
do
-- create and initialize the first window.
create first_window

if attached first_window as window then
-- Show the first window.
-- | TODO: Remove this line if you don’t want the first
\begin{tabular}{|p{0.5\textwidth}|}
\hline
\textbf{C. Code} & \textbf{C.3.2 BUFFER} \\
\hline
\end{tabular}

\begin{verbatim}
-- | window to be shown at the start of the program.
    window.show
end
-- add idle action to check the buffer whenever the application is idle.
add_idle_action (agent check_buffer)
end

feature {NONE} -- Communication
    check_buffer
    do
      if attached first_window as window then
        window.check_buffer
      end
    end

feature {NONE} -- Implementation
    first_window: detachable MAIN_WINDOW
    -- Main window. Made detachable.
end -- class APPLICATION

C.3.2 BUFFER

\begin{verbatim}
note
  description : "The buffer used by the MAIN_WINDOW and the CONNECTION to communicate."

class BUFFER
create make

feature make
  do
    create incoming_values.make (10)
    create outgoing_values.make (10)
  end

set_incoming (a_value: separate STRING)
do
  incoming_values.extend (create {STRING}.make_from_separate (a_value))
end

get_incoming: STRING
do
  if incoming_values.count > 0 then
    Result := incoming_values.item
    incoming_values.remove
  end
\end{verbatim}

\end{verbatim}
else
  Result := ""
end

set_outgoing (a_value: separate STRING)
do
  outgoing_values .extend (create {STRING}.make_from_separate (a_value))
end

get_outgoing: STRING
do
  if outgoing_values .count > 0 then
    Result := outgoing_values .item
    outgoing_values .remove
  else
    Result := ""
  end
end

incoming_values: ARRAYED_QUEUE[STRING]
outgoing_values: ARRAYED_QUEUE[STRING]

---

C.3.3 CONNECTION

note
description : "The class is responsible for the communication with the server"

class
  CONNECTION

INHERIT
  EXECUTION_ENVIRONMENT

create
  make

feature

make (a_buffer: separate BUFFER)
do
  buffer := a_buffer
  create tcp_socket .make_client_by_port (2000,"84.75.135.127")
  tcp_socket .connect

  tcp_socket .put_string ("register:running")
  tcp_socket .put_new_line
end

run
do
  from until false loop
  receive (buffer)
end
C. Code

```plaintext
send (buffer)
end
end

receive (a_buffer: separate BUFFER)
do
if tcp_socket.is_readable then
tcp_socket.read_line
a_buffer.set_incoming(tcp_socket.last_string)
end
done

send (a_buffer: separate BUFFER)
local l_string: STRING
do
if not tcp_socket.is_connected then
tcp_socket.connect
end
create l_string.make_from_separate(a_buffer.get_outgoing)
if l_string.count > 0 then
tcp_socket.put_string(l_string)
tcp_socket.put_new_line
end
done

close
do
tcp_socket.close
end
tcp_socket: NETWORK_STREAM_SOCKET
buffer: separate BUFFER
end

C.3.4 INTERFACE_NAMES

note
description: "Strings for the Graphical User Interface"
author: "Generated by the New Vision2 Application Wizard."

class
INTERFACE_NAMES

feature -- Access

Button_ok_item: STRING = "OK"
  -- String for "OK" buttons.

Menu_file_item: STRING = "&File"
  -- String for menu "File"

Menu_file_new_item: STRING = "&New%TCtrl+N"
```
−− String for menu "File/New"

Menu_file_open_item: STRING = "&Open...%TCtrl+O"
−− String for menu "File/Open"

Menu_file_save_item: STRING = "&Save%TCtrl+S"
−− String for menu "File/Save"

Menu_file_saveas_item: STRING = "Save &As..."
−− String for menu "File/Save As"

Menu_file_close_item: STRING = "&Close"
−− String for menu "File/Close"

Menu_file_exit_item: STRING = "E&xit"
−− String for menu "File/Exit"

Menu_help_item: STRING = "&Help"
−− String for menu "Help"

Menu_help_contents_item: STRING = "&Contents and Index"
−− String for menu "Help/Contents and Index"

Menu_help_about_item: STRING = "&About..."
−− String for menu "Help/About"

Label_confirm_close_window: STRING = "You are about to close this window.%NClick OK to proceed."
−− String for the confirmation dialog box that appears
−− when the user try to close the first window.

end −− class INTERFACE_NAMES

C.3.5 MAIN_WINDOW

note
description : "Main window for this application"
author: "Generated by the New Vision2 Application Wizard."

class
MAIN_WINDOW

inherit
EV_TITLED_WINDOW
redefine
create_interface_objects ,
initialize ,
is_in_default_state
end

INTERFACE_NAMES
export
{NONE} all
undef
default_create, copy
end

create
default_create

feature
check_buffer
do
  check_buffer_scoop (buffer)
end

check_buffer_scoop (a_buffer: separate BUFFER)
local
message: STRING
do
  message := create {STRING}.make_from_separate (a_buffer.get_incoming)
if message.is_equal ("running:yes") then
  button.set_text ("running")
elseif message.is_equal ("running:no") then
  button.set_text ("not running")
end
end

feature {NONE} -- Initialization

create_interface_objects
  -- <Precursor>
do
  -- Create main container.
create main_container
create buffer.make
create manager.make (buffer)
end

manager: separate CONNECTION
buffer: separate BUFFER

run_manager (a_manager: separate CONNECTION)
do
  a_manager.run
end

initialize
  -- Build the interface for this window.
do
  Precursor {EV_TITLED_WINDOW}
run_manager (manager)
build_main_container
extend (main_container)
  -- Execute 'request_close_window' when the user clicks
C.3. RS_DISTRIBUT_GUI_SIMPLE_GUI

--- on the cross in the title bar.

\texttt{close\_request\_actions} .\texttt{extend} (agent \texttt{request\_close\_window})

--- Set the title of the window.

\texttt{set\_title} (\texttt{Window\_title})

--- Set the initial size of the window.

\texttt{set\_size} (\texttt{Window\_width}, \texttt{Window\_height})

\texttt{is\_in\_default\_state} : \texttt{BOOLEAN}

--- Is the window in its default state?

--- (as stated in 'initialize')

do

\texttt{Result} := (\texttt{width} = \texttt{Window\_width}) \texttt{and then}
(\texttt{height} = \texttt{Window\_height}) \texttt{and then}
(\texttt{title} . is\_equal (\texttt{Window\_title}))

done

\textbf{feature} \{\texttt{NONE}\} --- Implementation, Close event

\texttt{request\_close\_window}

--- Process user request to close the window.

\texttt{local}

\texttt{question\_dialog} : \texttt{EV\_CONFIRMATION\_DIALOG}

do

\texttt{create} \texttt{question\_dialog} . \texttt{make\_with\_text} (\texttt{Label\_confirm\_close\_window})
\texttt{question\_dialog} . \texttt{show\_modal\_to\_window} (\texttt{Current})

\textbf{if} \texttt{question\_dialog} . \texttt{selected\_button} = (\texttt{create} \{\texttt{EV\_DIALOG\_CONSTANTS}\}).\texttt{ev\_ok}

--- Destroy the window.

\texttt{destroy}
\texttt{if}\texttt{attached} (\texttt{create} \{\texttt{EV\_ENVIRONMENT}\}).\texttt{application\ as\ a\ then}
\texttt{close\_connection\_scoop} (\texttt{manager})
\texttt{scoop\_destroy} (\texttt{a})
end
end

\texttt{scoop\_destroy} (\texttt{a\ application} : \texttt{separate} \texttt{EV\_APPLICATION})

do
\texttt{a\ application} . \texttt{destroy}
end

\texttt{close\_connection\_scoop} (\texttt{a\ manager} : \texttt{separate} \texttt{CONNECTION})

do
\texttt{a\ manager} . \texttt{close}
end

\textbf{feature}

\texttt{close\_connection}

do
C. Code

```plaintext
C.4 RS_DISTRIBUT_GUI_SIMULATOR

close_connection_scoop (manager)
end

feature {NONE} -- Implementation

main_container: EV_VERTICAL_BOX
-- Main container (contains all widgets displayed in this window).

build_main_container
-- Populate 'main_container'.
do
create button
button.set_text("not running")
button.select_actions.extend(agent.clicked)
main_container.extend(button)
ensure
main_container.created: main_container /= Void
end

button: EV_BUTTON

clicked
do
if button.text.is_equal("not running") then
send("start", buffer)
elseif button.text.is_equal("running") then
send("stop", buffer)
end
end

send(a_message: separate STRING; a_buffer: separate BUFFER)
do
a_buffer.set_outgoing(a_message)
end

feature {NONE} -- Implementation / Constants

Window_title: STRING = ""
-- Title of the window.

Window_width: INTEGER = 550
-- Initial width for this window.

Window_height: INTEGER = 750
-- Initial height for this window.
end
```
note
description : 
"A simple simulator of a Robot."
class
RS_DISTRIBUT_GUI_SIMULATOR
inherit RS_DISTRIBUT_GUI_APPLICATION
create
make
feature {NONE} -- Basic Methods

at_create
    -- there is nothing to do at creation of the class.
do
end
initialize
    -- initialize statuses and commands.
do
    initialize_statuses
    initialize_command_instructions
    set_time_interval (100)
end
initialize_statuses
    -- initialize default statuses.
local
    l_status : RS_DISTRIBUT_GUI_STATUS
do
    create l_status.make("x", "0")
x := 0
send_status (l_status)
create l_status.make("y", "0")
y := 0
send_status (l_status)
create l_status.make("speed", "0")
speed := 0
send_status (l_status)
create l_status.make("angular_speed", "50")
angular_speed := 50
send_status (l_status)
create l_status.make("angular_rotation", "0")
angular_rotation := 0
send_status (l_status)
end
initialize_command_instructions

--initialize command instructions.
local
l_command_instruction: RS_DISTRIB.GUI.COMMAND_INSTRUCTION
do
create l_command_instruction.make ("stop")
l_command_instruction.allow_none
send_command_instruction (l_command_instruction)

create l_command_instruction.make ("start_accelerate")
l_command_instruction.allow_none
send_command_instruction (l_command_instruction)

create l_command_instruction.make ("stop_changing_speed")
l_command_instruction.allow_none
send_command_instruction (l_command_instruction)

create l_command_instruction.make ("start_brake")
l_command_instruction.allow_none
send_command_instruction (l_command_instruction)

create l_command_instruction.make ("start_turn_left")
l_command_instruction.allow_none
send_command_instruction (l_command_instruction)

create l_command_instruction.make ("stop_turn")
l_command_instruction.allow_none
send_command_instruction (l_command_instruction)

create l_command_instruction.make ("start_turn_right")
l_command_instruction.allow_none
send_command_instruction (l_command_instruction)

create l_command_instruction.make ("set_angular_speed")
l_command_instruction.allow_range (0, 100)
send_command_instruction (l_command_instruction)
end

execute

--calculations of new position and speed each loop iteration
do
angular_rotation := angular_rotation + angular_speed * dt * rotating / 50
if angular_rotation > {SINGLE_MATH}.pi*2 then
    angular_rotation := angular_rotation - {SINGLE_MATH}.pi*2
end
x := x + {SINGLE_MATH}.sine (angular_rotation.truncated_to_real) * dt * speed
y := y + {SINGLE_MATH}.cosine (angular_rotation.truncated_to_real) * dt * speed
speed := speed + accelerating * dt * 10
if speed < 0 then speed := 0
elseif speed > 100 then speed := 100 end
end

handle_command (a_command: RS_DISTRIB.GUI.COMMAND)
do
    print("command handled\%N")
    command_error := false
    if a_command.name.is_equal ("stop") then
        speed := 0
        accelerating := 0
        rotating := 0
    elseif a_command.name.is_equal ("startAccelerate") then
        accelerating := 1
    elseif a_command.name.is_equal ("stopChangingSpeed") then
        accelerating := 0
    elseif a_command.name.is_equal ("startBrake") then
        accelerating := -1
    elseif a_command.name.is_equal ("startTurnLeft") then
        rotating := 1
    elseif a_command.name.is_equal ("stopTurn") then
        rotating := 0
    elseif a_command.name.is_equal ("startTurnRight") then
        rotating := -1
    elseif a_command.name.is_equal ("setAngularSpeed") then
        angular_speed := a_command.argument.to_integer
    else
        command_error := true
    end
end
ensure then
    command_error = false
end

execute every time interval
    -- send statuses every specified milisecond to not overflow the network. Default is 200 milisecond.
local
    l_status : RS DISTRIBUT GUI SIMULATOR
    do
        create l_status.make("x", x.rounded.out)
        send_status (l_status)
        create l_status.make("y", y.rounded.out)
        send_status (l_status)
        create l_status.make("speed", speed.rounded.out)
        send_status (l_status)
        create l_status.make("angular_speed", angular_speed.out)
        send_status (l_status)
        create l_status.make("angular_rotation", angular_rotation.out)
        send_status (l_status)
    end
feature {NONE} -- Global Variables
    x: DOUBLE
C. Code

-- x coordinate of robot.
y: DOUBLE
    -- y coordinate of robot.

angular_rotation: DOUBLE
    -- radial orientation

speed: DOUBLE
    -- speed of robot.

angular_speed: INTEGER
    -- angular speed of robot.

rotating: INTEGER
    -- is robot rotating? 1 for left, -1 for right and 0 for not.

accelerating: INTEGER
    -- is robot accelerating? 1 for accelerating, -1 for braking and 0 for not.

command_error: BOOLEAN
    -- is the command received from the server not recognizable.

end

C.5 RS_DISTRIB_GUI_SIMPLE_SIMULATOR

note
description:
    "A very simple simulator of a Robot which can only start and stop"

class
    RS_DISTRIB_GUI_SIMPLE_SIMULATOR

inherit
    RS_DISTRIB_GUI_CONTROLLER

create
    make

feature {NONE}

at_create
    -- initialize the global variable, no server needed.
    do
        create running.make("running", "no")
    end

initialize
    local
        -- initialize status and command instruction at the server
        command_instruction: RS_DISTRIB_GUI_COMMAND_INSTRUCTION
        do
            send_status (running)
        end
create command_instruction.make ("start")
command_instruction.allow_none
send_command_instruction (command_instruction)

create command_instruction.make ("stop")
command_instruction.allow_none
send_command_instruction (command_instruction)
end

execute
  -- There is no need to calculate something in this very simple Simulator.
do
end
every_time_interval
  -- Send the status to the server
do
  send_status (running)
end

handle_command (a_command: RS_DISTRIB_GUI_COMMAND)
do
  if a_command.name.is_equal ("start") then
    running.set_value ("yes")
  elseif a_command.name.is_equal ("stop") then
    running.set_value ("no")
end
end

running: RS_DISTRIB_GUI_STATUS
Bibliography


Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor’s thesis, Master’s thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Graphical user interfaces for Robocoop applications.

Authored by (in block letters):
For papers written by groups the names of all authors are required.

Name(s):

Stulz

First name(s):

Jonas

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the ‘Citation etiquette’ information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date

10.11.2014 Winterthur

Signature(s)

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.