University of Stavanger

Faculty of Science and Technology

MASTER'S THESIS

Study program/ Specialization: Computer Science	Spring semester, 2011 Open
Writer: Eirik Nordbø	…………………………………………. Eirik Nordbø
Faculty supervisor: Chunming Rong External supervisor(s): Einar Landre, Statoil	
Titel of thesis: Inter-Agent Communication In Multi-Agent Systems	
Credits (ECTS): 30	
Key words: Intelligent Agent, anget-communication, JACK Agent Framework, Lego Mindstorms, LeJos, Java	Pages 82 + enclosure: CD Stavanger, June 8, 2011 Date/year

MIDMAS – MASTER THESIS

Inter-Agent Communication In Multi-Agent Systems

Author:
Eirik NORDBØ

Supervisor
Einar LANDRE

Supervisor:
Chunming RONG

8. June 2011

Abstract

The oil and gas industry experience an increased dependency on IT and particular software based capabilities to achieve its business objectives. Core business processes such as exploration, well construction, production optimization and operations are all fueled by software and information technology. In coming years we will see that software will fill more and more advanced features, including central control functions in autonomous and collaborative robots and it is believed that agent technology may be of use in this scenario.

The primary reason for this is the practical benefit from goal oriented systems is a simplification of the human-machine interface. A goal oriented system is able to communicate and react to events in its environment in context of their goals. This is the primary driver for autonomous systems: simplifying and securing operation of machines in an unstructured / highly dynamic environment.

Inter-agent communication is an important aspect of agent software, as it helps in the process of decision-making, be it an individual decision or even group decision-making. It also enables agents to share its beliefs and desires among each other. In this thesis I will look at possible models for collaboration and coordination of autonomous robots, and how this can be addressed through the use of software agents. To do this a multi-agent solution for controlling Lego Mindstorms robots has been developed in cooperation with Rune Johansen.

The solution is based on three Lego robots operation on a line-based grid. One robot is set to explore the grid, finding object, and sharing this information (beliefs) with a second robot that is responsible for collecting and delivering these objects to a robot that sorts these object according to color. The solution enables investigation in relation to intelligent software agents combined with autonomous robot systems, such as inter-agent communication and coordination.

The agent system is developed using the Prometheus methodology for the design and the JACK Intelligent Agents framework for the implementation. Regular Java is used combined with the LeJos - Java For Mindstorms framework to implement the robot side of the system.

Preface

This master thesis has been written as a continuance of a preliminary project we performed in cooperation with Statoil during the autumn semester of 2010. The preliminary project was collaboration between Rune Johansen and Eirik Nordbø where the objective was to do a feasibility study of interfacing an agent platform (JACK) with a robot control system (LEGO Mindstorms). The cooperation has continued as we have developed a common technical solution with separate angle of approach for our master thesis. Eirik has focused on inter agent coordination and communication while Rune‘s area of focus has been human robot interaction, both in relation to multi-agent systems controlling robots or machines.

We would like to thank our supervisor Einar Landre at Statoil for all help and support during our work with this thesis and the opportunity to gain insight into the exiting field of intelligent agent technology. We would also like to thank professor Chunming Rong at the University of Stavanger and last but not least Jossi for great coffee and moral support throughout the semester.

Stavanger, 8 June 2011

Eirik Nordbø

Contents

List of Figures

List of Tables

1 Introduction

This Master's thesis has been written in collaboration with Statoil, an international energy company with operations in 34 countries. Building on more than 35 years of experience from oil and gas production on the Norwegian continental shelf, they are committed to accommodating the world's energy needs in a responsible manner, applying technology and creating innovative business solutions. They are headquartered in Norway with 20,000 employees worldwide, and are listed on the New York and Oslo stock exchanges. [2]

As a technology based energy company, Statoil experience an increased dependency on IT and particular software based capabilities to achieve its business objectives. Core business processes such as exploration, well construction, production optimization and operations are all fueled by software and information technology. In coming years we will see that software will fill more and more advanced features, including central control functions in autonomous and collaborative robots [3].

Due to the fundamental properties of Intelligent Agents they provide a intuitive and robust solution to the robot tasking challenge met in complex multi machine systems. Their ability to communicate and coordinate amongst them selves, as well as the natural mapping between robots and agents makes them not only a viable, but a good approach for this type of autonomous systems.

It's one thing to give one robot a task, but it's a whole other matter to give many robots a complex task, which requires cooperation and coordination in time and space. In this thesis I will look at possible models for collaboration and coordination of autonomous robots, and how this can be addressed through the use of software agents.

1.1 Motivation

Automation of reactive behavior has its roots in the need to control dynamic systems. Dynamic systems are systems where the system state changes as a function of time, systems that can be described using differential equations.

The scientific platform for controlling dynamic systems is known as control theory or cybernetics, where feedback and/or feed-forward techniques are used to control a systems reactive response to external events for the purpose of keeping the system within its operational envelope. The forces motivating automation are many, but most often falls into one of the categories illustrated in Figure 1.

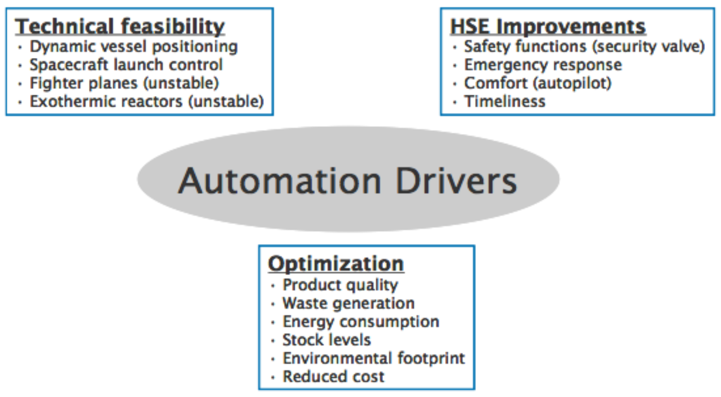


Figure 1: Forces motivating automation

Autonomous systems are motivated by the same forces, but the value from automating decisions, and moving to more goal oriented designs are easier to grasp using an example. The archetype example is human operation of complex processes or vehicles (robots) in unstructured and dynamic environments where time is a key. In these systems three concerns must be managed:

1. Communications loss. Communication links breaks and there is a need for the vehicle to maintain its own integrity as well as the integrity of its operating environment.
2. Communications latency. Remote operation over some distance has latency. In space applications like the Mars rovers, the latency is in minutes. For many earth bound applications latency in the magnitude of seconds might be unacceptable.
3. Operator information overload. Remote control is often more demanding than piloting the vehicle in a more traditional way. When a human is piloting a manned vehicle, vibrations, sounds and vision provide information that is easily lost in a remote control scenario, leading to unnecessary stress and mistakes.

By introducing autonomy, the vehicle (process) becomes able to store its mission objective (goal) and continuously assess its objective (goal) against environmental changes. Assuming the vehicle is an airplane, it will not only be able to detect a thunderstorm ahead, but it will be capable of validating the threat from the thunderstorm in context of its assigned mission.

In such situation the aircraft will recalculate its route, and validate if it has sufficient amount of fuel to pursue its original objective using the new route. In the case the aircraft is not able to accomplish its objective it will request permission from the human operator (pilot) to abort its assigned mission and update its objective to return home safely. The human in charge might reject or acknowledge such request.

Independent of what the human decides the role of the human operator has changed from flying the aircraft to perform mission management in collaboration with the vehicle. As a consequence the abstraction level in the man-machine interaction is raised, and the machine can interact with the human operator in a more human way.

Given industrial challenges such as smaller and more marginal resources (NCS), deeper waters, more demanding operational conditions (arctic), and the need for a reduced environmental footprint (regulations) will require smarter and more lightweight operational concepts. These new operational concepts will drive the development of more sophisticated and automated systems within all Statoil's core business processes (drilling, operations and production optimization), systems that need to be designed for unmanned / remote operations, systems utilizing the power of automated decision making, to enforce and secure prudent operations.

For Statoil to maintain a leading position as a technology based energy company it is important to understand and master autonomous systems including the software engineering challenges that comes with building goal oriented, collaborating physical systems to operate in unstructured environments. [4]

1.2 Problem definition

The following problem has been formulated in cooperation with our teaching supervisor at Statoil:

,,*The students should investigate some of the more complex problems related to autonomous systems, such as inter agent coordination and communication as well as agent human interfacing. This should be done through demonstrating how software agents can communicate with each other, with graphical interfaces, with external systems such as robots and human robot interaction (HRI).*"

To achieve this goal, Jack Intelligent Agent development platform [5] will be used together with Lego Mindstorms robots. [6]

Based on this problem definition and further discussion with our supervisors one common implementation goal was defined:

,,*Design and implement a proof-of-concept software with a graphical user interface (GUI), robot communication and human interaction. The GUI should implement two-way communication with the agents; provide functionality for relevant HRI challenges and display real time information and results provided by the agents. The solution also needs to provide standard interfaces specifying the robot functionality required, as well as a Lego Mindstorms specific implementation of these interfaces.* "

This thesis will discuss inter-agent-communication and address the following research hypostisis:

1. Intelligent agents are a suitable platform for modeling and development of interacting robots
In our preliminary project we found that the JACK framework where a robust approach for developing an agent solution controlling one robot. In this thesis I will look at how suitable intelligent agents are for controlling several interaction robots.

2. In a multi - agent systems, robot interaction can be modeled as interacting agents.

1.3 Report outline

Chapter 2, Software Agents
Agent-oriented software engineering is a rapidly developing area of research. This chapter will present basic agent theory, how they differ from traditional software paradigms and in which contexts they are useful.

Chapter 3, Agent Communication
Gives an overview over the different aspects of agent communication.

Chapter 4, Methodology and tools
This chapter describes the different methodologies and tools used for modeling and development of the agent-solution and robot application.

Chapter 5, Application
In order to design an application relevant to our problem definition and hypotheses, several approaches where considered. This chapter describes the different solutions.

Chapter 6, System Design
The chapter describes the different phases in our design using the Prometheus methodology. The overall system structure presented in this chapter is probably the most important and useful artifact resulting from the initial two phases of the Prometheus methodology.

Chapter 7, System Development
The different agents and how they communicate are presented in this chapter, including use case scenarios.

Chapter 8, Results
Summarizes the results of our work in light of the thesis hypotheses, as well as the challenges met.

Chapter 9, Conclusion
Based in our original problem definition, we here discuss the further implications of our result.

Chapter 10, Further work
We here present some possible approaches for further work.

2 Software Agents

The notion of AI was first introduced in the 1950's when it went from being fantasy/science fiction to becoming an actual research area. In addition to the design and implementation of robots to model the behavioral activities of humans, AI scientists eventually started to focus on implementing devices (software and hardware) that mimic human behavior and intelligence, Intelligent agents (agents) [7]. As of today no formal definition of an agent exists, but the Wooldridge and Jennigs definition is increasingly adopted.

The following definition is from (Wooldrigde 2002), which in turn is adapted from (Wooldridge and Jennigs 1995):

> „An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives".

Wooldridge distinguishes between an agent and an intelligent agent, which is further required to be reactive, proactive and social (Wooldridge 2002, page 23).

An intelligent agent is characterized as being autonomous, situated, reactive, proactive, flexible, robust and social [8]. These properties for an agent differ from traditional objects in several ways as shown in Table 1.

Property	**Agent**	**Object**
Autonomous	Agents are independent and make their own decisions.	Objects do not exhibit control over their own behavior because an object's method can be invoked by other entities.
Situated in an Environment	Agents tend to be used when the environment is dynamic, unpredictable and unreliable.	Objects tend to be used when the environment is static, predictable and reliable.
Reactive	Agents perceive changes in their environment and will respond to these changes to achieve goals.	Objects can be reactive, but their reactiveness is dependent on how well they manage changes in the environment.
Proactive	Agents are proactive because they persistently pursue goals, i.e. they have goal-directed behavior.	Objects are not proactive because they do not have goal-directed behavior and they lack reasoning ability.
Flexible	Agents are flexible because they can achieve goals in multiple ways.	Objects do not have the ability to choose between different ways to achieve a goal.
Robust	Agents recover from failure and choose another way to reach their current goals.	Objects are not flexible, and as a consequence they are less robust than agents.
Social	Agents have the ability to cooperate, coordinate and negotiate with each other to achieve common or individual goals.	Objects can exchange information and data with each other, but they lack the social aspect of the interaction.

Table 1: Difference between Agents and Objects [1]

2.1 Belief-Desire-Intention model

The Belief-Desire-Intention (BDI) model is based on human behavior and reasoning and can therefor provide a control mechanism for intelligent action. It is developed by Michael Bratman [9] to explain future-directed intention.

The Belief-Desire-Intention software model is a software model developed for intelligent agent programming. A BDI agent is a particular type of bounded rational software agent with some specific architectural components.

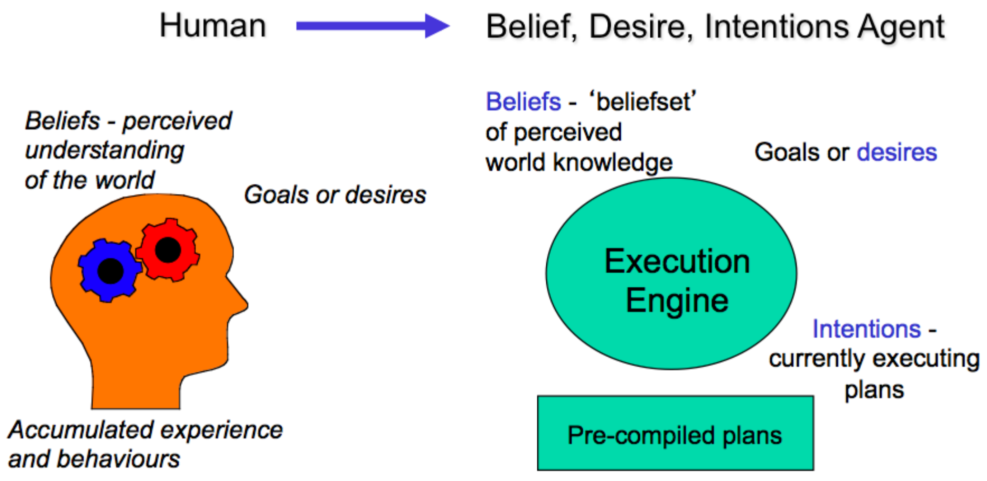

Human
Belief, Desire, Intentions Agent
Beliefs - perceived understanding of the world
Goals or desires
Accumulated experience and behaviours
Beliefs - 'beliefset' of perceived world knowledge
Goals or desires
Execution Engine
Intentions - currently executing plans
Pre-compiled plans

Figure 2: The BDI agent model

- Beliefs: Represent the informational state of an agent, what the agent believes about the world. The term belief is used instead of knowledge as the beliefs may be false although believed true by the agent. Beliefs are organized in Beliefsets.
- Desires: Represents the motivational state of an agent, objectives or situations the agent would like to accomplish or bring about. An agent can have goals which are desires actively pursued by the agent.
- Intentions: Represent the deliberative state of an agent, what an agent has chosen to do to accomplish a goal/desire. Plans are sequences of actions, which an agent can use to fulfill its intentions.
- Events: Events are the triggers for reactive activity by an agent. An event may change beliefs, update goals or trigger plans.

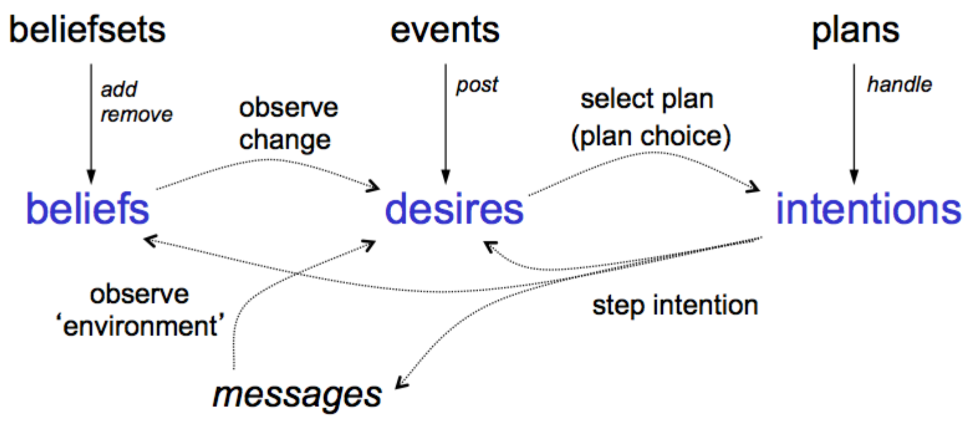


Figure 3: The JACK BDI Execution

2.2 Why are agents useful?

An important advantage of agents is that they reduce coupling. The coupling is reduced by encapsulation provided by autonomy, the robustness, reactiveness and pro-activeness of agents [8]. Because of its properties an agent can be relied upon to persist in achieving a given goal by trying alternative approaches depending on environment changes. Being proactive and reactive agents are human-like in the way they deal with problems. This provides a very natural abstraction and decomposition of complex problems. Leading to agents being used in a number of applications such as planning and scheduling, business process systems, exploration of space, military operations/simulation and online social communities.

3 Agent Communication

As mentioned in Section 2, the field of multi-agent systems is a rapidly growing research area. Although there is no real consensus on what exactly what an agent is, there are some generally accepted properties that an agent should have [10]. An agent is viewed upon as an autonomous entity with both reactive and proactive behaviors as described in Table 1, responding to external occurrences and persistently pursuing goals. An agent is additionally assumed to have a mental state comprised of informational attitudes (like knowledge and belief) and motivational attitudes (like goals, desires and intentions). Parts of this mental state may also be shared among agents in the same system. Moreover, it has an ability to interact with other agents in a multi-agent environment. Because there are different interpretations and implementations of agents, there are several different approaches to inter-agent communication, therefor this chapter will focus on inter-agent communication in the JACK framework. The content of this chapter is important in order to address the research hypotheses described in section 1.2 and to draw relevant conclusions based on the developed application.

3.1 Events

An event is a element that can trigger the choice of execution of a plan. These events can derive from percepts coming from the environment or messages delivered from other agents. Events are the origin of all activity within an agent-oriented system. In the absence of events an agent sits idle. When an event occurs, one or more agents starts a process to handle the event. The task causes the agent to choose between the eligible plans it has to execute the given event. A normal event is said to be successful if the chosen plan executed without failing, a goal event is said to be successful is one of the applicable plans succeeds, even though one or more fails before it.

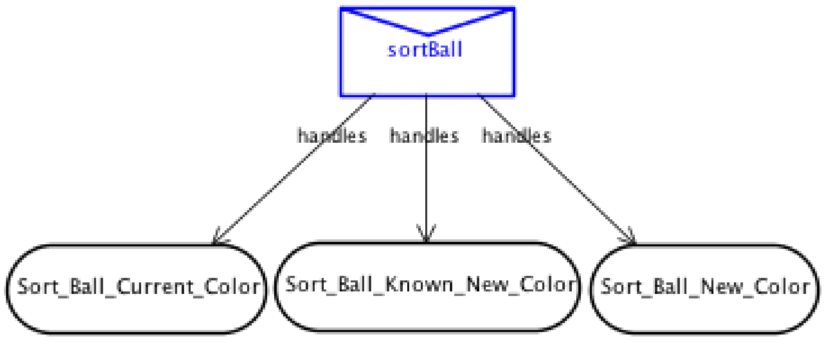


Figure 4: This figure displays a event that can be handled by three different plans.

An event may also contain information vital to the plan that processes the event, or even to check if a plan is viable to handle that exact event. As an example, Figure 4 shows 3 plans which handles the same event. In this case, the event contains the color of the ball, which is to be sorted. Based on this information, the first eligible plan is selected to handle the event. As explained above, the event has to be a goal event in order to have the event try more than one plan, if the first fails.

3.2 Inter-agent Events

JACK provides a runtime-networking environment on which different agent processes can operate. Agents can address messages (MessageEvents, and BDIMessageEvents) to one another by specifying the name of the destination agent and, if applicable, its portal and host. The JACK runtime network then takes care of routing this message to its desired destination.

3.2.1 Local communication

Agents often share the same process, and if more than one agent runs in the same process, the communication between them is said to be local. In this situation, the sending of messages between the agents is trivial. All the sender agent needs to know is the receiver agent's name to send the message.

The following requirements must be meet for to agents to successfully communicate locally:

- The sending agent, needs to know the name of the destination agent.
  The receiver however, will get the senders name in the events "from" data member. This information is automatically generated.
- The source agent needs to be able to send a message event.
  This is achieved by including a #sends event declaration in the agent's definition for the required class of message event. Note that the event must be defined as a MessageEvent. The source agent must then include code to send the message event. This can be done by calling the send method from code outside a reasoning method, or the @send statement from within a reasoning method.
- The destination agent needs to be able to handle this message event.
  This is achieved by including a #handles event declaration for this message event in the destination agent's agent definition or capability. To handle this event correctly, the destination agent must also include at least one plan with the same #handles event declaration, and declare that it uses this plan via a #uses plan declaration. If there is no plan that handles the event, JACK will output a error message saying that the event is not handled by any plans.

3.2.2 Remote communication

Agents may run on different computers and communicate over a network, using the JACK communication layer known as JACK DCI (Distributed Communication Infrastructure). The DCI network is layered in such a way that different underlying transport mechanisms can be accommodated. DCI is built as a thin layer over UDP [11], guaranteeing delivery.

When agents in remote processes need to be able to communicate, they need to be told how to communicate with the other agents. Each agent process has its own "portal", and these connect to each other to provides their agents to communicate. The portals can be connected explicitly, but they can also be connected on the fly using name-servers. To use this technique we simply assign one of the processes to be a name-server, and when an agent tries to send a message to an other agent at an unknown portal, the agent queries the name-server to try to locate the portal. If it succeeds, the message is sent, but if the portal name is not available from the name-server, an error occurs.

3.2.3 Message synchronization

JACK offers functionality to synchronize messages between agents. If an agents plan depends on input from an other agents, it can simply request the needed information and wait for a reply before continue executing the plan as shown in Figure 5. If the agent has received a message and performed a task in response to this message, one of the steps in the plan that responds to this message may be to send another message back to the originating agent in the form of a reply. This may be to confirm that the task has been completed, or for the originating agent needs some information from the receiver before continue executing its plan.

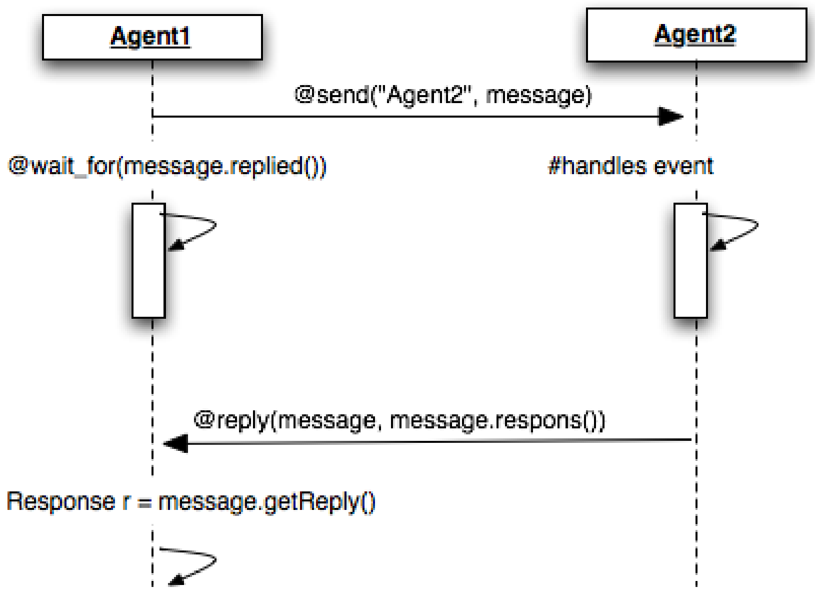

Agent1
Agent2
@send("Agent2", message)
@wait_for(message.replied())
#handles event
@reply(message, message.respons())
Response r = message.getReply()

Figure 5: This figure displays message synchronization between two agents.

3.3 Shared resources

Resources such as beliefsets and views can only be shared among agents in a given process. The declaration within the agent takes the following form:

- #private data DataType ref (arg_list); or
- #agent data DataType ref (arg_list); or
- #global data DataType ref (arg_list);

To share a data resource among agents it has to be declared as global in the agent's definition. Even though it is possible for all agents to modify a global data, agents should in general only modify the data that appears in their private user-defined data structures due to synchronization problems. Even though it is suggested that shared beliefsets and views are read-only it is possible to write to these data resources as well, but it may lead to synchronization problems if two or more agents try to update a beliefset or other data source at the same time.

4 Methodology and tools

This chapter describes the different methodologies and tools used for modeling and development of the agent-solution and robot application.

4.1 Prometheus methodology

Prometheus is intended to be a practical methodology. As such, it aims to be complete: providing everything that is needed to specify and design agent systems. The methodology is widely used in university courses, by industry workshops and the company behind JACK, Agent-Oriented Software [12].

4.1.1 Why a new agent methodology?

Although there are many methodologies for designing software, none of these are well suited for developing agent oriented software systems. Even though there are similarities between agents and objects there are some significant differences justifying the use of the Prometheus methodology over object oriented methodologies. This despite the fact that object oriented methodologies are extensively studied and developed compared to Prometheus.

Some of the main differences between Prometheus and object oriented methodologies are:

1. Prometheus supports the development of intelligent agents which use goals, beliefs, plans, and events. By contrast, many other methodologies treat agents as simple software processes that interact with each other to meet an overall system goal.
2. Prometheus provides explicit modeling of goals which is needed to support proactive agent development. This is generally not a part of object-oriented methodologies.
3. To provide flexibility and robustness a message (or an event) should be allowed to be handled by several plans, not just as a label on arcs, which is common for object oriented methodologies.
4. Agents are situated in an environment, thus it is important to define the interface between the agent and its environment.
5. In object oriented programming everything is a passive object, but in agent oriented programming it is needed to distinguish between passive components such as data, and beliefs and active components like agents and plans.

4.1.2 The three phases

The Prometheus methodology consists of three phases, as shown at figure 6.

1. The system specification phase intends to describes the overall goals and basic functionality, including the illustration of the systems operations with use case scenario schemes. The phase is also intended to specify inputs (for example sensor readings) and outputs (actions), namely the interface between the system and its environment.
2. The second phase, called the architectural design phase decides which agent types the system will contain and how they interact based on the previous phase.
3. The detailed design phase looks at each agent individually and describes its internal behavior to fulfill its goals within the overall system.

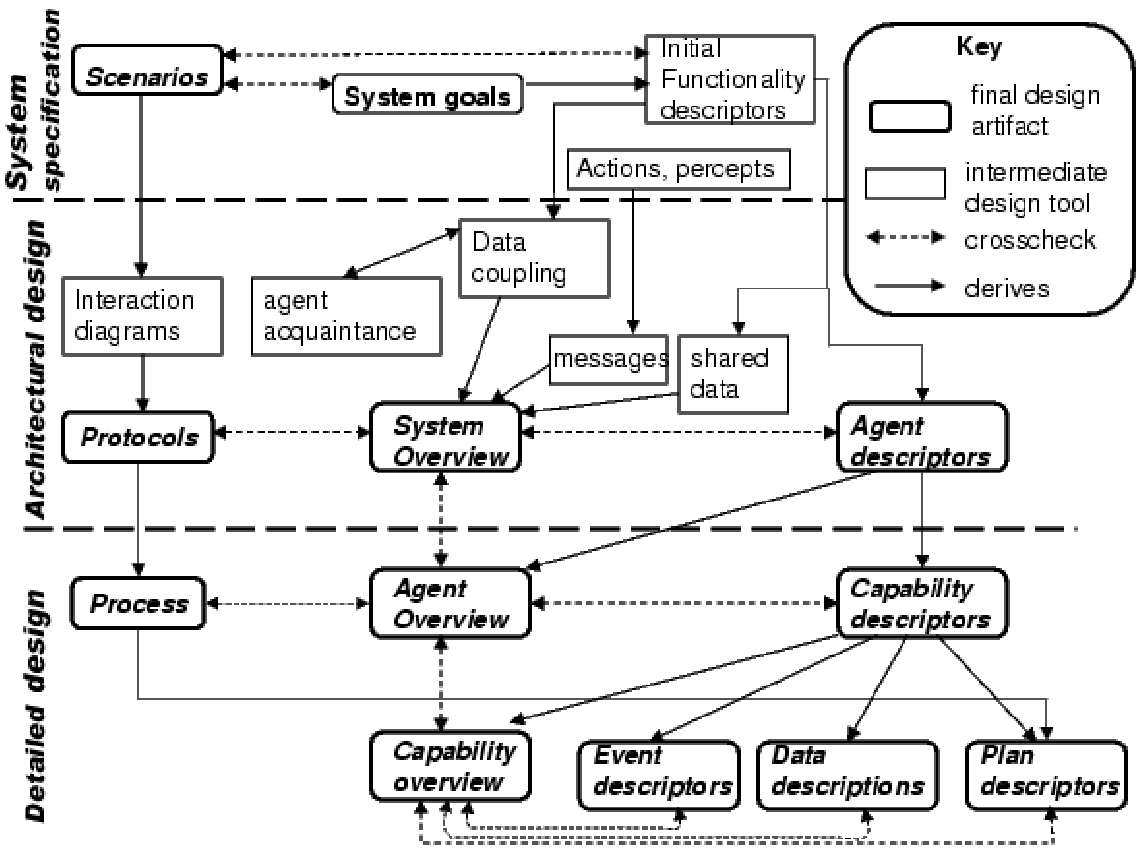


Figure 6: The phases of the Prometheus methodology

System specification As mentioned in the start of Chapter 3.2, the system specification phase focuses on the following:

- **Identifying the system goals.**
  The system goals might be thought of as the overall goals of the system, what the system should be able to achieve. In agent software these goals are important because they control the agents behavior. The system goals are often high-level descriptions; therefor they tend to be less likely to change over time than functionalities.

- **Creating use case scenarios that presents how the system works.**
  Use case scenarios are used to describe how the system operates through a sequence of steps combined with description of the context in which the sequence occurs. These scenarios are useful to understand the structure of the system works.

- **Identify the fundamental features of the system.**
  The fundamental features are groups of related goals, data and input/output that describe the main functionalities of the system. In a ATM system these might be; "Withdraw money", "Check account balance" and "Change card PIN code". As the system is created, the need for new functionality will be introduced. In our ATM system there might be a need to add "Charge cell phone account".

- **Describe the interface connecting the system to its environment, inputs and outputs.**
  An agent is situated in an environment, and we need to specify how the agent affects the environment and what information the agent gets from the environment. Using our ATM system example, the agent gets input in form of credit card data, withdrawal requests and so on. The output might be money or a message on the ATM display.

Architectural design The architectural design phase uses the outputs from the previous phase to determinate which agent types the system will contain and how they will interact. It also captures the system's overall structure using the system overview diagram.

- **Agent Types**
  One of the most important aspects of the Architectural design is to determine which agents are to be implemented and to develop the agent descriptors. The functionalities established in the first phase are grouped into agent types, so that each agent consists of one or more functionalities. The functionalities are grouped together based on coupling and cohesion.

- **System structure**
  Once the agent types are decided upon, the system structure is determined by dividing input and output responsibility among the agents. The major shared data repositories are also specified in this process. These items are modeled in the

system overview diagram, which is perhaps the single most important product of the design process. It ties together agents, data, external input and output, and shows the communication between agents.

- **Interactions**
  The System structure defines who talks to who, while the interactions part defines the timing of communication. This is done trough use case scenarios and is modeled in agent interaction diagrams.

Detailed design The last of the three phases focus on the individual agent's internal design, constructing its capabilities, including plans, events and data so that it can fulfill its responsibilities as outlined in the functionalities it is to provide. It is also important to refine the interaction protocols the agents use for internal and external communication.

4.2 JACK Intelligent Agents

AOS [5] offers a number of products for developing autonomous systems: JACK, JACK-Teams, JACK Sim, C-BDI, CoJACK and Surveilance agent. JACK is the worlds leading autonomous systems developing platform. It is entirely written in Java making it able to run on any system of which Java is available from laptops to high-end multi-CPU enterprise servers. JACK thus has access to all Java features including multiple threads, platform independent GUIs and third party libraries. JACK also provides a JDE (JACK Development Environment) for developing and designing JACK applications.

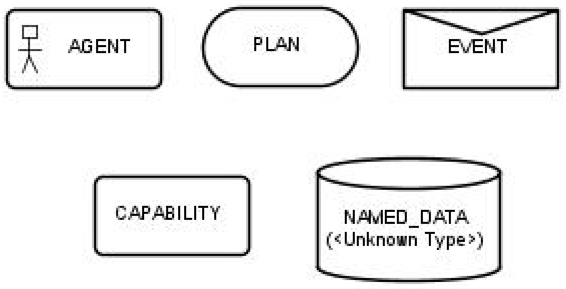


Figure 7: The JACK Components / Agent Model Elements

4.2.1 JDE

Components and links (See Figure 7) can be added/removed in the JDE browser window or graphically using the design tool. These express relationships between agent model, elements and skeleton code is automatically generated for them. The JDE saves in a .prj file and a gcode directory and when you select compile application the corresponding JACK files are generated before the compilation proceeds as it would on the command line.

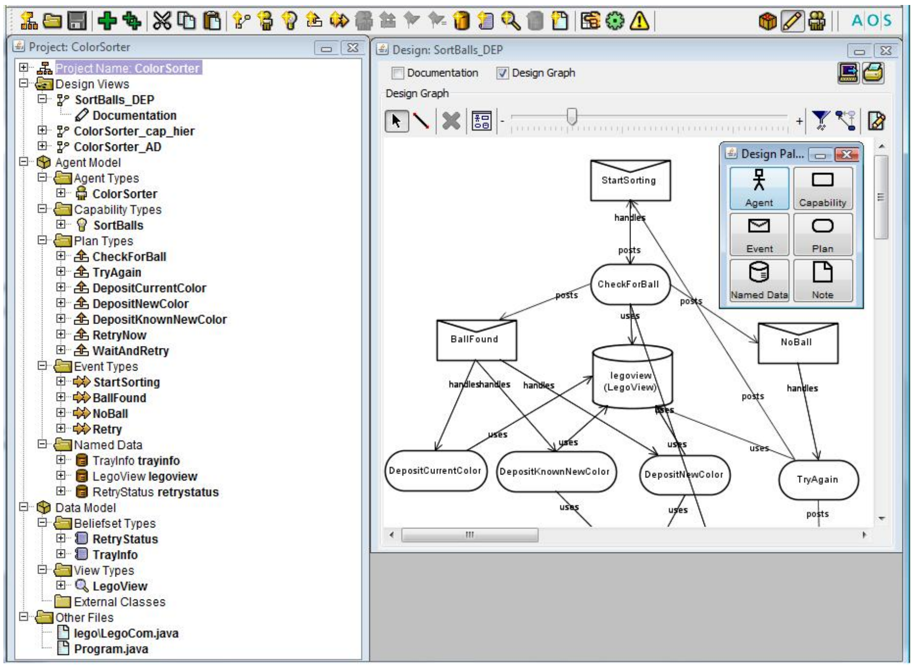


Figure 8: The Jack JDE

4.2.2 DCI

JACK DCI (Distributed Communication Infrastructure) enables agents to communicate within a process, across processes and between different machines. A DCI portal for a process is defined by giving the process a portal name and a port number to identify it. The full name for an agent is agent_name@portal and the DCI will ensure message delivery across portals.

4.2.3 JACOB

Provides machine and language independent object structures that can be stored or transmitted. The object structures are defined using the JACOB Data Definition Language and stored in definition files, which are compiled using JACOB Build.

4.2.4 JACK agent language

The JACK agent language is an extension of Java to support an agent oriented programming paradigm

It introduces new base classes: agent, capability, event, plan, view, beliefset, and extensions to the java syntax to support these e.g. #declarations and @reasoning statements.

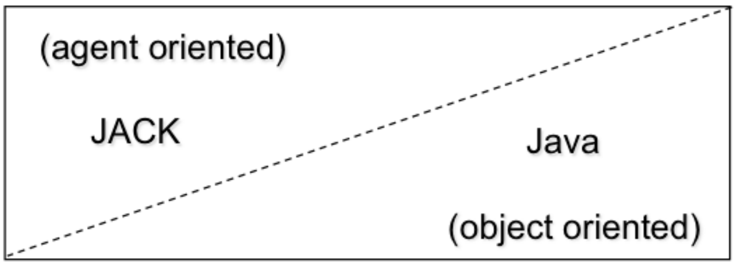


Figure 9: Agent oriented vs Object oriented

It uses the BDI (Belief Desire Intention) agent model

4.2.5 Agent:

The agent type encapsulates knowledge and behavior through beliefsets, events and plans which can be represented as capabilities. It reacts to events and receives messages to perform tasks and services.

4.2.6 Event

All activity in JACK originates from an Event. The event provides the type safe connection between agents and plans as both the agents and plans must declare the events they handle, post and send. JACK supports several different types of events depending on desired plan processing behavior. The different types are:

- Normal:

  A 'Normal' event corresponds to conventional event driven programming. Causes the plan behavior to be that if the plan fails the agent does not try again. There are two base classes for normal events, these are Event, which is the base class for all events and can only be posted internally and MessageEvent which can be sent between agents (a message for the sender, an event for the receiver).

- BDI:

  A BDI event represents the desire to achieve a goal and it may cause both meta-level and practical reasoning. This can result in agents trying several different plans and even recalculating the applicable plan set. There are three different base classes for BDI events, BDIGoalEvent, BDIMessageEvent, and BDIFactEvent. The BDIGoalEvent is typically used in @achieve, @insist, @determine etc and will cause an agent to try all applicable plans until one succeeds. The receiver of a BDIMessageEvent uses BDI processing and so does a receiver of a BDIFactEvent but in a non persistent way. The BDI events can be customized to specify how and when to determine the applicable plan set and how to form it, when to do meta-level reasoning, how to choose plan without meta-level reasoning , how to deal with plan failure and how to handle exceptions.

- Rule:

  The event base class for rule events is InferenceGoalEvent. This type of event will cause all plans in the applicable set to be executed regardless of success or failure.

- Meta: The event base class for Meta events is PlanChoiceEvent. This is the mechanism the agent uses to perform meta-level reasoning.

4.2.7 Plan

A plan describes the actions an agent can take when an event occurs. Each plan can only handle a single event and it will either succeed or fail. A plan contains logic to determine if the plan is relevant or not for a given event. It also has at least one reasoning method, which defines the actions of the plan. This method can contain JACK agent language @statements and each of these are handled as a logical condition. These are handled sequentially and if a statement fails the method fails and terminates, only if all statements succeed the plan succeeds.

4.2.8 Capability

Capabilities are used to wrap events, plans and data into reusable components. An agent can 'have' a capability that again can be composed of other capabilities (capability nesting).

4.2.9 Beliefset

JACK Beliefsets are a form of representing an agents belief. A Beliefset is a relational representation where the individual belief representations are propositional. It's like a relational database, but not used for long-term storage or shared between agents. The reason for not sharing Beliefsets amongst agents is to avoid concurrent data updates. A Beliefset may be shared, but there are concurrency issues due to multi-threading and it's therefore normally not done. Technically a Beliefset is a relation which is a set of tuples where each tuple is a belief/fact that can be either true or false. The tuples must have one or more fields, with an unique key field and value field(s). Beliefs can be queried on and changed/added/removed as the agent changes it's beliefs in run time. The change of an agents belief may result in change of behavior and this is invoked by callback methods posting Events that in turn are handled by relevant plans. Beliefsets must be declared in the agents, capabilities and plans that use them.

4.2.10 View

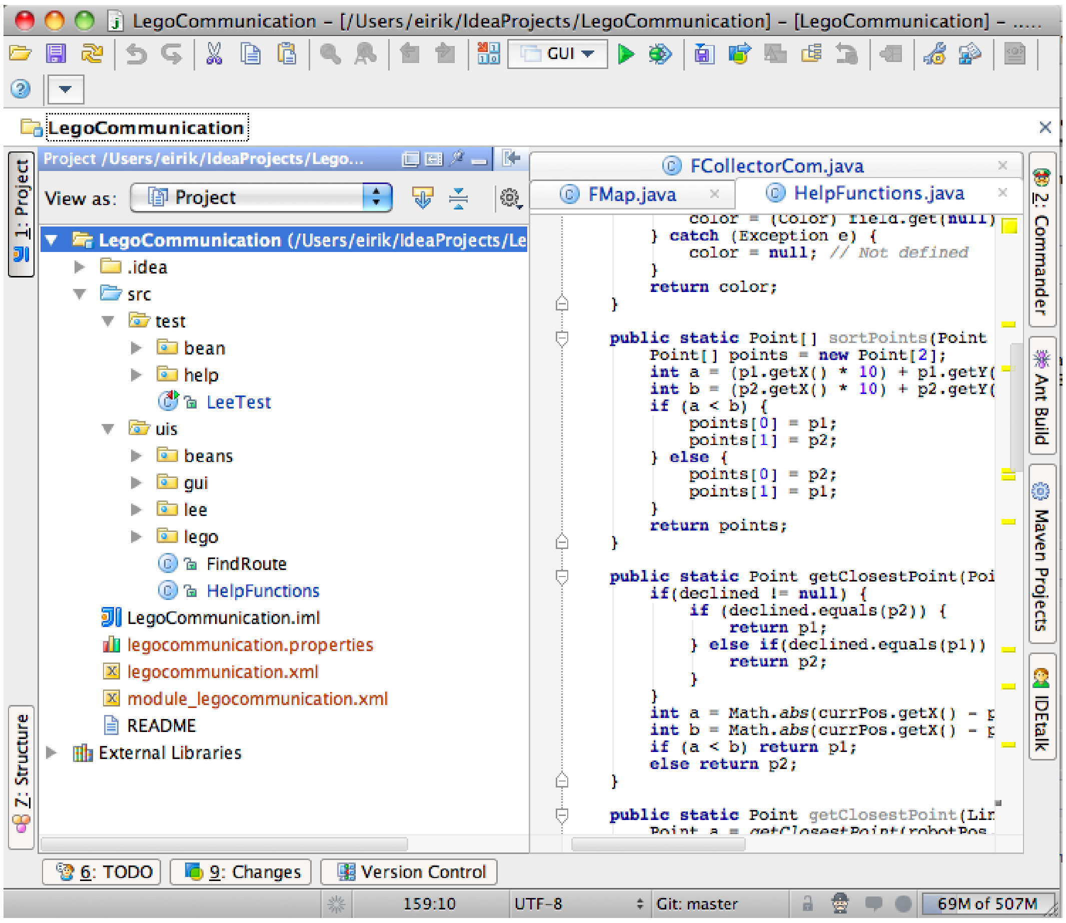
A JACK view is a way to interface between JACK and other systems. Using views it is possible to integrate a range of data sources into the JACK framework like Beliefsets, java data structures and legacy systems. Views must be declared in the agents, capabilities and plans that use them.

4.3 Java

Java [13] is a programming language originally developed by Sun Microsystems. The language derives much of its syntax from C and C++ but has a simpler object model and fewer low-level facilities. Java applications are typically compiled to byte-code that can run on any Java virtual machine (JVM) regardless of computer architecture.

4.4 IntelliJ IDEA

IntelliJ IDEA is a commercial Java IDE by JetBrains [14]. It is often simply referred to as "IDEA" or "IntelliJ." IntelliJ IDEA offers smart, type-aware code completion. It knows when you may want to cast to a type and is also aware of the run-time type checks that you made, after which you can perform cast and method invocation in a single action.



```
            color = (Color) field.get(null)
        } catch (Exception e) {
            color = null; // Not defined
        }
        return color;
    }

    public static Point[] sortPoints(Point
        Point[] points = new Point[2];
        int a = (p1.getX() * 10) + p1.getY(
        int b = (p2.getX() * 10) + p2.getY(
        if (a < b) {
            points[0] = p1;
            points[1] = p2;
        } else {
            points[0] = p2;
            points[1] = p1;
        }
        return points;
    }

    public static Point getClosestPoint(Poi
        if(declined != null) {
            if (declined.equals(p2)) {
                return p1;
            } else if(declined.equals(p1))
                return p2;
            }
        }
        int a = Math.abs(currPos.getX() - p
        int b = Math.abs(currPos.getX() - p
        if (a < b) return p1;
        else return p2;
    }

    public static Point getClosestPoint(Lin
        Point a = getClosestPoint(robotPos
```

Figure 10: The IntelliJ IDEA graphical user interface

4.5 LeJOS, Java for Lego Mindstorms

To allow us to program our LEGO robots using Java we used LeJOS NXJ which is a Java programming environment for the Lego Mindstorms NXT. The leJOS NXJ is a complete firmware replacement for the standard Lego Mindstorms firmware that includes a Java Virtual Machine. LeJOS is an open source project and was originally created from the tinyVM project that implemented a Java VM for the older Mindstorms system RCX. The current newest version and the one we used is lejos-NXJ 0.8.5 beta and it is supported by three operating systems: Microsoft Windows, Linux and MAC OS X. It consists of [15]:

- Replacement firmware for the NXT that includes a Java Virtual Machine.
- A library of Java classes (classes.jar) that implement the leJOS NXJ Application Programming Interface (API).
- A linker for linking user Java classes with classes.jar to form a binary file that can be uploaded and run on the NXT.
- PC tools for flashing the firmware, uploading programs, debugging, and many other functions.
- A PC API for writing PC programs that communicate with leJOS NXJ programs using Java streams over Bluetooth or USB, or using the LEGO Communications Protocol (LCP).
- Many sample programs

4.6 LEGO Mindstorms

LEGO Mindstorms is a programmable robotic kit created by LEGO. The LEGO Mindstorms NXT 2.0, which is the newest version, comes with a NXT Intelligent Brick, two touch sensors, a color sensor and an ultrasonic sensor. It also includes three servomotors as well as about 600 LEGO Technic parts.

The NXT Intelligent Brick is the main component of the robot. It can take input from up to four sensors and control up to three motors simultaneously. The brick also has a LCD display, four buttons and a speaker.

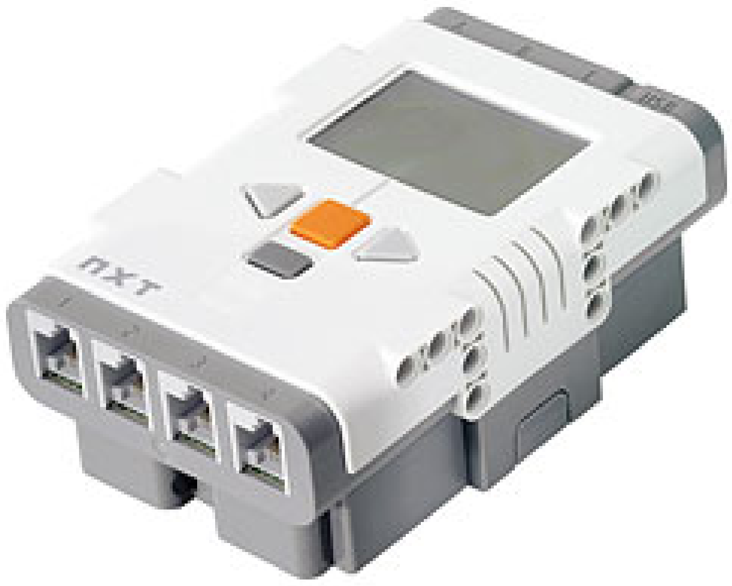


Figure 11: The NXT 2.0 Intelligent Brick

Originally the brick comes with software based on National Instruments LabVIEW [16], and can be programmed trough a visual programming language. LEGO has however released the firmware for the brick as open source [6], and several developer kits are available. Due to this, third party firmware has been developed to support different programming language, such as Java, C++, python, Perl, Visual Basic and more.

5 Application

This chapter will present the chosen application scenario and describe the process of defining it before ending up with the final approach. To achieve the common application goal as well as address the inter agent communication hypothesis, the application needed to contain several robots, collaborating to reach a common goal. An other important aspect was to create a scenario where the robots continuously could benefit from sharing each other views and information about the environment. Several approaches where tried before the final solution was chosen.

5.1 Scenario

Based on the implementation goal specified together with our supervisors we defined a scenario which includes all the desired aspects described in Section 1.2. The scenario is: 3 robots with different properties, which in cooperation are to explore a restricted, unstructured and dynamic operational environment where different types of objects are located randomly. These objects are to be collected and sorted by color. The robots are to coordinate amongst themselves cooperating in achieving a common goal. Each robot is assigned a specific task depending on its abilities, one explores and locates the objects, one collects and deposits the objects found, while the last robot sorts the delivered objects based on object color.

5.2 First approach

We first started out wanting to have the robots operate within a map only specified by a set of boundaries. The robots where to do the navigation an positioning using a sonar sensor measuring distances to the boundary walls and possible obstacles, for example other robots. There are several localization algorithms/techniques used in robotics, but one has proven to be both computationally efficient and accurate making it the most widely used, this is the Monte Carlo Localization Algorithm (MCL) [17] [18].

5.2.1 Monte Carlo Localization

The basic idea of this approach is to estimate the robots position using sensor readings. Initially only the map boundaries are known and not the robots position. MCL generates a set of poses distributed randomly within the boundaries all having a weight representing the probability of the pose representing the actual robot position and a heading. Each time the robot moves MCL generate N new samples that approximate the robots position after the move. These samples are generated by randomly drawing a sample from the previous computed sample set with likelihood determined by their previous weight combined with the new sensor reading. This resampling is done each time the robot moves and will eventually determine the robots most likely position with high accuracy.

5.2.2 First approach development

After deciding on this approach we "built" a simple map and a robot with a sonic sensor shown in Figure 16 and Figure 17. The MCL algorithm was implemented in java with a graphical user interface showing the robots current pose set within the boundaries shown in Figure 12 13 14 15. These figures show a typical scenario where the robot moves several times before its most likely position is determined accurately.

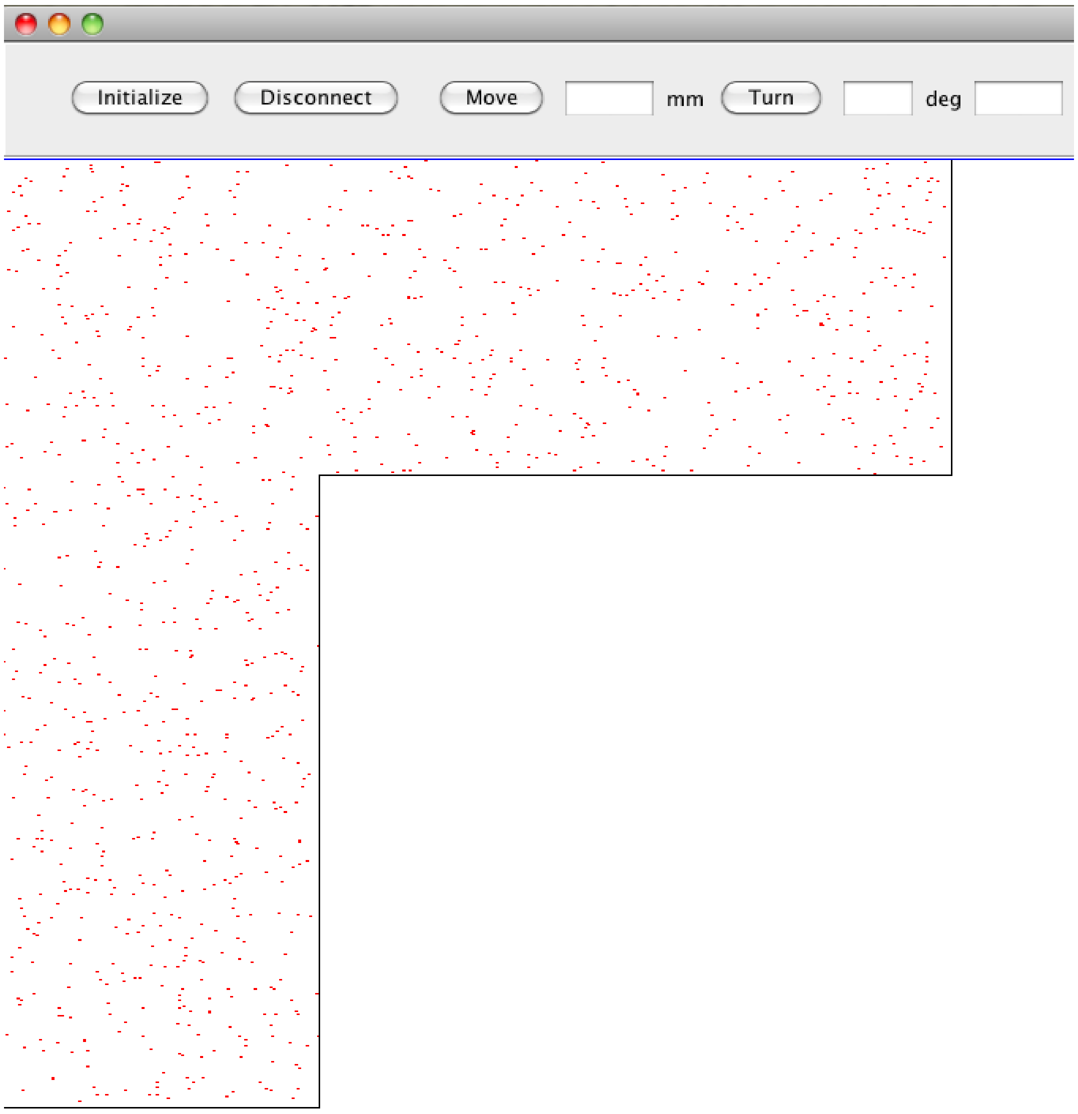


Figure 12: Monte Carlo Localization App initial pose.

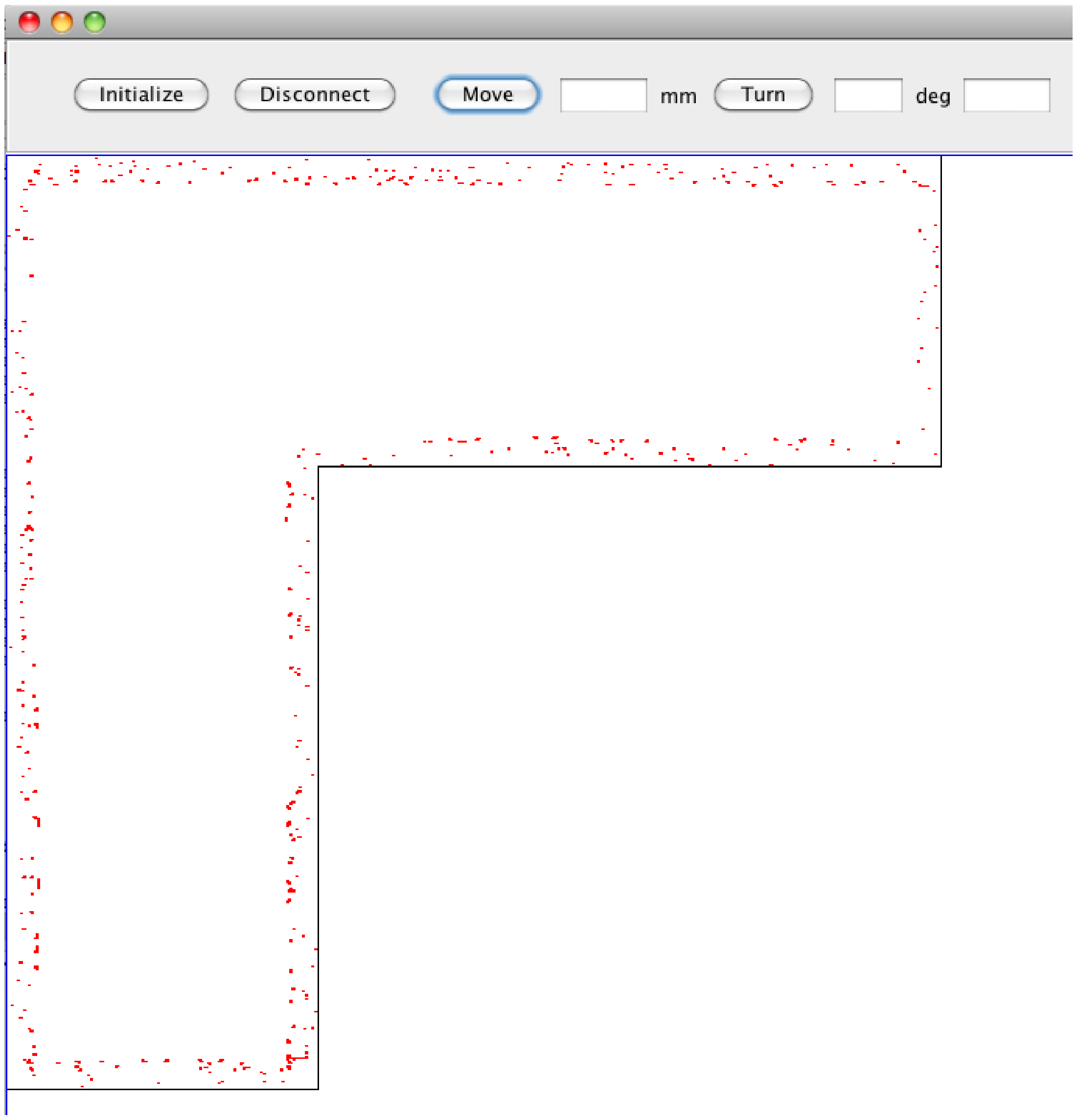


Figure 13: Monte Carlo Localization resampled pose set after first move

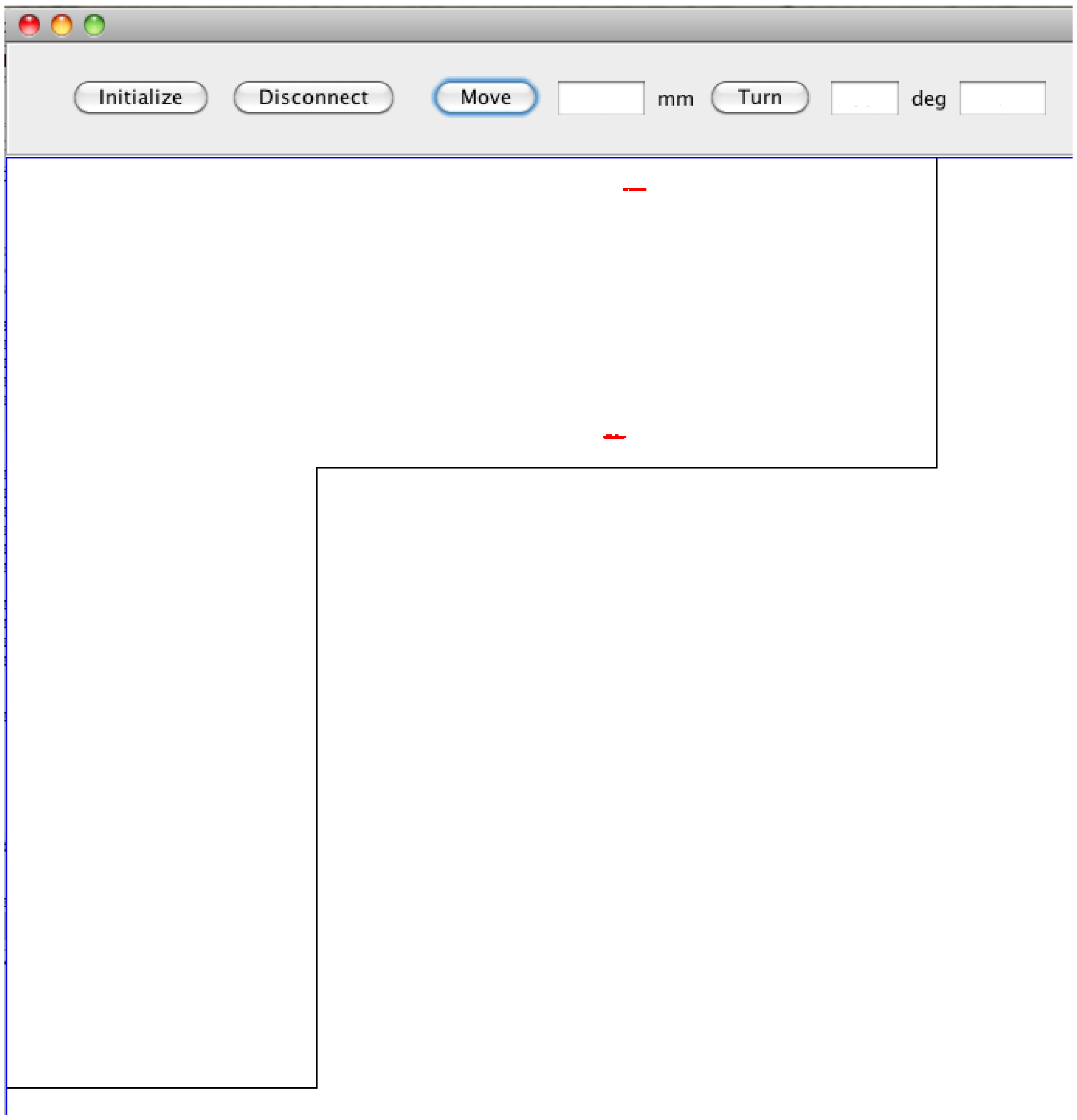


Figure 14: Monte Carlo Localization resampled pose set after several moves

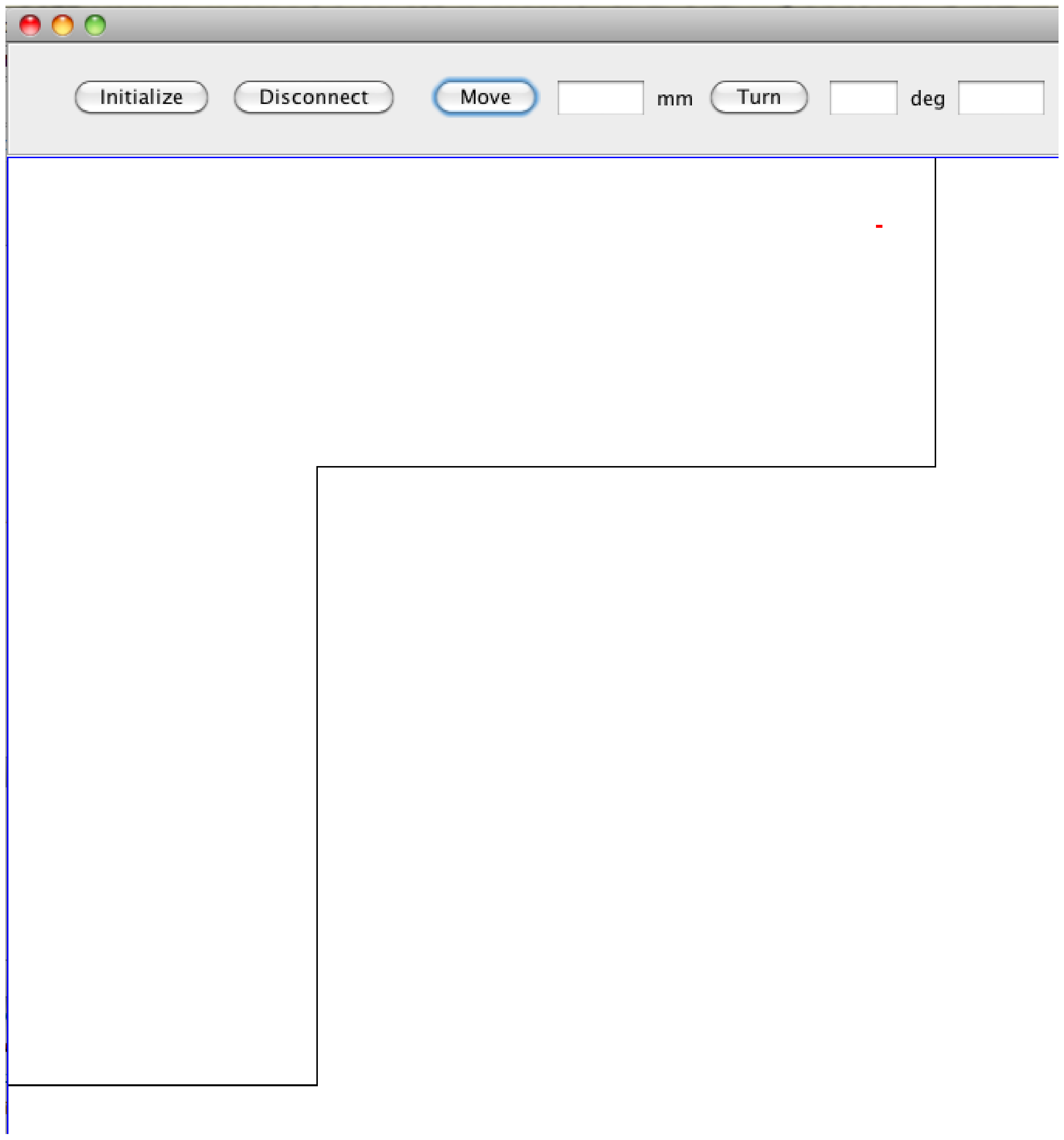


Figure 15: Monte Carlo Localization resampled pose set after location found.

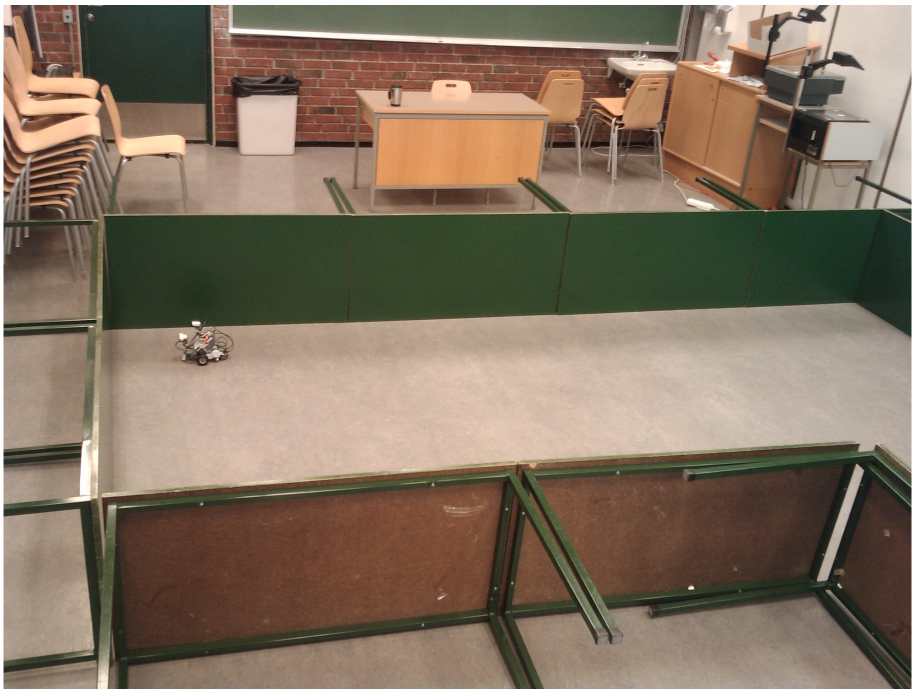


Figure 16: Robot located in MCL map

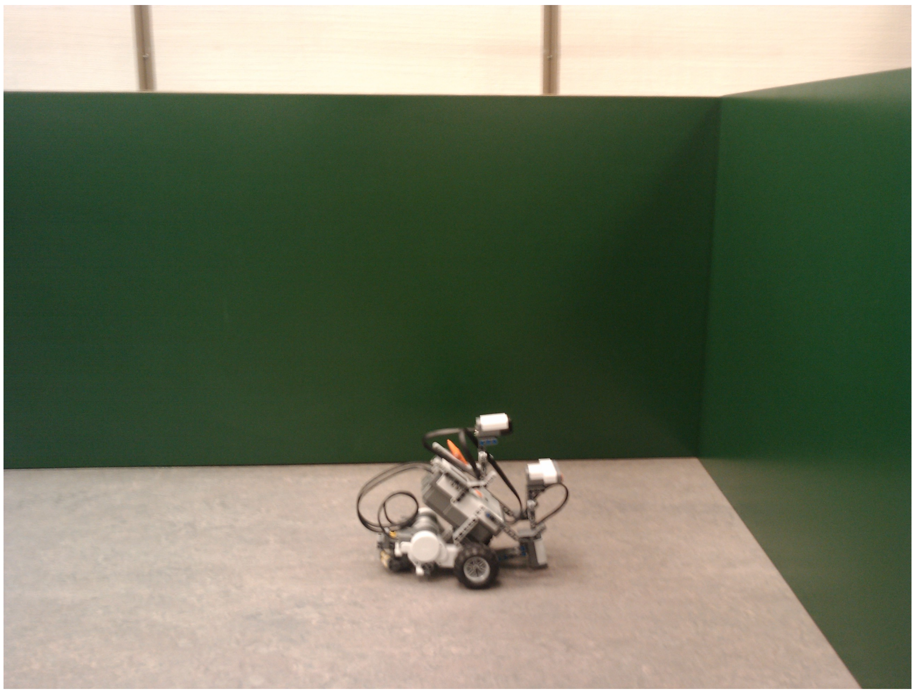


Figure 17: Robot located in MCL map, close up

5.2.3 First approach results

The MCL implementation was satisfactory in terms of accuracy and computational efficiency. Despite this the cons presented during testing heavily outweighed the pros of this approach. The LEGO Mindstorms sonic sensor was unreliable. Uncertainty in exact degrees turned and distance moved where both challenges, and the level of complexity in dealing with these issues increased drastically when more than one robot was introduced into the system. Due to time limitation and the main focus of the thesis being the software agent/HRI challenges we were forced to drop this approach after 1 month of development.

5.3 Final approach

After considering time limitations and the main focus of thesis, the final approach was specified. This approach is based on the robots operating on a line-based map/grid. This approach is preferable as Mindstorms robots have fairly good support for this kind of navigation (line following). There has been done quite a lot of projects on this leaving us to focus on more relevant challenges for the thesis, being the agent implementations and human-agent interfacing. The basic idea of robot setup and common goal remains the same as described in initial approach. A sketch of the overall grid design is presented in Figure 18

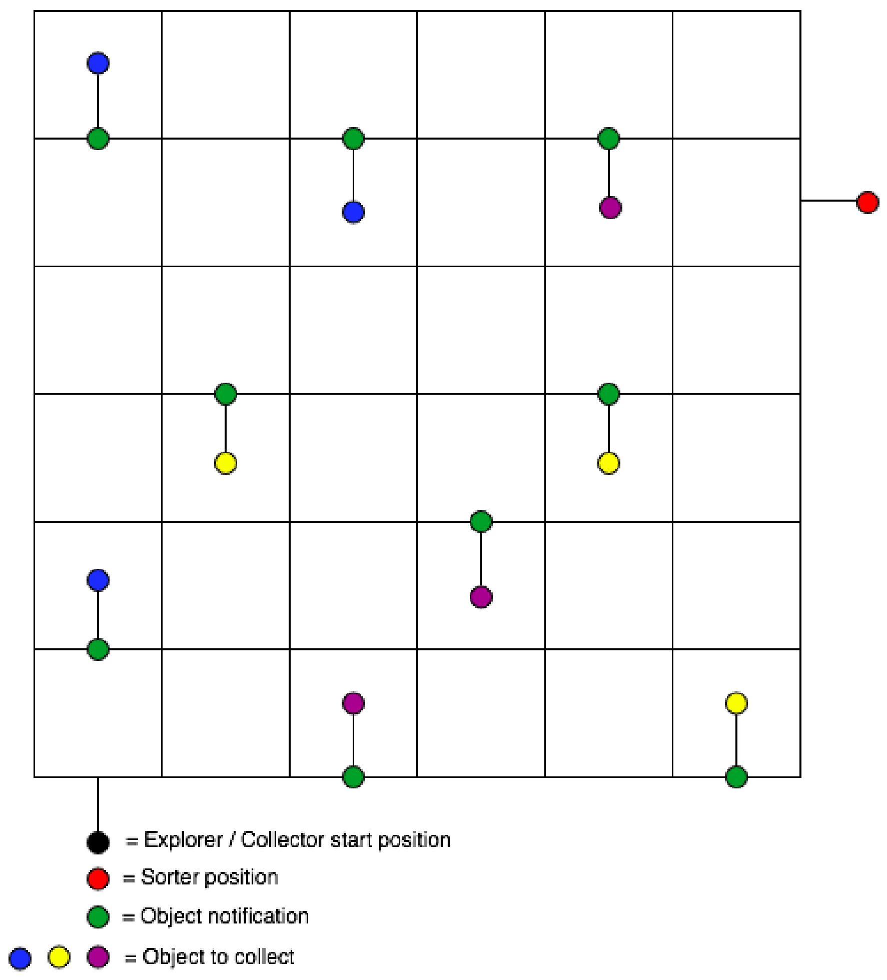


Figure 18: Grid-based map sketch

6 System Design

Our design is developed using the Prometheus methodology described in Section 4.1. This chapter will present the main phases of the design process and our design choices.

6.1 System specification

The system goals are derived from the scenario described in Section 5.3. To realize the system a set of main goals and sub goals where defined:

- Explore map
  −Find all drivable lines on the grid.
  −Find all objects located on the grid.
- Collect items
  −Pick up located items.
  −Deliver picked up items to be sorted.
- Sort all items located on the grid.
  −Sort items into trays based on color.
- Collision avoidance
  −Robots yield according to specified priority list. −Determine alternative routes on deadlock.
- GUI design based on best practice approach for successfull HRI.
  −Intuitiv GUI.
  −Keep operator focus on cruicial information.
  −Present results/data in user friendly manor.
  −Ease the load of data analysis for operator.

The required functionalities are defined based on these goals illustrated in Figure 19.

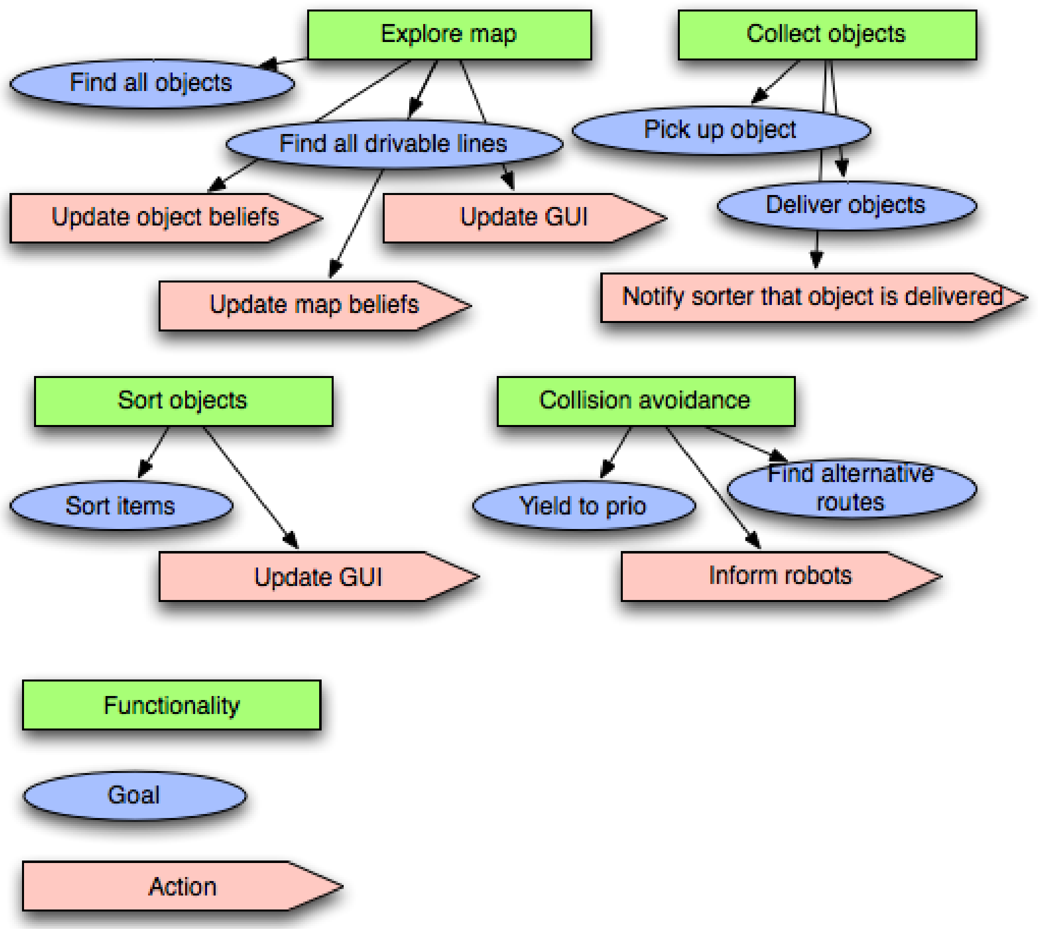


Figure 19: System functionalities based on goals

6.2 Architectural design

After defining goals and functionalities in the previous stage, 5 agents where identified to provide these functionalities and achieve the system goals. The agents and their specifications are shown in Figure 20:

- Agents for controlling the robots.
  −Explorer Agent.
  Agent with plans for controlling the explorer robot according to the defined goals. This agent communicates GUI updates and coordination requests as well as notifying the collector when items are discovered on the grid.
  −Collector Agent.
  Agent with plans for controlling the collector robot according to the defined goals.

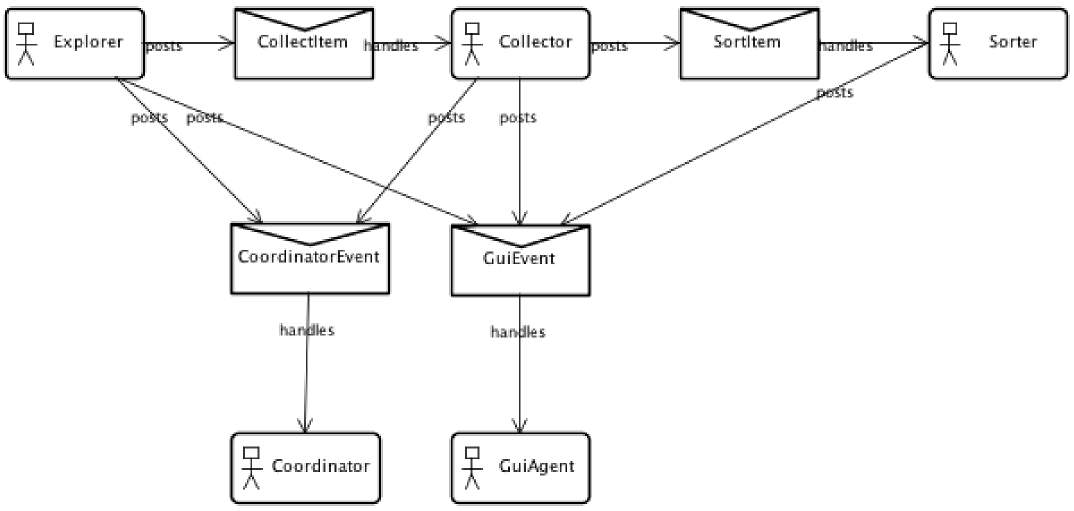


Figure 20: System Agents with basic interaction

Communicates GUI updates and coordination requests as well as notifying the sorter when items are deposited for sorting.
−Sorter Agent.
Agent with plans for controlling the sorter robot according to the defined goals. Communicates GUI updates and coordination, and handles sort requests from collector.

- Coordinator Agent.
  The agents main task is to handle the movement coordination between the 3 robots. Keeps track of robot positions and headings to ensure collision avoidance.

- GUI Agent.
  Handles all communication with the GUI/operator. Updates of the GUI as the robots gain more knowledge about their environment and also passes on user input to the robots/robot agents.

6.3 Detailed design

The system overview shown in Figure 23, describes how the agents communicate and access data. The design also supports the second hypothesis saying that robot interaction can be modeled as interacting agents. To complete the design it was however beneficial to add two additional agents, serving as support roles in the system. One agent for updating and interacting with the GUI, and one to handle coordination between the robots. It would have been possible to develop de application without the two extra agents, but this would have led to a less clearly design and would required sharing of data (views), as the three robot agents would have been forced to update the GUI through the GUI view.

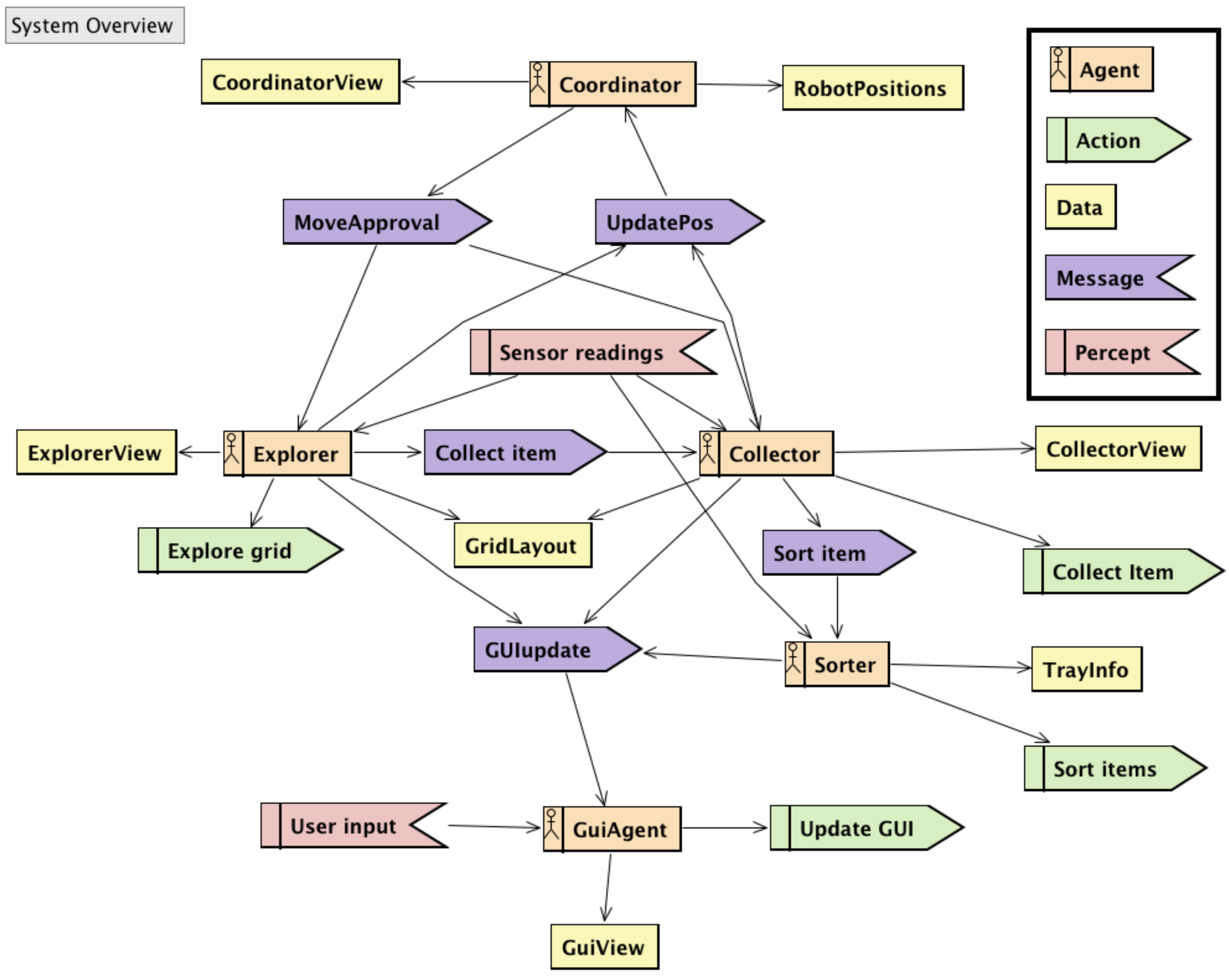


Figure 21: System overview

6.4 System - Robot communication design

The communication between the robots and the system will be done through Bluetooth. The Communication classes system side will send commands to the different robots where code for executing these commands will be running. Results and sensor readings sent from the robots will be received and interpreted by the communication classes before being passed on to the agents. An illustration of this design is shown in Figure 22.

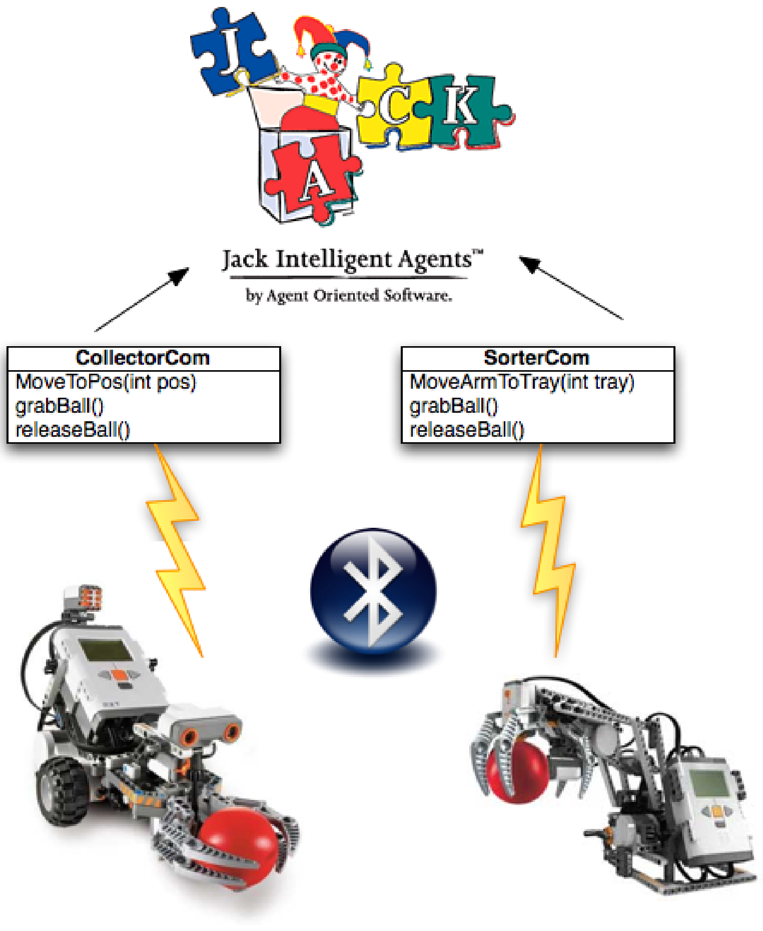


Figure 22: Communication design

6.5 Scenarios

Figure 23 shows the different scenarios that take place in the system. They are also described below together with their appurtenant steps. Note that only scenarios relevant to inter agent communication are described in detail.

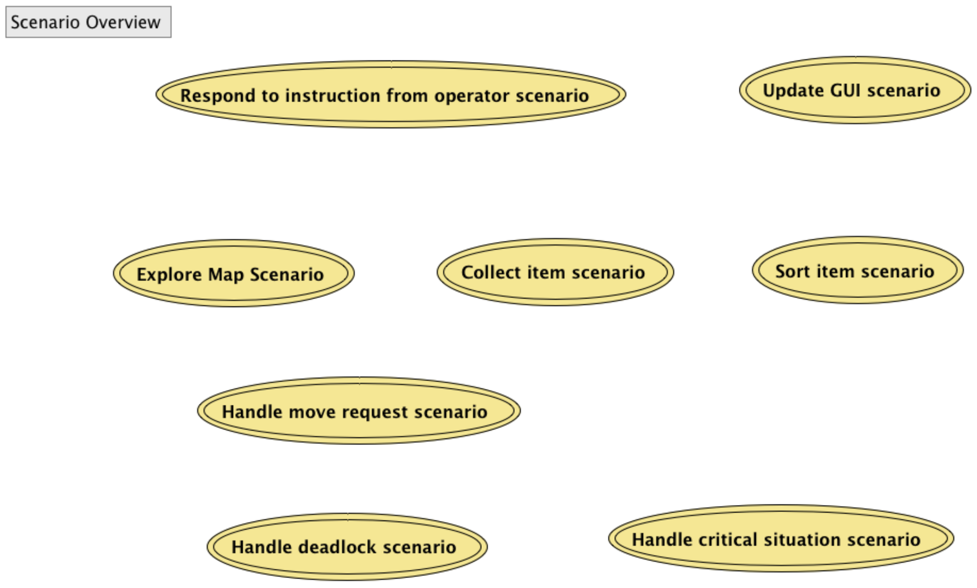


Figure 23: System scenarios

[S1] Explore Map

Trigger: User requests to start sorting

When the application starts, the operator has to initialize the robots connections and trigger the exploring. The Explorer robot then starts to explore the grid, and reporting back to the operator trough the user interface.

1. PERCEPT: New environment information through sensor readings.
2. GOAL: Explore grid
3. ACTION: Find available directions and line color
4. GOAL: Find objects
5. SCENARIO: S2

OR

6. SCENARIO: S5

[S2] Collect Item
Trigger: Explorer locates object to collect
When the explorer finds a object to collect, it notifies the collector which drives to the object, picks it up and delivers it to the sorter.
1. GOAL: Collect item.
2. GOAL Find shortest available path to object.
3. ACTION: Move to object.
4. ACTION: Collect object.
5. GOAL: Find shortest available path to sorter.
6. ACTION: Move to sorter.
7. ACTION: Deliver object to sorter.

[S3] Sort Item
Trigger: Collector delivers object to sorter.
The objects collected is delivered to the sorter, and is then sorted into the correct tray according to color.
1. PERCEPT: Object color through sensor.
2. GOAL: Sort item.
3. ACTION: Put object in tray according to color.

[S4] Handle Move Request
Trigger: Explorer or Collector requests to move
Every time a robot wants to move, he has to consult the coordinator agent to avoid deadlocks.
1. PERCEPT: Request from robot to move.
2. GOAL: Validate movement request.
3. ACTION: Reply to requestor.

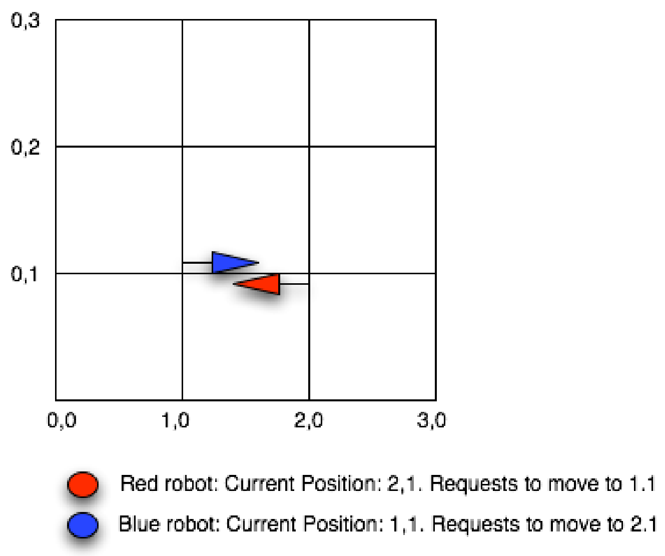

0,3
0,2
0,1
0,0
1,0
2,0
3,0
Red robot: Current Position: 2,1. Requests to move to 1.1
Blue robot: Current Position: 1,1. Requests to move to 2.1

Figure 24: A deadlock scenario, where two robots wants to move to each others location.

[S5] Handle Deadlock

Trigger: Two robots are located in a deadlock scenario, see Figure 24.
The robot with the ”lowest” priority needs to find an alternative route to its destination.
1. GOAL: Find a new route, that excludes driving through the other robots position.
2. ACTION: Follow new route to destination.

7 System Development

This chapter describes the implementation, see Figure 7 for symbol explantation.

7.1 Agents

This section will in short present the agents implemented in the system with a description and corresponding figures illustrating the workings of the individual agents.

7.1.1 Explorer

The explorer agent starts exploring when notified by the operator through the GUI. It uses a set of plans to achieve its objective to map out the available grid. It first checks available directions at its current position/intersection and stores this information in a beliefset. Based on available directions it chooses where to move and repeats step one at the next intersection until the entire grid is traversed. In addition to mapping it detects items to collect and notifies the collector agent during the exploration. The information obtained is continuously passed on to the GUI agent so that is can be presented to the operator. An overview of the explorer agent is shown in Figure 25.

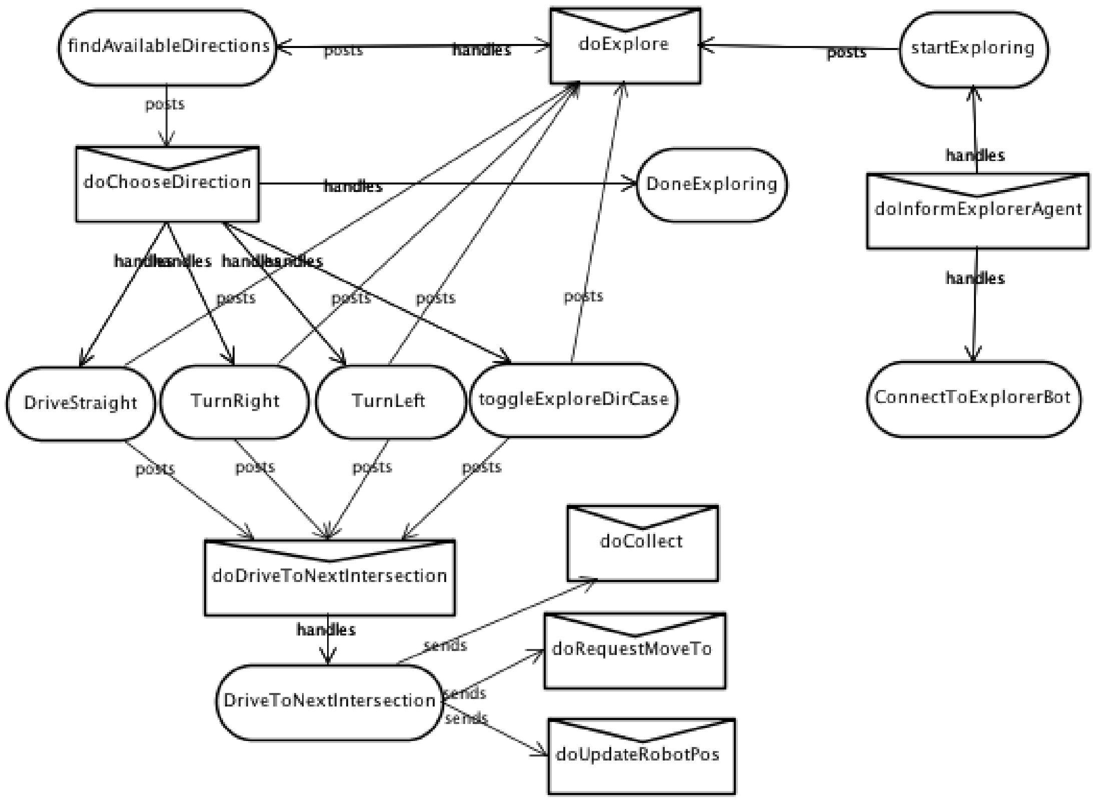


Figure 25: Explorer Agent overview

7.1.2 Collector

After being activated by the explorer, the collector agent first determines the shortest route to the item that is to be collected, then it moves to the item. The item is collected and a new shortest route to the sorter is determined before moving to deliver the item. After depositing the item the collector either repeats this sequence for next object to be collected or waits for a new notification from the explorer with item to collect. The GUI agent is continuously given information representing location and status of collection.

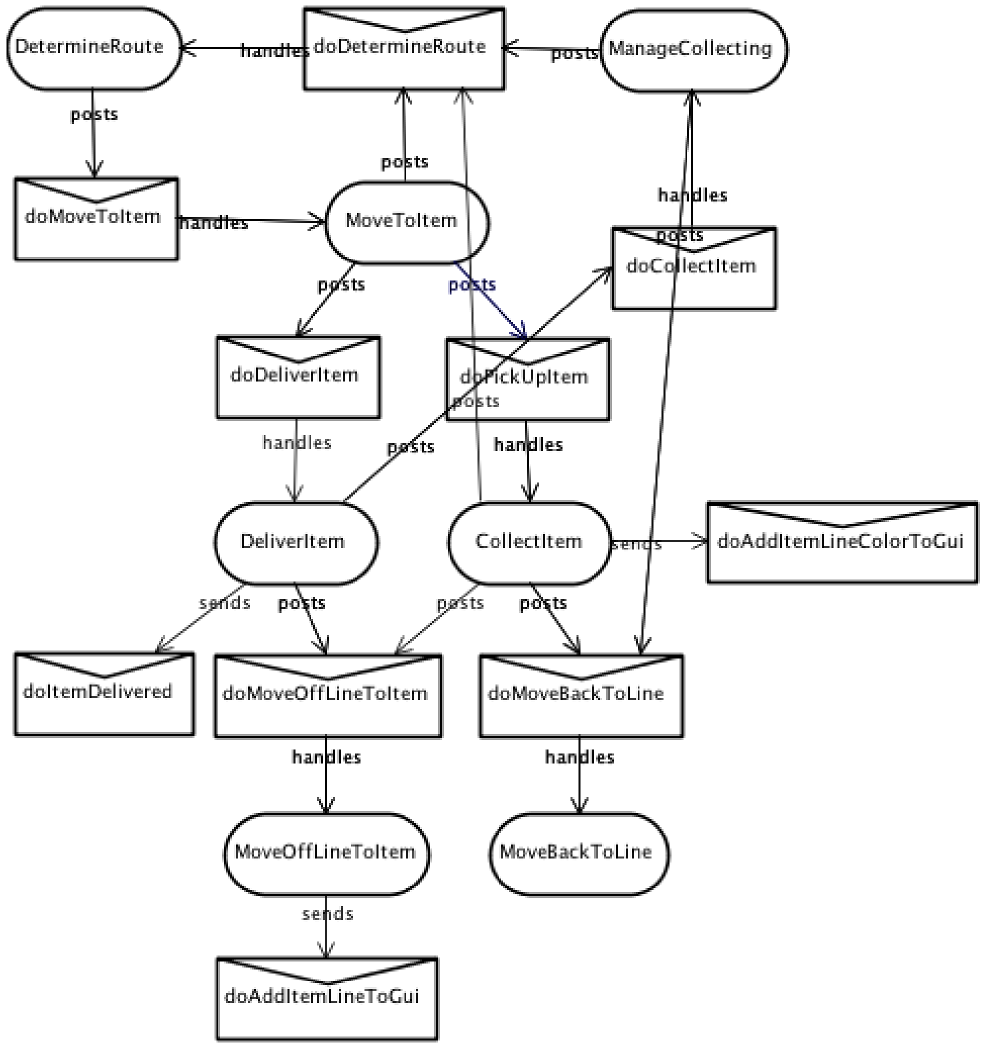


Figure 26: Collector Agent overview

7.1.3 Sorter

The Collector notifies the Sorter agent when a new object is ready to be sorted. The sorter then checks the object's color and queries its beliefset to see if the color already has a tray. If it has, the object gets placed in the same tray as the other objects of the same color, if not the object is put in to a new tray. The sorter also notifies the GUI agent that the object is sorted as displayed in Figure 27.

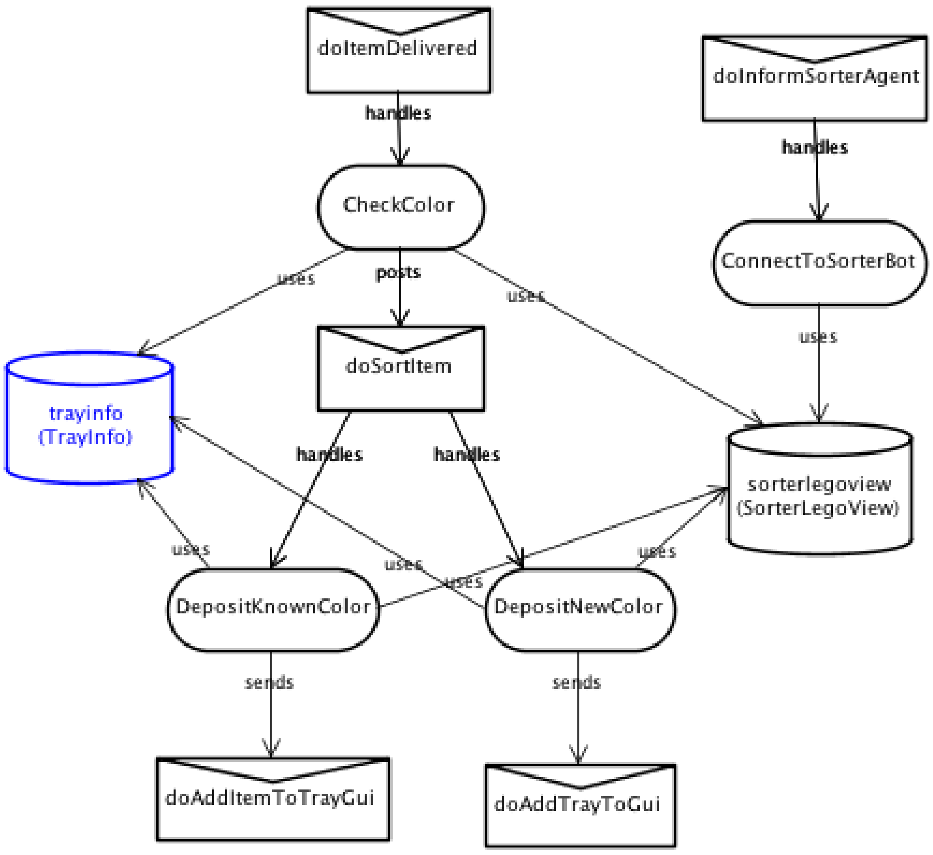


Figure 27: Sorter Agent overview

7.1.4 GUI Agent

The GUI Agent is responsible for handling communication with the external java graphical user interface. It handles events from the other agents and has plans for updating the GUI accordingly to the information received in these events. It also reacts to input from the GUI, and forwards the information to the relevant agents. Figure 28 and Figure 29 illustrate the workings of the GUI agent.

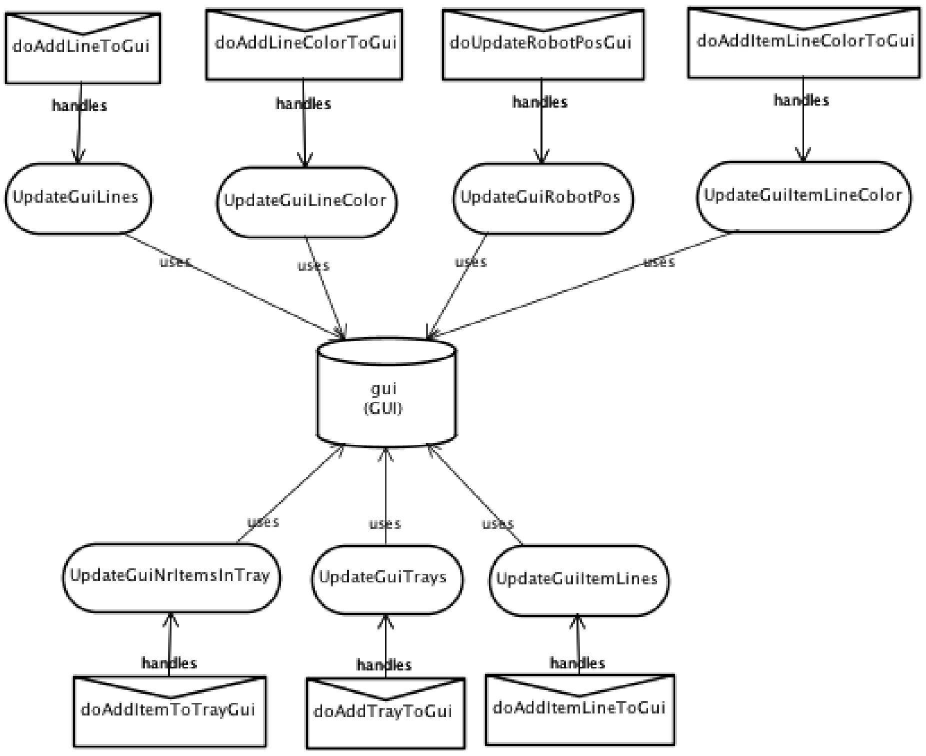


Figure 28: External communication from JACK to the GUI

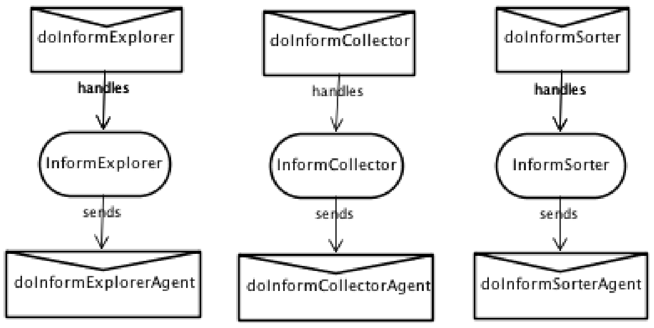


Figure 29: External communication from GUI to JACK

7.1.5 Coordination Agent

The Coordinator Agent is responsible for keeping track of the robots position and avoids deadlocks. The agent is also responsible for informing the GUI Agent about robot movement, as seen in Figure 30.

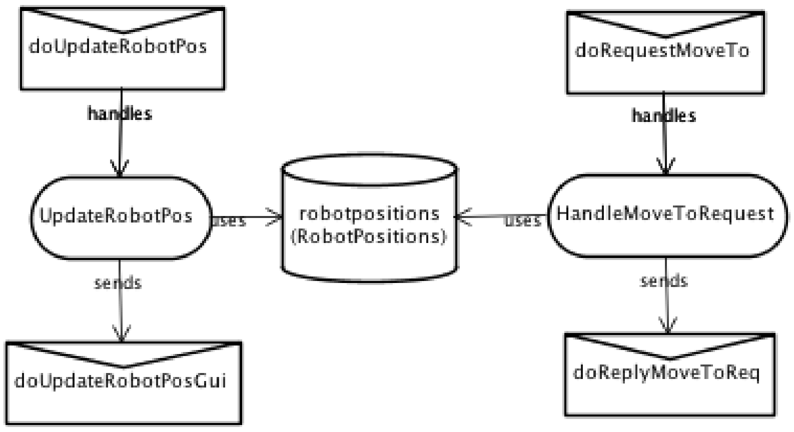


Figure 30: Coordinator Agent overview

7.2 Inter-Agent Communication

7.2.1 Interaction diagrams

To illustrate the interaction between agents, a set of sequence diagrams are created based on the scenarios in Section 6.5. The diagrams are based on object-oriented sequence diagrams [19].

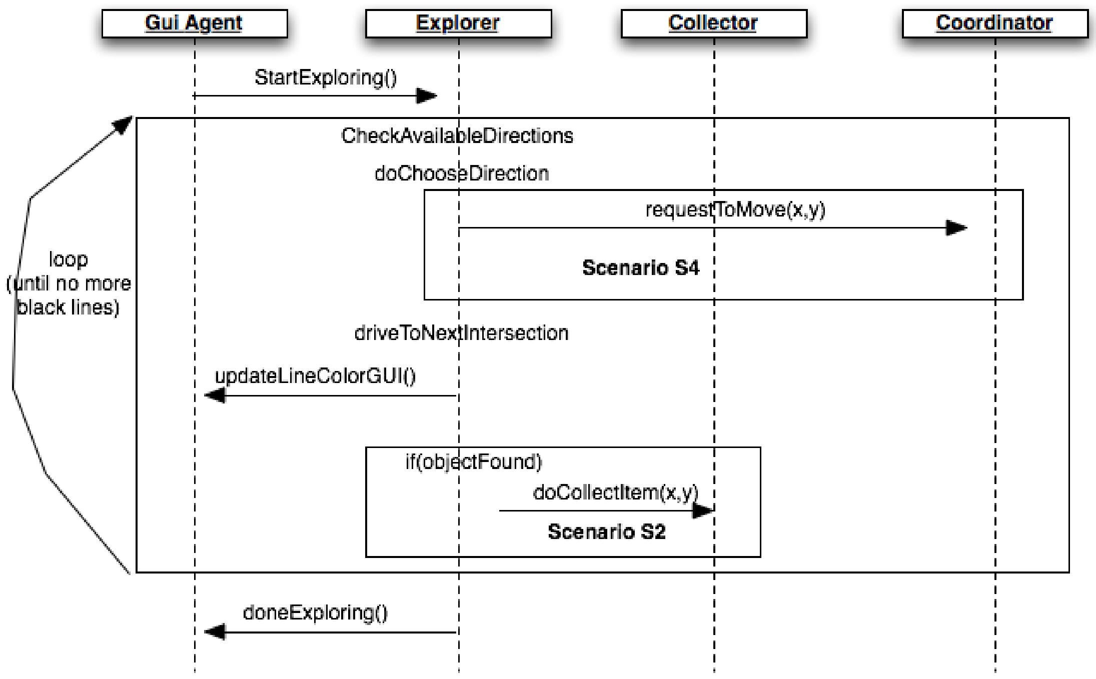


Figure 31: Scenario **S1** - Explore grid sequence diagram

Scenario S1 describes how the Explorer agent traverses the map as well as how and when it interacts with other agents. The operator pushing a button in the GUI triggers the exploring. The explorer starts out with checking which directions are drivable, updates it beliefs about the grid, and determines which way to drive. It then queries the coordinator agent, to asking if the selected line is not occupied, if the coordinator agent gives a positive reply the explorer moves to next intersection, if not it queries the coordinator until he is allowed to move till the requested destination. This is done each time the explorer comes to a new intersection, until the whole map is explored. The explorer also checks the line color when traveling over the lines, if a color other than yellow is found, it sends an event to the collector, informing that there is either a object located at the line, or the sorter is located on the line, depending on the color.

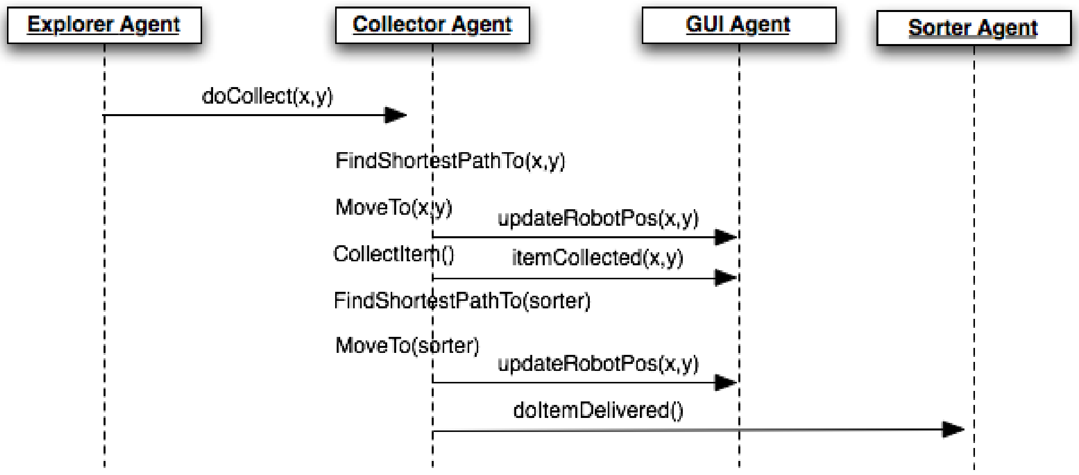


Figure 32: Scenario **S2** - Collect Item sequence diagram

Scenario S2 illustrates how the Collector Agent pick up objects. When the Explorer Agent located an object, it notifies the Collector, which then find the shortest path to the object using an recursive algorithm. It then moves one step at the time, querying the Coordinator to check that there is no obstacles or other robots in the chosen route. When it arrives to its destination the robot picks up the object, and updates the GUI with the information. It then moves to the sorter robot, which position is received from the explorer when found. The collector then delivers the item, notifies the sorter that the object is ready for sorting. When the object is delivered the collector queries its beliefset to see if a new item is found and ready to be collected, if it is not, it waits until a new object is found.

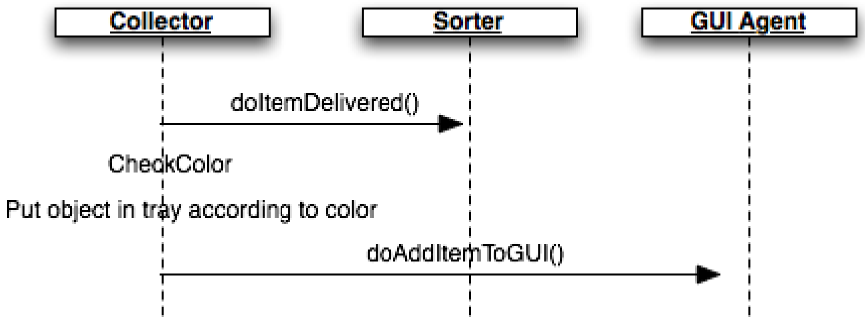


Figure 33: Scenario **S3** - Sort Item sequence diagram

Scenario S3 displays the workings of the sorter agent. When the Collector agent delivers a new object, the sorter picks up the object, checks its color, and sorts it into the corresponding tray. If a object of that color already has been sorted, the agent finds the tray number from its beliefset, if not it adds the color to the beliefset with a new tray number. It also updates the GUI, with the new tray, and number of objects in that tray.

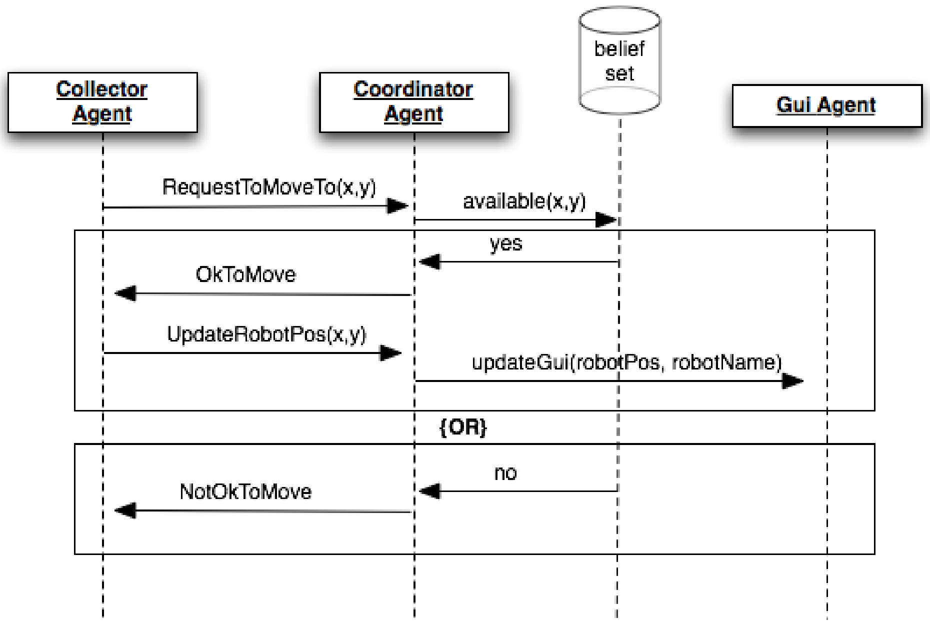


Figure 34: Scenario **S4** - Request to move sequence diagram

Scenario S4 describes in detail how the robot controlling agents communicate with the coordinator agent. When one of the robots wants to move, they have to ask the coordinator agent if they can move to that location, this is done to avoid collisions. If there currently is an other robot at that location, the coordinator declines the request, and the agent that made the request has to wait or find an other route to its destination. If the robot is allowed to move, it moves to the desired destination and sends a new notification to the coordinator, with its updates position. The coordinator forwards this to the GUI agent, which updates the GUI.

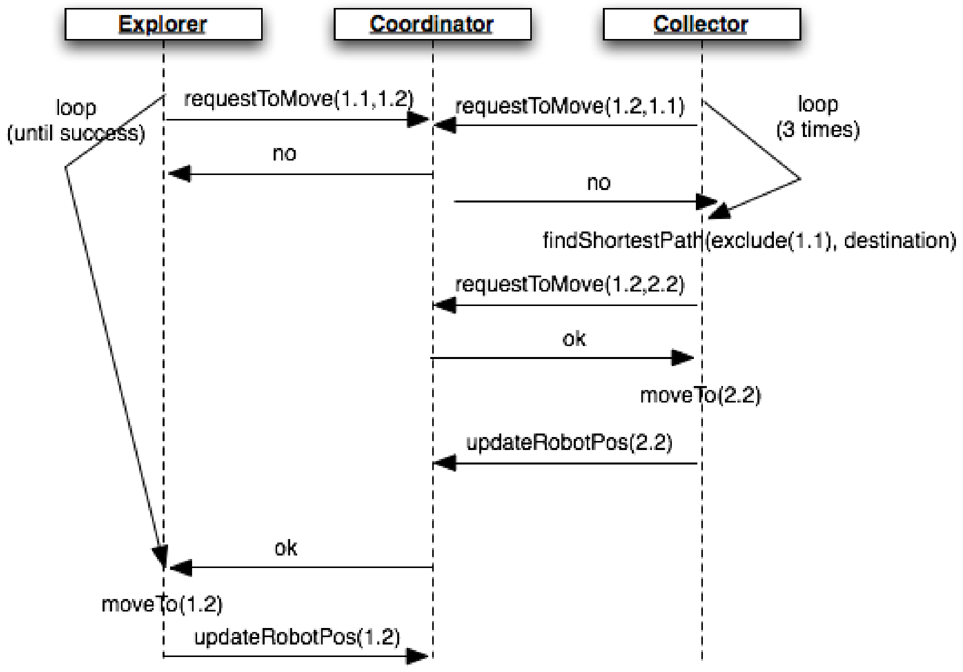


Figure 35: Scenario **S5** -Deathlock sequence diagram

Scenario S5 covers the deadlock scenario. In our system, we only have two robots, which travels the map, and hence the Explorer is given priority when a deadlock occurs. When this happens, as displayed in Figure 24, they both get declined to move by the coordinator, however, when the Collector is declined three times, it generates a new route, excluding the position which it is not allowed to move till, and the problem is solved.

7.2.2 Messages

This section will give a brief overview of how and what information the different agents communicate to each other and how the messages effect the other agents.

Explorer To Collector

The Explorer communicate two different events to the Collector, the first and most obvious event is sent every time the Explorer locates a new object that needs to be collected. Also, since the Sorter position is not initially known, the Explorer needs to inform the Collector when he has found the Sorters location.

Message:doCollect	
Description	A message event notifying the Collector that there a new object to collect.
Sender	Explorer agent
Receiver	Collector agent
Information	The line where the object is located

Table 2: New object to collect

Message:doRegisterSorterPos	
Description	A message event informing the Collector where the Sorter is located.
Sender	Explorer agent
Receiver	Collector agent
Information	The position of the Sorter

Table 3: Sorter position information

Collector to Sorter

Every time the Collector delivers a item to the Sorter, he needs to notify the Sorter that a new object is ready to be sorted.

Message:doItemDelivered	
Description	A message event informing the Sorter that there is a new object available to sort.
Sender	Collector agent
Receiver	Sorter agent
Information	nothing

Table 4: New item delivered to Sorter.

Explorer and Collector To Coordinator

Both the Explorer and the Collector needs to cooperate with the Coordinator agent to be able to move around the grid. This is to ensure that they do not crash into each other. Before they can move to a new position, they request to move, and if they get approval, they send a new message to the Coordinator with their updates position. This is done each time they move on the grid.

Message:doRequestToMove	
Description	A message event used to request to move to a new position.
Sender	Explorer or Collector agent
Receiver	Coordinator agent
Information	Robot name, current position and requested position

Table 5: Request to move event

Message:doUpdateRobotPos	
Description	A message event informing the coordinator that the sending agent has moved to a new position
Sender	Exporer or Collector Agent
Receiver	Coordinator agent
Information	Robot name, robot position and robot heading.

Table 6: Update robot position event

Coordinator to Explorer and Collector

The Coordinator needs to reply to the Explorer and Collector each time they request to move, with either yes, you can move, or no, you can not move to that position at this time.

Message:doReplyMoveToReq	
Description	A message event replying to a move event from either the Explorer or Collector agent.
Sender	Coordinator agent
Receiver	Explorer or Collector agent
Information	an answer to the request, either yes or no.

Table 7: Reply from Coordinator to move request.

7.3 Robot development

The Lego implementation was not a priority during the development due to the limitations discovered relatively early in the process. Because of this the only fully implemented robot code is for the explorer robot. The collector robots code is partially implemented.

7.3.1 Communication protocol

Bluetooth is used to send commands between the robot and system. Due to the limitations of Bluetooth technology such as high latency and low bandwidth we want to keep the communication protocol as simple as possible. The server sends its command in the form of three bytes, the first byte is the command it self, and the two following bytes are optional parameters. The robots reply is always 8 bytes which is enough to accommodate the most advances replies needed. For the different robot commands there are several cases to consider shown in tables 8, 9, 10.

Description	**Command**	**Reply**
Battery voltage request	[0,0,0]	[millivoltage,0,0,0,0,0,0,0]
Request to travel a given distance with or without checking the traveled lines color	[1, distance, boolean checkcolor]	[linecolor, 0,0,0,0,0,0,0]
Request to turn given degrees	[2,degrees,0]	[0,0,0,0,0,0,0,0]
read the color at current position	[3,0,0]	[color, 0,0,0,0,0,0,0]
Perform sweep at current location to discover available directions	[4,0,0]	[boolean straight, boolean left, boolean backwards, boolean right, 0, 0, 0, 0]
Disconnect bluetooth	[5,0,0]	[255, 255, 255, 255, 255, 255, 255, 255]

Table 8: Explorer robot communication protocol

Description	**Command**	**Reply**
Battery voltage request	[0,0,0]	[millivoltage,0,0,0,0,0,0,0]
Request to travel a given distance with or without checking the traveled lines color	[1, distance, boolean checkcolor]	[linecolor, 0,0,0,0,0,0,0]
Request to turn given degrees	[2,degrees,0]	[0,0,0,0,0,0,0,0]
read the color at current position	[3,0,0]	[color, 0,0,0,0,0,0,0]
Perform sweep at current location to discover available directions	[4,0,0]	[boolean straight, boolean left, boolean backwards, boolean right, 0, 0, 0, 0]
Disconnect bluetooth	[5,0,0]	[255, 255, 255, 255, 255, 255, 255, 255]
Grab object	[6,0,0]	[0, 0, 0, 0, 0, 0, 0, 0]
Release object	[7,0,0]	[0, 0, 0, 0, 0, 0, 0, 0]

Table 9: Collector robot communication protocol

Description	**Command**	**Reply**
Battery voltage request	[0,0,0]	[millivoltage,0,0,0,0,0,0,0]
Move object to tray position	[1, traynumber, 0]	[0, 0, 0, 0, 0, 0, 0, 0]
Read the color of object	[2,0,0]	[color, 0,0,0,0,0,0,0]
Grab object	[3,0,0]	[0, 0, 0, 0, 0, 0, 0, 0]
Release object	[4,0,0]	[0, 0, 0, 0, 0, 0, 0, 0]
Disconnect bluetooth	[5,0,0]	[255, 255, 255, 255, 255, 255, 255, 255]

Table 10: Sorter robot communication protocol

7.3.2 Internal robot code

The code located on the robot NXT brick is intended to provide as much functionality as possible with minimal amount of data send using Bluetooth. At first, the robot waits for a Bluetooth connection. Once a connection is made, it waits to receive its three-byte command. Once the command is received, the robot moves or turns, if necessary, and then sends back its eight-byte reply one byte at a time. The robot then waits for its next command. If the robot is commanded to terminate its Bluetooth connection, the robot sends back its acknowledgement, disconnects, and its program terminates on the brick.

The traveling is implemented using a PID algorithm [20] which ensures that the robot stays on the line by constantly reading light values and readjusting accordingly. The code for this is shown as follows:

```
private void PIDmove(int length) {
    int lightValue;
    int turn;
    int powerA;
    int error;
    int powerC;
    int lastError = 0;
    int derivative;
    resetTacho();
    while (getMM(motorA.getTachoCount()) < length) {
        lightValue = colorLightSensor.readValue();

        error = lightValue - offset;
        derivative = error - lastError;
        turn = (kp * error) + (kd * derivative);
        turn = turn / 100;
        powerA = tp - turn;
        powerC = tp + turn;

        if (powerA > 0) {
            motorA.setPower(powerA);
            motorA.forward();
        } else {
            powerA = powerA * (-1);
            motorA.setPower(powerA);
            motorA.backward();
        }
        if (powerC > 0) {
            motorC.setPower(powerC);
            motorC.forward();
        } else {
            powerC = powerC * (-1);
            motorC.setPower(powerC);
```

```
                motorC.backward();
            }
            lastError = error;
        }
        motorA.stop();
        motorC.stop();
    }
```

7.3.3 System side code

On the system side a communication class is developed for each of the robots interfacing between the robots and the agents. These classes are responsible for sending the commands one byte at a time to the robots and await replies. Once a reply starts being sent, the communication classes read each byte, one at a time, placing them in eight-byte arrays for interpretation before the results in turn are sent to the agents. The communication classes must implement interfaces defining required functionality for the given robot.

7.4 GUI implementation

The graphical user interface displays state information of the system with explored parts of the grid, items discovered and sorted. The different robots are also shown together with their corresponding movements and headings. The GUI implementation does not provide much functionality for operator input/influence as the implementation of the agent system is based on a structured environment due to time and LEGO Mindstorms limitations. Currently the only influence an operator has is to initialize the connections between the agents and the robots and start the system with a "Start" button. Figure 36 shows what the different components represent. A screenshot of the GUI with connections initialized is shown in Figure 37.

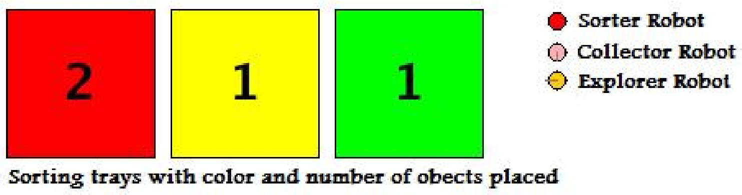


Figure 36: Gui components and what they represent.

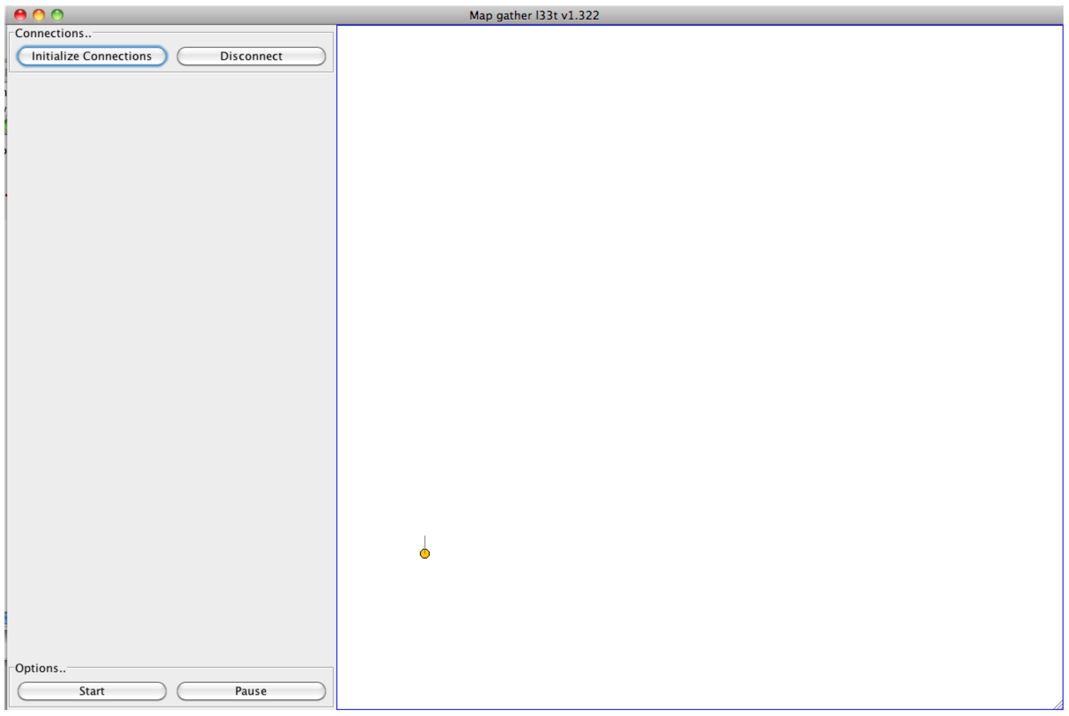


Figure 37: Gui after connections have been initialized.

After initialization of connections the operator can start the system by pressing the start button. Figure 38 shows the system during a normal run.

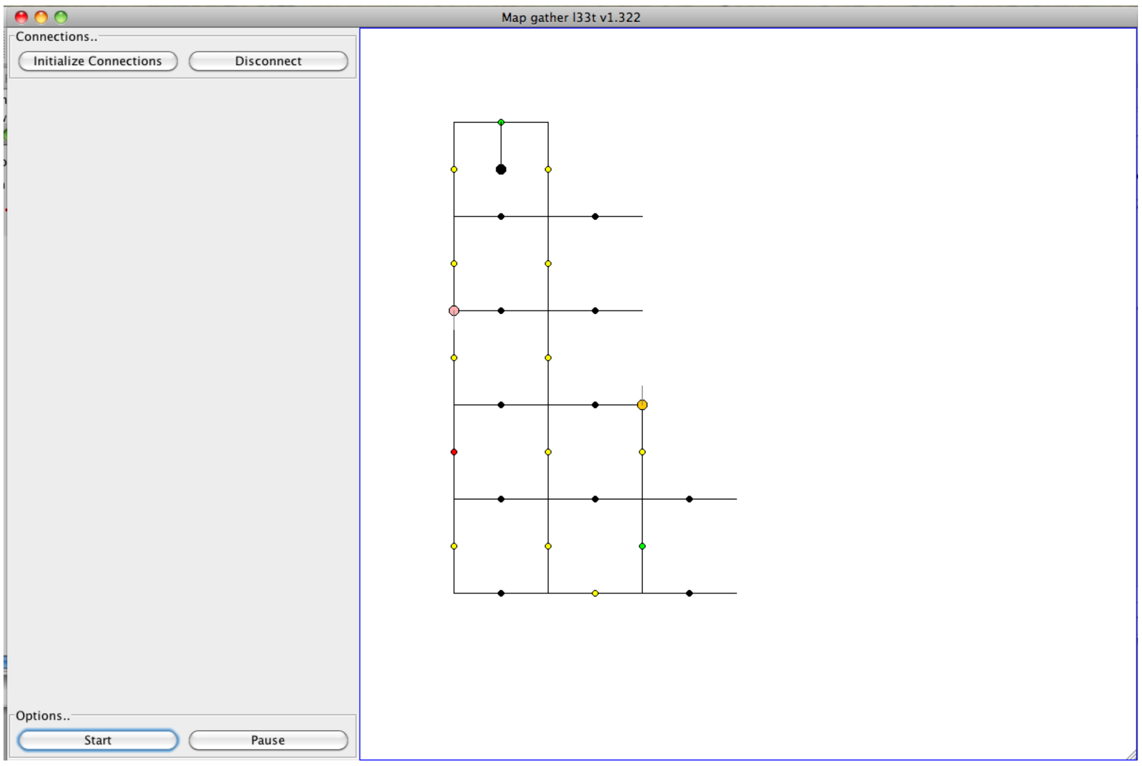


Figure 38: Gui some time after the start command is given.

While the explorer has traversed the entire grid the collector has collected items and delivered them to be sorted. In Figure 39 the entire grid is explored and a set of items have been collected and sorted by color.

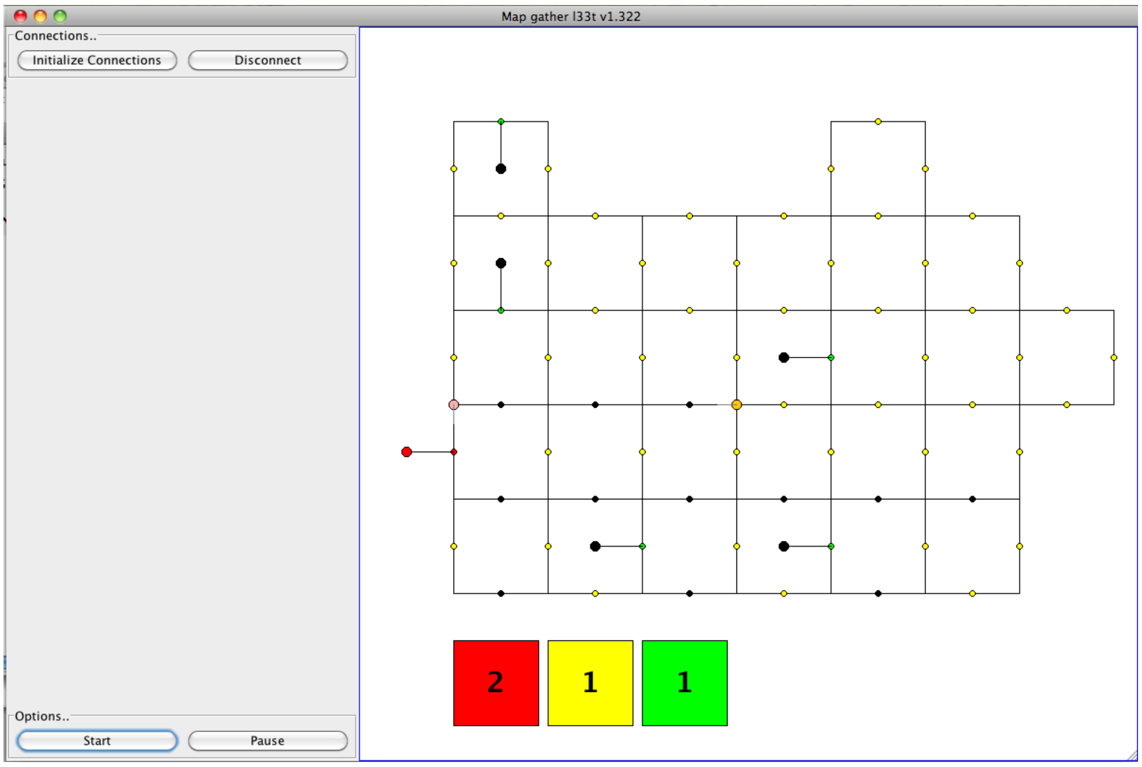


Figure 39: Gui after complete exploration (all objects not yet collected and sorted).

8 Results

This chapter presents the final solution with corresponding implementations, an overview of challenges met during the thesis work and an evaluation of the issues presented in Section 1.2 relative to the final implemented solution.

8.1 Final solution

The implementation goals set for this thesis where achieved with exception of a complete LEGO Mindstorms specific implementation of the defined interfaces. The Lego implementation was not a priority during the development due to the limitations discovered relatively early in the process. This lead to the final solution of implementing a set of java classes (mocks) [21] representing the robots and simulating replies and sensor readings. The downside of this approach is the obvious structured environment in which the agents now operate opposed to the desired unstructured and dynamic environment where the benefits of intelligent agents would be more visible.

8.1.1 LEGO robots and code

Three LEGO Mindstorms robots built according to TriBot [22] and RobotArm [23] schematics with modifications to meet our specific needs. To enable java programming on the Mindstorms intelligent brick the firmware was replaced with LeJOS [15]. Code for continuously receiving user commands and replying with results is implemented for the robots to run on the intelligent brick.

8.1.2 GUI and external java code

A graphical user interface is developed for the operator to interact with the robots. The operator can give input and observe a graphical representation of the robots, sensor readings and results during runtime. The GUI uses both color and placement to direct operator focus towards critical information. The GUI is shown in Figure 40.

Most of the algorithms used are implemented in pure Java, and used as external classes by the agents.

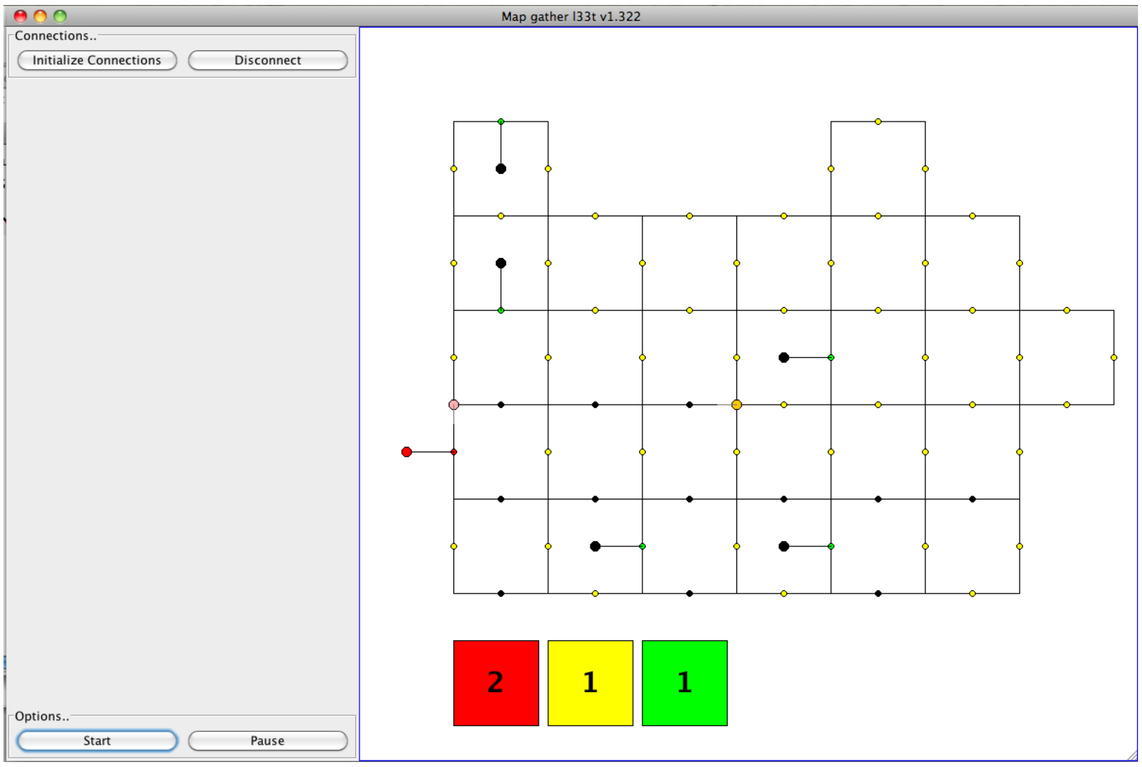


Figure 40: Graphical User Interface

8.1.3 Agent system

A total of 5 agents with respective views, beliefsets, plans and events where implemented, where three represent robots, one interacts with the GUI and the last agent is responsible for robot movement coordination:

- Explorer Agent
- Collector Agent
- Sorter Agent
- GUI Agent
- Coordinator Agent

8.2 Challenges

During the thesis work we have encountered several challenges both practical and technical. A summary of these challenges and how they where solved is presented in this section.

- JACK IDE
  The JACK IDE has several shortcoming compared to other well known IDEs such as Microsoft Visual Studio, Eclipse and InteliJ. The most apparent being the lack of syntax highlighting, syntax error correction/help and code completion. Shortcomings of this kind in general result in slower development as well as unnecessary frustration as we are used to these features in all other IDEs. No other solution to this problem except just accepting the shortcomings and working with them.
- JACK compiler
  The JACK compiler does not support any Java language features above JDK 1.4 which includes java generics, simplified for statements, optional method arguments etc. Using these language features in plain Java files and compiling these files separately with javac solved this.
- LEGO Mindstorms
  Generally robotics is a field with many challenges and with Mindstorms being a simple programmable robotics kit the weaknesses are more severe and not easily handled. The weaknesses we have encountered include non-accurate sensor readings, limited computational power and poor communication support. These limitations resulted in excessive time usage and thus we where required to give this part of the development less priority especially because this was not the main focus of the thesis. In addition to less priority we where forced to adjust the desired complexity in our implementation goal. Instead of having an unstructured environment as intended a structured grid solution was adopted and implemented.

8.3 Hypotheses

1. Intelligent agents are a suitable platform for modeling and development of interacting robots

We have found that intelligent agents are a very good software solution for modeling and development of interacting robots, which also is one of its main applications of agents. Especially the JACK intelligent agent framework, where the support for external robot interfacing is both well documented and supported. As shown in Figure 41, the actual communication is done between the agents and not the robots them selves, they only communicate with their respective agent. This ensures that we can utilize all features that agents provides, such as event driven communication and event handling, which makes applications both robust and reliable. Even though we had no prior experience with the approach, agent development was intuitive and efficient when we got the hang of the concepts involved.

Based on our results we believe that hypothesis 1 is true, but further research and comparison with other systems should be done to verify that intelligent agents are a better approach than other systems.

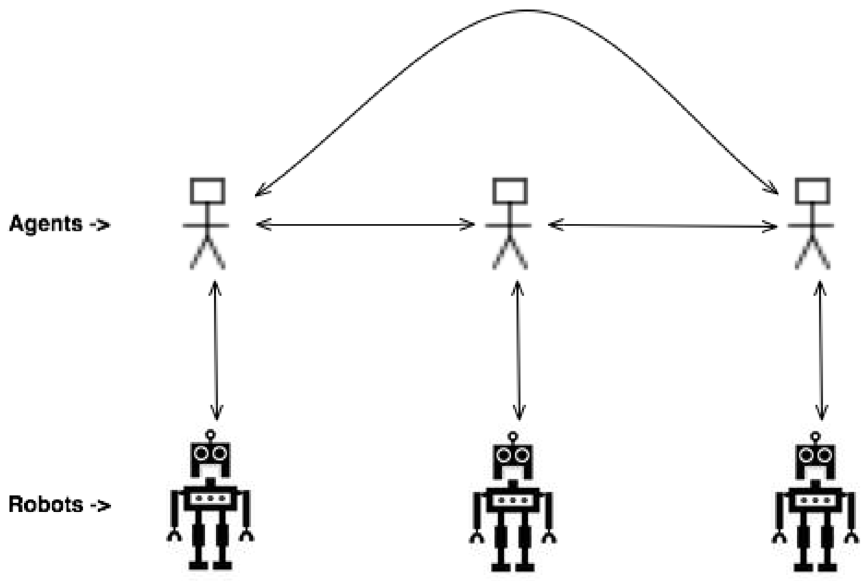


Figure 41: Robot communication done through use of agents

2. In a multi - agent systems, robot interaction can be modeled as interacting agents.

Our design supports that robot interaction can be modeled as interacting agents. As mentioned in Section 6.3, we did add two extra agents to the design, one to interact with the GUI and one to manage coordination between the robots. This does however not interfere with the hypothesis. Having a one to one robot/agent relation gives a very intuitive way of implementing functionality for both interpreting percepts and executing actions. It also gives a straightforward design that can be intrepid without extended knowledge of intelligent agents.

9 Conclusion

The project aim was to investigate if multi agent systems can help us to improve robot coordination and coordination. In order to achieve this, we implemented a multi agent system designed for controlling a set of Lego Mindstorms robots, Lego specific code for realizing the needed robot functionalities as well as classes for interfacing between the different parts, GUI, agent solution and robots. The solution is based on three Lego robots operation on a line-based grid. One robot is set to explore the grid, finding object, and sharing this information (beliefs) with a second robot that is responsible for collecting and delivering these objects to a robot that sorts these object according to color.

Two issues of research where formulated; "How can agent technology help us improve robot coordination and communication problems?"and "Given an more harsh and unstructured environment, how would our developed solution scale?"

Although the operational environment of the robots was simplified from unstructured to a structured environment and implemented as more of a simulator rather than actual robots working, the solution still leaves room for investigation of the research issues. A team of robots is given a common goal where they all need to perform different roles to achieve the desired results. The robots must cooperate and coordinate amongst themselves while constantly updating and reporting results to an operator. Despite the structured nature of the environment and the high level of autonomy implemented to communication between the robots is an important aspect.

The agent system is capable of controlling the robots and running the scenario for any given grid map using our simulated environment. The agents act according to sensor data and information shared between the agents with some additional operator input. Our test runs show that the agents are able to handle all the defined scenarios regardless of map layout and report accurate results through the user interface. Given the good performance achieved for our specified scenarios, it is important to point out the structured nature of the operational environment as being an important factor. This being the case our experience with the use of software agents to realize operator - multi robot machine systems has been very positive and we believe it to be a good approach.

10 Further Work

The problem definition we started out with turned out to be to excessive and complex due to time limitations. We ended up making several simplifications to the initial implementation goal. The main simplification was degrading from an unstructured and dynamic environment to a structured simulated one. Even though alot of work was put into the physical robots we also had to abandon this part of the project unfinished allowing us to focus on the more important aspects of the thesis.

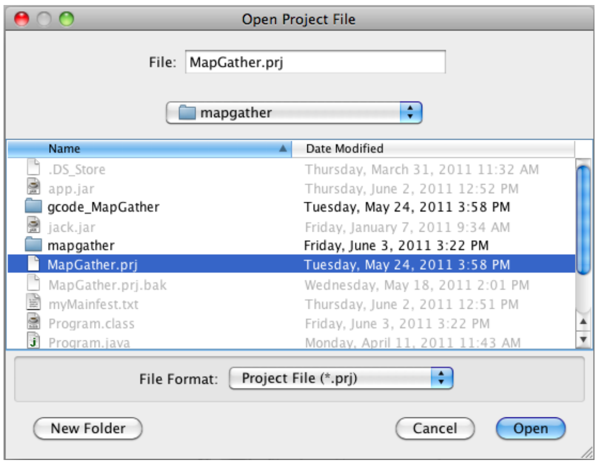
Further work on this project will be to finalize the actual robot implementation and have the physical robots working together with the agent solution as initially intended. This would require refinement of algorithms partly implemented for the different types of sensor input analysis and navigation. The next challenge is to change the operational environment and have the robots function without structured and predictable surroundings. Applying these environmental changes would lead to a greater need for operator involvement in context of critical situations. This involvement will include both input and decision making enabling us to utilize the agents capabilities even better. For example by having the agents provide the operator with a set of suggested solutions for a problem at hand relieving the operators workload.

We find the topic of agent systems very interesting and we would like to spend more time investigating it further both practical and theoretical. Although the agent community is rather small it will be exiting to follow future development within this field.

A JACK installation guide

The JACK framework is available for trial download at the aos group homepage [24]. After downloading the trial version install with the corresponding key you will receive by email after registration.

After installing JACK the thesis project can be opened through the standard file ⇒ open project menu as shown in Figure 42.



File: MapGather.prj

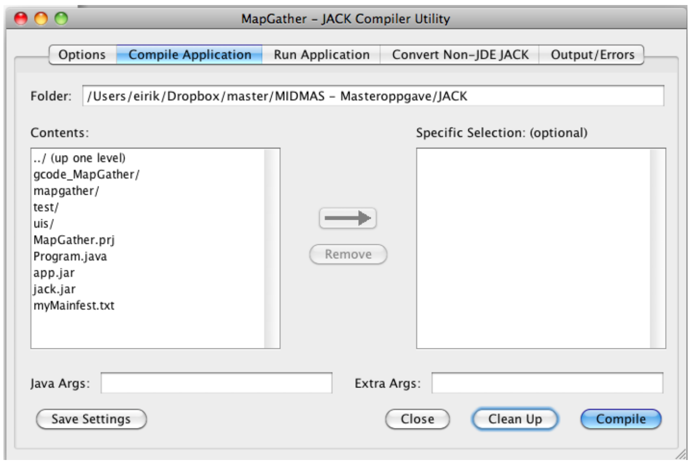
mapgather

Name	Date Modified
.DS_Store	Thursday, March 31, 2011 11:32 AM
app.jar	Thursday, June 2, 2011 12:52 PM
gcode_MapGather	Tuesday, May 24, 2011 3:58 PM
jack.jar	Friday, January 7, 2011 9:34 AM
mapgather	Friday, June 3, 2011 3:22 PM
MapGather.prj	Tuesday, May 24, 2011 3:58 PM
MapGather.prj.bak	Wednesday, May 18, 2011 2:01 PM
myMainfest.txt	Thursday, June 2, 2011 12:51 PM
Program.class	Friday, June 3, 2011 3:22 PM
Program.java	Monday, April 11, 2011 11:43 AM

File Format: Project File (*.prj)

Figure 42: Open project in JACK

To compile the program press Tools ⇒ Compiler utility from the menu bar. The compiler window is shown in Figure 43.



Folder: /Users/eirik/Dropbox/master/MIDMAS – Masteroppgave/JACK

Contents:
../ (up one level)
gcode_MapGather/
mapgather/
test/
uis/
MapGather.prj
Program.java
app.jar
jack.jar
myMainfest.txt

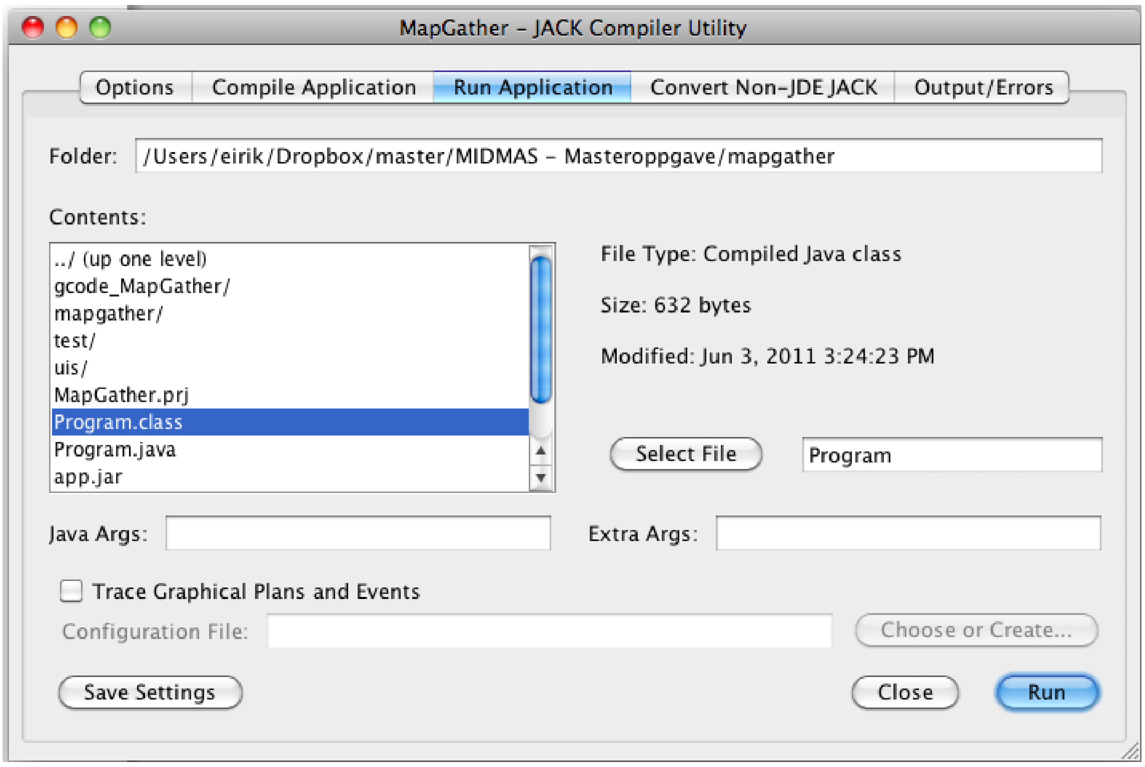
Specific Selection: (optional)

Java Args:

Extra Args:

Figure 43: Compile project in JACK

When the program is compiled successfully it can be run from the "Run Application" tab in the Compiler Utility window as shown in Figure 44.



Folder: /Users/eirik/Dropbox/master/MIDMAS – Masteroppgave/mapgather

Contents:

```
../ (up one level)
gcode_MapGather/
mapgather/
test/
uis/
MapGather.prj
Program.class
Program.java
app.jar
```

File Type: Compiled Java class

Size: 632 bytes

Modified: Jun 3, 2011 3:24:23 PM

Program

Java Args:

Extra Args:

☐ Trace Graphical Plans and Events

Configuration File:

Figure 44: Run compiled project in JACK

For more details see the JACK Development Environment Manual [25].

B User Guide

The system requirements for running the application are Java 1.5 or newer. There is no need to install the JACK framework, as the jack.jar is included on the cd. Remember to have the jack.jar file in the same folder as mapgather.jar for the application to work.

The jar file(mapgather.jar) for running the program is located on the attached CD. After starting the program the connections to the robots need to be established, this is done by pressing the "Initialize Connections" button in the top left corner of the GUI. With the connections up press the "Start" button in the bottom left corner to run the collection scenario. The buttons are shown in Figure 45

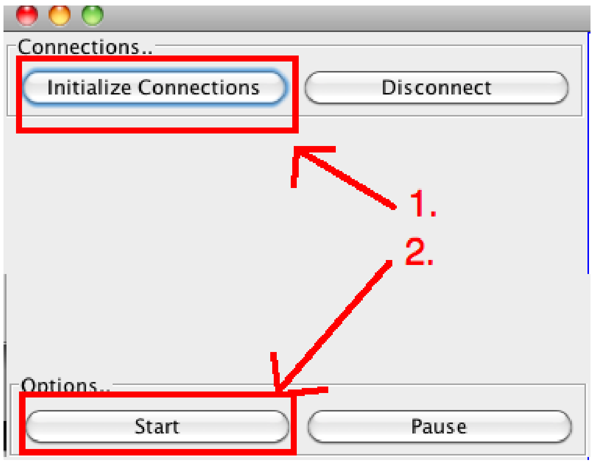


Figure 45: Buttons for running the program

References

[1] Elin Marie Kristensen. Agent technology. Master's thesis, Norwegian University of Science and Technology, 2005.

[2] Statoil ASA. Statoil in brief. `http://www.statoil.com/en/About/InBrief/Pages/default.aspx`.

[3] Hege J. Tunstad. Munin, the autonomous submarine. `http://www.forskning.no/artikler/2008/mars/1205410155.15`.

[4] Einar Landre. Autonomous systems & technologies, research and competence strategy, 2010.

[5] AOS Group. About aos group. `http://aosgrp.com/`.

[6] Lego. Lego mindstorms. `http://mindstorms.lego.com/en-us/Overview/NXTreme.aspx`.

[7] Raymond S.T. Lee. *Fuzzy-Neuro Approach to Agent Applications*. Springer, `http://www.springeronline.com`, 1st, edition, 2006.

[8] Lin Padgham and Michael Winikoff. *Developing intelligent agent systems - a practical guide*. WILEY, `http://www.wileyeurope.com`, 1st, edition, 2005.

[9] Michael E. Bratman. *Intention, Plans, and Practical Reason*. CSLI Publications, `http://http://csli-publications.stanford.edu/`, 1st, edition, 1999.

[10] M. Wooldridge and N. Jennings. Intelligent agents: theory and practice. *The Knowledge Engineering Review*, 1995.

[11] J. Postel. User datagram protocol. `http://tools.ietf.org/html/rfc768`.

[12] Lise Engmo and Lene Hallen. Software agents applied in oil production. Master's thesis, Norwegian University of Science and Technology, 2007.

[13] Oracle. What is java? `http://java.com/en/download/whatis_java.jsp`.

[14] Jetbrains. About intellij idea. `http://www.jetbrains.com/idea/`.

[15] LeJOS. Lejos - java for lego mindstorms introduction. `http://lejos.sourceforge.net/nxt/nxj/tutorial/Preliminaries/Intro.htm`.

[16] LabVIEW. About labview. `http://www.ni.com/labview/`.

[17] Dieter Fox, Wolfram Burgard, Frank Dellaert, and Sebastian Thrun. Monte carlo localization: Efficient position estimation for mobile robots. *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, 1999.

[18] Sebastian Thrun, Dieter Fox, Wolfram Burgard, and Frank Dellaert. Robust monte carlo localization for mobile robots. *Artificial Intelligence*, 2001.

[19] IBM. Uml basics: The sequence diagram. http://www.ibm.com/developerworks/rational/library/3101.html.

[20] K.H Ang, G.C.Y. Chong, and Y Li. Pid control system analysis, design, and technology. *IEEE Transactions on Control Systems Technology 13*, 2005.

[21] Alexander Chaffee and William Pietri. Unit testing with mock objects. http://www.ibm.com/developerworks/library/j-mocktest/index.html.

[22] Ro-botica.com. Tribot building instructions. http://ro-botica.com/img/NXT/Build-Tribot.pdf.

[23] Active-Robots.com. Robotarm building instructions. http://www.active-robots.com/products/mindstorms4schools/building-instructions/Build-RoboArm.pdf.

[24] AOS Group. Jack dowload site. http://aosgrp.com/products/jack/index.html.

[25] AOS Group. Jack development environment manual. http://www.aosgrp.com/documentation/jack/JDE_Manual_WEB/index.html.