GOAL DELIBERATION AND PLANNING IN COOPERATIVE MULTI-ROBOT SYSTEMS

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Yongho Kim

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

December 2018

Purdue University

West Lafayette, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF DISSERTATION APPROVAL

Dedicated to my wife, Seongha Park,

who guides me always to the right way.

ACKNOWLEDGMENTS

First and foremost, I would like to thank Dr. Eric T. Matson for all his advise and supports. The way how he sees the world inspired me to grow my own insight about research studies and things in life. Without his advise and both financial and mental supports, I would not be able to finish this dissertation. I am positively impressed by the way of how he treats students, people, fellows. He provides students many opportunities and challenges. For me, I was able to take many benefits from those opportunities and challenges he gave me. He is the best academic advisor that I wish to follow.

Next, I would like to thank my committee members. Dr. Eric J. Dietz, Dr. Julia T. Rayz and Dr. Jin-Woo Jung are always willing to share their thoughts on issues I have, and teach me to think wisely. Their academic advises on the topic of this research study helped me a lot to design and analyze the study. I especially thank Dr. Jin-Woo Jung for not only being my committee member, but also his warm-hands on supporting me while I visited his lab in South Korea in 2015.

I express my sincerely gratitude to Dr. Donghan Kim, Dr. Jongho Seon, and Dr. Jaesoo Kim. They showed me the joy in teaching and researching in academia. I thank Dr. Byung-Cheol Min for his support and advise. His enthusiasm on researching and learning always motivates me. I thank our department staffs, Stacy Lane and Cynthia Salazar, for their immense supports.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

ABM	Agent-Based Modeling
AI	Artificial Intelligence
API	Application Programming Interface
AOP	Agent-Oriented Programming
BDI	Belief-Desire-Intention
CNF	Conjunctive Normal Form
CPS	Cyber-Physical Systems
CTL	Computation Tree Logic
DAI	Distributed Artificial Intelligence
FOW	Fog of War
GG	Good Game
GHz	Giga-Hertz
IoT	Internet of Things
IP	Internet Protocol
ISM	Industry-Science-Medical
KB	Knowledge Base
LTL	Linear Temporal Logic
MAS	Multi-Agent Systems
ML	Machine Learning
OOP	Object-Oriented Programming
OR	Operation Research
OS	Operating System
ROS	Robot Operating System
SCV	Space Construction Vehicle

SC2	StarCraft II
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

ABSTRACT

Kim, Yongho Ph.D., Purdue University, December 2018. Goal Deliberation and Planning in Cooperative Multi-Robot Systems. Major Professor: Eric T. Matson.

Intelligent robots are rational agents. The rationality of robots working cooperatively is significantly different from robots working independently. Cooperation between intelligent robots requires the high level of reasoning and complex interactions for successful operations. The required reasoning process includes knowledge representation and sharing as well as the ability to understand the context of a situation. The reasoning process heavily influences on the planning of deciding what actions need to be taken. Goal deliberation and planning is the process that deals with those requirements. This dissertation investigates the problem of goal deliberation and planning to enable such cooperation between goal-oriented intelligent robots, working as a team. The dissertation then proposes a multi-robot system model that embraces results of the investigation. The proposed model is realized on the top of the platform 'robot operating system' (ROS). The implemented system, named 'goal-oriented multi agent systems' (GOMAS), is demonstrated with the computer game, StarCraft II. Units in StarCraft II are individually controlled by the GOMAS robots and work cooperatively to attain a set of goals given from operators. The demonstration with the three different scenarios validates that the GOMAS system successfully and efficiently deliberates and plans the given goals.

CHAPTER 1. INTRODUCTION

This chapter introduces the area of research and its environment to help dive into the topic of this research study. The area mainly falls into the field of distributed artificial intelligent (DAI). DAI focuses mainly on the intelligence of individual agents and its potentials for situations where robots coexist in the same environment and possibly work cooperatively. And then, the research motivation and problem are described in detail. Later, assumptions of this research study are listed and discussed.

1.1 Automation and Reasoning

Automation in service robots is defined differently from factory robots. Factory robots generally deal with things from a static environment using a single capability (a couple of capabilities at most) whereas service robots serve multiple tasks using multiple capabilities in a dynamic environment. Automation of the factory robots is simply defined as reactive agents, meaning that the robots do not think, but act whenever they perceive a certain level of stimulus. Multiple factory robots can easily be controlled by one that plans every task that the robots need to perform. In contrast, service robots are automated in a way that they perceive information from the environment, think what actions are available according to the perceived information, and select and perform an action while perceiving the environment to avoid any failure on the action. Those service robots are classified as behavioural or proactive agent. Reasoning requires many facts and relationships of the facts to conclude another fact that may activate another conclusion. For such service robots, the reasoning process may be the key to drive them to appropriately serve complex goals.

1.2 Multiple Agents and Their Environment

Automated computational agents have performed excellently in the fields that require precise, accurate, and simple operations, such as in a factory. In particular, autonomous robots in factories have outperformed tasks that any human can do, in terms of their productivity. Moreover, those factories have made the robots parallel and cooperative to increase the productivity even a way further. In practice, it is generally believed that running multiple copies of single-capable robot is more time and resource efficient than a single multi-capable robot as their task generally requires physical presence to work.

Operating multiple robots, however, may become significantly difficult when robots perform their action in a shared environment. As robots work closer they may face more uncertainties of information and unexpected events from their perception. In this environment those factors are usually perceived not because their sensors did not work properly, but because the actions that other robots committed to do and consequences of the actions unintentionally conflicted with an action from a robot. Often, neighbor robots disrupt one robot's perception (e.g., a robot does not see an object hindered from other robots). This implies that intelligent robots are required to be able to have an additional ability (e.g., make an organization to work together, adjust the system model to continue working), to deal with the uncertainties while continuing their task. Such uncertainties may not be crucial – the uncertainties might be trivial to robots and do not interrupt the robots while attaining a given goal. For example, an autonomous ground vehicle is driving on a road. The vehicle could possibly encounter unexpected obstacles on the road while driving. If the vehicle has an obstacle avoidance capability, it could overcome the uncertainty and drive to its destination to complete the given task. The vehicle could even keep driving over the obstacle even if it does not have such capability. However, if one of the tires is punctured by a sharp object and becomes flat or other vehicle runs into the vehicle while driving, the vehicle could not be able to arrive to the destination and thus the goal can not be attained. Because this change is critical for reliability of the robot (or the system), designers should be able to address those changes before deploying their robots.

In multi-robot systems, the difficulty mentioned above becomes more intensified because of complexity of the system itself, and because of a higher level of uncertainty brought by the complexity. Although multi-robot systems and behavior rules for robots in the systems are modeled correctly and triggered properly to serve a given goal, any changes that the other robots would make through the environment can interfere in events that a robot uses as a trigger to execute one of pre-defined actions. This falsified event may lead to a starting condition of another action which should not be executed, and may eventually result in failure of the mission. However, it is almost impossible to predict and prepare unexpected changes while designing the system. Instead, each robot in multi-robot systems has to have the ability to make them adaptive to the environment, and change its course of actions when necessary.

Multi-agent systems allow their robots to eliminate some of the aforementioned uncertainties, particularly coming from others. Communication in an organization can be used to inform actions and intentions of committing to a course of actions to one or more robots in the system. In fact, knowing others' actions helps plan the next action with regard to the others' (possible) actions.

1.3 Research Motivations

The internet of things (IoT) has accelerated spreading the concept of using multiple entities that are interconnected to transfer a stream of data. IoT applications have started a variety of services that improves the quality of human life. As manufacturing hardware components (e.g., sensors and actuators) is getting cheaper and the components are more accessible to service providers, the trend of spreading IoT services to the world has been accelerated even more. This resulted in deploying so many network-capable CPS that serve one goal, e.g. a service of sensing environments for environmental research studies. Various types of multi-agent based models have been proposed to manage IoT devices (Ayala, Amor, Fuentes, & Troya, 2015), Smart Grid (Merabet et al., 2014), and so on.

Operators of multi-robot systems have been willing to have their intelligent robots perform autonomously as a team in order for the robots to attain a given complex goal with little or zero control. Controlling them from an operator is even impossible for cases where a goal requires complex sequence of actions that are dependent on context and environmental conditions, and for other cases where the number of robots is too many (e.g., tens or hundreds) to control in assigning them to a specific task. Scaling up systems even further to tens of hundreds or thousands escalates the willingness and turns it into a must-have capability. Formation control is one of the research fields in multi-agent systems that utilizes the capability in controlling from a few tens to hundreds of robots with minimal intervention (Antonelli, Arrichiello, Caccavale, & Marino, 2014). Ultimately, operators do not provide a sequence of actions, but just toss a goal and have the robots figure out how to attain the goal.

Intelligent multi-agent systems have been utilized to solve complex problems. According to Durfee, Lesser, and Corkill (1987), the complex problems in multi-agent systems domain tend to require multiple activities from agents in a coherent way. This means that those activities are interrelated in that fitting the activities draws the overall picture of the problem. And thus, agents may need to hold any action until present activities are finished, and should avoid duplication of activities when unnecessary. Moreover, scope of the perception of an agent and its view in multi-agent systems is limited and information is perceived locally. Due to these circumstances, it is desirable that agents are able to share the knowledge of information and their status, including their intention toward solving the problem.

Computer games, e.g. ATARI games and StarCraft from Blizzard Entertainment, have been used as a test-bed for intelligent agents and machine learning techniques (Guo, Singh, Lee, Lewis, & Wang, 2014; Mnih et al., 2013; Ontanón et al., 2013). At the same time, such tested and validated AI techniques have been applied back to those games to improve the quality of non-player units in the game. This increases the difficulty of the game play so much in that human players do feel that the player plays with another human player. One fascinating example of it is the match between the non-human player, AlphaGo from Google, and the human player, Sedol Lee, in Go (Chouard, 2016). There is no reason why utilizing such computer games cannot be used in simulating multi-agent systems, especially simulating multi-robot systems. It is hoped that autonomy in multi-robot systems and emergence may arise from studies using those computer games and those empower operators to control groups of multiple robots to deal with complex goals.

1.4 Research Scope and Aims

Kraus (1997) well described the target situation of this research study as,

> "... all the agents work together toward the satisfaction of a joint goal; the designer of the automated agents can develop, in advance, protocols for cooperation between the agents; the number of agents is not large; and the agents can communicate and have computation capabilities."
> (Kraus, 1997, p. 88)

To deal with the situations, Kraus (1997) recommended employing *operation research techniques*. Operation research (OR) is the domain that deals with problems of how to model and operate organizations of agents when available resources are limited and sparse (Hira, 2007; Winston & Goldberg, 2004).

This research study aims to address two research topics from the situation described above. The first topic is to investigate and understand process of deliberation and goal planning in a cooperative multi-robot system. Deliberation allows robots to reason about what they know about the goal and how they can attain it. To understand the process of deliberation toward a complex goal, all details about the reasoning process for deliberation need to be addressed. Interactions to share information between robots during the reasoning process are the key that enables the robots to track progress of sub-goals. Goal planning addresses how robots plan a series of actions with regard to current circumstances as well as other's intention and ongoing actions. A robot's capability may restrict what they can do for the goal. Defining types of information for the goal planning and generalizing the types are also the key components that make agents intelligent. Before and after robots are committed to perform actions, they need to again update their knowledge set to avoid any conflicts from their actions and intentions with others.

The second topic is to propose a system that embodies all the features described above, and to prove that the computer games, aforementioned, can be used to run a simulated environment for the proposed multi-robot system. Agents running outside of the simulated environment control each unit rendered in the simulation environment. The agents perceive information and act through their unit. Inter-communications between the agents happen from outside of the simulation, but interactions with the environment occur within the simulation environment. Through the simulation, it is hoped to see how intelligent robots deliberate goals and plan accordingly while tracking how other cooperative robots have attained other goals.

1.5 Research Assumptions

Because this research study aims at capturing phenomena from high-level intelligence, several concepts need to be defined in order to focus on the research problem, deliberation and goal planning.

- **Social Agents** According to the definition of rationality (Binmore, Castelfranchi, Doran, & Wooldridge, 1998), rational agents act to maximize benefits that they earn as a result of their action. In multi-agent systems environment the total amount of resources and benefits are limited. Therefore, rational agents in such environment may conflict on their actions and may compete with each other. System designers and operators in multi-agent systems however do not wish their agents compete, but rather cooperate unless desired. Because this research study investigates how rational (i.e., intelligent) agents cooperate to attain a given goal, the agents are socially rational, which makes them act to maximize social benefits, not individual benefits.

- **Single-minded Agents** Intelligent agents sometimes need to reconsider the plan that they are committed to do, whenever the plan or the intention becomes inapplicable. Thorne (2005) described three main types of agent based on their commitment strategy: *blind-minded*, *single-minded*, and *open-minded*. Blind-minded agents never reconsider or revise the plan it is committed to do, whereas both single-minded and open-minded agents do revise the plan whenever their perception tells them that the plan or goal is not applicable to pursue. Open-minded agents are capable of modifying a plan as well as creating a new one when necessary. This research study considers single-minded agents such that agents may change their plan, but are not able to generate plans that do not exist.

- **Pre-defined Goal Decomposition** Decomposing a goal is a process that breaks down a high-level goal (i.e., an abstracted goal) into sub-goals (i.e., more specific goals). This process needs to be done before agents take the goal because decomposed sub-goals look more specific and understandable to agents in that they are able to manage the sub-goals to attain the high-level goal. Goal decomposition is another area of study and is out of scope of this research study (Alford, Shivashankar, Roberts, Frank, & Aha, 2016; DeLoach & Miller, 2010). Therefore, this research study does not tackle how decomposition of a high-level goal is proceeded. Instead, it is assumed that a high-level goal is well decomposed into sub-goals using pre-defined rules that all agents know. This means that a goal is always decomposed into the same set of sub-goals.

- **Stable Communication** Communication protocol is assumed to be defined well enough to support interactions between agents. Exchanging messages happens instantly without any delay between neighbors to eliminate temporal constraints discussed in (Calvaresi, Marinoni, Sturm, Schumacher, & Buttazzo, 2017). Since most of intelligent agents are capable of using network communication through wireless and Bluetooth, communication barrier applies only when agents are too far to communicate; presumably more than distance of 100 m in open space could cause this barrier.

- **Semi-Closed System** The proposed model is a semi-closed system where agents can be joined to the system, but none of the member agents allow to retire or withdraw themselves from the system.

- **Honesty in System** Because this research study investigates cooperation and coordination in intelligent systems, agents in the proposed system are responsible ones.

1.6 Structure of The Thesis

Chapter 2 reviews relevant literature on DAI and intelligent agent for goal deliberation and planning. Relevant sub-fields are also addressed to understand the research problem in detail. In addition, the chapter discusses existing simulation tools for multi-agent systems. Chapter 3 describes the proposed system model to deal with the problem. Chapter 4 then illustrates a simulation tool that validates the proposed system model. Chapter 5 discusses conclusions of the research study with limitations and future works.

CHAPTER 2. REVIEW OF RELEVANT LITERATURE

This chapter first introduces intelligent systems and Distributed Artificial Intelligence (DAI), a principle of artificial intelligence in multi-agent systems, to discuss characteristics of multi agent systems in dynamic environment. Next, different aspects of multi-agent systems (MAS) models are described to clarify target system of this research study. Later on, in-depth exploration of practical reasoning techniques that enable agents to express and exchange their knowledge for goal planning are presented. Lastly, existing simulation tools for multi-agent systems and their features are discussed.

2.1 Distributed Artificial Intelligence

Centralized multi-agent systems are usually efficient in a static environment. When agents are working in the same environment, coordination of their tasks can greatly improve performance by bringing the maximum utility. Moreover, the coordination can be easily modeled and analyzed because it considers only a small number of variables and narrower variation of these variables. However, centralized systems have huge drawbacks such as single point of failure, centralized computation that makes the system slow and complex, and consistent and stable connection between peers and the central unit. As the number of agents increases these drawbacks significantly impact to reliability of the system.

M. J. Wooldridge (1992) described the key features of DAI systems for reasoning about multi-agent systems as follows,

> “agents have a set of explicitly represented beliefs, which are formulae of some internal, logical language of belief; agents are able to derive some though not necessarily all of the logical consequences of their beliefs;

> agents have computational resources upon which they may draw, by performing private, internal, cognitive actions (for example, a database agent might perform a retrieve action); agents are able to affect the beliefs of other agents by communicating with them through message passing" (M. J. Wooldridge, 1992, p. 5)

DAI has received attention from researchers because DAI well describes characteristics of environments in the world and models intelligent agents working in a shared environment (Weiss, 1999). DAI focuses more on groups and interactions than individuals and actions. Interactions allow individuals to deliver knowledge of beliefs, desires, and plans. Then, the individuals are able to reason socially, in order to take an action towards attaining a complex goal, which requires concurrency of multiple actions.

Various communication protocols provide ways of such interactions in that agents in a DAI system are able to packetize information into a form of message, send the message over a network where others are involved in, and understand information sent from the others.

Communication protocols do not necessarily use explicit forms, but rather utilize an implicit way of expressing information in some situations where information is simple and easily explainable. These types of communication are usually happening throughout artifacts that nearby agents are aware of. In nature, ants are a perfect example of the implicit communication in that ants use pheromone in their trail to notice a way to the pray. In engineering, the ants model inspired from the nature has been utilized to solve complex problems such as Travel Salesman Problem (TSP) (Dorigo & Gambardella, 1997), assembly sequence planning in factory (Wang, Liu, & Zhong, 2005), and data aggregation problem (Liao, Kao, & Fan, 2008).

2.1.1 Intelligent Agents and Robots

Cyber Physical Systems (CPS) are computational systems that have abilities to intelligently interact with the cyber, e.g., Internet, and the physical world, the world we live in. The use of CPS has shifted from industries to human-friendly environments such as home, office, school, and public places, in order to utilize those abilities for services to humans. Autonomous robots, one kind of such CPS, are a physical system designed to closely interact with the surrounding environment. Software agents are one other kind that generally exists in the cyber world and provides services (e.g., booking a flight, broadcasting an announcement to employees) that typically do not require physical actions.

Not all kinds of CPS are necessarily intelligent. A electrical thermometer, for example, interacts with an environment by measuring temperature of the area, but it does not consider why and how it measured. A washing machine may pour water inside the drum to wash, based on the weight of the clothes in the drum. This reactive behaviour may be called *intelligent* in a broader context. A certain level of intelligence can enable CPS to reason about the environment and allow them to decide what to do (i.e., decision making). According to the term 'autonomy' (Castelfranchi, 1995; M. Wooldridge & Jennings, 1995), the self-reasoning requires the following contexts,

- There should be alternatives to attain a given goal
- Agents have an utility function that describes expected outcomes from the alternatives in a numeric value
- Agents have a form of knowledge representation that contains intentions and beliefs on what behaviors the agents prefer

Those which satisfy the aforementioned requirements are called *intelligent agent*. The washing machine from the above example can be intelligent if it measures how much dirty the clothes are and adds the proper amount of detergent based on the measurement. It can be even more intelligent if it optimizes the number of running cycles based on the dirty level.

Intelligent robots have three key components: sensors, actuators, and an intelligence. Those three components are closely related to bring an action, as a result of reasoning. Sensors and actuators are usually limited in a way that they interact with the local environment only. As the intelligence relies on both the sensors and actuators, reasoning and decision making processes concern information mainly from the surroundings. Many machine learning techniques and AI applications have focused on this process – how to make the process appropriately react with and without prior experiences upon a given set of inputs. Ultimately, those techniques attempt to make the robots think and act *rationally*.

2.1.2 Rationality in Multi-Agent Systems

The term '*rationality*' can be seen differently from many views as the term is defined subjectively. In economics, the rationality is defined as " ... by attributing to every actor a *utility function*, which assigns to every possible outcome a value. An agent is then rational if it acts so as to maximize its utility." (Binmore et al., 1998, p.309). In cognitive and social sciences the term is defined in a different way as "Agents are not moved to action by the principle of maximum utility, although actions are controlled by this principle. Actions are motivated by needs, desires, etc."(Conte & Castelfranchi, 1995, p.6). This difference implies that the rationality of an intelligent agent cannot be explained from one view. Nonetheless, those definitions of 'rationality' coexist in human society.

Self-rational decision making in multi-agent systems can be dangerous as the agents are then greedy for pursuing their maximum utility only. In general, environments do not provide an ample amount of resources for all robots playing upon. Performing a series of actions that would bring the highest profit to the robots may lead them into the zero-sum game. In the zero-sum game, none of the robots would earn benefit from the actions that they are committed to. Research studies have investigated how robots and agents can be rational in their society to avoid such unwilling situations (Boman, 1999), or to maximize the utility even more (Kalenka & Jennings, 1999).

2.1.3 Social Agents and Cooperation in Multi-Agent Systems

Socially rational agents generally assess social benefits as much as their individual benefits. In order for them to balance those two types of benefits, they cope with social-based decision making as well as self-interested decision making. However, it is still arguable that when and how those rational decision makings need to be adjusted for itself or its society, or even for both. Agents in a competing environment may choose self-rational approach as they concern only themselves. In contrast, socially-rational based approaches would be needed for cooperative multi-agent environments. This in fact depends on what the system designers desire to achieve through agents in the system.

Parker (2008) described and compared the four interaction types used in the field of distributed intelligence: collective, cooperative, collaborative, and coordinative. The four types are distinguished and applied according to the characteristics of the target environment. The characteristics are awareness of others, types of goals, and influences of actions. Collective interactions are used when the given goals are shared among agents and actions from the agents affect the goals of others, but they are not aware of each other. Coordinative interactions can be utilized when agents have individual goals and do recognize others. However, their actions do not influence the goals of others. Collaborative and cooperative interactions seem similar as both require awareness of others and actions of agents affecting the goals of others. The difference between the two interaction types comes from the type of goals – collaborative interactions require individual goals while cooperative interactions are more suitable for shared goals. Such distinction is however not clear because the type of goals can be viewed differently. For example, a given goal can be decomposed into multiple individual goals. Interactions that help achieve the decomposed individual goals can be collaborative. However, the interactions can also be cooperative as the achievements of the individual goals can also help achieve the given goal.

A cooperation among intelligent agents can be achieved by coordinating actions of the agents toward a shared goal. According to results from the coordination process, agents are committed to perform a course of actions to attain each given goal of the shared goal. Since the cooperation has been defined from a variety of research fields, such as controls, multi-agent systems, economics, networks, transportation, etc., many computational multi-agent system models support the cooperation in different ways. Doran et al. (1997) categorized the cooperation based on characteristics of multi-agent systems (See Figure 2.1).

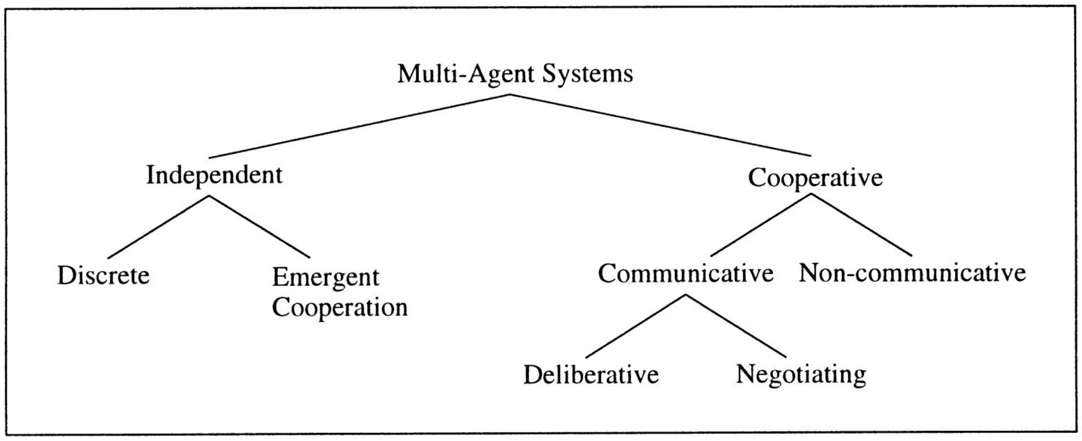

Figure 1 Cooperation Typology

Figure 2.1. Different types of cooperations based on characteristics of multi-agent systems (Doran et al., 1997)

Swarm intelligence and colony-based multi-agent system models, inspired from the nature (Bonabeau, Dorigo, & Theraulaz, 1999; Olfati-Saber, 2006; Parunak, 1997), fall into the areas of the *independent* cooperation. Individuals in an organization perform their actions based only on perceptions from the local environment. They do not attempt to transfer any form of information to others – to them, other agents are the same as artifacts. A formation control of the birds flocking model (Olfati-Saber, 2006), multiple unmanned aerial vehicles (UAV) for formation control (Kuriki & Namerikawa, 2014), and building a structure without a blueprint (Allwright et al., 2014; Werfel, Petersen, & Nagpal, 2011) are applications of the multi-agent systems in this category.

On the other hand, some of the research studies from the swarm or colony-based multi-agent system models can also be categorized in *non-communicative* under *cooperative* because agents in those models do attempt to communicate with each other through nearby objects (i.e., artifacts) in the surrounding environment. Dorigo, Maniezzo, and Colorni (1996) and Karaboga and Basturk (2007) proposed a colony-based cooperative multi-agent system to solve the problem of optimization.

Cooperative multi-agent systems are generally capable of dealing with high-level goals as individuals in the systems handle different types of tasks using their distinctive capabilities. A coordination to assign cooperative actions to agents can be performed according to some criteria. One of the criteria is capability (Deloach, Oyenan, & Matson, 2008). Agents are assigned to a task, which the agent can serve it the best among them (i.e., the agent that brings the highest profit from the task). The holonic model (Fischer, Schillo, & Siekmann, 2003) groups agents in a holon according to the roles of the agents. Every holon has a leader which takes a goal from outside of the holon. The leader then delegates the goal to the agent that its role is associated with the goal. Because holons take actions in parallel, the holonic model is capable of solving problems of which the '*divide-and-conquer*' strategy is applicable. The game theory based approaches in multi-agent systems manipulate the cooperation process using contract net protocol (CNP) (Akbarimajd & Barghi Jond, 2014; Kuwabara, Ishida, & Osato, 1995; Zhao, Wang, Cheng, Yang, & Huang, 2010). In this case, an agent that proposes the most profitable bid to the coordinator (i.e., a group leader or an agent in the group) is selected to take the goal. Those approaches require self-rational agents that pursue obtaining profits from attainments of goals. Chien, Barrett, Estlin, and Rabideau (2000) however showed that socially rational agents are also capable of employing CNP to deal with the task of distributing goals. In the study, the robots bid if the goal can fit in their schedule, otherwise they do not bid. In another case, coordinators may rely on how much the agents trust others when allocating tasks. The trust and reputation model has been proposed to allocate a task to the most trustworthy agent (Ramchurn et al., 2009). In the model, the trustworthiness of individuals can be accumulated based on the mental state toward the individual, or on an aggregation of parts from the utilities that the individual has brought.

The research study is categorized in *deliberative cooperation* as the study designs a situation, in which socially rational agents communicate with each other to deliberate a set of goals. Those agents do not take into account individual benefits. Rather, they count on social benefits such that their basic behavior is to attain goals as much as possible to maximize social benefits, but not to debate which agent takes the goal. This type still requires the high-level of intelligence and an explicit process of a coordination to make such deliberation and delegation possible.

2.1.4 Coordination and Consensus in Multi-Agent Systems

A coordination in DAI-based multi-agent systems is a challenging problem in that agents encounter dependencies and constraints from goals. O'Hare and Jennings (1996) stated that the main reasons of a coordination among agents as follows:

- A course of actions is dependent on other courses of actions. This dependency comes from different levels of a goal structure. For example, when a house is being built rooftop cannot be built prior to the completion of lower level structures and the ground. In addition, it comes from conflicts of the same level goals. Escaping hundreds of agents from a room through a single door in a brute-force way may cause delays and damages as the agents would run into themselves at the door.

- Agents are constrained by shared resources. An agent is pausing its action until a certain amount of resources is gathered, while another agent is spending the resources whenever gathered. The agent is unlikely to take the action unless the other agent stops spending the resources.
- A given goal is too complex that no individual is capable of attaining the goal. This occurs when the individuals do not have sufficient resources, information, or capabilities – most of multi-agent systems, particularly multi-agent systems with DAI, run into this situation.

When a coordination process is taken into account, a success of the process puts the system into a situation where the agents participated in the process have an agreement of a plan or a state, called 'consensus'. According to Olfati-Saber, Fax, and Murray (2007), the consensus is defined as:

> "In networks of agents (or dynamic systems), "consensus" means to reach an agreement regarding a certain quantity of interest that depends on the state of all agents. A "consensus algorithm" (or protocol) is an interaction rule that specifies the information exchange between an agent and all of its neighbors on the network." (Olfati-Saber et al., 2007, p. 215)

Ren, Beard, and Atkins (2005) surveyed consensus protocols used in many sub-areas of control systems. The consensus in control systems is an agreement of the most desirable state and can be found by exploring possible state space. In the control system (Cao, Yu, Ren, & Chen, 2013), a convergence of a state is the mean of input variables. The variables can vary depending on geographical distances of individuals (Mastellone, Stipanović, Graunke, Intlekofer, & Spong, 2008), logical distances in a network topology (Ji & Egerstedt, 2007; Li & Zhang, 2010), or some values measured locally by the individuals (Seyboth, Dimarogonas, & Johansson, 2013). In the study (Papadopoulos, Jenkins, Cipcigan, Grau, & Zabala, 2013), the most efficient strategy for an agreement was found using computational search methods and statistical approaches. Research studies in this research area have been trying to converge some values that are shared by individuals. These approaches, however, do not consider individuals' beliefs toward the value. Instead, they focus on merging values and providing one that the individuals are supposed to accept without any complaints, even when the one neglects some of the individuals. For example, systems in formation control assume that all agents drive toward the same direction while keeping a certain distance between them. When some of the agents in the group sense that they will run into an obstacle, they may try to avoid the obstacle by maneuvering themselves toward a direction perpendicular to the direction of the group. However, nearby agents from the maneuvering agents may not sense the obstacle, but sense the change in distance that the maneuvering agents are making. The nearby agents may then try to adjust the distance by maneuvering themselves toward the agents. As a result, the nearby agents heading to the agents that are avoiding the obstacle may run into the obstacle. The example implies that such deviation can influence consensus in an unwilling way.

The consensus in agent-oriented systems and DAI systems is an agreement of a selection among alternatives by making the alternatives compete each other. The most high level of cooperation falls into this domain. Kraus, Sycara, and Evenchik (1998) proposed a logical model to reach an agreement through the process of argumentation. In the model, a set of agents exchanges information to persuade the others for bringing the intention that they desire to achieve. Because the agents in the model are self-rational, they evaluate (future) rewards and costs when they negotiate. On the other hand, Liu et al. (2014) proposed a decentralized multi-agent system model consisting of socially rational agents that pursue a common goal to deal with cooperative power distribution in a power grid. Because the agents in the system are socially rational, they try to share their beliefs and intentions to estimate the amount of power that they can generate and compare the amount with the demand. The agents may adjust their intention based on how much power they need to generate to meet the demand.

2.2 Belif-Desire-Intention Model in Multi-Agent Systems

Belief-Desire-Intention model, introduced by Rao and Georgeff (1995), is a core architecture for intelligent agents behaving proactively. The model has been recognized a promising modeling approach as it bases on the theory of practical reasoning (Bratman, 1987). The BDI model illustrates agent's mental states and enables reasoning to perform a course of actions triggered by its intention. BDI agents perceive states of interests from the surrounding environment as well as from other agents through a form of communication. The perceived states are then stored in their belief set, i.e. a knowledge base. The belief set is also capable of generating a new knowledge from existing knowledge statements using the process of inference. More details about inference can be found in the books (Pearl, 2014; Russell & Norvig, 2016). Desires illustrate a state of affairs that agents desire to achieve. Desires are usually generated by events perceived from an environment, as well as internals of an agent (e.g., a perceived statement *not enough battery* may trigger the desire *charge the battery*). Intentions are a desire that is committed by a set of knowledge and information from beliefs.

Since BDI-based systems in DAI cover a broad area of the problem domain, the problem of determining which alternative (i.e., desire) is the best depends on a situation and what the system pursues. Weiss (1999) explained several criteria that help evaluate the alternatives:

- Social Welfare: alternatives are compared based on what these bring (i.e., utility), and thus the alternative that brings the most utility is selected

- Pareto Efficiency: an alternative that is Pareto efficient is selected; a solution is Pareto optimal if there is no solution that brings better to any agent and there is no solution that brings worse to any agent
- Individual Rationality: an alternative is selected when the selection is individually rational for all the agents
- Stability: all alternatives are dependent on others; each agent selects an alternative if it is the best response on the other agents' selections — Nash equilibrium. The research study conducted by Bowling, Jensen, and Veloso (2005) falls into this category
- Computational Efficiency: the alternative that requires the lowest computational overhead is selected
- Distribution and Communication Efficiency: an alternative that makes the agents distributed is selected. Also, an alternative that minimizes the cost of communication is selected. According to Weiss (1999), the two criteria may conflict in some cases

BDI-based agents are goal-oriented in a way that they look for events generated from both inside and outside of them, and those events may trigger generating a goal (i.e., a desire) to attain. Desires are therefore a precondition for them to act. In order to generate goals appropriately, statements in the belief set need to be updated in real-time, even while agents are in their action.

2.2.1 Knowledge Representation

Knowledge representation is defined as, "*... is the application of logic and ontology to the task of constructing computable models for some domain*" (Sowa, 2000, p. xii). Knowledge statements provide clues for a following statement. Logic and set functions are commonly used to represent knowledge in computational agents. *NOT*, *AND*, *OR*, *IMPLY*, and *IF_AND_ONLY_IF* are connectives expressed as $\neg$, $\wedge$, $\vee$, $\Rightarrow$, and $\iff$, respectively. Using those connectives makes a relationship between logics and constructs complex propositions such as,

- $\neg$P: negation of proposition P
- (P $\wedge$ Q) $\vee$ R: P and Q, or R
- P $\Rightarrow$ Q: P implies Q (i.e., if P, then Q)
- P $\iff$ Q: if P, then Q and if Q, then P (i.e., biconditional)
- (P $\wedge$ Q) $\Rightarrow$ R: if P and Q, then R

For example, a knowledge statement '*FooHasAnApple*' represents a state where the agent *Foo* has an apple. Logics are strictly explicit. Negation of the statement '$\neg$*FooHasAnApple*' does not necessarily mean a state that the agent *Foo* does not have an apple. It is reasonable to interpret '$\neg$*FooHasAnApple*' as 'it is not believed that the agent *Foo* has an apple'.

Propositional logic is useful to express knowledge statements, but is less flexible than predicate logic (i.e., first-order logic). When representing '*ten Foo agents each have an apple*', propositional logic would represent it as,

'*FooOneHasAnApple* $\wedge$ *FooTwoHasAnApple* $\wedge$ ... $\wedge$ *FooTenHasAnApple*'

Predicate logic however would represent the statement using propositional variables as,

'*Has(FooOne, Apple)* $\wedge$ *Has(FooTwo, Apple)* $\wedge$ ... $\wedge$ *Has(FooTen, Apple)*'

Because predicate logic uses the propositional symbol '*Has*', it is much easier to make relationships between the variables (i.e., FooOne, FooTwo, ... FooTen), and thus more expressive. For example, other variables can easily be expressed using predicate logic: '*Has(FooOne, Banana)*'

In the BDI model, the belief set is a knowledge base that stores knowledge statements (i.e., beliefs) that express a state of affairs of an object or a value of a property. '*the robot$_r$ is working*' and '*the resource currently obtained is 100 units*' are examples that can exist in the belief set. Representation of a belief can be formed in a variety of ways including descriptive and logical (Brachman, Levesque, & Reiter, 1992; Szolovits, Hawkinson, & Martin, 1977; M. J. Wooldridge, 1992). Maleković and Čubrilo (1999) proposed the framework for a knowledge base in multi-agent systems that provides a set of functions. With the provided functions, agents are allowed to tell the knowledge base statements to store and to query a stored knowledge statement from the knowledge base.

In order to utilize beliefs in a knowledge base for reasoning, agents need to understand the meaning of beliefs and the relationships of them. Rao (1996) proposed a language, called 'AgentSpeak', that expresses events and actions using first-order logic. AgentSpeak defines belief literals as well as goals. Beliefs and goals are related with symbols such as '+', '-', '!', and '?', in order for agents to reason about the goals. For example, adding a state depicting that the agent a takes the task t using the resource r can be logically expressed as +possess(a, t, r). After the agent a finishes possessing the task t using the resource r, the belief can be dropped by adding -possess(a, t, r) in the knowledge base.

Beliefs in a knowledge base also need to be represented in a form to be transferred to knowledge bases of other agents. Knowledge interchange format (KIF) is a language that manipulates information in a format (Genesereth, Fikes, et al., 1992). With using KIF, the intended meaning of information is logically expressed and can be reproducible by another agent. Laclavık, Balogh, Babık, and Hluchỳ (2006) proposed a knowledge model, called 'AgentOWL', that integrates the communication model of agents with semantic web standards (OWL). With the model, agents are able to transfer their beliefs over existing web standards. This makes the agents (in multi-agent systems) bonded with existing technologies and services.

Similarly, the ontology-based natural language exchange model proposed a way that allows agents, which have different languages, to exchange information using their languages (Matson, Taylor, Raskin, Min, & Wilson, 2011). Information described in a language is interpreted into a form, called Text Meaning Representation (TMR), when the information is being sent to others. TMRs enable agents being indistinguishable in terms of what language they use. Ontological Semantics Technology (OST) to represent the meanings of information helps such interpretation of information with a specific domain. As OST accommodates larger domain, the meanings of information can be transformed more precisely and transmitted to broader spectrum of agents including humans. In 2012, IEEE-RSA (Robotics and Autonomous Systems) group has been formed to provide a standard ontology for robotics and automation that provides a unified way to represent knowledge to share (Schlenoff et al., 2012).

Shapiro and Iwánska (2000) argued that representing information from natural language into first-order logic may not capture the exact meaning of the information. The first-order logic referred here is "the standard, classical, first-order predicate logic, using its standard syntax" – from the footnote (Shapiro & Iwánska, 2000). The reason for the argument comes from the difference between logic and natural language when used to interpret information. SNePS is one of the knowledge representation systems that builds rational agents using natural languages (Shapiro & Rapaport, 1991).

2.2.2 Goal Model

When multiple agents perform their actions in a shared environment, their actions and intentions may conflict even if they are cooperative. Resolving conflicts of actions is relatively easier as the agents participating in the situation need to establish an interaction locally and adjust their actions (e.g., prioritizing the actions) until resolved. It is assumed that they are capable of communicating with each other to exchange information in a form of representations, such as TMR (Matson et al., 2011), XML (Laclavık et al., 2006), etc. In contrast, resolving conflicts of intentions needs higher level of interaction in order to bring the best outcome from possible adjustments. This type of interaction may cost very expensive as higher number of agents involved, and as their knowledge differs (i.e., larger number of alternatives to resolve).

A complex goal is not easily attained by simply deploying multiple intelligent agents. In general, such complex goals are not meant to be attained by individuals, but by multiple agents working in a cooperative manner. Complex goals can be analyzed and possibly decomposed into sub-goals such that agents take the sub-goals to attain the goal initially given. Sub-goals are associated with each other in the following ways (see Figure 2.2),

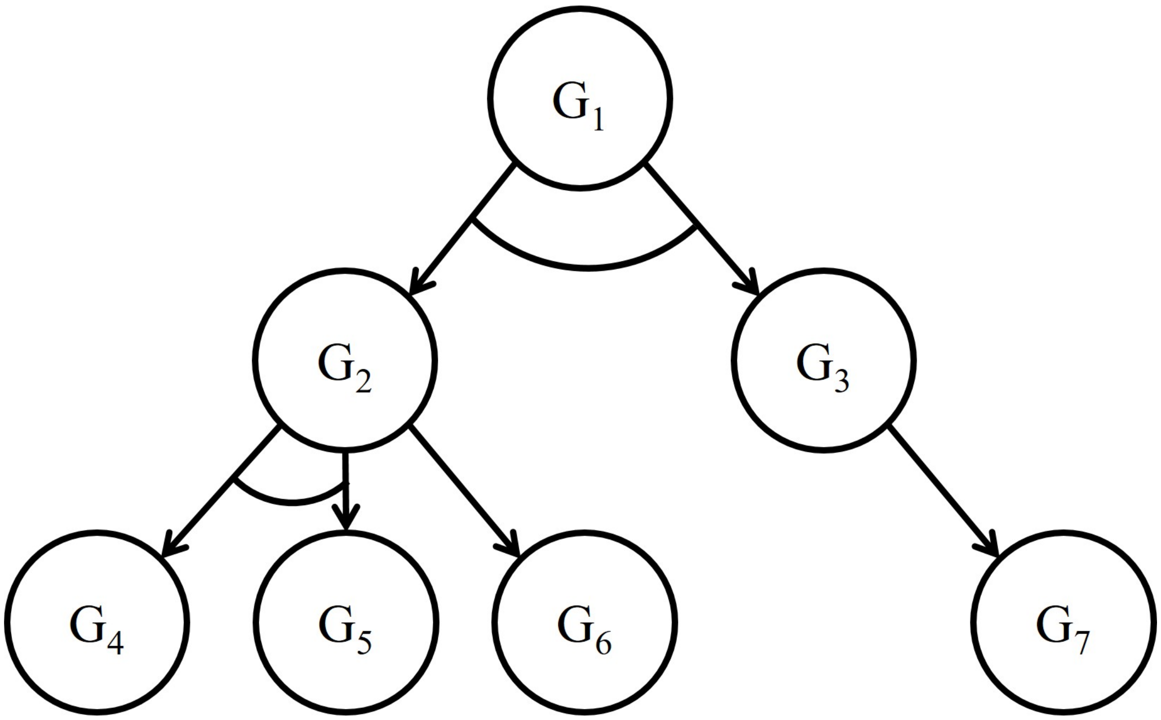

Figure 2.2. Representation of the two types of sub-goals; pairs of G_2 – G_3 and G_4 – G_5 are conjunctive sub-goals whereas G_6, and G_7 are disjunctive sub-goals

- Conjunctive: all the sub-goals should be attained to attain the parent goal
- Disjunctive: accomplishment of one of the sub-goals can attain the parent goal

If a goal can be decomposed into both conjunctive and disjunctive sub-goals (e.g., the relationships between the sub-goals of G_2 in Figure 2.2), the goal can be attained from attainments of the conjunctive sub-goals or an attainment of one of the disjunctive sub-goals. The effects of those attainments between the conjunctive and disjunctive sub-goals are the same. This is because these two types are distinguished only by characteristics of their parent goal.

Excessive assignment of agents to disjunctive sub-goals or shortfall in assigning agents to conjunctive sub-goals can result in poor performance. This implies that for conjunctive sub-goals agents need to be coordinated and assigned properly to all of the sub-goals, whereas one agent needs to be assigned to the best sub-goal among disjunctive sub-goals, in order to bring the most efficient outcome.

The research studies have illustrated different types and structures of goals (Braubach et al., 2004; Thangarajah, Padgham, & Harland, 2002; Van Riemsdijk, Dastani, & Winikoff, 2008), and its operational semantics (Harland, Morley, Thangarajah, & Yorke-Smith, 2014). In multi-robot systems, *Perform* and *Maintain* goals are the common types of goals as both goals typically require a physical action. The study (Braubach et al., 2004) also showed that many multi-agent modeling tools support those two types of goals.

Figure 2.3, brought from Braubach et al. (2004), illustrates life-cycle of a goal. During the cycle, beliefs stored in a knowledge base may be used as creation and drop conditions of a goal. Once created, goals can be in different states before dropped. Whenever a goal is created, it first transitions to ‘option’ state. In this state, the goal is treated as a desire for agents to pursue. When an agent decides to take this goal, the state transitions to ‘active’ state. In ‘active’ state, the agent that takes the goal tries to attain the goal by performing a course of actions that leads to the state 'finished'. The goal may be suspended and may become an option again, based on the context currently being perceived from the environment.

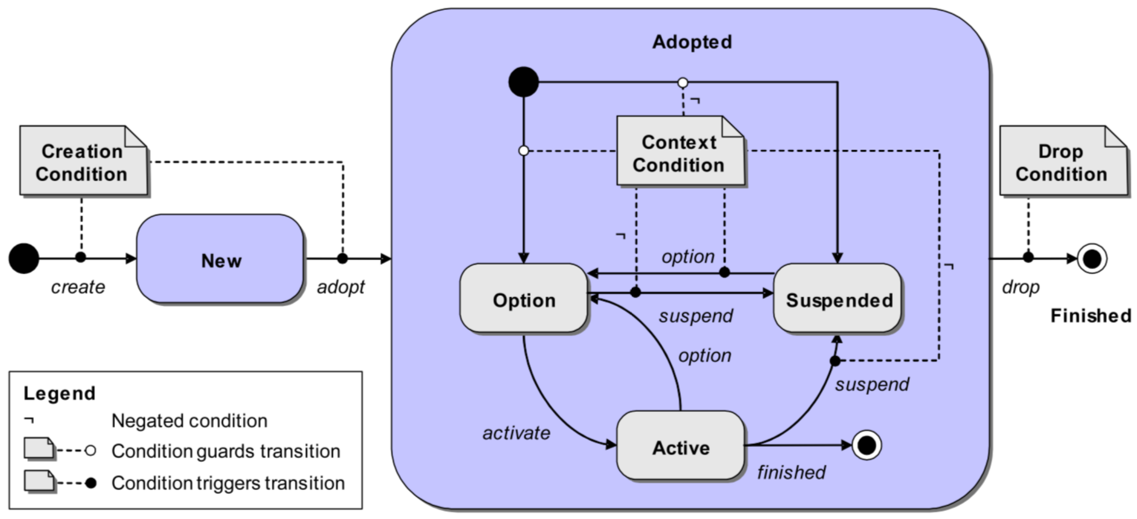

Figure 2.3. Life-cycle of a goal, from the study (Braubach et al., 2004)

2.2.3 Goal Deliberation and Reasoning

Whenever agents sense or receive a desire, they start reasoning to evaluate if the desire can be pursued. If it can be pursued, the desire (i.e., goal) becomes an intention and the appropriate plan is selected to attain the desire. In BDI-based agent systems, this process is called *deliberation*. From the view of goal life-cycle (see Figure 2.3), this is the process that brings a goal from 'option' state to 'active' state. To make this process occur, the context conditions of the goal need to be checked whenever necessary (e.g., when agents do not have any intention currently pursued).

Deliberation evaluates context conditions of goals with relevant beliefs, represented in predicate logic. If all context conditions of a goal can be entailed by beliefs currently believed to be true, the conditions are met. This entailment can be driven by '*forward chaining*' algorithm. On the other hand, the deliberation process can also examine what other context conditions including the context conditions currently being evaluated need to be met to activate the goal. This examination can be done by '*backward chaining*' algorithm. The forward and backward chaining approaches are described as (some points are highlighted to emphasize),

> "Backward chaining is **a form of goal-directed reasoning**. It is useful for answering specific questions such as What shall I do now? and Where are my keys? Often, the **cost of backward chaining is much less than linear in the size of the knowledge base, because the process touches only relevant facts**. In general, an agent should share the work between forward and backward reasoning, limiting forward reasoning to the generation of facts that are likely to be relevant to queries that will be solved by backward chaining." (Russell & Norvig, 2016, p.220)

For example, let a knowledge base consist of the following beliefs,

P

$\wedge Q$

$\wedge (P \wedge Q) \Rightarrow T$

$\wedge (Q \wedge R) \Rightarrow S$

$\wedge (P \wedge S) \Rightarrow W$

From the knowledge base above, forward chaining algorithm finds that the symbol T can be entailed because both P and Q are satisfied (i.e., both symbols are true in the knowledge base). The algorithm also finds that the symbols S and W are not entailed due to the lack of the symbols R and S. However, backward chaining algorithm first finds that W needs P and S to be met. Since P already meets, it looks for S. The symbol S needs R to be met. The backward chaining algorithm finally finds that the symbol R needs to be met to satisfy W. Backward chaining algorithm allows agents to find missing pieces to meet the goal that they are currently pursuing. The missing pieces may turn into sub-goals that require to be attained prior to the goal. The goal and its sub-goals can then be structured using the goal tree (See Figure 2.2) to illustrate their relationships.

Once a goal tree is organized from the process of deliberation and reasoning, it is time to select one from the goal tree. When multiple goals including sub-goals are available (i.e., the goals in 'option' state), BDI-based agents need to be able to prioritize the goals because the goals are dynamically suspended and re-activated in operation. Khan and Lespérance (2010) proposed a logical framework to prioritize goals based on priority and dynamics of goals. In addition, Thangarajah, Harland, and Yorke-Smith (2007) proposed the constraint optimization problem (COP) model to reflect preferences toward goals and utility measurements in the process of prioritization. Pokahr, Braubach, and Lamersdorf (2005a) proposed 'Easy Deliberation' that allows agents to specify relationships between goals to help decide what goal to pursue.

Selection of a plan to attain a goal needs to consider environmental properties and preferences of agents. Nunes and Luck (2014) proposed the plan selection algorithm that takes into account costs of execution of actions and agent's preferences. In the example of the study, the algorithm selects the best applicable plan with consideration of the factors – 'safety', 'security', 'performance', 'cost', and 'comfort' – to attain the goal '*go to work from its home*'.

2.2.4 Distributed Goal Planning in Cooperation

Intelligent agents in multi-agent systems usually do not have information that describes the whole world; they only know a part of the world (refer to the blind man and the elephant from the article (Saxe & Schwartzott, 1994)). The lack of information may fail to allocate a complex goal in cooperative MAS. Pieces of information perceived from individuals need to be merged and shared among the individuals to make rational decisions for the organization that the individuals belong to.

For both homogeneous and heterogeneous multi-agent systems, coordination of agents toward goals occurs based on the results from interactions between agents participating to the interactions. In general, conducting the process of coordination is easier and simpler in homogeneous systems than heterogeneous systems because agents in homogeneous systems are single-capable that brings fewer alternatives to coordinate, i.e. less complex. Unless agents are originally structured in a hierarchy, no agent possesses or directly controls other agents, i.e., the hierarchy of commands is flat.

Although any agent can initiate a series of communication at anytime for coordination, it typically happens when agents perceive a goal, which cannot be attained by the individuals, as described in the previous section. The agents then broadcast the perceived goal to others in the boundary of influence to initialize interactions for coordination. Such interactions allow them to exchange information of what they believe to be true. The interactions directly update beliefs in their knowledge base, and thus bring alternatives to serve the given goal. The alternatives usually bring social benefits or individual benefits or both. In the environment where resources are limited, alternatives that bring only individual benefits are difficult to be selected, but other alternatives that bring either social benefits or both benefits are likely to be selected. Once the most beneficial alternative is selected, all participants of the interactions agree to be committed to take a course of coordinated actions.

As those social interactions enable communications between agents toward social decision making, a variety of interactions has been studied. In particular, the market-based approaches have received an attention from the research community as the approaches well illustrate the cooperation problem and relevant solutions (Dias, Zlot, Kalra, & Stentz, 2006; Semsar-Kazerooni & Khorasani, 2009).

The auction-based method initiates interactions between agents to select who bids the best. The bidder who wins an auction is selected to take what it desires to do, as a price of the auction. In task distribution and planning, some of bidders usually bid for the same item because their preferences toward tasks are similar (for most of time they bid for the item that gives them the most profit). A bid usually represents a cost that the bidder proposes to spend for the auction item. The best bid is then the minimum cost to do the task (i.e., the auction item). From an organizational perspective, it is optimal when all tasks for a given set of goals are distributed to the agents which consume minimal resources and bring the maximum profit obtained from the allocated tasks. However, in cooperative multi-agent systems within dynamic environments it is difficult to find an optimal solution of the problem so that calculating cost should consider not only resources based on bidders' capabilities toward the task, but also agent's independent variables (Hoeing, Dasgupta, Petrov, & O'hara, 2007), agent's instant limitations (Michael, Zavlanos, Kumar, & Pappas, 2008), communication cost (Xuan, Lesser, & Zilberstein, 2001), and social welfare (Jennings & Campos, 1997). For example, if the agent a is more capable of serving the task t than the agent b, the agent a's bid, i.e., cost, is likely to be always lower than the agent b's bid so that the agent a always takes the task and the agent b never wins. In order to prevent the point of starvation, a bid needs to include bidder's independent variables such as geographical location toward the auction items or remaining battery life. In addition, social factors need to be considered when bidders calculate a cost. One of social factors can be formalized by the fact that a goal requires a form of cooperation. In order to deal with such social factors, the research studies have proposed recursive auction (Viguria, Maza, & Ollero, 2008), utilization of combinational bids (Lin & Zheng, 2005), and use of combination of auction-based methods and others (Liekna, Lavendelis, & Grabovskis, 2012).

Intelligent agents in multi-agent systems always wonder how the world is shaped and try to map the world into their mind. Because those agents perceive incomplete of information about the world, they are willing to fill the missing information by asking their companions, which have another piece of information. In cooperative multi-agent systems, agents need to know what others do in order to coordinate itself to the plan that they have agreed to be committed. Placing a blackboard in such systems allows agents to check and change status of the world (Ram & Ramesh, 1995). Using blackboard is beneficial in that it reduces communication cost and time that takes to wait responses from others (Vallejo, Albusac, Castro-Schez, Glez-Morcillo, & Jiménez, 2011). The blackboard method has been used in multi-agent systems mostly to support concurrent tasks. Blackboard can even be divided into sub-blackboards when agents are limited in their communication; each communication group shares one sub-blackboard (Jiang, Xia, Zhong, & Zhang, 2005).

2.3 Simulating Multi-Agent Systems

Various forms of tools for simulating intelligent agents and robots have been developed in order to support actions and interactions that such intelligent ones perform in a simulated environment. Those tools are extremely helpful to validate the concept of proposed systems, and to prototype autonomous agents and robots. The tools are even more valuable when agents and robots that are designed are multiple and expensive due to physical and economical limits, i.e. size and cost. Testing Mars exploration robots (Carsten, Rankin, Ferguson, & Stentz, 2009), swarm bots (Mondada et al., 2004), and disaster-rescue robots (Schwarz et al., 2017) are examples of showing why using simulation tools are crucial. The following subsections review existing simulation tools and computer games that have been utilized for multi-agent systems.

2.3.1 Simulation Tools for Multi-Agent Systems

In the agent modeling approach, simulation tools are required to support at least one of the two features: modeling agent and defining interaction rules between agents, and between agents and an environment. Many studies and their tools have defined their own modeling language to model an agent and tried to standardize the modeling language for general use in agent modeling: Jason in AgentSpeak (Bordini & Hübner, 2005), Jadex (Pokahr, Braubach, & Lamersdorf, 2005b), NetLogo (Tisue & Wilensky, 2004), agent modeling language (AML) (Trencansky & Cervenka, 2005), SARL (Rodriguez, Gaud, & Galland, 2014), and so on. The reason that they support such modeling language is to implement agent-oriented programming (AOP) architecture because traditional object-oriented programming (OOP) architecture, supported from general purpose programming languages such as C++ and JAVA, does not fully capture characteristics of rational agent. For the other feature, the tools provide ways to design communications between agents, e.g. exchanging messages, and between agent and its surrounding environment, e.g. changing properties of nearby objects in the environment.

Agent-based modeling (ABM) is a one popular principle to model agents. ABM platforms well describe behaviors of agents and reflections of the behaviors in an ABM environment. AnyLogic (Emrich, Suslov, & Judex, 2007), NetLogo (Tisue & Wilensky, 2004) are ones of the known ABM-based simulation tools for multi-agent systems. However, ABM lacks of reasoning – it does not describe how and why an agent acts. Padgham, Scerri, Jayatilleke, and Hickmott (2011) proposed a way of integrating BDI agents into a simulation using ABM. The integration of BDI-based agents into ABM enables ABM agents to understand perceived information and reason what to do. A result of reasoning then fires an action linked to the behavior model of ABM. The study also designs an interface to support message exchange between ABM agents to share perceptions and intentions.

2.3.2 Simulation Tools for Multi-Robot Systems

To validate system models, simulation tools also need to provide a realistic simulation environment as closer to the real world as possible. Autonomous robots particularly require realistic simulation tools more than other types of agents because robots change the environment more actively, using their actuators. Hence, failure in simulating behaviors of a robot may fail to deploy the system to the real world. A massive number of research studies and products in multi-robot systems have been published since various types of robots including cyber-physical agents, internet of things (IoT) concept, and their applications started actively being utilized and realized in the real world (do Nascimento & de Lucena, 2017; Zhong & DeLoach, 2011). These efforts try to integrate multi-agent system models and CPS including robots, in order to provide more realistic services to users, particularly to humans. Nevertheless, it also indirectly emphasizes simulation developers that simulation tools need to support characteristics of their models and targeted robots for successful integration.

Principles of multi-agent systems and reasoning of rational agent have started being embodied into the field of robotics as robots evolve and become more intelligent. This has happened more actively in simulation environments then the real world because robots are expensive entities – as described at the beginning of the section. Gerkey, Vaughan, and Howard (2003) described Player/Stage tool to simulate multi-robot, distributed-robot, and sensor network systems. Similar to ABM, Player models actors (i.e., robots) playing roles and Stage describes the simulated environment. Development of Players is done through general purpose programming languages. This enables flexibility in modeling various types of robots and their sensor models. The flexibility has led the tool being popularly used by studies in the field of robotics since 2002. Gazebo is an open-source simulation tool developed in cooperation with Player/Stage (Koenig & Howard, 2004). Gazebo provides in-depth modeling framework for both Player and Stage. Robots modeled in Gazebo are able to control each joint of the body and obtain more precise (i.e., more realistic) information from modeled sensors. Moreover, the 3D rendered simulation environment in Gazebo enhances reality of robots and their interactions.

Later, robot operating system group, known as ROS, has launched a platform consisting of supportive libraries and simulation tools to reduce the gap between the simulation environment and the real world (Quigley et al., 2009). ROS has successfully maintained and updated libraries and packages, and has been growing to continue supporting research studies involved mostly in robotics. ROS-powered robots can precisely be simulated in the tool, Gazebo. With this, any robot verified in the simulation can easily be deployed to the real world with a little or zero modification. However, multi-agent modeling principles and architectures such as Belief-Desire-Intention (BDI) and holonic organization model, studied in multi-agent systems, are yet weakly supported – existing libraries are not well maintained and their applications are stil sparse, comparing to other commonly used libraries and packages in ROS.

Use of simulation tools has recently been accelerated along with the fact that unmanned aerial vehicles (UAV) has been highlighted in the research field because of its potential for future applications toward intelligent robots (Ma'sum et al., 2013; Wagoner, 2017; Zhang et al., 2015). Schmittle et al. (2018) introduced a platform, OpenUAV, that supports testing UAVs in a simulation environment. It incorporates many state-of-art technologies such as Docker[1], ROS, MAVLink[2], and Cloud to make the platform extremely accessible from existing models and libraries. OpenUAV provides containers to run a simulation independently from other running simulations. This allows multiple simulations to run concurrently. Each container runs an instance of ROS that interfaces with Gazebo simulation. Simulations running in containers can be monitored using TensorBoard[3] or web interfaces.

2.3.3 Computer Games for Simulating Intelligent Agents

Computer games typically have a goal for players to attain. A list of actions to attain the goal are also provided to give players alternatives. Players including non-human players may compete or cooperate, depending on the genre of the game. In either situation, each player needs to reason about facts perceived from the game

[1]An automated program that encapsulates software components into a virtual container to run independently

[2]Micro Air Vehicle Link that defines a protocol for communication between UAV and host system

[3]A web interface that monitors and logs values of interests in time domain

and plan the best action to attain the goal. Each active player can be considered as an agent playing roles in the games. Computer games are similar to other multi-agent simulation tools such as Gezebo except the fact that they provide a specific context.

Computer games have been actively used as a test-bed in the area of practical AI and machine learning (ML). Norling and Sonenberg (2004) modeled BDI agents as a player in the game Quake2 to capture human behavior. Other techniques such as reinforcement learning (Mnih et al., 2013), deep learning (Guo et al., 2014), and case-based reasoning (Aha, Molineaux, & Ponsen, 2005) were also utilized in games to prove and validate those AI techniques. As the game industry expands, games and their market become more detailed (e.g., applying physics engines and larger number of types of behaviors in games), and become more realistic – similar to Gazebo in ROS.

StarCraft II is a real-time strategic game popularly played by over 200,000 users from many countries in the world since 2010 (with the best effort, no official record on the number of players was found such that it is estimated unofficially). All units in the game are originally designed to be controlled by one player and each unit has different capability. This allows players to build their own strategy to win the game. The AAAI conference on artificial intelligence and interactive digital entertainment and the conference of computational intelligence in games have held annual tournaments (Churchill et al., 2016), in which intelligent bots that are either scripted their strategy or trained by machine learning algorithms compete in StarCraft for research purposes (Ontanón et al., 2015; Wender & Watson, 2012) and fun. In addition to the efforts of using StarCraft as a simulation tool, in 2017, Blizzard entertainment, the maker and publisher of StarCraft II, and Google DeepMind team have released a more standardized way to access the game from external and control units in game (Vinyals et al., 2017). Many artificial intelligent related research studies have started using this game as a simulated environment (Gajurel, Louis, Mendez, & Liu, 2018; Park et al., 2018; Rashid et al., 2018).

Because all StarCraft II units and their actions are well modeled and verified from both the game designers and users for many years, only modeling a multi-robot system that controls those units to test and validate the system is required.

2.4 Summary

This chapter provided a review of the literature relevant to goal deliberation and planning problem in cooperative multi-agent systems. Although heterogeneous agents are capable of serving complex goals, coordinating agents to goals needs to be carefully proceeded in order to prevent confusions about what to do and conflicts between agents while performing actions. Moreover, dynamic environment changes the criteria used for the coordination process such that the system needs to address those changes.

Running BDI agents in simulations still has a technical gap between modeling BDI agents and use of appropriate simulation tools. Many existing multi-agent simulation tools did not elaborate actions and consequences of the actions enough to deploy and use the acting entity to the real world. One cannot say "*let's deploy our robot to pick up the box on the floor, after the simulator prompted in terminal 'the robot has successfully picked up a box'.*". This is because the modeling tools have focused more on the mindset of agents and cyber-interactions between agents. It is hoped that more multi-agent simulation tools such as Player/Stage are introduced in the future to apply BDI principles for better intelligence in robotics.

Next chapter deals with the problem of goal deliberation and planning in detail and propose a computational model that accommodates the desired features discussed in this chapter.

CHAPTER 3. GOAL DELIBERATION AND PLANNING IN COOPERATIVE MULTI-AGENT SYSTEMS

This chapter introduces a multi-agent system model that copes with the goal deliberation and planning problem, discussed and detailed in the previous chapters. Agents in the multi-agent system are socially rational, as assumed in Chapter 1, and eager to attain goals whenever possible. To realize it, the system model is required to accommodate representation of information, reasoning using the information, and planning and sharing intentions that the reasoning would bring. The rest of this chapter first describes the proposed model, and then addresses each of the accommodations.

3.1 Goal-Oriented Multi-Agent Systems (GOMAS)

The system model that this research study proposes is named goal-oriented multi-agent system (GOMAS). Despite of the implication from the name that the model would work for all types of agents, this research study strictly targets robots as the intelligent entity in the system – cyber agents do not fall into the consideration. The two main reasons are because the system assumes that the agents have actuators to influence directly on objects in the environment, and because it mainly considers applications where the connection between robots and operators of the system is unreliable, while robots are interconnected as long as they present within their communication range. In other word, the applications require physical presence of agents, not the agents that exist in cyber world.

3.1.1 Overview

The architectural view of the GOMAS model is depicted in Figure 3.1. The arrows in the figure indicate flows of information. Since the actors in the GOMAS model are robots, they have physical sensors and actuators. Using those components the GOMAS robots are able to perceive information from the environment and alter properties of objects in the environment. The GOMAS robots have two core components: *goal reasoner* and *goal planner*. The core components actively deal with the problem 'goal deliberation and planning'. The knowledge base, abbreviated as KB, stores all perceptions from the sensors as well as actions that the robot is committed to perform. The plan library stores a set of plans and suggests any applicable plan toward a goal requested from the goal planner. The blackboard tracks progress of goals that the GOMAS robots are currently pursuing. It also allows them to share those progress to attain goals cooperatively.

3.1.2 Basic Behaviors of GOMAS

Once new instances of goals are given during life-cycle of robots, the robots show rational behaviors in a sense that they proactively pursue the instances of goals. The basic behaviors of robots in the GOMAS model is illustrated in Figure 3.2. These behaviors are repeated whenever a new instance of goals arrives to the system. The four basic behaviors are detailed as follows,

Distribution: Operators do not specify a robot when a set of goals is given to the system. They rather give the set of goals to any robot that they can reach at the moment. Operators may choose the closest robot – the closest in geometry or network topology. Since agents in the GOMAS model are robots (i.e. a kind of cyber physical systems), they are capable of receiving goals using their physical sensors such as microphone and camera as well as networking modules. As soon as a robot receives the set of goals from the operator, it begins to distribute the goals. This happens only in the boundary of influence of the robot – it does not send the

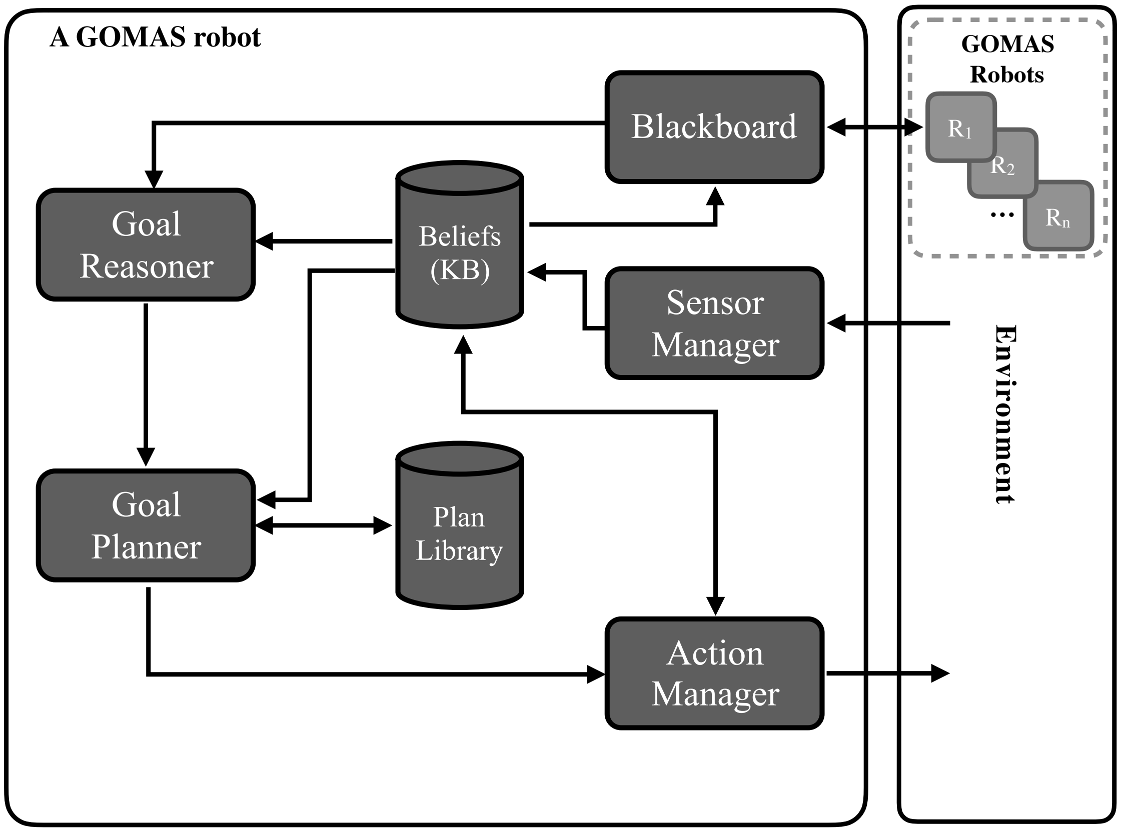

Figure 3.1. The architecture of the GOMAS model. The arrows indicate a flow of information.

goals to all robots in the system. The boundary of influence is determined based on network capability or sensor limits of the robots. For example, if they use commercial products that utilize 2.4 GHz ISM bands for tele-communication, their boundary of influence may be limited within 100 m. For another example, if robots use a speaker that produces about 60 decibels (similar to the sound level of normal human conversation) to describe the goals to other robots nearby, the boundary of influence can be significantly reduced by up to several tens of meters – it is assumed that the background noise from the actuators of the robots disturbs perception of the sound under 30 decibels; then, a sound source that produces 60 decibels at 1 meter may travel approximately 31.62 m. Other robots which received the goals

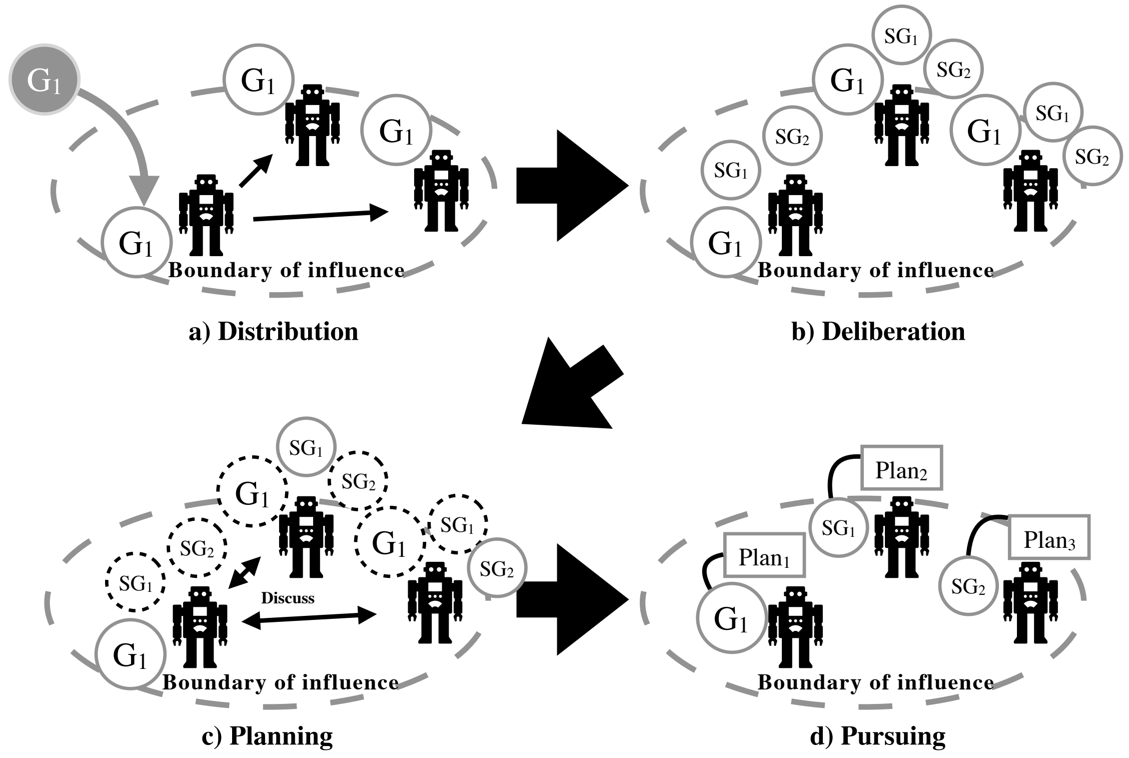

Figure 3.2. An illustration of showing how GOMAS system operates from the view of operators. G indicates a given goal while SG stands for a sub-goal.

may re-distribute the goals to another robots in their boundary of influence. However, the distribution only occurs when robots receive a new goal; joining to the boundary of influence of others and updating existing goals do not trigger this process unless requested.

Deliberation: During the process of deliberation robots do not interact with each other. Instead, they start inferring internally toward the goals that they are given. The goals may have prerequisites. The prerequisites that are not satisfied with the current context generate sub-goals (shown as SG in the Figure 3.2) and relate the sub-goals under the goal. Creating sub-goals may occur recursively as necessary – sub-goals of a goal may also have prerequisites to satisfy the sub-goals.

It is reasonable to say that robots may bring a set of sub-goals different from each others based on their level of intelligence. Since this research study does not address this difference in knowledge, it is assumed that the level of knowledge is the same across all the robots in this system. This makes the robots bring the same set of sub-goals for the goal that they shared. It is also assumed that the context that they perceived is the same unless perceived locally.

Planning: Once robots finished structuring all sub-goals along with their parent goals, they start planning their actions to attain the (sub-)goals. Because they constructed the same goal tree they would come up with the same set of (sub-)goals that they need to attain first. They now begin examining each of the (sub-)goals to see if they have an ability to attain. As a result, a set of (sub-)goals that are available and can be attained by the robot is listed. And then, the robots choose one that is the most desirable. Note that the strategy on selecting one goal from a set of goals is subjected to regulations of the organization, preferences of individuals, benefits from the goals, and so on. Later section will deal with the strategy used in the GOMAS model in detail. Nevertheless, the most important step in this process is to inform other robots about their selection from the set of goals. When this is informed, other robots either do not pursue the goal (i.e., mark the goal as not attainable by them) or suggest a conversation on deciding which takes the goal. Depending on the result of the conversation, they may take the goal or move onto the other available goals and repeat this process. If there are no goals that one can attain or all goals are currently pursued by other robots, the one may do nothing or do whatever that is designed to perform when idle.

Pursuing: At this stage, robots know their intention toward the goal that they selected in the previous stage. The plan that can attain the goal is in place. Now robots perform required actions for the plan in a sequence. While performing actions, the robots considerably and continuously verify the current context to react to any condition changes that may affect the intention as well as the plan. If such conditions are perceived, the robot adjusts its behavior. The robot may attain or fail to attain the goal. In any case, the robot is responsible for updating other robots on the status of the goal. Once this stage finished, robots may go back to *planning* stage until the top-level goal is attained.

3.2 Goals in GOMAS

Robots in the GOMAS model are goal-oriented, as the name of the model implies. The robots always look for goals. Because the GOMAS model considers goals as desires of what operators want the system to achieve, goals are given to the GOMAS robots from outside of the system, i.e. from operators. Many goals can be provided to the robots and further added in the future. However, only one goal or one sub-goal can be pursued by one robot at a time. The following sub-sections describe how goals are created, related, and pursued by the GOMAS robots.

3.2.1 Goal Model

The goal model in the GOMAS model structures goals and sub-goals in an hierarchical way. Goals that have no parent are top-level goals. Top-level goals are given from operators. Sub-goals (i.e., child goals) apparently have a parent goal. Goals that have sub-goals are not supposed to be pursued until the sub-goals are attained. As described earlier in the previous chapter, all sub-goals need to be attained to activate the parent goal if the sub-goals are in a conjunctive relationship. On the other hand, one of the sub-goals needs to be attained to enable the parent goal if the sub-goals are in a disjunctive relationship. Robots distribute a goal including its sub-goals and sub-goals of the sub-goals if exist.

This research study actively employs the concept of the goal model from the study (Braubach et al., 2004). Braubach et al. (2004) described the life cycle of a goal (See Figure 2.3). According to the concept, goals are created and dropped dynamically. The study also described different types of goals: *achieve*, *maintain*, *query*, *text*, *perform* and so on. Among the goal types the GOMAS model implements only the type *perform*. The reason comes from the fact that in the GOMAS model operators provide a goal to actively reach a state of affairs – a series of actions needs to be *performed* to transition toward the state. However, autonomous robots need to *maintain* their status as valid (e.g., charge battery, avoid collisions) to continue serving goals. In this research study, this remains as future works.

Figure 3.3 shows the GOMAS goal model. A goal can be transitioned to 'suspended', 'option', and 'active' after created. Transitions only occur by activation and context conditions of the goal. Those conditions are states currently perceived from the environment. Activation conditions consider state of sub-goals, while context conditions look for perceptions related to the goal. In 'suspended' state, the goal is not considered by robots, i.e. the goal is not desired.

In order for a goal to transition from 'suspended' to 'option', state of the sub-goals under the goal need to satisfy the activation conditions. This mechanism not only allows goals to be prioritized, by blocking one until others have attained, but also allows robots not to consider the goals – the size of considerable goals can be reduced. Once the activation conditions are met, the goal transitions to 'option' state. In this state, the goal becomes a desire. The desire now attracts robots as they proactively behave toward goals. When robots decide to pursue this goal, and when the context conditions of the goal are met, the goal can transition to 'active' state. The robot pursuing the goal is now performing a course of actions. Because the type of the GOMAS goal is *perform*, the goal is transitioned to the end and thus attained, as soon as the robot successfully delivered all the actions. However, the state of the goal may go back to 'option' state if one or more of the context conditions become unsatisfied while the course of actions in place. The goal that went back to 'option' state can be pursued again, as long as there is a robot that takes the goal and all the context conditions of the goal are met again.

The GOMAS goal including sub-goals is characterized by the following properties:

- Activate conditions: Activate conditions are conditions that each specifies desired state of a sub-goal. Depending on how sub-goals are related, the conditions can be satisfied from one sub-goal or all of the sub-goals. The conditions are represented and compared using first-order predicate logic.
- Context conditions: Context conditions control state of the goal. As long as the context conditions are met, the goal should eventually be attained. The conditions are also represented and validated using first-order predicate logic.
- Concurrency limit: The concurrency limit defines how many robots can pursue the goal at the same time.

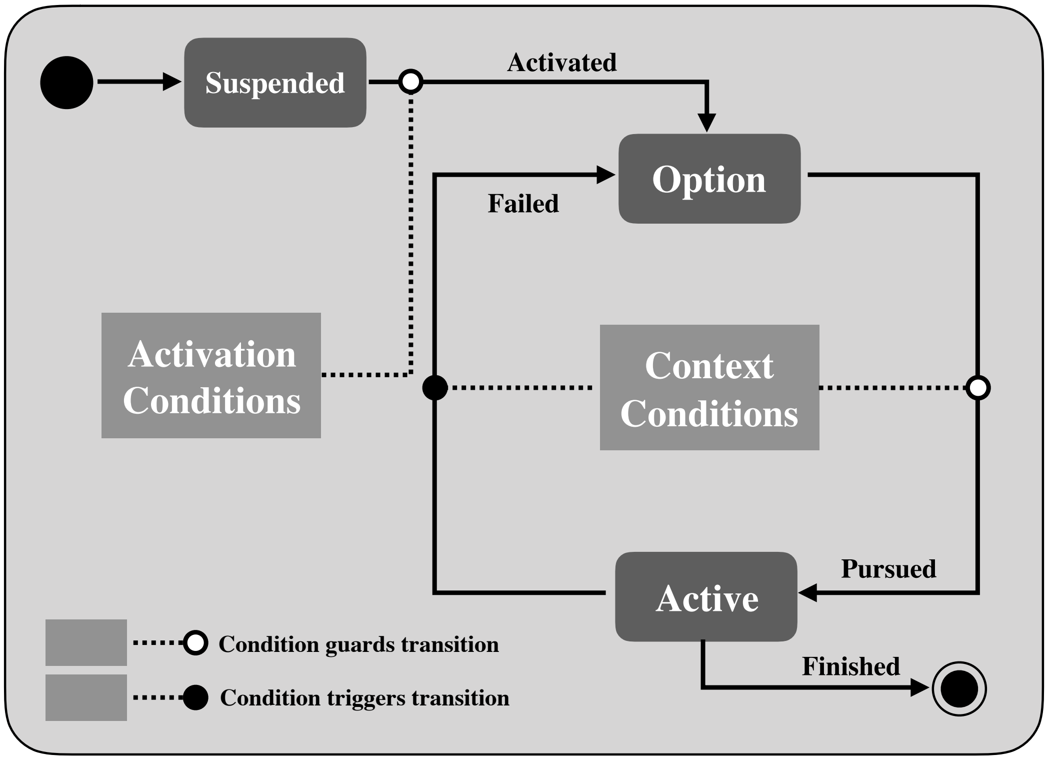

Figure 3.3. Goal lifecycle in GOMAS, inspired and referenced from (Braubach et al., 2004)

- Parent: There is always zero or one parent in a goal. If this is set as zero or none, the goal is a top-level goal. Otherwise, the goal is a sub-goal.
- Children: This defines a list of sub-goals. If the list is not empty, the goal contains activation conditions. If it is empty, the goal immediately transitions to 'option' state.

Goals that require only one robot do not use the 'concurrency limit' property. The property is rather set to 1 as default, to mean that only one can attain the goal at the same time. In this case, actions to attain such goals are typically constrained. For example, a goal '*Parked at the spot A*' does not need many mobile robots to park at the spot. Instead, only one mobile robot is required to park at the spot for the goal. On the other hand, goals that require multiple instances of actions concurrently happening set this property. For example, a goal '*Clean the room*' may set the property as 10 to allow that up to 10 cleaning robots can clean the room at the same time. This property ultimately brings the system higher efficiency in time.

3.2.2 Sequential and Parallel Goals

Operators can relate goals to make explicit priority between the goals, before giving them to the system. Relating goals is done through a process of placing goals in a hierarchical structure using the goal tree – the relationship of a parent and children (See Figure 2.2). Parent goals are not pursued until all children (or one of the children) of the parent are attained. Figure 3.4 shows an example of two general types of goals that structure operator's desires. As illustrated in Figure 3.4(a), goals without relationships are attained in a brute-force manner, depending on which conditions that satisfy the goals are met first. For example, G_2 would have been attained before G_1, if the satisfactory conditions of G_2 were met prior to G_1. This allows participating robots to decide the order of attainment based on the current context. On the other hand, Figure 3.4(b) shows an explicit order for the goals – G_3, G_4, G_2, and G_1. The order of attainment between G_3 and G_4 also depends on

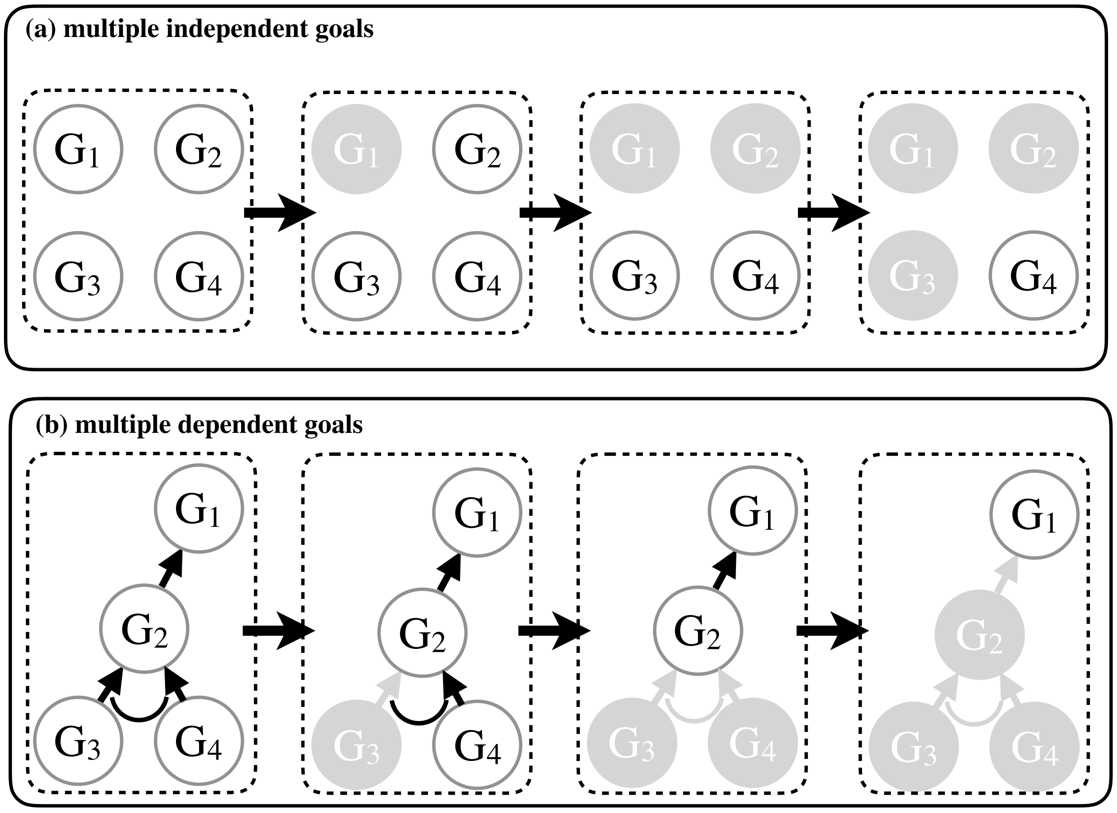

Figure 3.4. Examples of initial goals and showing how different types of goals would be attained over time. Goals that are attained are grayed while goals that are not yet attained are shaped in a solid circle. (a) depicts independent goals while (b) shows dependent goals.

the context conditions. However, G_2 cannot be pursued before G_3 and G_4. G_1 also cannot be pursued before G_2.

Goals and sub-goals that are not directly related to each other can be pursued in parallel by multiple robots. Once the goals and sub-goals turn into desires, i.e. in the ‘option’ state, and their context conditions are satisfied, they are considered by robots. The robots may select each goal and independently perform courses of actions to attain the goal. From the example of Figure 3.4 (a), the four goals can be simultaneously pursued and attained by four robots. G_3 and G_4 from

Figure 3.4 (b) can also be pursued and attained at the same time from two robots. This greatly improves time-efficiency until the number of robots exceeds the number of the goals and sub-goals. Sub-goals of a parent goal are also considered for this parallel process. If the sub-goals are conjunctive each robot takes each sub-goal and attains it. On the other hand, if the sub-goals are disjunctive each robot takes each sub-goal, but the earliest attainment of the sub-goal may terminate the other sub-goals being pursued by the other robots.

3.3 Deliberation of Goals in GOMAS

As described in the previous section, a goal may require a set of prior states to be satisfied prior to attaining the goal. The GOMAS robots need to be able to reason about the goal and its prerequisites, and expand it further in depth to understand '*what things need to be done for the goal*' for successful operation. The robots also need to cautiously keep watching their actions toward the goal and consequences of the actions, in order to track '*when the things are satisfied*'. The GOMAS model provides a knowledge base for each robot to store every piece of information that the robot perceived from the environment and others. Beliefs stored in the knowledge base are later used to reason about goals.

3.3.1 Knowledge Representation and Logical Rules

Beliefs are represented using first-order predicate logic. Logical relationships between literals are made using the connectives: $\neg$, $\wedge$, $\vee$, $\Rightarrow$, and $\iff$. Then, a belief can be elaborated by combinations of those relationships. Predicate logic uses propositional variables to represent different states. For example, '*Has(he, car)*' representing that 'he has a car' is different from '*Has(she, car)*' that apparently represents 'she has a car'.

Unlike linear temporal logic (LTL) and computation tree logic (CTL) that support temporal operators (e.g., **X** and **U** representing next and until, respectively) as well as logic operators, predicate logic itself does not represent information in time domain. For example, a predicate logic '*Hungry(me)*' can be true if '*me*' is hungry now. The predicate logic can however be false if '*me*' is not hungry now. Even, the predicate logic can be true and become false in the future. Some may argue that predicate logic can represent temporal property, '*Hungry(me, now)*' or '*Hungry(me, 10 seconds ago)*' as examples. However, that makes the knowledge base too complex – too many beliefs to store, and to retrieve when inferring.

Multi-robot systems in the real world need to consider temporal properties of the environment to appropriately act in time. The GOMAS model therefore generates all time-dependent beliefs whenever the deliberation process occurs – noting that beliefs are only valid when reasoning. When not deliberating, pieces of information are stored as variables. For example, a piece of information indicates that the level of hunger is low. Then, the predicate logic '*Hungry(me)*' is not generated in the process of deliberation – of course the predicate logic is generated if the level indicates high. In another example, '*CanBuy(Car)*' may be generated if it is believed that the robot has sufficient amount of the resources (i.e., money, a place to park the car) to buy a car. Deciding if those variables fall within the threshold can be a fuzzy process. For instance, medium level of hungry may or may not generate '*Hungry(me)*'. This fuzziness, however, is not further discussed in this research study because the topic is out of the scope of this research study.

However, the GOMAS knowledge base does keep some beliefs that are permanent over time. Those beliefs are known as rules and permanent facts. Rules are pre-defined by the system designer.

$$\mathit{Bought(me,\ car)} \Rightarrow \mathit{Has(me,\ car)}$$
$$\mathit{Has(me,\ apple)} \wedge \mathit{Ate(me,\ apple)} \Rightarrow \neg \mathit{Hungry(me)}$$

are examples of rules. Permanent facts are beliefs that never change after once generated.

$$\mathit{Identifier(me,\ xyz)}$$

Name(me, John)

NetworkAddress(me, 127.0.0.1)

are examples of permanent facts.

The GOMAS knowledge base supports many well-known rules to help inferring. Modus Ponens is one of the best-known inference rules. This rule is expressed as follows,

$$\frac{P \Rightarrow Q, P}{Q} \tag{3.1}$$

The expression means that if $P \Rightarrow Q$ and P are given, then Q can be inferred. This rule allows the GOMAS robots to infer something that has not explicitly perceived. For example, a relationship $CleanedRoom \Rightarrow RoomCleaned$ makes robots believe that the room is cleaned when they perceived a fact that a robot cleaned the room, even if they never perceived a fact that the room is actually cleaned. And-Elimination is another useful rule and is expressed as follows,

$$\frac{P \wedge Q}{Q} \tag{3.2}$$

This simply means that Q can be inferred if P and Q are given. For example, from a fact that $YouLive \wedge ILive$, $ILive$ can be inferred. Unification is a process that unifies propositional variables of predicate logic symbols that make the logic symbols look identical. For example, the unification process produces a dictionary $\{x : me\}$ from '*Hungry(x)*' and '*Hungry(me)*'. Unification process is useful for inferring because it allows to examine if the predicate symbol '*Hungry(x)*' satisfies with the given variable '*me*'. However, a unification of unrelated predicate symbols (e.g., '*Hungry(x)*' and '*Has(me, car)*') outputs an empty dictionary.

It is claimed that '*every sentence of propositional logic is logically equivalent to a conjunction of disjunctions of literals*' (Russell & Norvig, 2016, p. 215). A conjunction of disjunctions of literals is called 'conjunctive normal form' (CNF). According to Russell and Norvig (2016), logically represented beliefs can be

expressed in a form of CNF after converting the beliefs using the logic rules (For details, refer to Appendix A). A Horn clause is a disjunction of literals that contains at most one positive literal. For example, $(P \wedge \neg Q \wedge \neg R)$ is a Horn clause. Horn clauses that have only one positive literal are called Definite clauses.
And-Elimination and Definite clauses play an important role for inference because of their use in CNF.

3.3.2 Backward Chaining for Goal Deliberation

Backward chaining is a process that proves that a predicate literal Q is true. In the GOMAS model, this process tells the GOMAS robots what they need to do to satisfy a goal (i.e., a query). The algorithm 3.1 shows the backward chaining process used in GOMAS. The backward chaining process first finds all implications that conclude the query. At the same time, it also checks if the query is already satisfied. If not satisfied, it tries to prove all literals of the found implications. If a literal of any implication is not satisfied and there is at least one implication that concludes the literal, the process continues further proving the literal. This process recursively runs until it visits all of the relevant implications. If the query Q can be satisfied from this recursive process, then the query is logically satisfied. However, if there is no satisfied implication to support satisfaction of the query, then the process outputs that the query is not logically satisfied. Russell and Norvig (2016) claimed that computational complexity of the backward chaining process is less than the size of the knowledge base because it visits only relevant beliefs.

The backward chaining process evaluates if a query is logically satisfied. However, this process should not only tell the result of the evaluation, but also should provide why the query cannot be satisfied if unsatisfied. In other word, it should tell what should be done to satisfy the query. The GOMAS backward chaining process supports this feature by providing all literals required to satisfy the query. The process accumulates all unsatisfied predicate symbols during the

Algorithm 3.1 The GOMAS backward chaining process

```
Require: the knowledge base $KB$, the query $Q$
Ensure: : result of the query $Q$, and unsatisfied symbols for the query $Q$
 1: function GOMASBackwardChaining($KB$, $Q$)
 2:     for all beliefs in $KB$ do
 3:         Let $b_i$ the CNF-converted i-th belief from $KB$
 4:         Let $lhs_i$ a list of left-hand side literals of $b_i$
 5:         Let $rhs_i$ the right-hand side literal of $b_i$
 6:         if $rhs_i$ and $Q$ are related then
 7:             Let $usym_i$ an empty list for unsatisfied symbols for $b_i$
 8:             if $rhs_i$ and $Q$ are identical then
 9:                 Return True and $usym_i$
10:             else
11:                 for each belief from $lhs_i$ do
12:                     Let $lhs^i_j$ j-th belief from $lhs_i$
13:                     Let $u_i$ unified literal of $rhs_i$ with $Q$
14:                     Let $st^i_j$ $lhs^i_j$ with the variables substituted by $u_i$
15:                     Call GOMASBackwardChaining($KB$, $st^i_j$)
16:                     Let $r^i_j$ a satisfactory result of $lhs^i_j$ returned from the call
17:                     Let $usym^i_j$ a list of unsatisfied symbols of $st^i_j$ returned from the
    call
18:                     if $r^i_j$ is false then
19:                         Append $usym^i_j$ to $usym_i$
20:                     end if
21:                 end for
22:                 if $r^i_j$ is true for $\forall$ $lhs^i_j \subset lhs_i$ then
23:                     Return True and $usym_i$
24:                 else
25:                     Return False and $usym_i$
26:                 end if
27:             end if
28:         end if
29:     end for
30: end function
```

evaluation, and returns a list of the unsatisfied symbols. The GOMAS robots use those symbols to plan appropriately toward the query (i.e., the goal).

3.4 Goal Planning in GOMAS

The GOMAS backward chaining process, detailed in the previous section, should now tell the GOMAS robots any unsatisfied symbols for the goal. The process can actively be used for the robots to expand a goal. Once the goal is fully expanded, the GOMAS robots begin planning how to attain the goal. All the decisions and consequences of their actions made during this process need to be informed to other robots. Informing to others is essential to a successful and efficient goal planning. The following subsections will describe how an expanded goal can be tracked and attained.

3.4.1 Goal Expansion

To decide which sub-goal to pursue, the goal first needs to be fully expanded with regard to the current context. The goal may have some sub-goals that are already satisfied. All pre-satisfied goals and sub-goals are not generated in this process. This means that all generated sub-goals are not currently satisfied and thus need to be pursued. Figure 3.5 illustrates an example of the process of expanding a goal. In this example, G_1 is given as a top-level goal. In a), the deliberation process first visits the top-level goal G_1 and figures out that G_2 and G_3 are sub-goals of G_1. Since G_2 and G_3 are not satisfied with regard to the current context, both sub-goals are generated in the goal tree. Because goals are expanded in a post-order manner, G_2 is examined by the process prior to G_3. In b), the process reveals that G_4 and G_5 are the sub-goals of G_2. G_5 is not generated because it is already satisfied by the current context. Next, G_4 is considered by the process and G_6, G_7, and G_8 are added to the goal tree. G_6, G_7, and G_8 are however leaf goals. This means that the literals of the goals do not have any implication in the knowledge base to expand. The deliberation process would of course visit those leaf goals, but would come up with no sub-goals, disregarding satisfied or unsatisfied. In d), G_3 has two sub-goals G_9 and G_a – indicating 10 in decimal. Since G_9 is already satisfied only G_a is

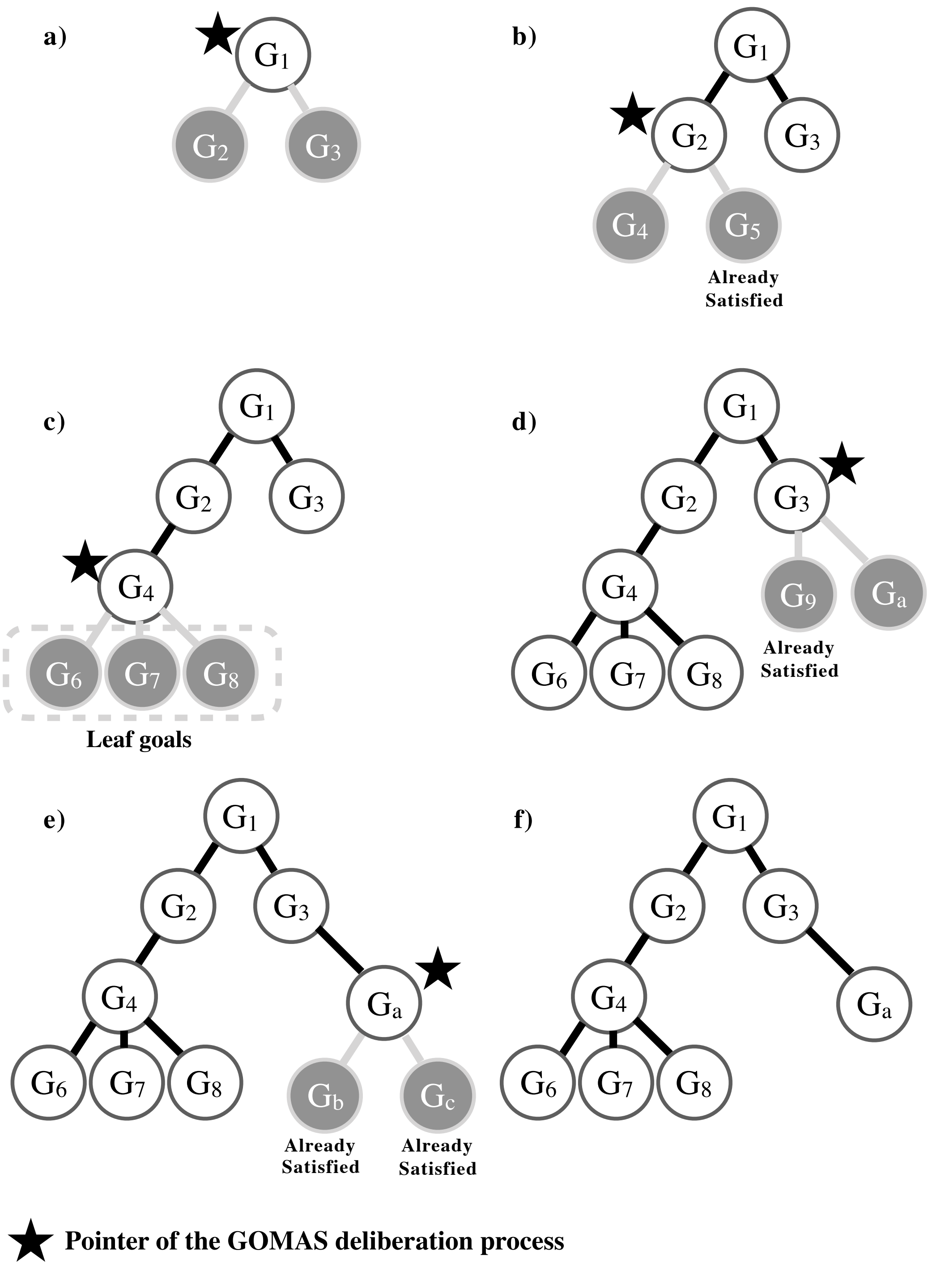

Figure 3.5. The process of expansion of a goal

added. In e), the process figures out that G_b and G_c are the sub-goals of G_a, and are already satisfied. Thus, they are not added to the tree. f) finally shows the fully expanded goal tree of G_1.

3.4.2 Goal Selection and Planning

The goal selection visits leaf goals first (i.e., the bottom-up approach). For the fully expanded goal tree from Figure 3.5 f), G_6, G_7, G_8, and G_a are the alternatives for the consideration of selection. Although G_6, G_7, and G_8 are placed in the same level (i.e., the same priority), the order of the visit would be G_6, G_7, and G_8 for technical reasons. One technical reason would be the way of visiting nodes in a tree structure. In this research study, left nodes are visited first (i.e., post-order). If goals have a utility function that approximates benefits from attaining the goals, the order would start from the most valuable goal. Moreover, the order could be determined by the order of preferences toward the goals.

While the goal selection sequentially visits the leaf goals, it evaluates two things: the activation conditions of the leaf goals and applicable plans for the leaf goals. The activation conditions should be empty for those goals when they are generated – they are leaf goals, and thus have no sub-goals. This may not be true in some cases. Recall Figure 3.5 for an example. It is known that G_5 is a sub-goal of G_2 and it was satisfied when considered by the deliberation process. When the selection returns to G_2 after G_4 and its sub-goals are attained, G_5 may become unsatisfied at that moment in time. This can happen when robots cooperatively share resources. Consider the following scenario to understand the circumstance:

> "Robot$_a$ and robot$_b$ are attaining the goal 'build a wooden table'. The goal requires the two robots to make a wooden top as well as wooden legs. Robot$_a$ begins pursuing the sub-goal of making a wooden top while robot$_b$ tries to attain the other sub-goal 'make wooden legs'. Both sub-goals require them to prepare a pile of woods. Robot$_a$ prepares a

> pile of woods for the wooden top. While robot$_a$ is making (or buying) nails and screws for the wooden top, robot$_b$ takes the pile of woods that robot$_a$ prepared, and makes wooden legs. The behavior of robot$_b$ may seem reasonable because the sub-sub-goal 'prepare a pile of wood' under the sub-goal 'make wooden legs' was already satisfied (by robot$_a$), even if the pile of woods was actually for the sub-goal 'make a wooden top'. Because the robots are cooperative, sharing and occupying resources may be tricky unless robot$_a$ would have informed robot$_b$ that the pile of woods was for the wooden top."

Once the activation conditions of the leaf goal are checked again and are all satisfied, the selection process begins asking suggestions for applicable plans from the plan library. The plan library is a system component that stores pre-defined plans. The plan library defines a dictionary of matches between goals and plans. The relationship between goals and plans is 1:n. It means that multiple individual plans can attain the same goal. Because of the assumption 'single-minded agents' (see Section 1.5), the plan library does not attempt to create a new plan, nor change the context of the existing plans.

A plan consists of a course of actions. The course of actions actively changes conditions and properties of the world. The plan thus needs to provide the GOMAS robot that is committed to perform this plan additional information (e.g., consequences of the actions) and its context information. The additional information is stored in the knowledge base of the robot. This information can then be used by the robot to track progress of the actions, and to determine if the actions are performed successfully.

Figure 3.6 shows the process in detail. The GOMAS robots that utilize this process first store their capabilities. This step happens every time when the robots request a plan from the plan library. The reason is because their capability may change over time, due to malfunctioning or context conditions that restrict the use

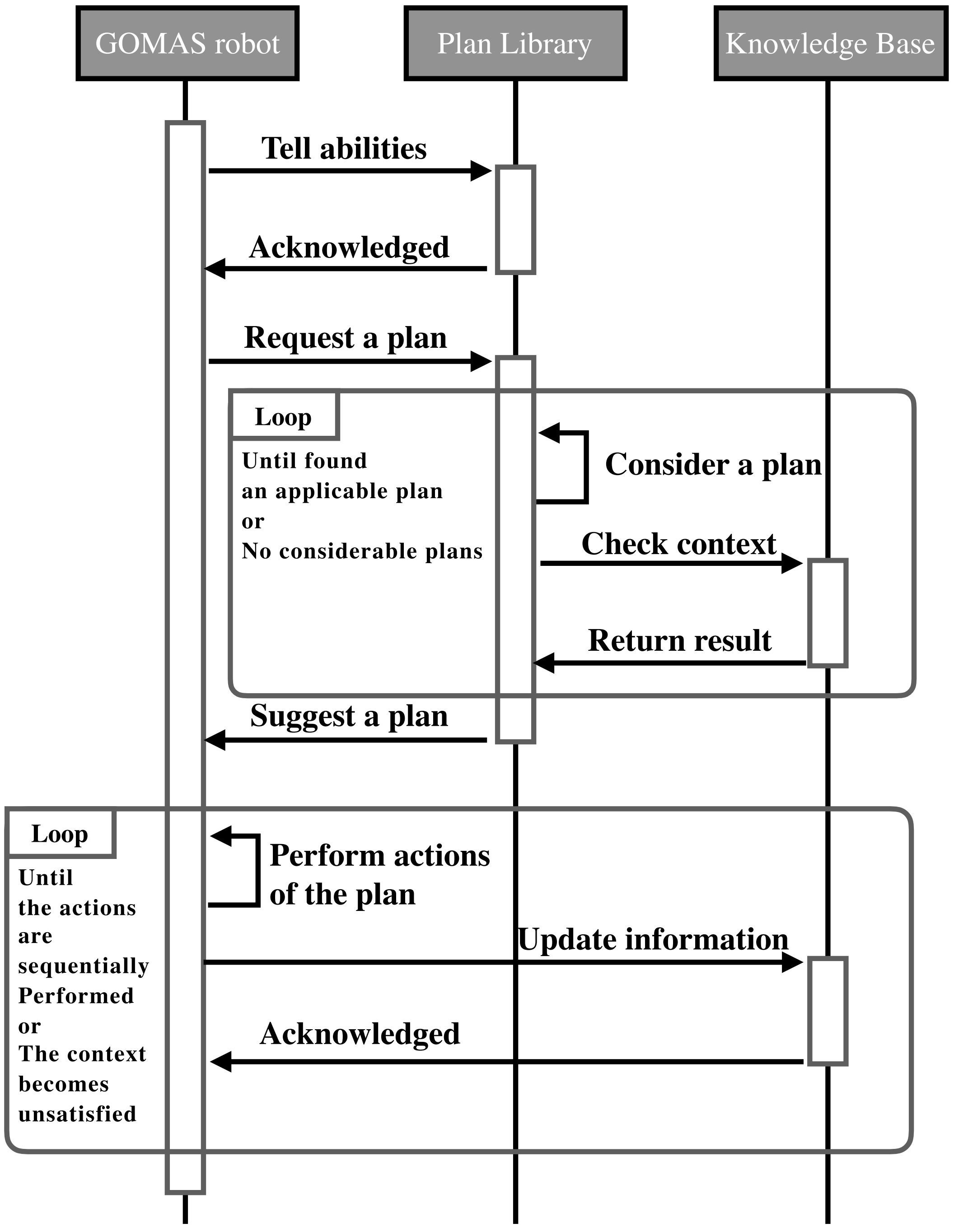

Figure 3.6. A sequential diagram of the goal selection and planning

of their capability. A flying robot for example may not fly because of the regulation that has been agreed by the organization that the flying robot belongs to.

The plan library suggests a plan whenever requested. During this step, the plan library lists all plans that can attain the goal. The listed plans are further pruned by matching the plans with the stored capabilities. If the GOMAS robot does not have the capability to perform a specific plan, the plan is not considered. And then, the plan library evaluates the context conditions of the remaining plans to determine if the plans are applicable now. For example, the plan library does not suggest a plan *Move* if there is no path currently available to go to the destination. After a plan is suggested from the plan library, the GOMAS robot begins to take actions from the suggested plan and perform. While performing actions, the robot also updates their knowledge base to see if the actions bring the expected consequences such that the plan is performed successfully. If the plan has failed, the GOMAS robot removes the plan and goes back to the goal selection process.

3.4.3 Asynchronous Blackboard for Goals

The GOMAS robots are required to inform their intention to others. As Ram and Ramesh (1995) stated, the act of informing others is the key to enable multiple robots to work collaboratively. In the GOMAS model, the GOMAS robots share information related to goals using the blackboard approach. The information includes any updates on goals, such as creations, modifications, attainments as well as occupations or releases of goals. Unlike the typical centralized blackboard approach, the GOMAS model employs the decentralized – distributed – blackboard approach, called GOMAS blackboard, in order to manage the implementation of the GOMAS goal tree. The GOMAS blackboard is not a singleton instance across the system, but is managed inside individuals. This liberates the GOMAS robots from maintaining any continuous connection to outside of them.

The GOMAS robots utilize the supported operations to maintain goals in their GOMAS blackboard. Table 3.1 shows the operations and its descriptions. Note that an operation of checking context conditions is not supported in the GOMAS blackboard because the operation should happen in the GOMAS robot model.

3.5 Robot Model in GOMAS

The GOMAS robots are the main actors of the system. They actively utilize the components of the GOMAS model to behave *intelligently.* Since quite intensive computations are used in the knowledge base and reasoning, the GOMAS robots need to be computationally powerful to react fast in real-time applications.

3.5.1 BDI Perspective

The GOMAS robots strictly follow the BDI principles. The robots reason and act only when there is at least a desire that may possibly become an intention. In traditional BDI architecture, desires are generated from events. The events include perceptions from environment as well as messages received externally. However, in

Table 3.1.

The supported operations of the GOMAS blackboard

Operation	Description
AddTopGoal	Adds a top-level goal
ExpandGoal	Initiates deliberation process for a goal
SearchAvailableGoals	Returns a list of leaf goals
CheckActivationConditions	Checks if the activation conditions of the goal are met
UpdateGoal	Updates properties of the goal
DropGoal	Drops the goal from the goal tree

the GOMAS model desires are generated only by messages containing goals, and those messages are expected to be sent from operators – desires of the operators.

Intentions do not coexist. There can be a situation in which many desires exist and context conditions of those desires are met. The GOMAS robots however pick one desire from the desires to pursue. They do not consider multiple intentions at the same time. In typical cases, they even cannot achieve multiple intentions at the same time.

3.5.2 Main Algorithm

The main control loop of the GOMAS robots is illustrated in Figure 3.7. The loop begins by checking if the robot has any intention *I*. If the robot does not have any intention that is being currently pursued, it searches all current desires (from the goal tree through the GOMAS blackboard). If the list of desires D_n is empty – there is no current desire presented to the robot –, then the robot does *noop* (i.e., no operation), because they are goal-oriented agents. If the list contains at least one desire after the search, the robot now can pick one desire from the list to examine if it can make the desire an intention. Making a desire an intention means that the robot wants to attain the desire and context conditions of the desire are met. The next behavior checks if the intention can be attained or it becomes impossible to be attained. This logic is important to prevent the GOMAS robots from blindly pursuing an intention that has already been achieved or will never be achieved in the future. The robot next checks any plan *P* available in hand to achieve the intention. If not, the robot asks the plan library for any suggestion. If the plan library brings no applicable plan for the intention, the robot drops the intention because it now knows that the intention is not possible to be achieved from the robot. If any application plan is suggested from the plan library, the robot begins performing the list of actions *Act* of the plan in a sequence. While performing the actions, the robot evaluates if the last action was performed correctly. Lastly, the

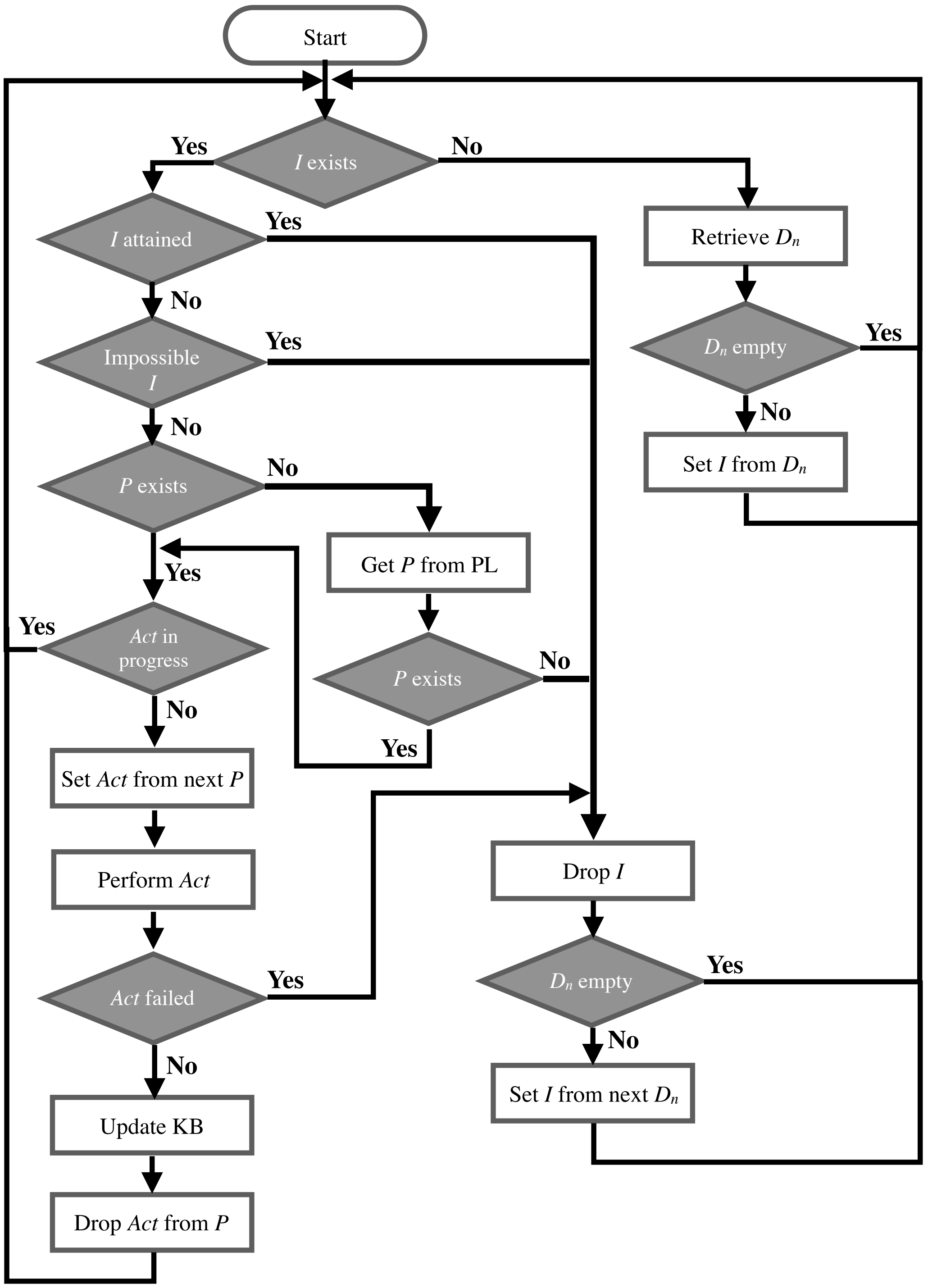

Figure 3.7. A sequential diagram depicting the behavior of the GOMAS robot

robot updates its knowledge base with the new information generated from the last action. And then, it proceeds to the next action of the plan until either there is no action remains or the intention is achieved.

The logic of the GOMAS robot is robust against dynamic changes of the context. While performing actions, the context may block the current intention from being achieved. For example, if all the paths to the destination are currently blocked by obstacles or other robots, the robot may give up its intention toward the goal ‘going to the destination’ and proceed to the next desire if exists. Some may argue that the robots may need to reattempt the plan. In the GOMAS model, the robots do reattempt the plan *after* they considered other goals and come back to the goal again.

3.6 Summary

This chapter provided all the details about the GOMAS model. By employing the BDI principles, the GOMAS robots are able to explore currently perceived desires, coming from the operators, and seek any applicable plan for the desires. The goal deliberation and planning process that utilizes the knowledge base, plan library, and blackboard provides a way to handle the distributed goal planning problem. Moreover, the architecture of the model allows the robots to reconsider their intention as well as plans to improve reliability of the system against dynamically changing environments.

Since the proposed model is elaborated using the abstracted concepts and principles, it is still too general to be applied to applications. The next chapter provides specific applications where the proposed model can be applied to solve the goal deliberation and distributed goal planning problem.

CHAPTER 4. A SIMULATION TOOL FOR MULTI-ROBOT SYSTEMS

This chapter focuses on realization of the GOMAS model, introduced and detailed in the previous chapter, in order to demonstrate its use in possible application domains. The GOMAS system is implemented on the top of the platform, robot operating system (ROS), and actively interacts with StarCraft II application programming interfaces (APIs). Technological backgrounds of those tools are introduced and described to help understand the GOMAS system. Multiple scenarios are later described to illustrate the application domains in detail. Simulation results and discussions of the results are discussed at the last of this chapter.

4.1 Technological Backgrounds

The GOMAS system relies on the tools: ROS and StarCaft II APIs. The system takes many advantages from the ROS platform. The features that are required to support the intelligence of the GOMAS robots are already implemented and supported by the platform. The supported features include message exchanges, handling of asynchronous requests, and running multiple instances (i.e., the GOMAS robots) in parallel. The computer game, StarCraft II, has been updated to reflect requests on balancing between the units. Those updates have made the players come up with different strategies based on situations from time to time. As the balancing changes the units more delicately, the strategies on deciding which unit does what in what circumstances become more complex. This is the situation where the GOMAS system needs to be applied to solve such complex planning problem. The following sub-sections detail the backgrounds of both ROS and StarCraft II.

4.1.1 Robot Operating System as A Platform

Figure 4.1 illustrates the ROS components and their relationships. The ROS core, shown in the figure, is a program that launches instances of a ROS master, a parameter server, and a logging service. Once the ROS core starts running, ROS can run instances of a program using the concept, named ROS node. A ROS node performs all computation of a program and thus acts as the main thread of the program. Every ROS nodes are required to have a stable connection to a server, called ROS master. The ROS master allows the connected ROS nodes to discover each other. Once ROS nodes are connected to the same ROS master, the nodes are able to send messages between them.

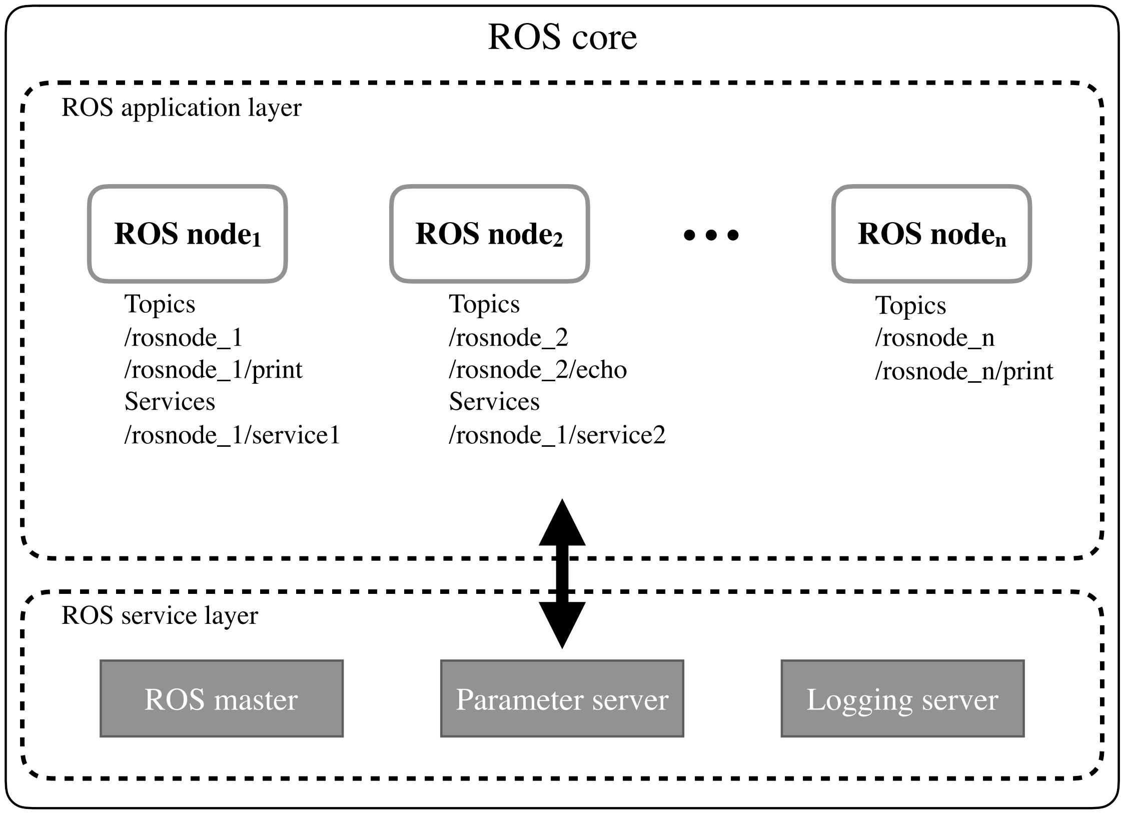

Figure 4.1. Overview of the ROS architecture

The ROS topics provide a method to transfer messages between the ROS nodes over the ROS master. Topics are defined by the ROS nodes as needed. ROS supports the publisher-subscriber mechanism. The ROS topics can be used to publish messages. The ROS nodes use those topics to subscribe any messages that the publisher topics are sending. The rate of publishing and subscribing a message can be adjusted as desired. However, the ROS messages are not meant to transfer a stream of bits (i.e., raw data such as an image or a clip of sound waves). The ROS messages are the best when used in transferring meaningful information (e.g., coordinates of a location, the name of a node, the number of detected pedestrians in a scene, and so on).

A message consists of one or more elements with different data types. The parameter server, shown in Figure 4.1, provides the data types used in ROS to construct a message. Table 4.1 shows the primitive data types supported by ROS. The table only shows the data types actively used in this research study – there are more data types available in ROS. ROS also supports nested data structures such as an array or a custom data structure using the primitive data types (See Table 4.2).

The ROS nodes not only publish and subscribe messages, but also handle requests. The ROS service offers an asynchronous request handling mechanism.

Table 4.1.

Some of the primitive data types in ROS

Primitive type	Description
int8	8-bit signed integer
uint8	8-bit unsigned integer
uint16	16-bit unsigned integer
uint32	32-bit unsigned integer
uint64	64-bit unsigned integer
float32	single-precision floating-point number
float64	double-precision floating-point number
string	an array of characters

Table 4.2.

Nested data types used in this research study

Nested type	Description
int16[]	an array of 16-bit signed integers
geometry_msgs/Point	a nested three double-precision floating-point numbers
sc2_ros/Point2D	a nested two double-precision floating-point numbers

Requests to a ROS service (and also subscriptions to a ROS topic) are handled asynchronously such that the ROS node does not actively handle the requests. The ROS services can be used when an immediate response from the ROS nodes is required.

ROS comes with a simulation, called Gazebo (when installed fully). Gazebo supports importing pre-modelled robots as well as any custom robots. ROS and Gazebo communicate with each other using TCP/IP and UDP/IP protocols. Because both ROS and Gazebo are standalone tools, ROS can be integrated with any other simulation tools, as long as those simulation tools support TCP/IP or UDP/IP protocol. Even if a simulation tool does not support those protocols, a proxy ROS node that supports the communication protocol that the simulation tool supports can bridge between the ROS core and the simulation tool.

4.1.2 A Proxy to StarCraft II

In 2017, Blizzard Entertainment, the maker of StarCraft II, and Google DeepMind team released APIs to control units in StarCraft II (Vinyals et al., 2017). The game accepts connections using the TCP/IP protocol from outside the game. Once a connection to StarCraft II is established, it is possible to send commands to the units in the game and receive information about the game (i.e., information about the environment where the units play upon). The StarCraft II

APIs support Protocol Buffers[1] to send requests and receive responses. Clients that access to the APIs need to install the Protocol Buffers library as well as the StarCraft II API library. All the libraries are available from the research study (Vinyals et al., 2017). The APIs currently support the major operating systems: Mac OS, Windows, and Linux.

There is a certain sequence of state changes required to run a single mode game in StarCraft II. The sequence is shortly described as,

0. *launched* state: This is the state entered when the game is just launched successfully. The game may be waiting for a connection or a connection is just made. To go to the *init_game* state, the request 'create_game' with the required parameters (i.e., a map to play, player race, real-time or discrete time, etc) needs to be sent to the game.

1. *init_game* state: In this state, the game awaits players to join the game. The 'join_game' request starts the game with the map and the player information that are already set in the previous state. After the game started, the state transitions to the *in_game* state.

2. *in_game* state: All in-game requests including actions and queries as well as observations are processed in this state. Requests for actions command the units in the game while requests for queries ask for an answer for the queries. Requests for observations retrieve information about the environment as well as the states of the units at the moment. The observation request is the only way to perceive information about the game from outside. Therefore, the observation request is usually requested repeatedly (e.g., once every second).

3. *ended* state: The game can transition to this state by the two requests: 'quit' or 'save_replay'. The two requests end the current game and in-game requests are no longer accepted by the game. The only difference between the two

[1]A messaging protocol, offered from Google, to serialize data

requests is that 'save_replay' stores all actions and environmental changes happened in the *in_game* state into a file with the 'SC2Replay' extension. This file can later be replayed to give further analysis on the actions and situations of the game.

The response of an observation request provides in-depth information about the game. Figure 4.2 shows the information through the player's screen. The information includes (but, not limited),

- states of the units (e.g., their position, orientation, remaining health, etc) appearing in the game; units in the fog of war (FOW) do not report their state as they are not perceivable from the player's perspective, nor the allied unit's perspective
- actions that the units (not in FOW) are currently performing
- resources that are shared among the allied units

There are about 56 different types of units in StarCraft II. The units are multi-capable and usually have counter units. The non-worker units cannot build structures and only structures can train new units. The number of units per player is limited to 200. Each unit is identified by a unique 64-bit integer number.

4.2 Implementation of The GOMAS Model

This section details the implementation of the GOMAS model using ROS and the StarCraft II APIs. The GOMAS system consists of two ROS packages: ros_sc2 and ros_gomas. The packages run independently on the same ROS master. The overall architecture of the GOMAS system is depicted in the Figure 4.3.

The ros_sc2 package serves as a proxy to an instance of StarCraft II. The SC2 node primarily delivers requests and responses of the requests between the Robot nodes to the instance of StarCraft II. The SC2 request and SC2 response

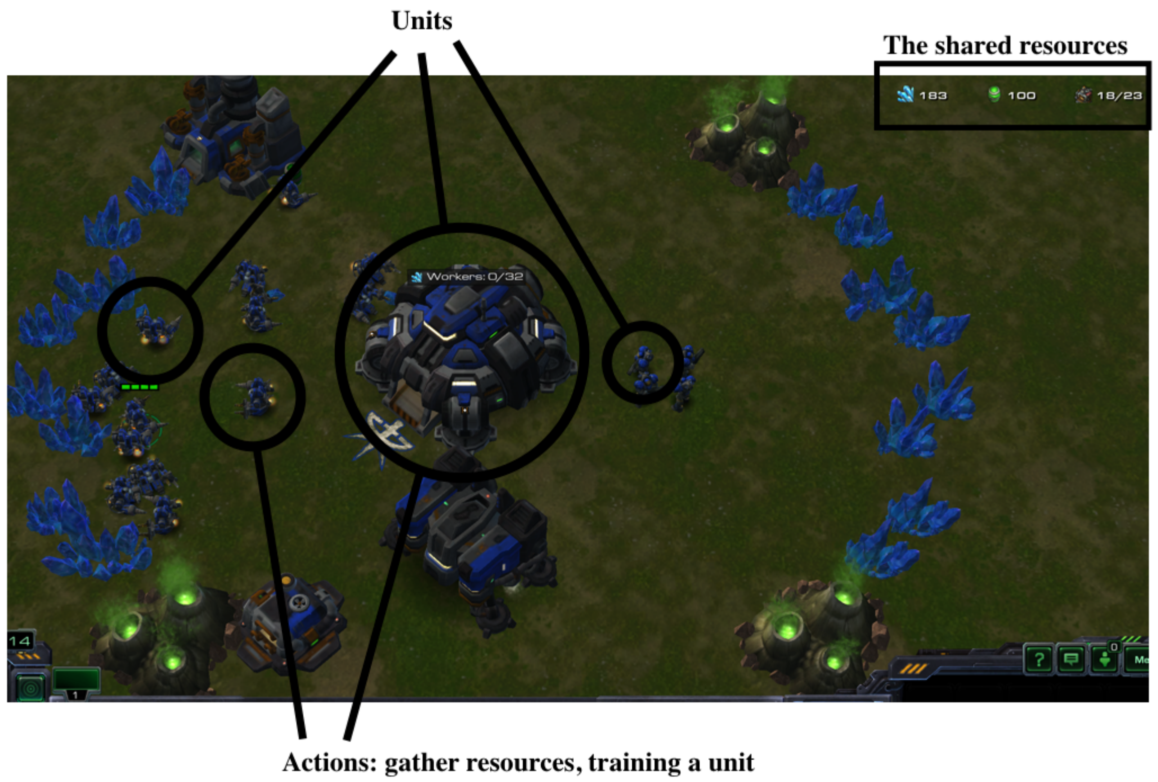

Figure 4.2. A StarCraft II screenshot illustrating states and actions of the units, and the shared resources

modules in the ros_sc2 service layer embed a parser for the SC2 messaging protocol, defined in Protocol Buffers. The modules utilize the parser to respectively support serialization and deserialization of requests and responses. The SC2 utilities provide a set of functions that both the SC2 node and robot nodes can use to understand the meaning of responses received from StarCraft II. The SC2 node on the other hand takes an action to retrieve information from the instance of StarCraft II and update any connected robot nodes for their real-time perception. This action is taken repeatedly, by default it occurs every second. Table 4.3 and table 4.4 respectively list the topics and services that the SC2 node provides to all robot nodes connected to the same ROS master.

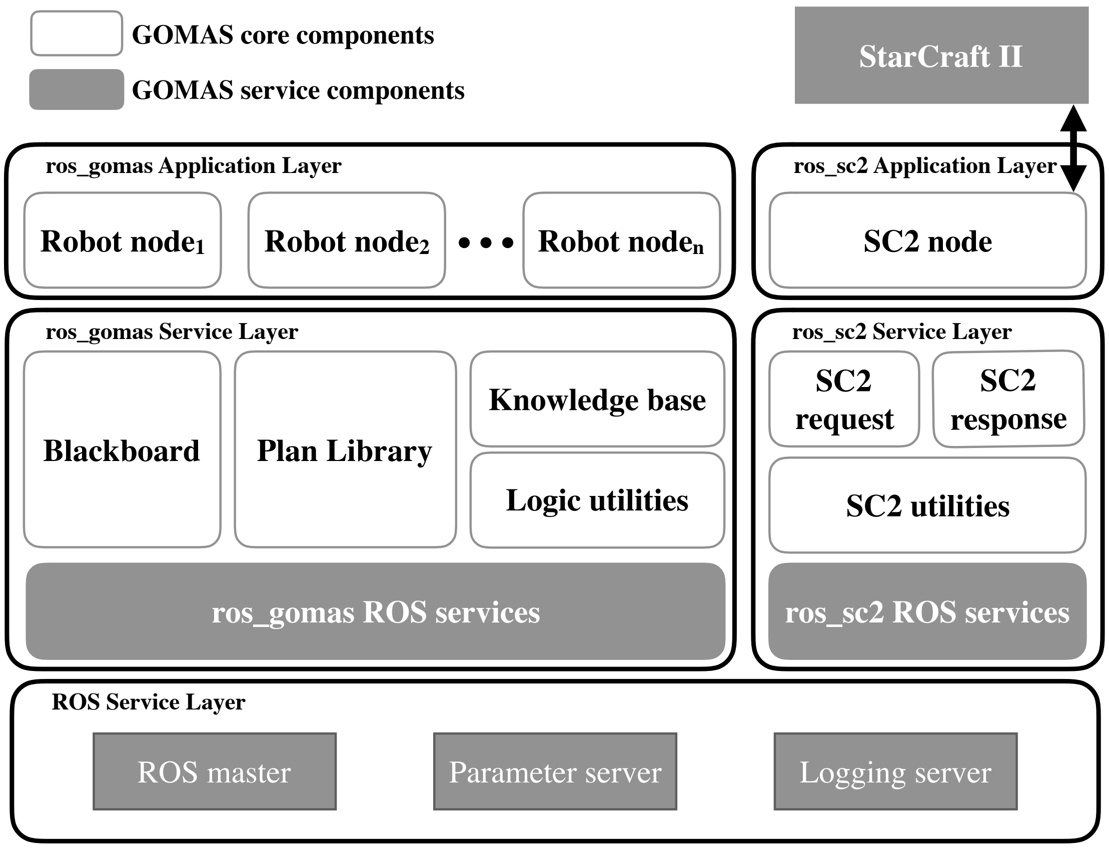

Figure 4.3. Architecture of the GOMAS system.

The ros_gomas package encompasses the core components of the GOMAS model: blackboard, knowledge base, and plan library. When a robot node is spawned (to control a unit in StarCraft II), the node instantiates instances of the

Table 4.3.

The ROS topics supported from the SC2 node

Topic name	Description
sc2/resource	publishes information of the shared resources in the game
sc2/x/y/z	publishes information of y type of the robot z that belongs to the team x

Table 4.4.

The ROS services supported from the SC2 node

Service name	Description
sc2/set_state	changes the state of the game
sc2/request	requests an action to the units in the game
sc2/query	queries information from the game
sc2/log	prints out messages in the game

core components to deliberate and plan goals. Each node should control only one unit in StarCraft II and the unit must exist in the game. Structures such as 'CommandCenter' or 'Barracks' are considered as a unit as they are also agents capable of bringing actions, e.g. training a unit).

The robot nodes constantly subscribe the topics that the SC2 node publishes. Those topics carry the environmental information of the game. The information is stored in the knowledge base of the nodes and changes existing beliefs in the knowledge base. Whenever the robot nodes need to control their unit for an action, they request and query through the SC2 services. The action requests are performed as soon as the requests arrive in the game. The SC2 node is able to handle multiple service requests from multiple nodes in parallel. The sc2/log service is used for debugging purpose.

Operators or system designers are able to define actions and rules. Those pre-defined actions and rules are stored in the plan library and knowledge base, respectively. The actions and rules should logically sound and complete since the robot nodes rely heavily on them for the deliberation and planning process. This means that the rules should not,

- have a path that circulates in the rules; for example, $A \Rightarrow B$, $B \Rightarrow C$, and $C \Rightarrow A$
- have any belief that is not entailed by the inference procedure, and

- make things up; that should not infer any belief that should not exist.

Russell and Norvig (2016) stated for the inference procedure that "*if KB is true in the real world, then any sentence α derived from KB by a sound inference procedure is also true in the real world*". This emphasizes why beliefs, including the pre-defined rules, in the knowledge base should fairly reflect the real world (i.e., the environment).

4.3 Demonstrations

This section describes the target scenarios in StarCraft II to demonstrate the GOMAS system. There are three scenarios considered in this research study. The first sub-section details the context of the scenarios. Then, all the three scenarios run in each of the following sub-sections.

4.3.1 The Context of The Target Scenarios

Multiple heterogeneous robots working as a team serve any given goals from an operator. The robots are types of 'SCV', 'CommandCenter', 'Barracks', and 'Marine'. The robots are shown in Figure 4.4. Some of the robots may already exist at the beginning of the simulation, some of them may be created during the demonstration. The capabilities of the robots are *Move*, *Build*, *Attack*, *Repair*, and *Train*. To support those capabilities in the GOMAS system, plans are defined and stored in the plan library. Each of the plans consists of a helper function for preparation and a sequence of actions that are actually used in StarCraft II. The helper function determines whether the plan is applicable with the current context. For example, if there is no path going to the destination, the *Move* plan for going to the destination is not suggested by the plan library. The defined plans are further described in Appendix B.

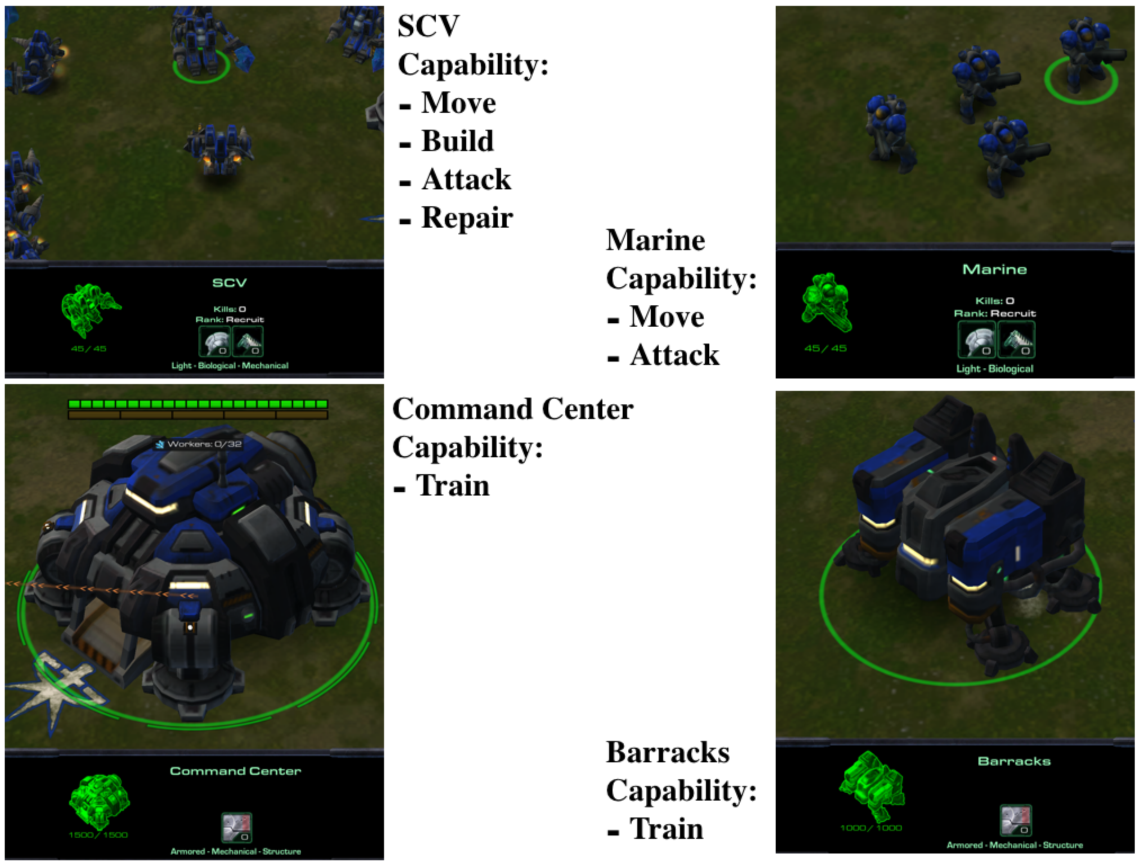

Figure 4.4. The StarCraft II units used in the demonstration

In each of the scenarios, an operator provides the robots a set of goals. Each of the robots expands the set of goals to reason what needs to do to attain the goals. The rules that are used in this context are listed as follows,

R1 : Trained(x, TERRAN_SCV) ⇒ TERRAN_SCV(y)

R2 : Built(x, TERRAN_BARRACKS, px, py) ⇒ TERRAN_BARRACKS(y)

R3 : Attack(x, y) ⇒ Attacked(x, y)

R4 : Located(x, y) ⇒ Scouted(x, y)

R5 : CloseTo(x, y) ⇒ Located(x, y)

R6 : Built(x, TERRAN_SUPPLYDEPOT) ⇒ EnoughFood(TERRAN_MARINE)

R7 : TERRAN_SCV(x) ∧ EnoughMineral(TERRAN_SUPPLYDEPOT) ∧ Build(x, TERRAN_SUPPLYDEPOT, px, py) ⇒ Built(x, TERRAN_SUPPLYDEPOT, px, py)

R8 : EnoughMineral(TERRAN_BARRACKS) ∧ TERRAN_SCV(x) ∧ Build(x, TERRAN_BARRACKS) ⇒ Built(x, TERRAN_BARRACKS, px, py)

R9 : TERRAN_COMMANDCENTER(x) ∧ EnoughMineral(TERRAN_SCV) ∧ Train(x, TERRAN_SCV) ⇒ Trained(x, TERRAN_SCV)

R10 : TERRAN_BARRACKS(x) ∧ EnoughMineral(TERRAN_MARINE) ∧ EnoughFood(TERRAN_MARINE) ∧ Train(x, TERRAN_MARINE) ⇒ Trained(x, TERRAN_MARINE)

The lower case symbols in the parentheses such as x, y, px, and py are propositional variables, whereas the upper case symbols are propositional constants.

Time-dependent beliefs need to be thoroughly generated. As described early in the section 3.3.1, in the GOMAS system variables make the time-dependent beliefs whenever the deliberation process occurs. The variables used in this context are listed as follows,

- mineral: the amount of Minerals shared across the robots
- food_cap: the amount of the total supplies
- food_used: the amount of supplies occupied by the existing units
- loc_x: the current position of a unit in x axis
- loc_y: the current position of a unit in y axis
- target_x: the position of the target in x axis
- target_y: the position of the target in y axis
- training_object: the object being trained

- training_progress: the progress of the training for the training object
- my_state: the current state of a unit
- my_last_state: the last state of a unit
- building_object: the object being constructed

The relationships between the variables and corresponding time-dependent beliefs are shown in Table 4.5. The symbol 'abs' in the table represents a mathematical function that calculates absolute value of the result from the calculation in the parentheses.

Since the process of inference sequentially searches all the beliefs in the knowledge base, the beliefs that are likely to be used more frequently are better to

Table 4.5.

The time-dependent beliefs and the variables used in the demonstration

Generated literal	Condition
EnoughMineral (TERRAN_SCV)	mineral $\geq$ the required amount of Minerals for a single SCV
EnoughMineral (TERRAN_MARINE)	mineral $\geq$ the required amount of Minerals for a Marine
EnoughMineral (TERRAN_SUPPLYDEPOT)	mineral $\geq$ the required amount of Minerals for a Supplydepot
EnoughMineral (TERRAN_BARRACKS)	mineral $\geq$ the required amount of Minerals for a Barracks
EnoughFood (TERRAN_SCV)	(food_cap - food_used) $\geq$ the required amount of food for a single SCV
EnoughFood (TERRAN_MARINE)	(food_cap - food_used) $\geq$ the required amount of food for a Marine
CloseTo (target_x, target_y)	abs(loc_x - target_x) ≤ 1 $\wedge$ abs(loc_y - target_y) ≤ 1
Train (x, training_object)	training_object exists $\wedge$ training_progress > 0.1
Build (x, building_object, px, py)	my_last_state = building_object $\wedge$ my_state = INVALID

be placed at front in the knowledge base. Even though the goal deliberation process visits every belief in the knowledge base, it is still better to place the beliefs (i.e., propositional symbols that do not contain an implication, $\Rightarrow$) at front, in order to reduce complexity of the search.

Once the deliberation process is finished, the robots look for their capabilities to see if they can attain any sub-goals of the goals. Table 4.6 shows the relationships between goals and plans in this context. With the helper function of the plans, the robots skip goals if they do not have the required capability for the goals. If they have, they take courses of actions from the plan to attain the goal. This behavior continues until there is no desires (i.e., sub-goals and goals).

4.3.2 Scenario I: Scout Areas

Scouting areas to identify enemy units is one of the basic strategies used in StarCraft II. This requires multiple entities visit multiple places to observe the areas. In this scenario, multiple SCVs that are capable of moving are deployed for the operation. The given goal tree consists of four instances of the goal *Scouted(location_x, location_y)* with different target locations. Figure 4.5 shows the given goals and generated sub-goals for the goals. To expand the given goals

Table 4.6.

The set of goals and plans in the plan library

Goal	Applicable plan
Train	PlanTrain
EnoughMineral	PlanGather
EnoughFood	PlanBuild
Build	PlanBuild
ClosedTo	PlanMove

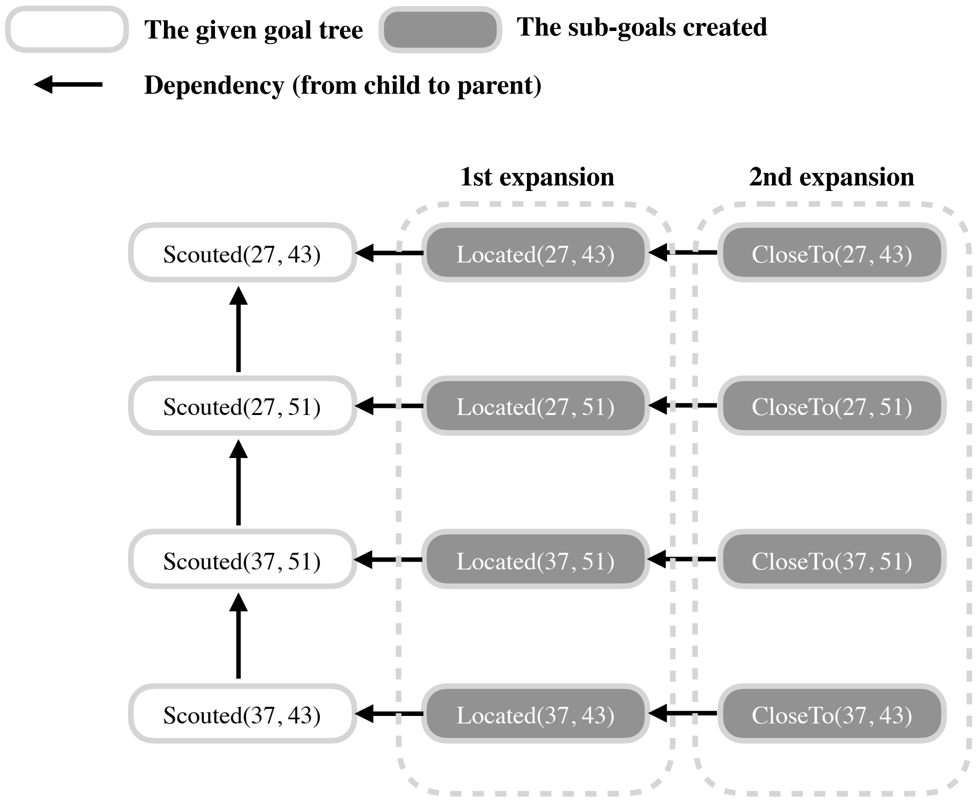

Figure 4.5. The expanded goal tree from the given goal in the scenario I

(illustrated as solid circle-shaped goals in Figure 4.5), the rules **R4** and **R5** are used in the deliberation process.

The snapshots of the demonstration in the Scenario I are shown in Figure 4.6, 4.7, 4.8, 4.9, and 4.10 in the chronological order. Since none of the sub-goals created from the deliberation process have been attained, the SCVs take the leaf goals 'CloseTo' first. As defined in Table 4.6, their plan library suggests them the plan 'PlanMove' to attain the leaf goals. Later, the leaf goals are marked as attained, as they reach to the points using the plan 'PlanMove'. As shown in Figrure 4.8, the first two SCVs that arrive at the right bottom point attain the leaf goal 'CloseTo(37, 43)', which also satisfies attainments of the parent goals 'Located(37, 43)' and 'Scouted(37, 43)'. When the two SCVs start the goal planning process again, they know that the other two sub-goals, going to the top points, are being pursued by the other two SCVs. Therefore, the goal planning process picks the sub-goal ‘CloseTo(27, 43)‘, going to the left bottom point, for the next available goal. Figure 4.9 illustrates this process. However, the other two SCVs located at the two top points conclude that there is no goals that need to be pursued at the moment, and thus become *noop* state – doing nothing. In Figure 4.10, all the given goals are finally attained by the four SCVs.

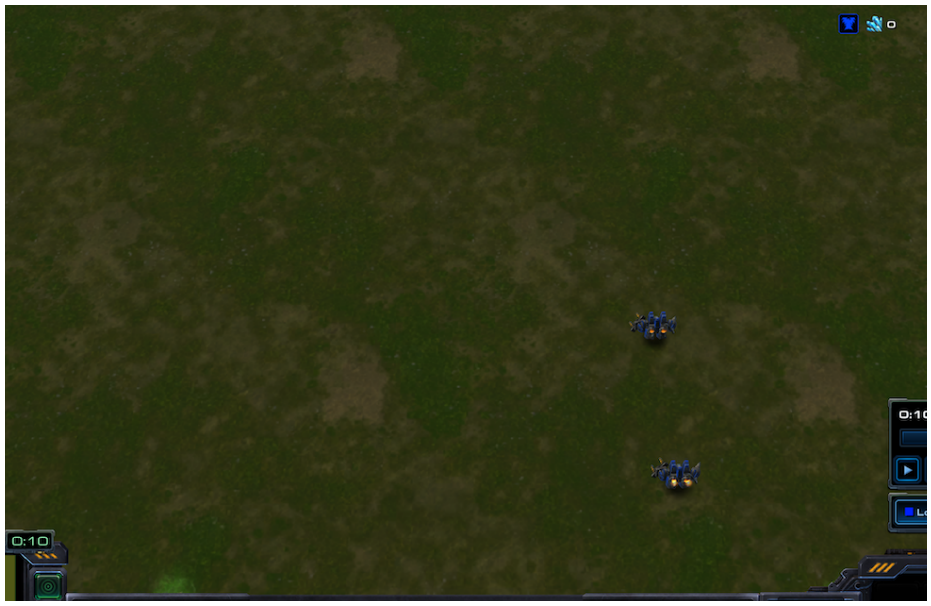

Figure 4.6. The snapshot of the demonstration for the scenario I at 0:10

The detailed movements of the SCVs are depicted through Figure 4.11, 4.12, 4.13, and 4.14. The figures show that each SCV tries to reach the coordinate to attain the goal ‘CloseTo’. For example, SCV II moves and stops at the coordinate (37, 43) from 3 to 9 seconds after the simulation starts and does the same for the coordinate (27, 43) approximately from 12 to 17 seconds. It is observed that the SCV II and IV are pursuing the same sub-goal ‘CloseTo(37, 43)’, which should be pursued by only one robot. The reason the two SCVs pursue the same sub-goal is because the four SCVs are spawned sequentially such that later SCV could not

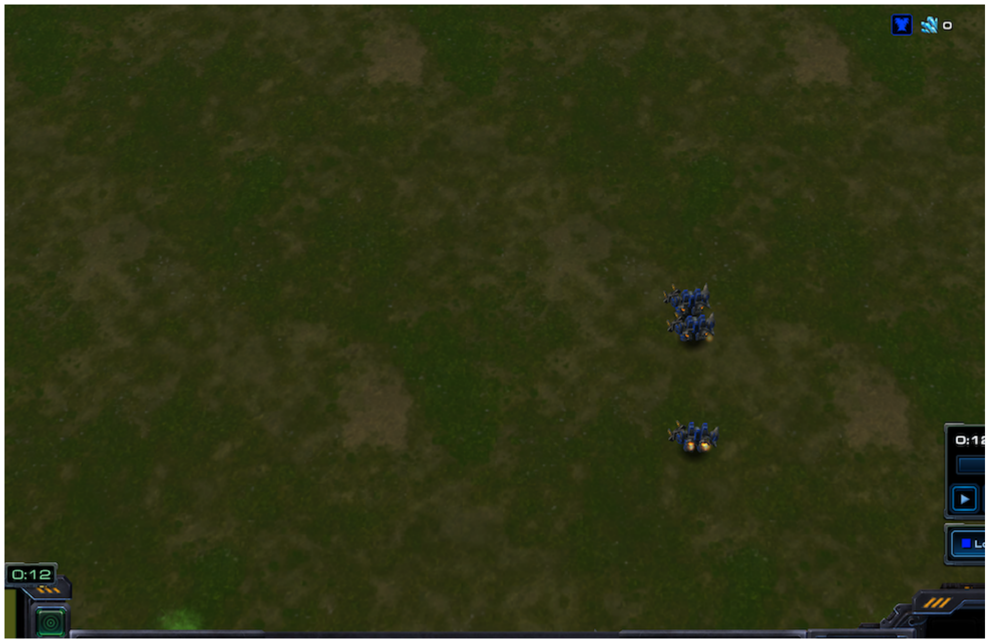

Figure 4.7. The snapshot of the demonstration for the scenario I at 0:12

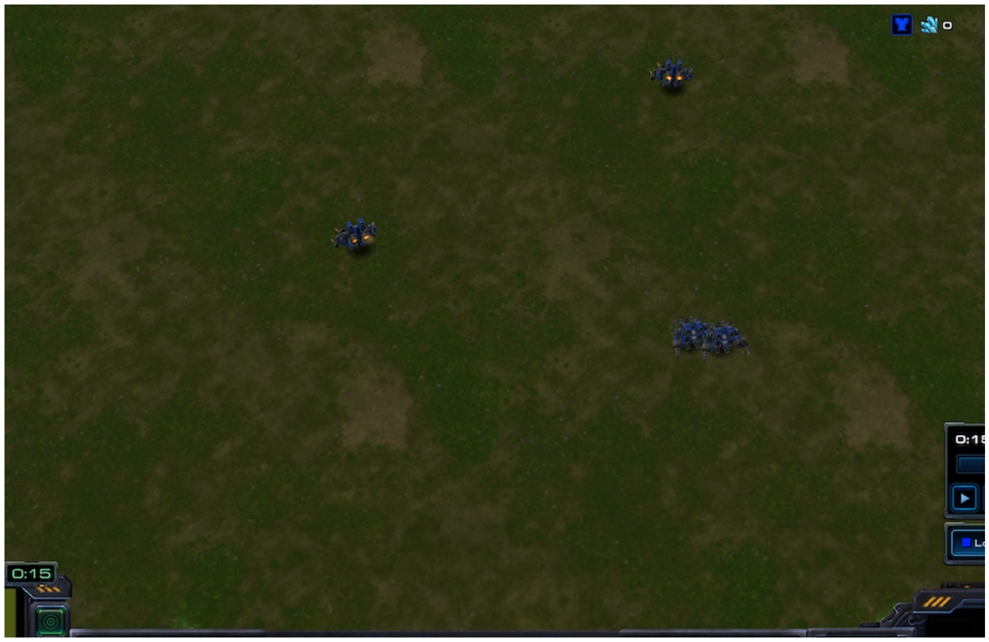

Figure 4.8. The snapshot of the demonstration for the scenario I at 0:15

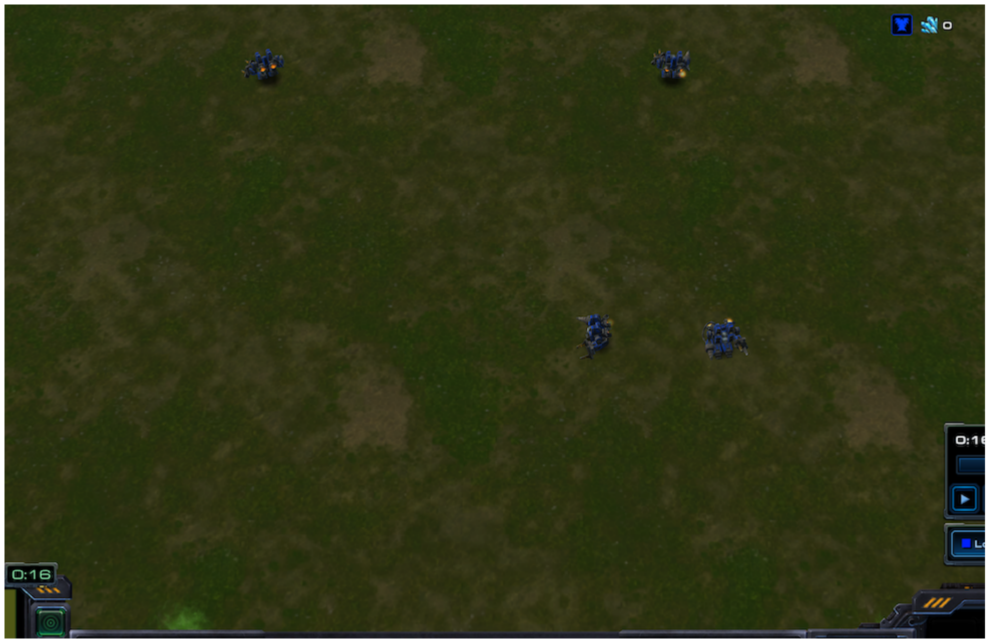

Figure 4.9. The snapshot of the demonstration for the scenario I at 0:16

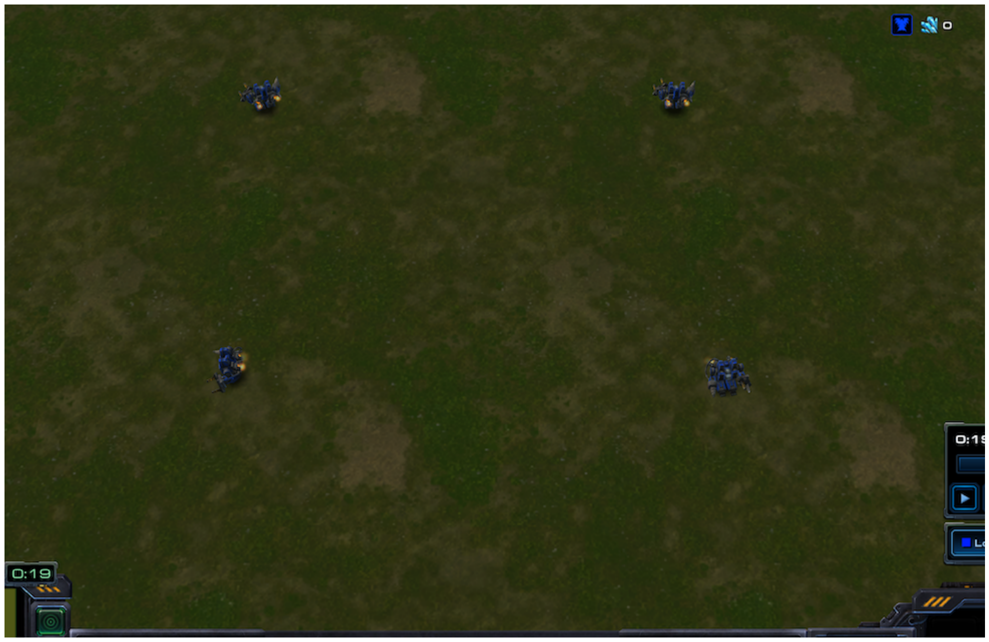

Figure 4.10. The snapshot of the demonstration for the scenario I at 0:19

notice the progress of the sub-goal that the former SCV updated. This will not occur if robots get the goal after they are all spawned (i.g., ready for operation).

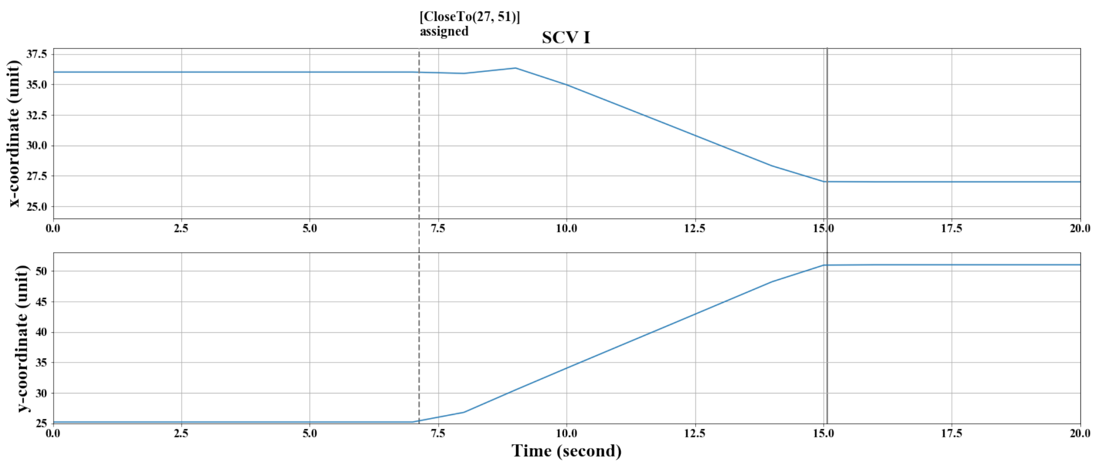

Figure 4.11. Movement of the SCV I. Dashed line indicates the time that the goal is pursued and solid line indicates attainment of the goal.

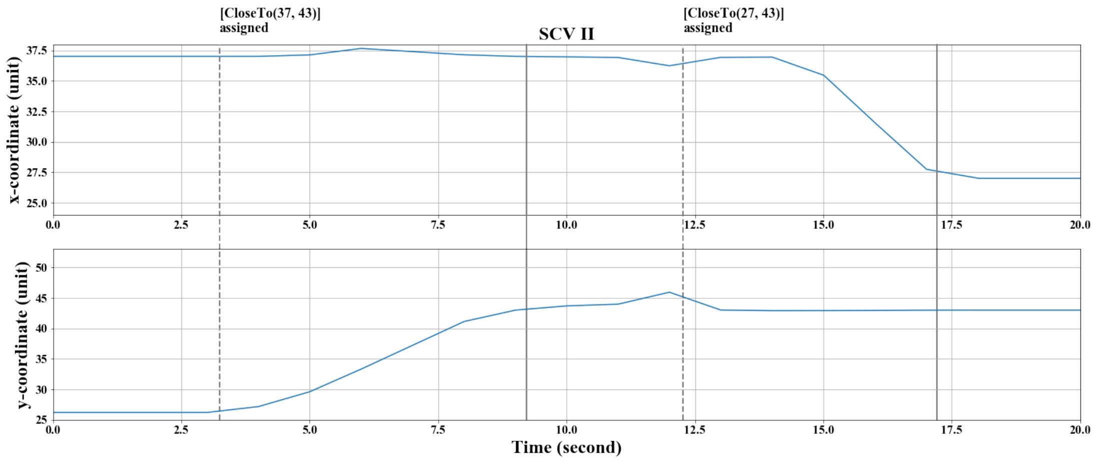

Figure 4.12. Movement of the SCV II. Dashed line indicates the time that the goal is pursued and solid line indicates attainment of the goal.

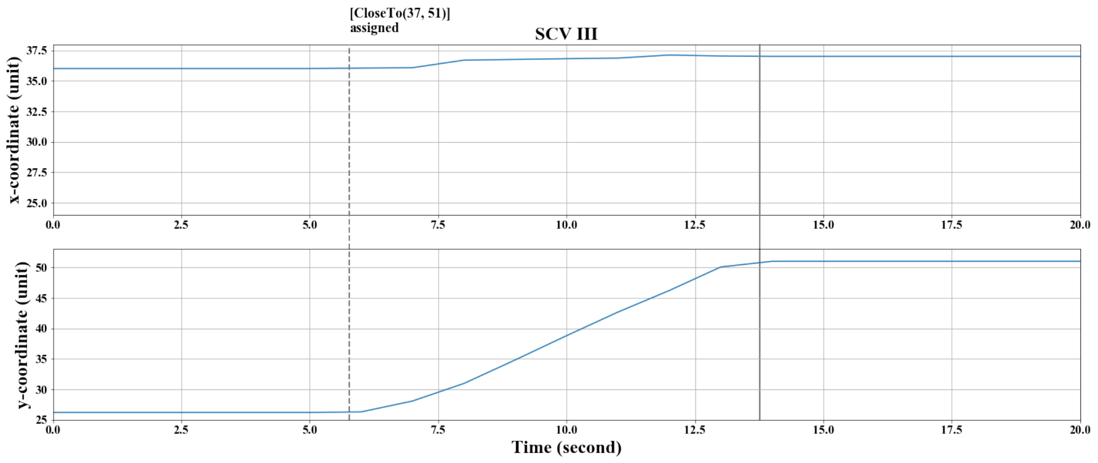

Figure 4.13. Movement of the SCV III. Dashed line indicates the time that the goal is pursued and solid line indicates attainment of the goal.

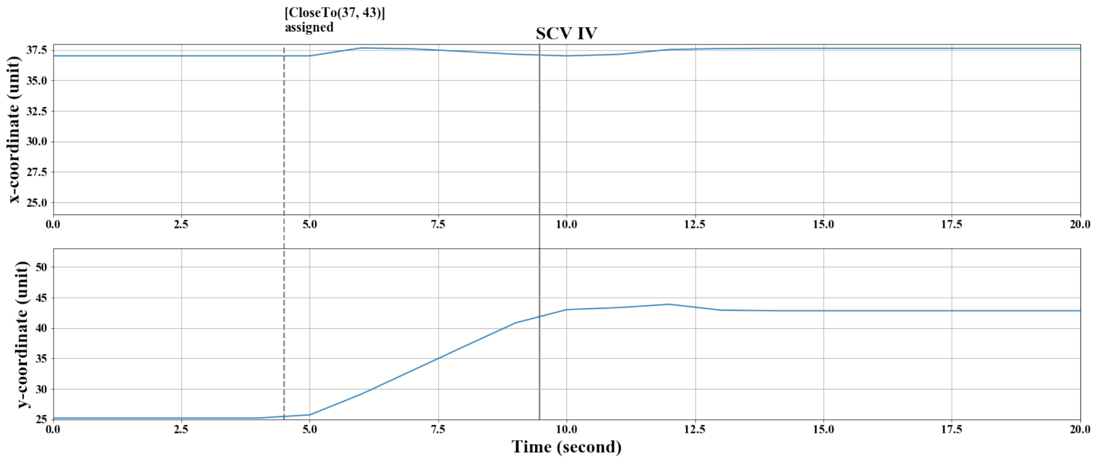

Figure 4.14. Movement of the SCV IV. Dashed line indicates the time that the goal is pursued and solid line indicates attainment of the goal.

4.3.3 Scenario II: Draw GG

The good game, ‘GG’, means the sign for the end of a match. This is usually used in a chat in order for players to express the good impression received from the opponent throughout the match. ‘GG’ has also been used by players to show off the almost certain victory when the player is overwhelming the opponent. In this situation, the player intentionally wastes the resources by constructing structures sequentially to illustrate GG-shape in the game.

In this scenario, multiple SCVs build structures at the designated locations to draw 'GG'. The operator constructs the goal tree to specify the order of the constructions. 20 structures are needed to draw 'GG' – each 'G' shape requires 10 structures. When the goal tree is given to the SCVs, they starts expanding the goal tree to see what to do. According to the rule **R7**, each construction, 'Built(x, TERRAN_SUPPLYDEPOT, px, py)', requires a set of sub-goals:

'TERRAN_SCV(x)'

'EnoughMineral(TERRAN_SUPPLYDEPOT)'

'Build(x, TERRAN_SUPPLYDEPOT, px, py)'

The fully expanded goal tree is partially shown in Figure 4.15. The figure does not show the entire goal tree in a full scale since there are too many goals to show – 20 goals that are given and 60 sub-goals that are generated by the SCVs. The figure however should easily be interpreted without the full scale of the goal tree as each goal is expanded using the same rule (i.e., each expanded goal has the same set of sub-goals with different parameters).

The demonstration is illustrated through Figure 4.16, 4.17, 4.18, 4.19, 4.20, and 4.21. The sub-goal 'TERRAN_SCV(x)' is already satisfied by the SCVs because the belief that satisfies the sub-goal already exists in their knowledge base. An interpretation of this in English would be '*the goal requires a worker and the robots are the worker*'. The sub-goal 'EnoughMineral (TERRAN_SUPPLYDEPOT)' however requires the SCVs to gather resources from the Minerals nearby because the SCVs start with no Minerals. The SCVs plan 'PlanGather' to gather the required amount of Minerals. When the SCVs gather 100 units of Minerals, 'EnoughMineral (TERRAN_SUPPLYDEPOT)' is attained. Next, the sub-goal 'Build(x, TERRAN_SUPPLYDEPOT, px, py)' requires a single SCV to construct the structure at the location of (px, py). Figure 4.22 supports the process of the goal planning and pursuing. The figure shows only the events between 250 seconds and 350 seconds after the simulation starts because the same pattern of the events happends over the entire simulation. The SCVs gather the resource to meet the sub-goal ‘EnoughMineral’ and construct whenever they see the sufficient amount of the resource is gathered.

Note that the sub-goal of gathering Minerals can be pursued by multiple SCVs whereas only one SCV can pursue the sub-goal of constructing a building. This makes the SCVs seem cooperative and distributed because they gather resources together while one of them constructs a structure. Moreover, this level of cooperation is also captured from a situation where a single SCV constructs a

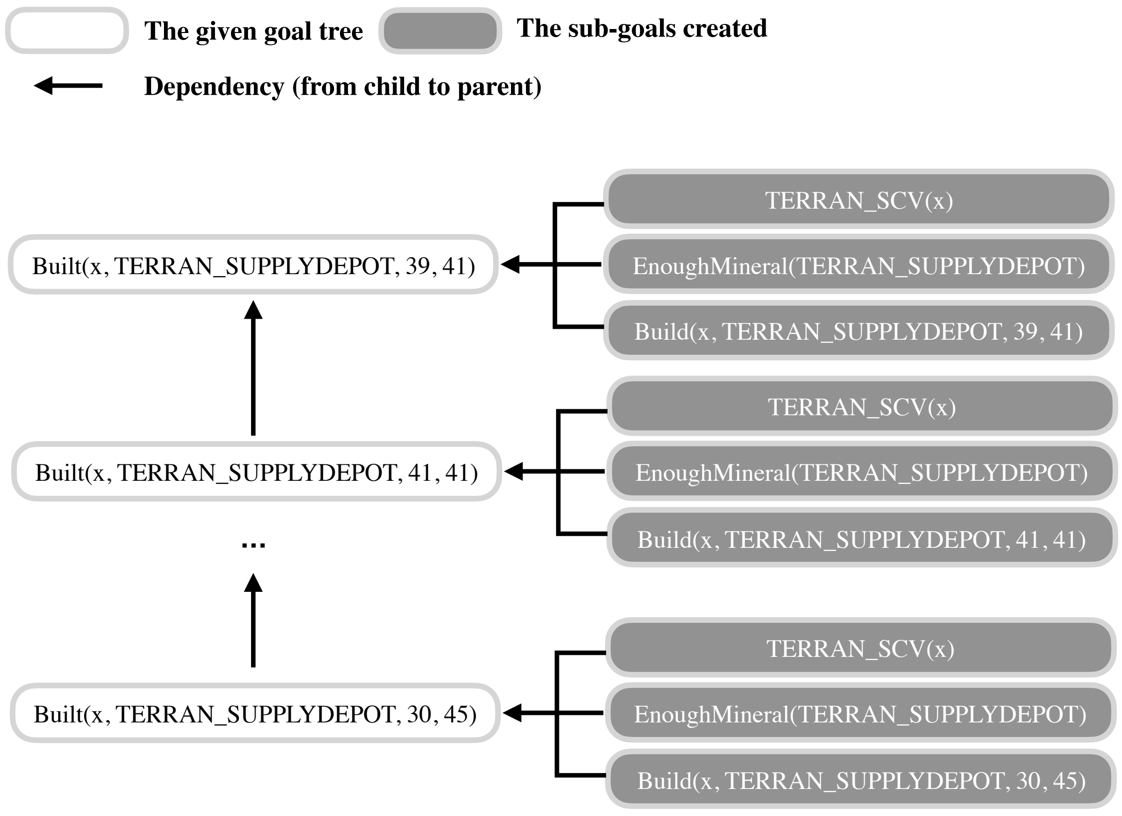
The given goal tree
The sub-goals created
Dependency (from child to parent)
Built(x, TERRAN_SUPPLYDEPOT, 39, 41)
TERRAN_SCV(x)
EnoughMineral(TERRAN_SUPPLYDEPOT)
Build(x, TERRAN_SUPPLYDEPOT, 39, 41)
Built(x, TERRAN_SUPPLYDEPOT, 41, 41)
TERRAN_SCV(x)
EnoughMineral(TERRAN_SUPPLYDEPOT)
Build(x, TERRAN_SUPPLYDEPOT, 41, 41)
...
Built(x, TERRAN_SUPPLYDEPOT, 30, 45)
TERRAN_SCV(x)
EnoughMineral(TERRAN_SUPPLYDEPOT)
Build(x, TERRAN_SUPPLYDEPOT, 30, 45)

Figure 4.15. The expanded goal tree from the given goal in the scenario II

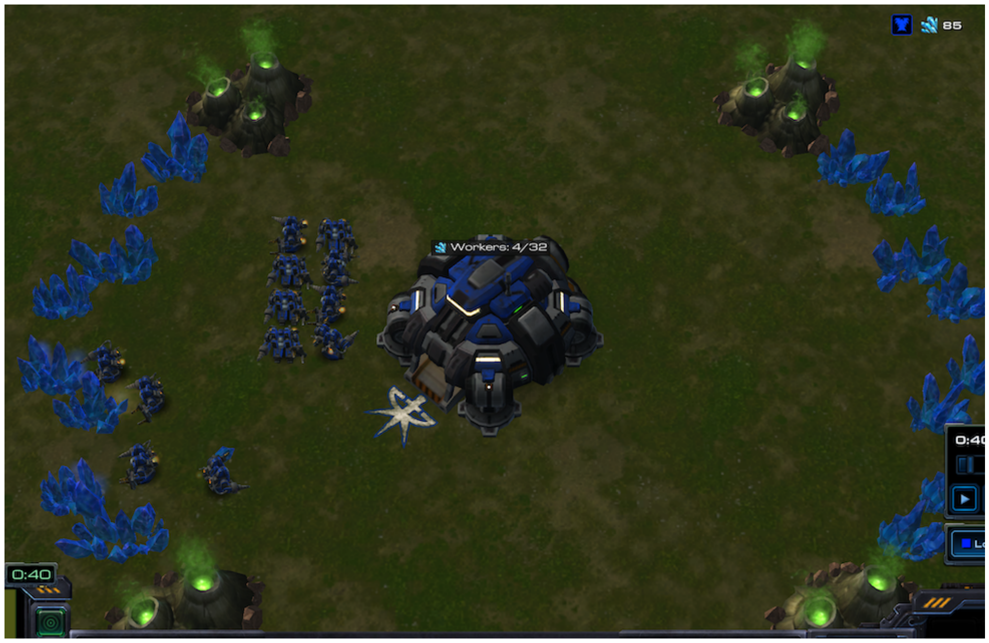

Figure 4.16. The snapshot of the demonstration for the scenario II at 0:40

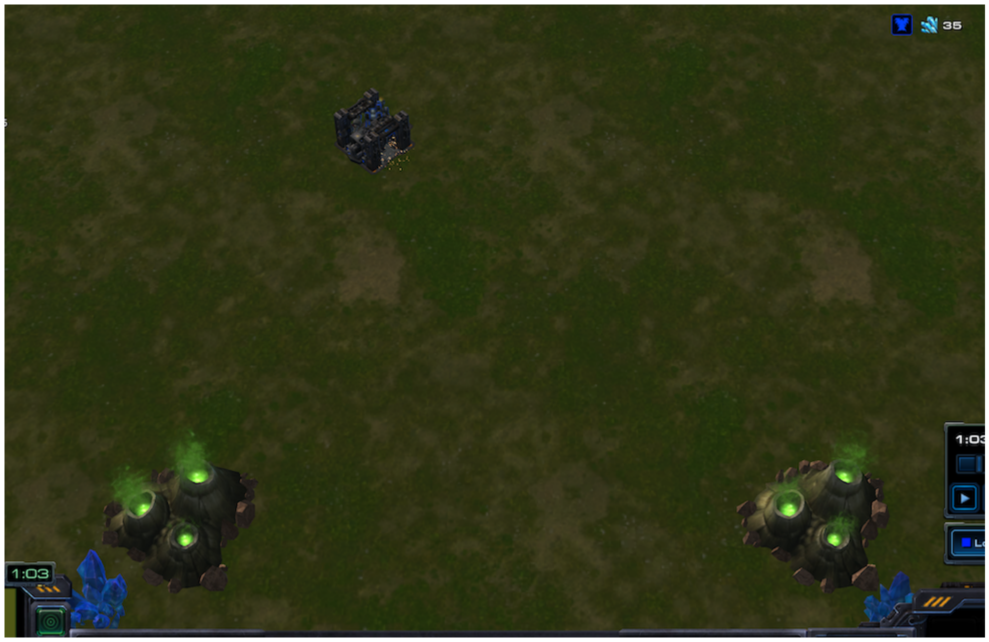

Figure 4.17. The snapshot of the demonstration for the scenario II at 1:03

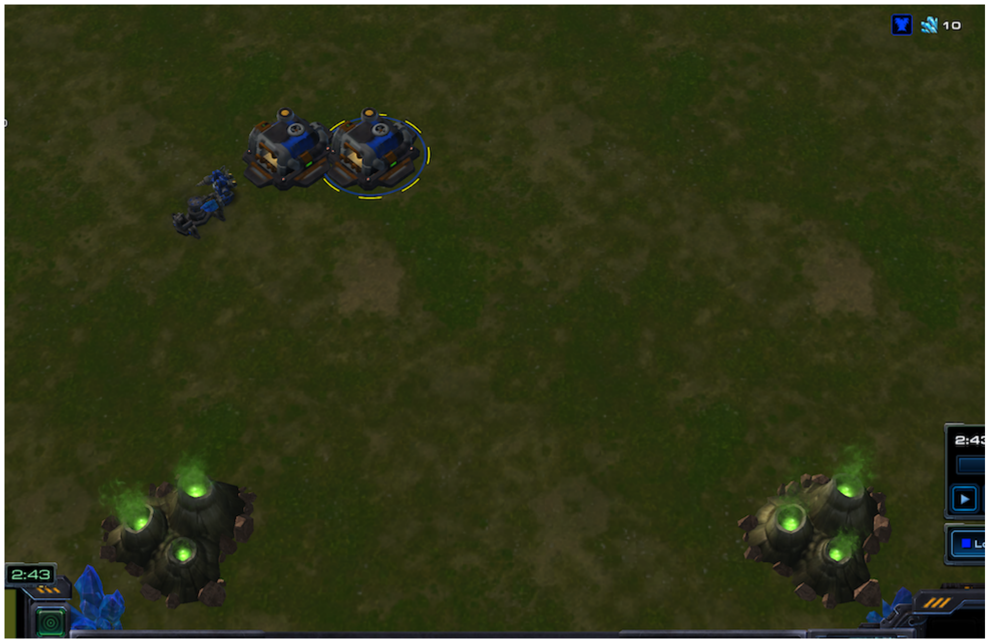

Figure 4.18. The snapshot of the demonstration for the scenario II at 2:43

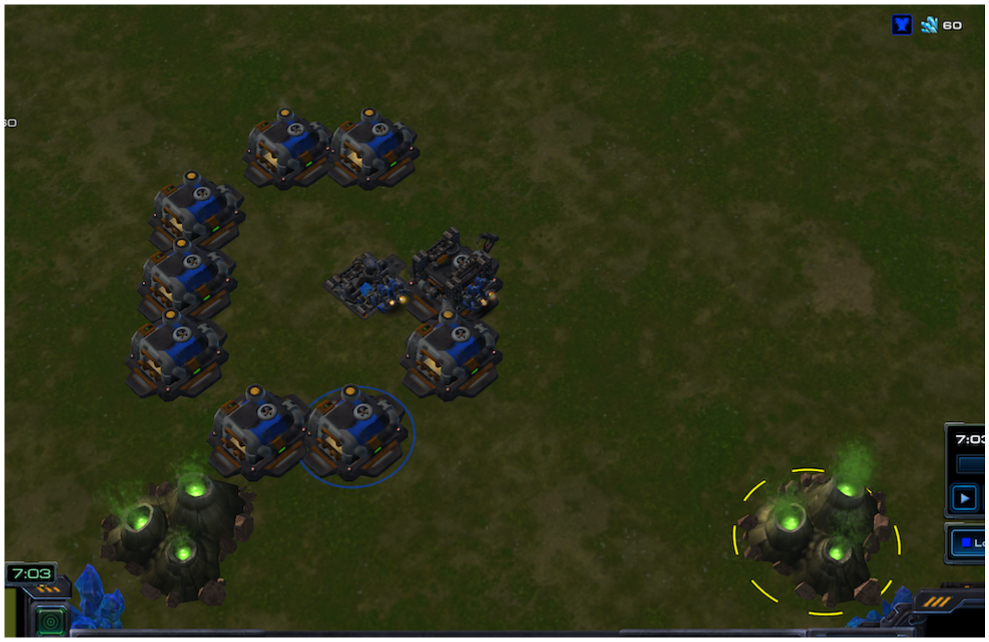

Figure 4.19. The snapshot of the demonstration for the scenario II at 7:03

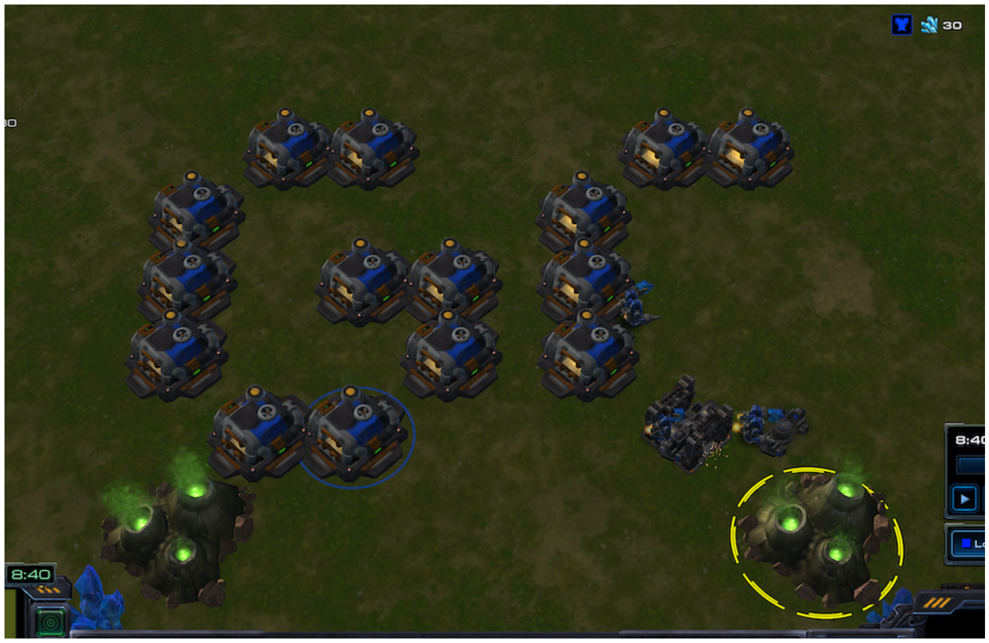

Figure 4.20. The snapshot of the demonstration for the scenario II at 8:40

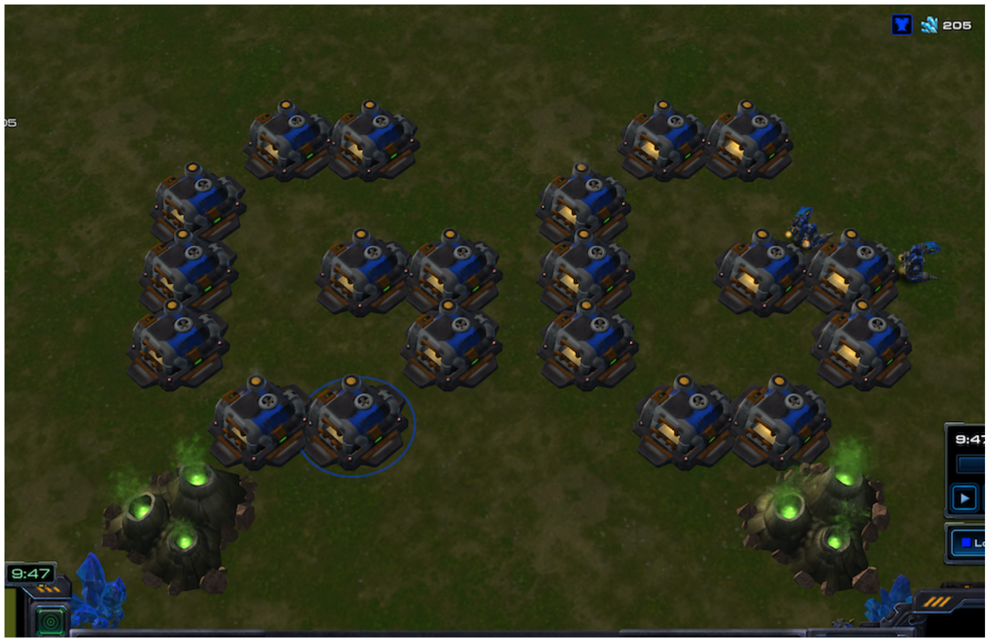

Figure 4.21. The snapshot of the demonstration for the scenario II at 9:47

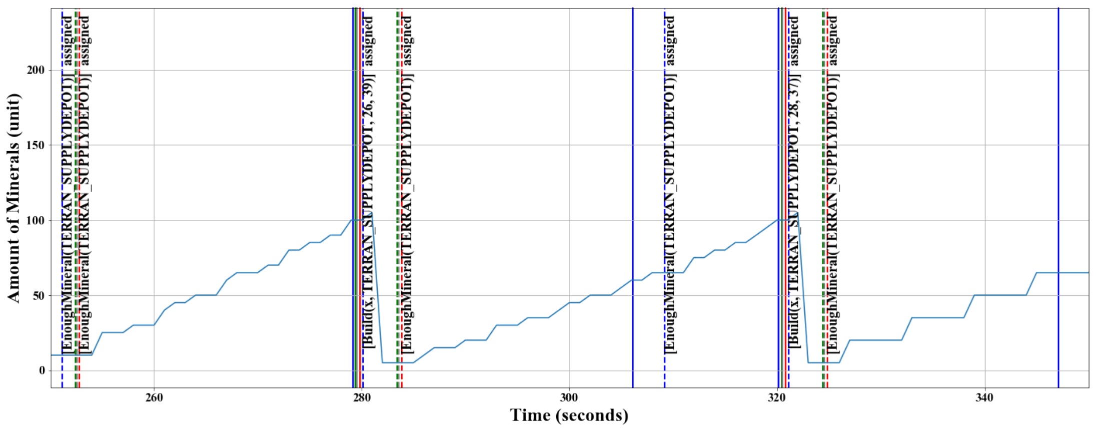

Figure 4.22. A fraction of the simulation that shows how the SCVs act and how the resource is spent. Dashed line indicates the time that the goal is pursued and solid line indicates attainment of the goal. Different color indicates different SCV.

structure at (x_1, y_1) while another SCV starts constructing another structure at (x_2, y_2). This indicates that deploying higher number of SCVs can improve efficiency in time and the level of collaboration.

4.3.4 Scenario III: Train Units

In StarCraft II, gathered resources are mostly used to train more new units to enhance the quality of tactical forces, leading to a victory of a match. StarCraft II requires players to follow the order of structures (i.e., tech-orders) to train higher-tier units. For example, Barracks, is required to train the unit, Marine. The Battlecruiser, one of the most advanced units in StarCraft II, requires much more complex order of constructions of structures – Barracks → Factory → Starport → Tech lab (an add-on to the Starport) → Fusion Core.

In this scenario, the SCVs are given a goal to train a Marine. The SCVs utilize the rule **R10** to deliberate the goal. According to the rule, the goal is expanded with 4 sub-goals. The first sub-goal 'TERRAN_BARRACKS(x)' cannot be satisfied because there is no Barracks at the beginning of the simulation. The

SCVs therefore further expand the sub-goal with the rule **R2**, in order to construct a Barracks. And then, the SCVs use the rule **R8** to finalize the expansion. The completed goal tree is depicted in Figure 4.23. The Barracks from the goal tree is constructed at a random location since the operator did not specify it (the operator may not consider the location of the structure).

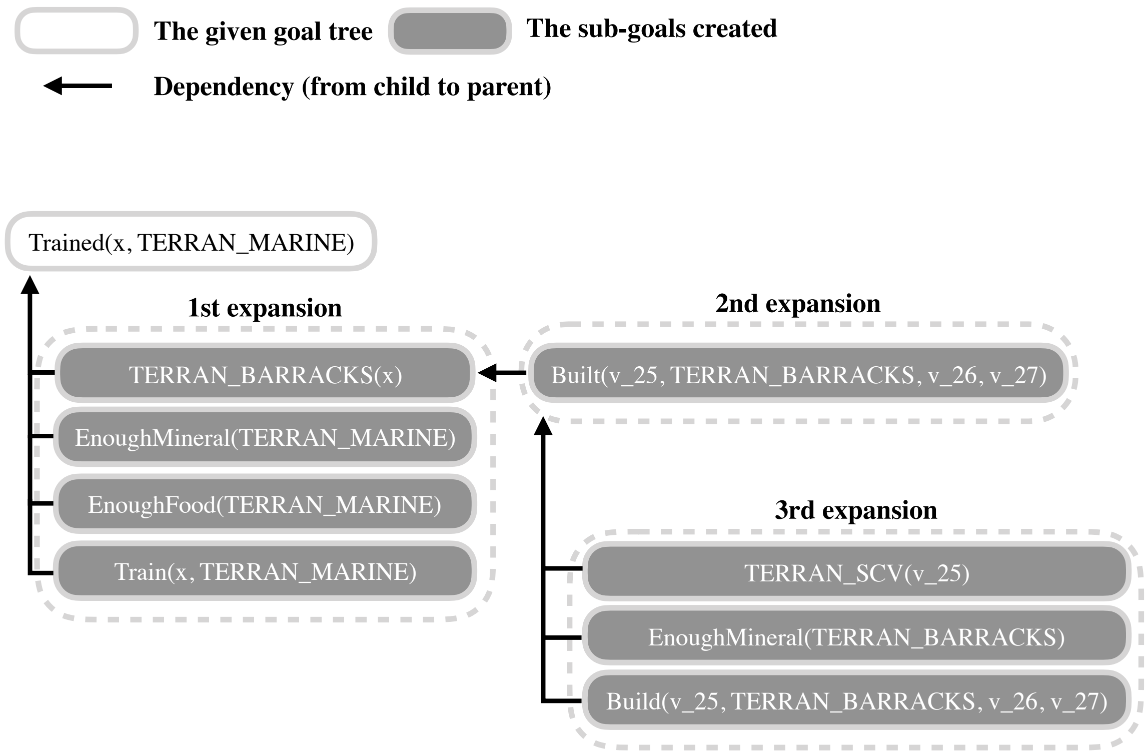
The given goal tree
The sub-goals created
Dependency (from child to parent)
Trained(x, TERRAN_MARINE)
1st expansion
TERRAN_BARRACKS(x)
EnoughMineral(TERRAN_MARINE)
EnoughFood(TERRAN_MARINE)
Train(x, TERRAN_MARINE)
2nd expansion
Built(v_25, TERRAN_BARRACKS, v_26, v_27)
3rd expansion
TERRAN_SCV(v_25)
EnoughMineral(TERRAN_BARRACKS)
Build(v_25, TERRAN_BARRACKS, v_26, v_27)

Figure 4.23. The expanded goal tree from the given goal in the scenario III

Figure 4.24, 4.25, 4.26, 4.27, 4.28, 4.29, 4.30, and 4.31 show the goal planning process in detail. In Figure 4.24, the SCVs try to gather the resource, Minerals, to build a Barracks. When the required amount of Minerals is gathered, one of the SCVs randomly picks a location from its current location and constructs a Barracks to attain the sub-goal of constructing a Brrracks (see Figure 4.26). While the Barracks is being constructed, the rest SCVs start looking at the other sub-goals and decide to pursue the sub-goal ‘EnoughMineral (TERRAN_MARINE)’ (see the moment at around 85 seconds from Figure 4.31 and also see Figure 4.27). The rest of the SCVs attain the sub-goal at around 108 seconds, after they gather the required amount of Minerals for a Marine (see Figure 4.28). Since they already have enough supply for the upcoming Marine, ‘EnoughFood(TERRAN_MARINE)’ is dropped without an action. Note that a structure is built to satisfy the requirement for the SCVs to build a Barracks. The 100 units of Minerals spent at around 38 seconds indicates that the structure is built by the simulation runner (see Figure 4.25). This is a game-specific requirement and is not recognized by the robots – it could have been added as another rule in the knowledge base though. Because the SCVs do not have the ability to train a Marine, they do not intend to pursue the sub-goal ‘Train(x, TERRAN_MARINE)’. It may seem that the SCVs do nothing, but they repeatedly re-visit all the sub-goals that are not yet attained and not assigned to any robot. At this moment, the sub-goal for training a Marine is the only sub-goal remained. The newly built Barracks has the ability such that it intends to pursue the sub-goal, as shown in Figure 4.29. In Figure 4.30, the Barracks attains the sub-goal of training a Marine. With the blackboard sharing, the SCVs and Barracks know that the given goal is finally attained.

4.4 Discussions

The GOMAS system is demonstrated through the three scenarios. The demonstration showed that the deliberation process is one of the keys that makes the robots intelligently reason about goals. The proposed system even granted the robots higher level of reasoning – the robots not only utilize what they currently have, but also create ones that do not exist (e.g., the barracks from the scenario III). This feature is extremely useful when the environment changes severely. For example, existing resources and entities may be wiped out by a giant sand storm

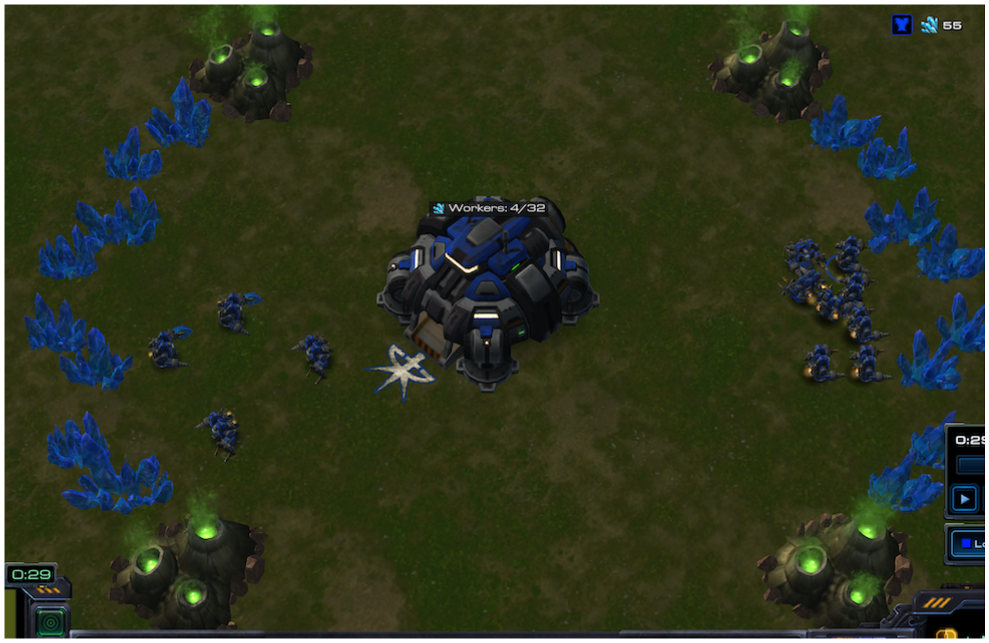

Figure 4.24. The snapshot of the demonstration for the scenario III at 0:29

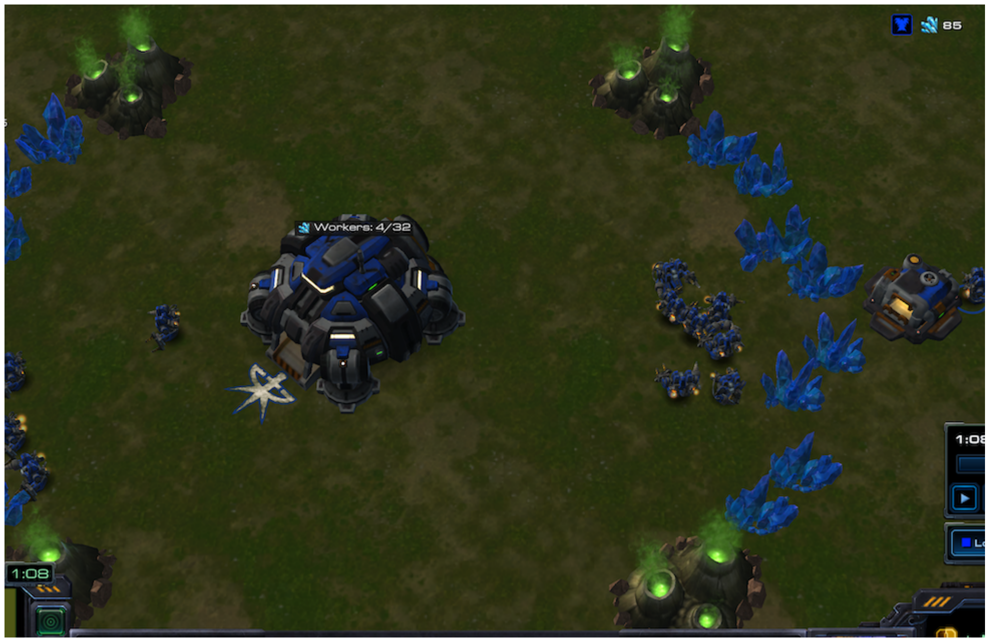

Figure 4.25. The snapshot of the demonstration for the scenario III at 1:08

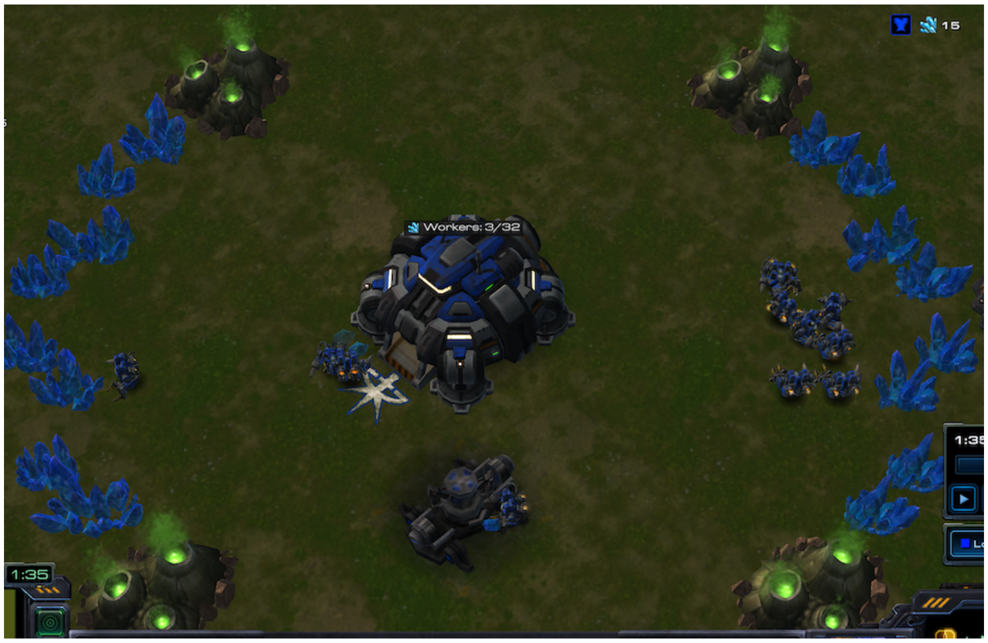

Figure 4.26. The snapshot of the demonstration for the scenario III at 1:35

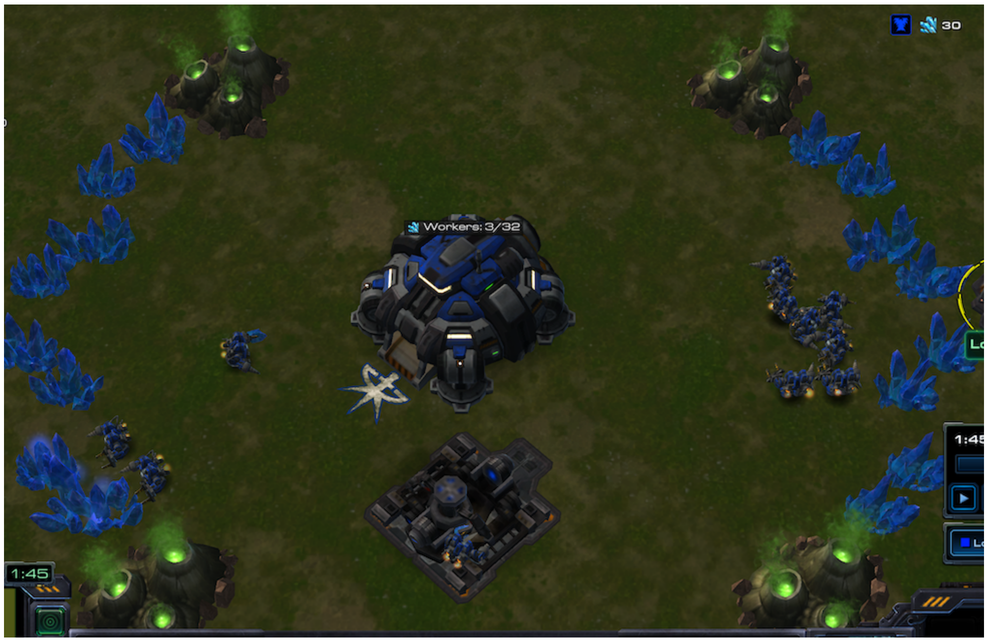

Figure 4.27. The snapshot of the demonstration for the scenario III at 1:45

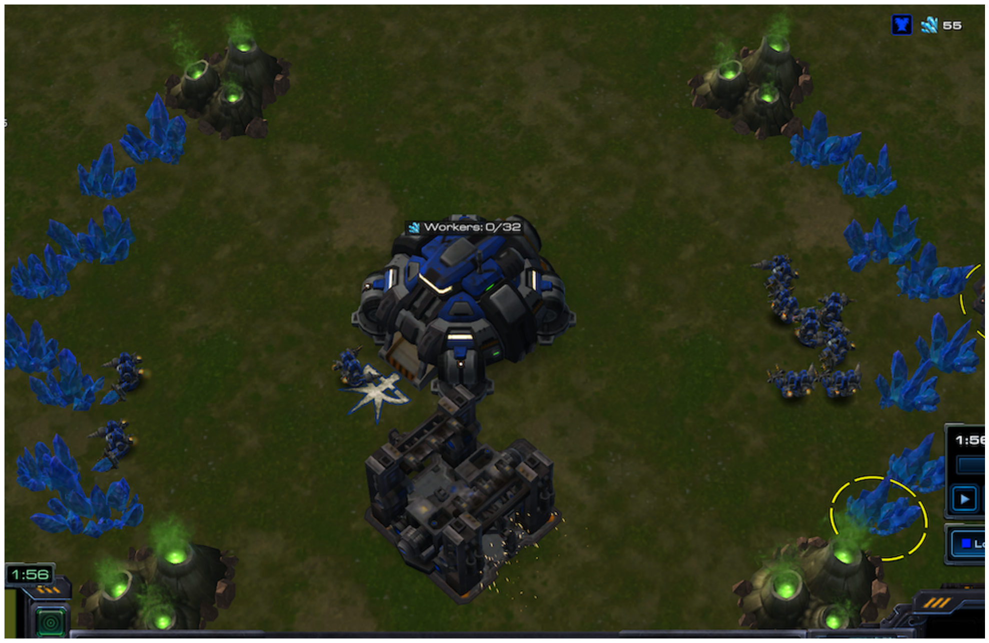

Figure 4.28. The snapshot of the demonstration for the scenario III at 1:56

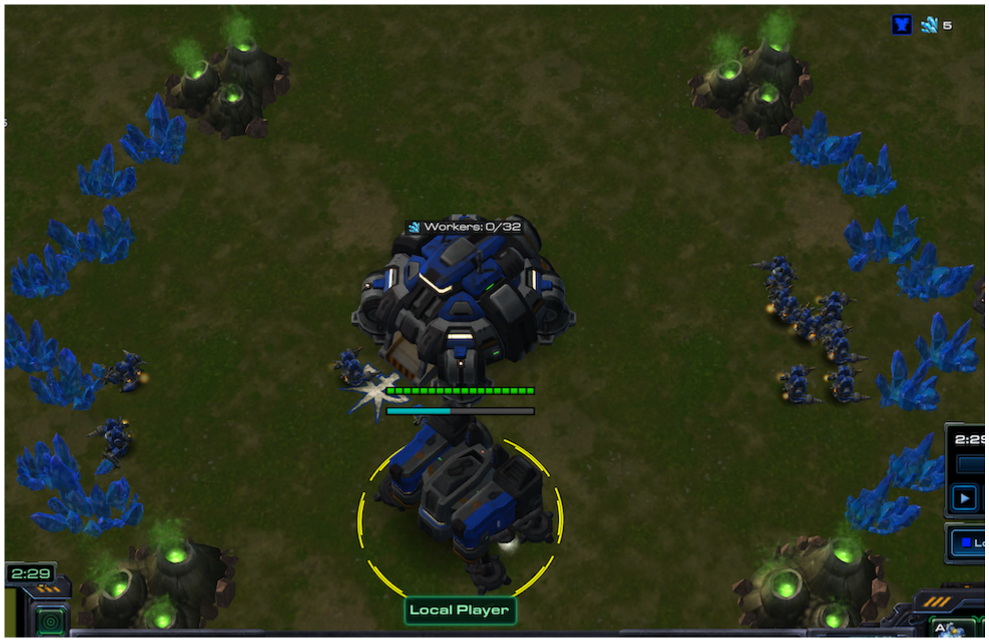

Figure 4.29. The snapshot of the demonstration for the scenario III at 2:29

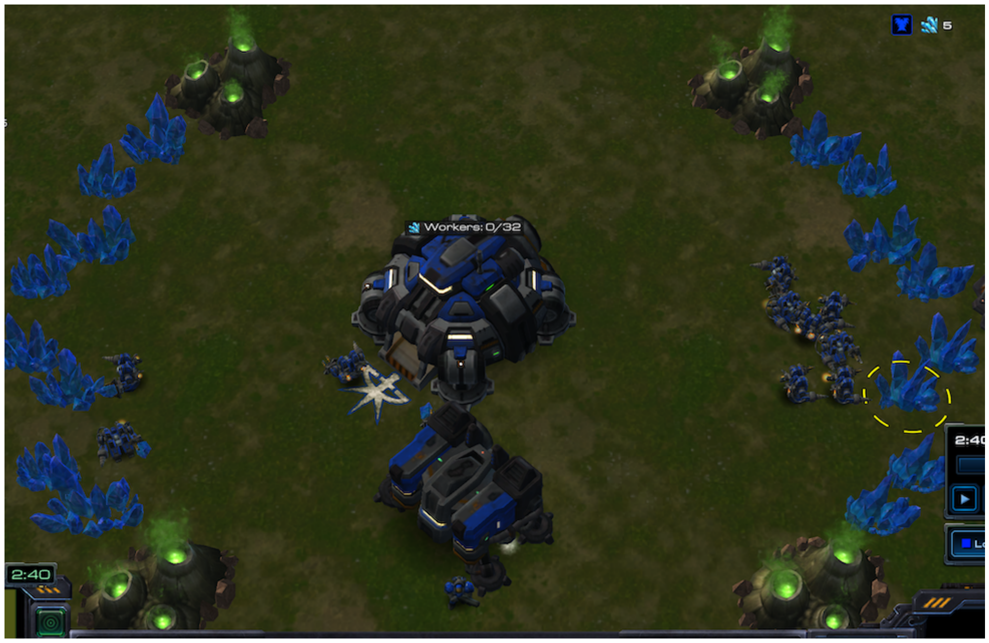

Figure 4.30. The snapshot of the demonstration for the scenario III at 2:40

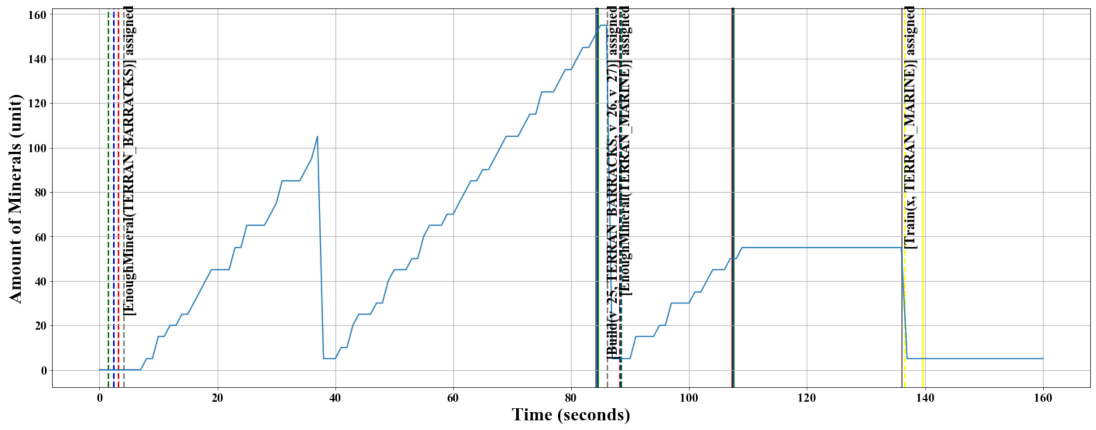

Figure 4.31. The simulation that shows how the SCVs act and how the resource is spent. Dashed line indicates the time that the goal is pursued and solid line indicates attainment of the goal. Different color indicates different SCV.

while exploring Mars. Robots that are able to gather and construct may be able to recover any failure caused by the lost from the giant sand storm.

The goal planning process, the other key, allowed the robots in the scenarios to plan which goals to pursue with regard to what other robots intend to pursue. The distributed blackboard approach makes this process technically possible. With the blackboard, the sub-goal ‘EnoughMineral’ allows multiple robots involved at the same time, whereas the other sub-goals such as ‘Train’ and ‘Build’ require only one robot to participate. This made the robots in the scenarios skip sub-goals that are not attained yet, but occupied by other robots.

One of the points for further discussions is synchronization of their intentions more firmly. A robot_a that intends to attain a goal_1 informs a robot_b of its intention. At the same time, the robot_b also intends to attain the goal_1 and tries to inform the robot_a of its intention, too. If the goal_1 accepts only one robot, then one of them should drop its intention to avoid the conflict of their intention. Robots in the GOMAS system generally follow the process to avoid it. However, there can be a timing issue that may cause the two robots pursuing the goal_1, which they should not. This happens when the two robots finished its goal process and notify the result exactly at the same time. Both robots cannot modify their intention because they already started performing an action for the goal. Moreover, the goal planning process does not support synchronization of goals after the goal deliberation process – the synchronization happens only while deliberating goals.

CHAPTER 5. CONCLUSIONS AND FUTURE WORKS

This chapter concludes this research study and the proposed system that is demonstrated in the previous chapter. The key contributions and challenges are discussed in the conclusion as well. And then, possible applications and future works to improve both proposed model and system are described.

5.1 Conclusions

Multi-agents model and its execution model are not intended to involve any active humans in the loop. M. J. Wooldridge (1992) stated it in his dissertation as,

> 'In contrast, the theory is emphatically not intended to be a model of human social systems' (M. J. Wooldridge, 1992, p. 5).

Instead, such multi-agent models are designed to help tasks that humans feel difficult because of physical or mental risks to them. Those multi-agent models that do not actively rely on human interventions still need a certain level of intelligence to deal with the aforementioned tasks, which usually require the high-level of intelligence. The GOMAS model and its goal deliberation process using first-order predicate logic allow robots to have a limited human-level intelligence such that the robots perform actions similar to what humans would have done. Employing more powerful computation units and an immense size of storage for the robots could make them deal with more complex tasks.

As more intelligent robots working cooperatively, their decisions impact more on other's decision. The goal planning process among the intelligent robots is essential to bring successful operations without conflicts of their actions and intentions. Many interaction methodologies have been proposed to support the process. The blackboard approach is one of the reliable and intuitive methods to model the planning process in cooperative goal-oriented multi-agent systems. The distributed blackboard approach, proposed and implemented in the GOMAS system, make the blackboard approach working even more reliable for distributed multi-agent systems.

With the best knowledge, this research study is the first attempt of integrating ROS with StarCraft II to run multi-robot simulations. Existing simulation tools using StarCraft: Blood War (the previous version of StarCraft II) have successfully supported research studies and practical applications. However, their tools are sparse in terms of the platforms, supportive tools, and messaging protocols that have been utilized in the tools. For example, Šustr, Malŷ, and Čertickŷ (2018) proposed the Docker[1] contained platform using remote control protocols to deploy multiple instances of the simulation. Synnaeve et al. (2016) proposed TorchCraft that utilizes the server-client approach over a messaging protocol, ZeroMQ[2]. Since both ROS and StarCraft II APIs support the well-known standards, this integration bonds them tightly to bring a unified simulation tool. This tool will then serve many applications where multiple heterogeneous robots play actions.

5.2 Future works

Since the topic of this research study touches many relevant fields in DAI, there are possible improvements to apply the model to the broader areas of applications. The future works are described as follows,

- **Speech-Based Goal Generation**: Goals can be generated from human operators by converting sentences spoken from operators into a goal structure. Interpretation of the sentences along with the technology of ontological

[1] A virtualized environment in computer systems to run instances of programs independently from each other

[2] A distributed messaging protocol that carries messages over IPC, TCP, and TIPC protocols

semantics extracts conditions for the goals to be satisfied. This may open a new area of applications in which humans closely interact with the systems.

- **Utility Functions for Goals and Plans**: Selecting goals from the goal tree and plans from the plan library may depend on how efficient the selected goal and plan would be in the current context. Many selection algorithms based on prior experiences, social norms, and individual preferences can be applied to bring the best goal and plan that maximize the utility.

- **Blackboard Synchronization**: Synchronization of distributed blackboards is the key to improve efficiency of collaboration and the key to solve complex problems. The synchronization allows distributed intelligent robots to understand the best of the current context. Synchronization algorithms such as Blockchain technology and fuzzy-based decision making algorithms can enhance reliability of the blackboard approach.

- **Means-End Reasoning**: Efficient goal planning improves performance of intelligent multi-agent systems. The BDI model used in this research study does not fully support the goal planning process. Thorne (2005) mentioned this point in the article as,

  > "BDI logics model only deliberation and not means-end reasoning. This is somehow a handicap, because planning is an essential part of practical reason. Woolridge in [17] remedies to this by introducing action modalities and action operators which lets us define conditional and iterative control structures." (Thorne, 2005)

  It is necessary to propose further mean-end reasoning models and algorithms to capture the missing part.

APPENDICES

CHAPTER A. THE CNF CONVERSION

Russell and Norvig (2016) described the steps to convert sentences into conjunctive normal form (CNF). The objective of the conversion is to simplify sentences such that inference can be possible on complex sentences. The steps are fully described in the section 7.5 in the book (Russell & Norvig, 2016, p. 215). The following steps describe the conversion briefly with an example $P \iff (Q \land R)$,

1. Convert bi-implications to implications
   replace $\iff$ with $\Rightarrow$
   $(P \Rightarrow (Q \land R)) \land ((Q \land R) \Rightarrow P)$

2. Convert implications to their logical equivalents
   eliminate $\Rightarrow$ using $\neg$ and $\lor$
   $(\neg P \lor (Q \land R)) \land (\neg(Q \land R) \lor P)$

3. Move negations inwards
   use the logic rules, double-negation elimination and de Morgan, to have negations only with literals
   $(\neg P \lor (Q \land R)) \land (\neg Q \lor \neg R \lor P)$

4. Distribute disjunction over conjunction
   $(\neg P \lor Q) \land (\neg P \lor R) \land (\neg Q \lor \neg R \lor P)$

   Now, '$(\neg P \lor Q) \land (\neg P \lor R) \land (\neg Q \lor \neg R \lor P)$' is called a form of CNF.

CHAPTER B. THE PLANS USED IN THE DEMONSTRATION

Plans in the GOMAS system are a software component that describes actions. Consequences of the actions change some states in the environment. Plans are defined by system designers and used by robots. In this research study, the plans are static – the contents of the plans never be changed. However, plans can be generated, used, and possibly evaluated by robots if the robots are the type of open-minded agents (Thorne, 2005).

Plans provide two features to support the plan suggestion process as well as the plan execution, described in Section 3.4.2. The features are,

- Preparation: evaluates the conditions for the plan to be proceeded.
- Consequence: tells information about the consequences that the plan brings, no matter if the plan succeeds or fails.

The plans used for the demonstration in Section 4.3 are detailed along with the description of the two features as follows,

PlanMove is to move a robot to a coordinated location in a two dimensional space.

- Preparation: evaluates if the destination is currently reachable from the current location of the robot. This also calculates how far the destination is from the current location.
- Consequence: provides coordination of the destination.

PlanGather is to gather resources.

- Preparation: evaluates source of the target resource and a destination where the resource is returned. Both source and destination need to exist. If the target resource or destination is multiple, the plan selects the closest one from the robot.

- Consequence: provides coordination of the closest target resource.

PlanTrain is to train a unit.

- Preparation: *not applicable.*
- Consequence: provides the type of the unit being trained. The progress of the training can be obtained from the knowledge base in order to determine if the training is done.

PlanBuild is to build a structure.

- Preparation: evaluates if the location for the structure is valid – see if the structure can be built on the location. If the location is not given, the plan randomly selects the building location nearby the robot.
- Consequence: provides the type of the structure and coordinates of the building location. Moreover, the progress of the building can be obtained from the knowledge base.

LIST OF REFERENCES

LIST OF REFERENCES

Aha, D. W., Molineaux, M., & Ponsen, M. (2005). Learning to win: Case-based plan selection in a real-time strategy game. In *International conference on case-based reasoning* (pp. 5–20).

Akbarimajd, A., & Barghi Jond, H. (2014). Multi-robot foraging based on contract net protocol. *Journal of Advances in Computer Research*, *5*(1), 61–67.

Alford, R., Shivashankar, V., Roberts, M., Frank, J., & Aha, D. W. (2016). Hierarchical planning: Relating task and goal decomposition with task sharing. In *Proc. of the intl joint conf. on ai (ijcai). aaai press.*

Allwright, M., Bhalla, N., El-faham, H., Antoun, A., Pinciroli, C., & Dorigo, M. (2014). Srocs: Leveraging stigmergy on a multi-robot construction platform for unknown environments. In *International conference on swarm intelligence* (pp. 158–169).

Antonelli, G., Arrichiello, F., Caccavale, F., & Marino, A. (2014). Decentralized time-varying formation control for multi-robot systems. *The International Journal of Robotics Research*, *33*(7), 1029–1043.

Ayala, I., Amor, M., Fuentes, L., & Troya, J. M. (2015). A software product line process to develop agents for the iot. *Sensors*, *15*(7), 15640–15660.

Binmore, K., Castelfranchi, C., Doran, J., & Wooldridge, M. (1998). Rationality in multi-agent systems. *The Knowledge Engineering Review*, *13*(03), 309–314.

Boman, M. (1999). Norms in artificial decision making. *Artificial Intelligence and Law*, *7*(1), 17–35.

Bonabeau, E., Dorigo, M., & Theraulaz, G. (1999). *Swarm intelligence: from natural to artificial systems* (No. 1). Oxford university press.

Bordini, R. H., & Hübner, J. F. (2005). Bdi agent programming in agentspeak using jason. In *International workshop on computational logic in multi-agent systems* (pp. 143–164).

Bowling, M. H., Jensen, R. M., & Veloso, M. M. (2005). Multiagent planning in the presence of multiple goals. *Planning in Intelligent Systems: Aspects, Motivations and Methods, John Wiley and Sons, Inc.*

Brachman, R. J., Levesque, H. J., & Reiter, R. (1992). *Knowledge representation.* MIT press.

Bratman, M. (1987). *Intention, plans, and practical reason.* Cambridge, MA: Harvard University Press.

Braubach, L., Pokahr, A., Moldt, D., & Lamersdorf, W. (2004). Goal representation for bdi agent systems. In *International workshop on programming multi-agent systems* (pp. 44–65).

Calvaresi, D., Marinoni, M., Sturm, A., Schumacher, M., & Buttazzo, G. (2017). The challenge of real-time multi-agent systems for enabling iot and cps. In *Proceedings of the international conference on web intelligence* (pp. 356–364).

Cao, Y., Yu, W., Ren, W., & Chen, G. (2013). An overview of recent progress in the study of distributed multi-agent coordination. *IEEE Transactions on Industrial informatics*, *9*(1), 427–438.

Carsten, J., Rankin, A., Ferguson, D., & Stentz, A. (2009). Global planning on the mars exploration rovers: Software integration and surface testing. *Journal of Field Robotics*, *26*(4), 337–357.

Castelfranchi, C. (1995). Guarantees for autonomy in cognitive agent architecture. *Intelligent agents*, 56–70.

Chien, S., Barrett, A., Estlin, T., & Rabideau, G. (2000). A comparison of coordinated planning methods for cooperating rovers. In *Proceedings of the fourth international conference on autonomous agents* (pp. 100–101).

Chouard, T. (2016). The go files: Ai computer wraps up 4-1 victory against human champion. *Nature News*.

Churchill, D., Preuss, M., Richoux, F., Synnaeve, G., Uriarte, A., Ontannón, S., & Čertickỳ, M. (2016). Starcraft bots and competitions. *Encyclopedia of Computer Graphics and Games*, 1–18.

Conte, R., & Castelfranchi, C. (1995). *Cognitive and social action*. Psychology Press.

DeLoach, S. A., & Miller, M. (2010). A goal model for adaptive complex systems. *International Journal of Computational Intelligence: Theory and Practice*, *5*(2), 83–92.

Deloach, S. A., Oyenan, W. H., & Matson, E. T. (2008). A capabilities-based model for adaptive organizations. *Autonomous Agents and Multi-Agent Systems*, *16*(1), 13–56.

Dias, M. B., Zlot, R., Kalra, N., & Stentz, A. (2006). Market-based multirobot coordination: A survey and analysis. *Proceedings of the IEEE*, *94*(7), 1257–1270.

do Nascimento, N. M., & de Lucena, C. J. P. (2017). Fiot: An agent-based framework for self-adaptive and self-organizing applications based on the internet of things. *Information Sciences*, *378*, 161–176.

Doran, J. E., Franklin, S., Jennings, N. R., & Norman, T. J. (1997). On cooperation in multi-agent systems. *The Knowledge Engineering Review*, *12*(03), 309–314.

Dorigo, M., & Gambardella, L. M. (1997). Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on evolutionary computation*, *1*(1), 53–66.

Dorigo, M., Maniezzo, V., & Colorni, A. (1996). Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, *26*(1), 29–41.

Durfee, E. H., Lesser, V. R., & Corkill, D. D. (1987). Coherent cooperation among communicating problem solvers. *IEEE Transactions on Computers*, *100*(11), 1275–1291.

Emrich, S., Suslov, S., & Judex, F. (2007). Fully agent based. modellings of epidemic spread using anylogic. In *Proc. eurosim* (pp. 9–13).

Fischer, K., Schillo, M., & Siekmann, J. (2003). Holonic multiagent systems: A foundation for the organisation of multiagent systems. In *International conference on industrial applications of holonic and multi-agent systems* (pp. 71–80).

Gajurel, A., Louis, S. J., Mendez, D. J., & Liu, S. (2018). Neuroevolution for rts micro. *arXiv preprint arXiv:1803.10288*.

Genesereth, M. R., Fikes, R. E., et al. (1992). *Knowledge interchange format-version 3.0: reference manual.* Computer Science Department, Stanford University San Francisco, CA.

Gerkey, B., Vaughan, R. T., & Howard, A. (2003). The player/stage project: Tools for multi-robot and distributed sensor systems. In *Proceedings of the 11th international conference on advanced robotics* (Vol. 1, pp. 317–323).

Guo, X., Singh, S., Lee, H., Lewis, R. L., & Wang, X. (2014). Deep learning for real-time atari game play using offline monte-carlo tree search planning. In *Advances in neural information processing systems* (pp. 3338–3346).

Harland, J., Morley, D. N., Thangarajah, J., & Yorke-Smith, N. (2014). An operational semantics for the goal life-cycle in bdi agents. *Autonomous agents and multi-agent systems*, *28*(4), 682–719.

Hira, D. (2007). *Problems in operation research (principles & solution): Principles and solutions.* S. Chand Publishing.

Hoeing, M., Dasgupta, P., Petrov, P., & O'hara, S. (2007). Auction-based multi-robot task allocation in comstar. In *Proceedings of the 6th international joint conference on autonomous agents and multiagent systems* (p. 280).

Jennings, N. R., & Campos, J. R. (1997). Towards a social level characterisation of socially responsible agents. *IEE Proceedings-Software*, *144*(1), 11–25.

Ji, M., & Egerstedt, M. (2007). Distributed coordination control of multiagent systems while preserving connectedness. *IEEE Transactions on Robotics*, *23*(4), 693–703.

Jiang, Y., Xia, Z., Zhong, Y., & Zhang, S. (2005). An adaptive adjusting mechanism for agent distributed blackboard architecture. *Microprocessors and Microsystems*, *29*(1), 9–20.

Kalenka, S., & Jennings, N. R. (1999). Socially responsible decision making by autonomous agents. In *Cognition, agency and rationality* (pp. 135–149). Springer.

Karaboga, D., & Basturk, B. (2007). A powerful and efficient algorithm for numerical function optimization: artificial bee colony (abc) algorithm. *Journal of global optimization*, *39*(3), 459–471.

Khan, S. M., & Lespérance, Y. (2010). A logical framework for prioritized goal change. In *Proceedings of the 9th international conference on autonomous agents and multiagent systems: volume 1-volume 1* (pp. 283–290).

Koenig, N. P., & Howard, A. (2004). Design and use paradigms for gazebo, an open-source multi-robot simulator. In *Iros* (Vol. 4, pp. 2149–2154).

Kraus, S. (1997). Negotiation and cooperation in multi-agent environments. *Artificial intelligence*, *94*(1-2), 79–97.

Kraus, S., Sycara, K., & Evenchik, A. (1998). Reaching agreements through argumentation: a logical model and implementation. *Artificial Intelligence*, *104*(1-2), 1–69.

Kuriki, Y., & Namerikawa, T. (2014). Consensus-based cooperative formation control with collision avoidance for a multi-uav system. In *American control conference (acc), 2014* (pp. 2077–2082).

Kuwabara, K., Ishida, T., & Osato, N. (1995). Agentalk: Describing multiagent coordination protocols with inheritance. In *Tools with artificial intelligence, 1995. proceedings., seventh international conference on* (pp. 460–465).

Laclavık, M., Balogh, Z., Babık, M., & Hluchỳ, L. (2006). Agentowl: Semantic knowledge model and agent architecture. *Computing and Informatics*, *25*, 421–439.

Li, T., & Zhang, J.-F. (2010). Consensus conditions of multi-agent systems with time-varying topologies and stochastic communication noises. *IEEE Transactions on Automatic Control*, *55*(9), 2043–2057.

Liao, W.-H., Kao, Y., & Fan, C.-M. (2008). Data aggregation in wireless sensor networks using ant colony algorithm. *Journal of Network and Computer Applications*, *31*(4), 387–401.

Liekna, A., Lavendelis, E., & Grabovskis, A. (2012). Experimental analysis of contract net protocol in multi-robot task allocation. *Applied Computer Systems*, *13*(1), 6–14.

Lin, L., & Zheng, Z. (2005). Combinatorial bids based multi-robot task allocation method. In *Robotics and automation, 2005. icra 2005. proceedings of the 2005 ieee international conference on* (pp. 1145–1150).

Liu, W., Gu, W., Sheng, W., Meng, X., Wu, Z., & Chen, W. (2014). Decentralized multi-agent system-based cooperative frequency control for autonomous microgrids with communication constraints. *IEEE Transactions on Sustainable Energy*, *5*(2), 446–456.

Maleković, M., & Čubrilo, M. (1999). Multi-agent systems: Modeling knowledge bases. *Journal of Information and Organizational Sciences*, *23*(2), 103–111.

Mastellone, S., Stipanović, D. M., Graunke, C. R., Intlekofer, K. A., & Spong, M. W. (2008). Formation control and collision avoidance for multi-agent non-holonomic systems: Theory and experiments. *The International Journal of Robotics Research*, *27*(1), 107–126.

Ma'sum, M. A., Arrofi, M. K., Jati, G., Arifin, F., Kurniawan, M. N., Mursanto, P., & Jatmiko, W. (2013). Simulation of intelligent unmanned aerial vehicle (uav) for military surveillance. In *Advanced computer science and information systems (icacsis), 2013 international conference on* (pp. 161–166).

Matson, E. T., Taylor, J., Raskin, V., Min, B.-C., & Wilson, E. C. (2011). A natural language exchange model for enabling human, agent, robot and machine interaction. In *Automation, robotics and applications (icara), 2011 5th international conference on* (pp. 340–345).

Merabet, G. H., Essaaidi, M., Talei, H., Abid, M. R., Khalil, N., Madkour, M., & Benhaddou, D. (2014). Applications of multi-agent systems in smart grids: A survey. In *Multimedia computing and systems (icmcs), 2014 international conference on* (pp. 1088–1094).

Michael, N., Zavlanos, M. M., Kumar, V., & Pappas, G. J. (2008). Distributed multi-robot task assignment and formation control. In *Robotics and automation, 2008. icra 2008. ieee international conference on* (pp. 128–133).

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Mondada, F., Pettinaro, G. C., Guignard, A., Kwee, I. W., Floreano, D., Deneubourg, J.-L., ... Dorigo, M. (2004). Swarm-bot: A new distributed robotic concept. *Autonomous robots*, *17*(2-3), 193–221.

Norling, E., & Sonenberg, L. (2004). Creating interactive characters with bdi agents. In *Proceedings of the australian workshop on interactive entertainment (ie04)* (pp. 69–76).

Nunes, I., & Luck, M. (2014). Softgoal-based plan selection in model-driven bdi agents. In *Proceedings of the 2014 international conference on autonomous agents and multi-agent systems* (pp. 749–756).

O'Hare, G. M., & Jennings, N. (1996). *Foundations of distributed artificial intelligence* (Vol. 9). John Wiley & Sons.

Olfati-Saber, R. (2006). Flocking for multi-agent dynamic systems: Algorithms and theory. *IEEE Transactions on automatic control*, *51*(3), 401–420.

Olfati-Saber, R., Fax, J. A., & Murray, R. M. (2007). Consensus and cooperation in networked multi-agent systems. *Proceedings of the IEEE*, *95*(1), 215–233.

Ontanón, S., Synnaeve, G., Uriarte, A., Richoux, F., Churchill, D., & Preuss, M. (2013). A survey of real-time strategy game ai research and competition in starcraft. *IEEE Transactions on Computational Intelligence and AI in games*, *5*(4), 293–311.

Ontanón, S., Synnaeve, G., Uriarte, A., Richoux, F., Churchill, D., & Preuss, M. (2015). Rts ai problems and techniques. *Encyclopedia of Computer Graphics and Games*, 1–12.

Padgham, L., Scerri, D., Jayatilleke, G., & Hickmott, S. (2011). Integrating bdi reasoning into agent based modeling and simulation. In *Proceedings of the winter simulation conference* (pp. 345–356).

Papadopoulos, P., Jenkins, N., Cipcigan, L. M., Grau, I., & Zabala, E. (2013). Coordination of the charging of electric vehicles using a multi-agent system. *IEEE Transactions on Smart Grid*, *4*(4), 1802–1809.

Park, S., Park, S., Lee, H., Hyun, M., Lee, E., Ahn, J., … Matson, E. (2018). Collaborative goal distribution in distributed multiagent systems. In *2018 second ieee international conference on robotic computing (irc)* (pp. 313–318).

Parker, L. E. (2008). Distributed intelligence: Overview of the field and its application in multi-robot systems. *Journal of Physical Agents*, *2*(1), 5–14.

Parunak, H. V. D. (1997). " go to the ant": Engineering principles from natural multi-agent systems. *Annals of Operations Research*, *75*, 69–101.

Pearl, J. (2014). *Probabilistic reasoning in intelligent systems: networks of plausible inference.* Elsevier.

Pokahr, A., Braubach, L., & Lamersdorf, W. (2005a). A goal deliberation strategy for bdi agent systems. In *German conference on multiagent system technologies* (pp. 82–93).

Pokahr, A., Braubach, L., & Lamersdorf, W. (2005b). Jadex: A bdi reasoning engine. In *Multi-agent programming* (pp. 149–174). Springer.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., … Ng, A. Y. (2009). Ros: an open-source robot operating system. In *Icra workshop on open source software* (Vol. 3, p. 5).

Ram, S., & Ramesh, V. (1995). A blackboard-based cooperative system for schema integration. *IEEE Expert*, *10*(3), 56–62.

Ramchurn, S. D., Mezzetti, C., Giovannucci, A., Rodriguez-Aguilar, J. A., Dash, R. K., & Jennings, N. R. (2009). Trust-based mechanisms for robust and efficient task allocation in the presence of execution uncertainty. *Journal of Artificial Intelligence Research*, *35*(1), 119.

Rao, A. S. (1996). Agentspeak (l): Bdi agents speak out in a logical computable language. In *European workshop on modelling autonomous agents in a multi-agent world* (pp. 42–55).

Rao, A. S., & Georgeff, M. P. (1995). Bdi agents: from theory to practice. In *Icmas* (Vol. 95, pp. 312–319).

Rashid, T., Samvelyan, M., de Witt, C. S., Farquhar, G., Foerster, J., & Whiteson, S. (2018). Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. *arXiv preprint arXiv:1803.11485*.

Ren, W., Beard, R. W., & Atkins, E. M. (2005). A survey of consensus problems in multi-agent coordination. In *American control conference, 2005. proceedings of the 2005* (pp. 1859–1864).

Rodriguez, S., Gaud, N., & Galland, S. (2014). SARL: a general-purpose agent-oriented programming language. In *the 2014 ieee/wic/acm international conference on intelligent agent technology.* Warsaw, Poland: IEEE Computer Society Press.

Russell, S. J., & Norvig, P. (2016). *Artificial intelligence: a modern approach.* Malaysia; Pearson Education Limited,.

Saxe, J. G., & Schwartzott, C. (1994). *The blind men and the elephant.* Carol Schwartzott.

Schlenoff, C., Prestes, E., Madhavan, R., Goncalves, P., Li, H., Balakirsky, S., ... Miguelanez, E. (2012). An ieee standard ontology for robotics and automation. In *Intelligent robots and systems (iros), 2012 ieee/rsj international conference on* (pp. 1337–1342).

Schmittle, M., Lukina, A., Vacek, L., Das, J., Buskirk, C. P., Rees, S., ... Kumar, V. (2018). Openuav: a uav testbed for the cps and robotics community. In *Proceedings of the 9th acm/ieee international conference on cyber-physical systems* (pp. 130–139).

Schwarz, M., Rodehutskors, T., Droeschel, D., Beul, M., Schreiber, M., Araslanov, N., ... others (2017). Nimbro rescue: Solving disaster-response tasks with the mobile manipulation robot momaro. *Journal of Field Robotics*, *34*(2), 400–425.

Semsar-Kazerooni, E., & Khorasani, K. (2009). Multi-agent team cooperation: A game theory approach. *Automatica*, *45*(10), 2205–2213.

Seyboth, G. S., Dimarogonas, D. V., & Johansson, K. H. (2013). Event-based broadcasting for multi-agent average consensus. *Automatica*, *49*(1), 245–252.

Shapiro, S. C., & Iwánska, L. M. (2000). Citeseer.

Shapiro, S. C., & Rapaport, W. J. (1991). Models and minds: Knowledge representation for natural-language competence. *Philosophy and AI: Essays at the Interface*, 215–259.

Sowa, J. F. (2000). *Knowledge representation: logical, philosophical, and computational foundations* (Vol. 13). Brooks/Cole Pacific Grove, CA.

Šustr, M., Malỳ, J., & Čertickỳ, M. (2018). Multi-platform version of starcraft: Brood war in a docker container: Technical report. *arXiv preprint arXiv:1801.02193*.

Synnaeve, G., Nardelli, N., Auvolat, A., Chintala, S., Lacroix, T., Lin, Z., ... Usunier, N. (2016). Torchcraft: a library for machine learning research on real-time strategy games. *arXiv preprint arXiv:1611.00625*.

Szolovits, P., Hawkinson, L. B., & Martin, W. A. (1977). *An overview of owl, a language for knowledge representation.* (Tech. Rep.). MASSACHUSETTS INST OF TECH CAMBRIDGE LAB FOR COMPUTER SCIENCE.

Thangarajah, J., Harland, J., & Yorke-Smith, N. (2007). A soft cop model for goal deliberation in a bdi agent. *Proc. of CP*, *7*, 61–75.

Thangarajah, J., Padgham, L., & Harland, J. (2002). Representation and reasoning for goals in bdi agents. *Australian Computer Science Communications*, *24*(1), 259–265.

Thorne, C. (2005). *The bdi model of agency and bdi logics* (Tech. Rep.). Technical report. Trento, Italy: Laboratory for Applied Ontology. Available at http://www. loa-cnr. it/Files/bdi. pdf.

Tisue, S., & Wilensky, U. (2004). Netlogo: A simple environment for modeling complexity. In *International conference on complex systems* (Vol. 21, pp. 16–21).

Trencansky, I., & Cervenka, R. (2005). Agent modeling language (aml): A comprehensive approach to modeling mas. *Informatica*, *29*(4).

Vallejo, D., Albusac, J., Castro-Schez, J. J., Glez-Morcillo, C., & Jiménez, L. (2011). A multi-agent architecture for supporting distributed normality-based intelligent surveillance. *Engineering Applications of Artificial Intelligence*, *24*(2), 325–340.

Van Riemsdijk, M. B., Dastani, M., & Winikoff, M. (2008). Goals in agent systems: a unifying framework. In *Proceedings of the 7th international joint conference on autonomous agents and multiagent systems-volume 2* (pp. 713–720).

Viguria, A., Maza, I., & Ollero, A. (2008). S+ t: An algorithm for distributed multirobot task allocation based on services for improving robot cooperation. In *Robotics and automation, 2008. icra 2008. ieee international conference on* (pp. 3163–3168).

Vinyals, O., Ewalds, T., Bartunov, S., Georgiev, P., Vezhnevets, A. S., Yeo, M., ... others (2017). Starcraft ii: A new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782*.

Wagoner, A. R. (2017). *A monocular vision-based target surveillance and interception system demonstrated in a counter unmanned aerial system (cuas) application.* Unpublished doctoral dissertation, Purdue University.

Wang, J., Liu, J., & Zhong, Y. (2005). A novel ant colony algorithm for assembly sequence planning. *The international journal of advanced manufacturing technology*, *25*(11), 1137–1143.

Weiss, G. (1999). *Multiagent systems: a modern approach to distributed artificial intelligence.* MIT press.

Wender, S., & Watson, I. (2012). Applying reinforcement learning to small scale combat in the real-time strategy game starcraft: Broodwar. In *Computational intelligence and games (cig), 2012 ieee conference on* (pp. 402–408).

Werfel, J. K., Petersen, K., & Nagpal, R. (2011). Distributed multi-robot algorithms for the termes 3d collective construction system..

Winston, W. L., & Goldberg, J. B. (2004). *Operations research: applications and algorithms* (Vol. 3). Thomson Brooks/Cole Belmont.

Wooldridge, M., & Jennings, N. R. (1995). Intelligent agents: Theory and practice. *The knowledge engineering review*, *10*(02), 115–152.

Wooldridge, M. J. (1992). *The logical modelling of computational multi-agent systems*. Unpublished doctoral dissertation, Citeseer.

Xuan, P., Lesser, V., & Zilberstein, S. (2001). Communication decisions in multi-agent cooperation: Model and experiments. In *Proceedings of the fifth international conference on autonomous agents* (pp. 616–623).

Zhang, M., Qin, H., Lan, M., Lin, J., Wang, S., Liu, K., ... Chen, B. M. (2015). A high fidelity simulator for a quadrotor uav using ros and gazebo. In *Industrial electronics society, iecon 2015-41st annual conference of the ieee* (pp. 002846–002851).

Zhao, Y., Wang, S., Cheng, T. E., Yang, X., & Huang, Z. (2010). Coordination of supply chains by option contracts: A cooperative game theory approach. *European Journal of Operational Research*, *207*(2), 668–675.

Zhong, C., & DeLoach, S. A. (2011). Runtime models for automatic reorganization of multi-robot systems. In *Proceedings of the 6th international symposium on software engineering for adaptive and self-managing systems* (pp. 20–29).

VITA

VITA

Yongho Kim

Education

- Doctor of Philosophy, Computer and Information Technology
  Purdue University, West Lafayette, Indiana, United States of America
  December 2018
- Master of Science, Electronics and Radio Engineering
  Kyung-Hee University, Yongin, Republic of Korea
  February 2012
- Bachelor of Science, Computer Science and Engineering
  Seoul National University of Science and Technology, Seoul, Republic of Korea
  February 2010

Work Experience

- Research and Teaching Assistant
  Purdue University, West Lafayette, Indiana United States of America
  August 2017 - December 2018
  August 2016 - May 2017
  January 2014 - May 2016
- Research Aide
  Argonne National Laboratory, Lemont, Illinois, United States of America
  May 2017 - August 2017
  May 2016 - August 2016
- Researcher

Moa payment Inc, Seoul, Republic of Korea

August 2013 - December 2013

- Research Assistant

  Kyung-Hee University, Yongin, Republic of Korea

  April 2010 - March 2013

- Research Assistant

  Seoul National University of Science and Technology, Seoul, Republic of Korea

  July 2008 - June 2009

Publications

Journal Articles

- Y. H. Kim, J. W. Jung, J. C. Gallagher, and E. T. Matson, 2016, An Adaptive Goal Based Model for Autonomous Multi-Robot Using HARMS and NuSMV. *International Journal of Fuzzy Logic and Intelligent Systems*, 16(2), pp. 95-103.
- S. M. Shin, S. H. Park, Y. H. Kim, and E. T. Matson, 2016, Design and Analysis of Cost-Efficient Sensor Deployment for Tracking Small UAS with Agent-Based Modeling, *Sensors*, 16(4), 575.
- B. C. Min, Y. H. Kim, S. J. Lee, J. W. Jung, and E. T. Matson, 2015, Finding the Optimal Location and Allocation of Relay Robots for Building a Rapid End-to-end Wireless Communication, *Ad Hoc Networks*
- Y. H. Kim et al, 2012, Computational Method for Analyzing the Cumulative Ionizing Effect from Solar-terrestrial Charged Particles and Cosmic Rays with Geant4, *Journal of the Korean Physical Society*, 61, pp. 653-657.
- Y. M. Seo, Y. H. Kim, S. H. Park, and J. H. Seon, (2012), Cumulative ionizing effect from solar-terrestrial charged particles and cosmic rays for CubeSats as simulated with GEANT4, *Current Applied Physics*, 12(6), pp. 1541-1547.

Book Chapters and Proceedings

- S. J. Park,, S. U. Park, H. G. Lee, M. J. Hyun, E. S. Lee, J. H. Ahn, L. Featherstun, Y. H. Kim, and E. T. Matson, 2018, Collaborative Goal Distribution in Distributed Multiagent Systems. 2018 *Second IEEE International Conference on Robotic Computing* (IRC), pp. 313-318
- J. M. Goppert, A. R. Wagoner, D. K. Schrader, S. Ghose, Y. H. Kim, S. H. Park, M. Gomez, E. T. Matson, and M. J. Hopmeier, 2017. Realization of an autonomous, air-to-air counter unmanned aerial system (CUAS). *IEEE International Conference on Robotic Computing* (IRC), pp. 235-240
- Y. H. Kim, and E. T. Matson, 2016. A Realistic Decision Making for Task Allocation in Heterogeneous Multi-agent Systems. *Procedia Computer Science*, 94, pp. 386-391.

- Y. H. Kim, and E. T. Matson, 2016. A Realistic Decision Making for Task Allocation in Heterogeneous Multi-agent Systems. *Procedia Computer Science*, 94, pp. 386-391.

- Y. H. Kim, M. Gomez, J. Goppert, and E. T. Matson, 2015. Model checking of a training system using nusmv for humanoid robot soccer. *In Robot Intelligence Technology and Applications 3*, pp. 531-540.
- S. H. Kang, Y. H. Kim, E. J. Lee, S. G. Lee, B. C. Min, J. U. An, and D. H. Kim, 2011, Implementation of Smart Floor for multi-robot system, *The 5th International Conference on Automation, Robotics and Applications* (ICARA), pp. 46.
- J. W. Kim, Y. H. Kim, B. C. Min, and D. H. Kim, 2010, Tacit Navigation Method for Multi-agent System, *Communications in Computer and Information Science*, Vol. 103, pp. 186.
- Y. W. Lim, Y. H. Kim, J. U. An, and D. H. Kim, (2010, September), Path planning algorithm based on the limit-cycle navigation method applied to the edge of obstacles. *In FIRA RoboWorld Congress*, pp. 226-233, Springer, Berlin, Heidelberg.

- Y. H. Kim, J. W. Kim, B. C. Min, and D. H. Kim, 2010, Dynamic Obstacle Avoidance Using Vector Function Algorithm, *International Conference on Electronics, Information and Communication*, pp. 229-231.

Academic Activities

Academic Services

- Program committee, The 17th International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS 19), Àvila, Spain, June 26 - 28, 2019.
- International program committee, The Third IEEE International Conference on Robotic Computing (IRC 2019), Naples, Italy, February 25 - 27, 2019.
- International program committee, The 4th International Workshop on Communication for Humans, Agents, Robots, Machines and Sensors (CHARMS 2018), The 2nd IEEE International Conference on Robotic Computing, Laguna Hills, California, United States, January 31 - February 2, 2018.
- International program committee, The Second IEEE International Conference on Robotic Computing (IRC 2018), Laguna hills, California, United States, January 31 - February 2, 2018.
- International program committee, The 3rd International Workshop on Communication for Humans, Agents, Robots, Machines and Sensors (CHARMS 2017), The 1st IEEE International Conference on Robotic Computing, Taichung, Taiwan, April 10 - 12, 2017.
- International program committee, The 14th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI 2017), Maison Glad Jeju, Jeju, Korea, June 28 - July 1, 2017.
- Workshop organizer, 2nd International Workshop on Communication for Humans, Agents, Robots, Machines and Sensors (CHARMS 2016), The 12th

International Conference on Mobile Systems and Pervasive Computing (MobiSPC), Montreal, Quebec, Canada, August 15 - 18, 2016.

- International program committee, 13th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI 2016), Sofitel Xian on Renmin Square, Xian, China, August 19 - 22, 2016.
- Technical program committee, Special Session on Humans, software Agents, Robots, Machines and Sensors (HARMS), 6th International Conference on Automation, Robotics, and Applications (ICARA 2015), Rydges Lakeland Resort, Queenstown, New Zealand, February 17 - 19, 2015.
- International program committee, 12th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI 2015), KINTEX, Goyang city, Korea, October 28 - 30, 2015.

Reviews

- Reviewer, IEEE Sensors Applications Symposium, 2015 - 2019.
- Reviewer, International Conference on Automation, Robotics, and Applications (ICARA), 2015.

Presentations

- Invited Speech, Collaborative Multi-Robot systems: Goal Representation and Delegation, Purdue University, West Lafayette, United States, July 28, 2018.
- Research Seminar, Model Checking for Interception of A Ball in Robot Soccer, Gacheon University, Gyunggi-do, South Korea, July 20, 2015.

Award

- 2016 Polytechnic Institute Summer Research Grant
  Purdue University, West Lafayette, Indiana, United States of America