A Generic Software Framework For Distributed Coordination And Control In Multiagent Systems

Thorbjørn Borup**, Alexander Larsen, Kim Mørk, Ebbe Nielsen & Sigurd Villumsen*

Abstract—This paper addresses the problem of designing a generic framework for distributed coordination and control in multiagent systems. Communication between agents increase their effectiveness, which is why this is the cornerstone of the software framework. Generic aspects are secured by making all layers of the software framework exchangeable, which secures compatibility with different wireless topologies, protocols and applications. A protocol for multiagent systems and an application regarding distributed household cleaning is designed. The parts of the design concerning task assignment and peripheral units is implemented on the designed software framework. The results show that it is possible to implement applications on the designed software framework. The designed task assignment algorithm is able to distribute tasks between multiple agents in a hierarchical manner by means of the protocol. Furthermore it is possible to interact with peripheral units in the form of a Ubisense localisation system and an automated door from Besam. The tests of the protocol show that optimizations regarding collision avoidance are needed in order to make the protocol usable. The designed algorithm for cooperative completion of tasks is verified by means of UPPAAL. From this it is concluded that all states of the algorithm are reachable and it contains no deadlocks.

I. INTRODUCTION

The world wide demographic trend shows that the average life expectancy is increasing. This means that the amount of individuals that need care is increasing. In Denmark FOA (a danish trade union) assesses that 6.6 percent more nursing staff is needed in the year 2015, when considering the demographic trend [1]. Furthermore one third of the nursing staff will retire within the next ten years. Taking these facts into consideration it is clear that the elder care workload is increasing.

The elder care can be divided into two categories; solicitude towards the elder and trivial tasks such as cleaning. To reduce the workload generated by trivial tasks automation can be introduced. The automation could effectively be implemented as cooperating agents because this improves completion time and ensures robustness of the execution of multiple tasks compared with non-cooperating agents [2]. One area that can be automated is vacuum cleaning in nursing homes. Today automated vacuum cleaning systems exists, but to our knowledge no cooperative multiagent solution is available. In order to make such a solution a generic software framework supporting distributed coordination and control is beneficial because the elements of the framework are exchangeable such that other applications can be implemented. Thus this paper seeks to develop a generic framework for distributed coordination and control in multiagent systems that involves previously researched areas. These include cooperative control, communication and mapping. Furthermore an application to automate cleaning tasks is designed and partial implemented using the framework.

Cooperative control can be divided into two sections. Formation control problems which regard unmanned aircrafts, satellites and spacecrafts whereas nonformation control problems regards subjects such as task assignment and payload transport. Such problems are of interest since they must be solved for cooperation in multiagent systems to be efficient. The information necessary for solving these cooperative problems can be communicated between the agents through a wireless network or it might be pre-programmed before a mission. This information enables the agents to reach agreement when sudden changes happen in their environment. It is said that consensus is reached when a group of agents agree on some coordination data. Consensus can be defined as a convergence towards a common value. When information is decentralized reaching consensus can be challenging since a single agent may have unique information. Reaching
consensus in multiagent systems have been studied considerably. Wei Ren and Beard R.W. has shown that consensus can be reached even if the interaction topologies changes dynamically [5]. Olfati-Saber R. and Murray R.M. introduce two consensus protocols for networks and study cases of fixed and switching topologies [7]. Our designed application assumes a fixed topology in the sense that all agents can communicate directly.

Various characteristics of cooperation have been investigated within the robotics community. According to [3] there are different characteristics which define the group architecture of a cooperative multi agent system. The planning of the cooperative decisions can be categorized in two ways, centralized or decentralized. The agents can be either homogeneous, meaning they have the same capabilities whereas heterogeneous agents have different capabilities. The communication structure can be divided into three groups: Interaction via Environment (all actions are based on the environment), Interaction via Sensing (actions based on the presence of other agents) and Interaction via Communication (actions based on explicit communication). Furthermore multiagent problems can, according to Tucker Balch and Ronald C. Arkin [2], be divided into three types. These are foraging, consuming and grazing. Foraging covers searching for objects and bringing them back to a certain location. Consuming involves doing a task at a specific location. Grazing tasks such as vacuum cleaning involves to cover a certain area. Our application uses a decentralized method to solve the nonformation control problem regarding task assignment whereas the cooperative completion of tasks are done centralized. The application is made for homogeneous agents with a communication structure of the type Interaction via Communication, but the framework does support the use of heterogeneous agents.

A.F.T. Winfield and O.E. Holland have demonstrated a communication and control infrastructure for distributed mobile robotics. They use commercial off-the-shelf software and hardware such as Transmission Control Protocol/Internet Protocol (TCP/IP) and WLAN in order to make a platform to use for experiments with robotics [8]. Their solution has been demonstrated to successfully and reliably control a multiagent system. Our research focuses on an application regarding household cleaning and a protocol for this purpose is designed. A reason for not using TCP/IP as A.F.T. Winfield and O.E. Holland is that it is proven to lead to unacceptably low throughput with high packet loss [6].

The problem of concurrent indoor mapping has been a subject of attention in the mobile robotics community. The problem of mapping an indoor environment concerns localisation of obstacles and other objects of interest[4], [10], [11] . Our application uses a static solution for mapping and localisation, which is provided by the localisation system Ubisense [9].

This paper presents a generic software framework for distributed coordination and control in multiagent systems. Beside this an application regarding distributed household cleaning is designed. The parts of the design concerning distributed task assignment and peripheral units are implemented on the software framework. The part of the design regarding cooperative completion of tasks is tested by means of UPPAAL in order to verify the design.

II. THE SOFTWARE FRAMEWORK DESIGN

This section describes the different levels of the designed software framework. Initially the structure of the software framework is presented, which leads to a description of the protocol design.

Figure 1 presents the structure of the software framework, which consists of three levels.

![Software framework](image)

Fig. 1. Shows the structure of the designed software framework, which have been divided into three levels. Each level is exchangeable in order for the framework to be generic. The right side of the Figure shows the layers of the OSI model and what level these are placed on in the framework.
The first level has the purpose of ensuring the wireless topology. This level forms the basis for the use of the communication level in the sense that it provides the service regarding the establishment of a wireless connection.

The communication level contains the communication and message protocol. The communication protocol establishes connections and secures data transmission whereas the message protocol ensures that the content of transmitted data can be recognized and understood by the agents. The communication level can incorporate a routing algorithm if necessary because the implemented application assumes that all agents can communicate directly at all times. The communication layer is exchangeable meaning that the designed protocol can be replaced if it does not fit a certain application.

The application layer contains elements which are needed to make multiple agents able to carry out a task. These elements are mapping and localisation, task assignment and cooperative task completion. By isolating these elements in the application layer the framework support different applications as long as the interface to communication is obeyed.

Protocol Design

One of the key elements of the designed software framework is the protocol. The purpose of the protocol is twofold, firstly it should enable intercommunication between agents. As any agent potentially holds unique information about the common objective the combined information of all agents is viewed as a distributed database. This fact leads to the second purpose of the protocol, namely that it should provide the agents with means of distributing information about the common objective to the distributed database.

The seven-layer OSI reference model is used as a basis for the protocol design, but it should be noted that the application and transport layer are separated by an intermediate embedded application layer with interfaces to application lists.

Figure 2 shows the software architecture used for the protocol design. The Presentation and Session layer of the OSI reference model are omitted since there is no need for presentation or session services. The application lists and individual layers are described in the following sections.

The Application Layer: The application layer is divided into two distinct sublayers. The first is a normal application layer which contains the applications which define the functionality of the agents and what objective they are to accomplish. The second sublayer is an embedded application layer which handles the interface to the subjacent transport layer. Both of these sublayers have a common interface to the application lists which contain all the distributed information needed by any applications using the network. The common interface is defined by use of at least two data identifiers. These contain the list type, and the filter used to describe which part of the list that is accessed. In Figure 3 an illustration of how the identifiers are used for accessing one of the lists in the distributed database is presented. The list contains an ID pointer identifying the default list element of which the data is extracted. When writing data to the list the ID can either be specified by altering the ID pointer with the ID interface, or by appending the ID at the start of the data.

The embedded application layer handles the interface to the transport layer and holds a list of jobs to be sent. These jobs contain the data identifiers needed to retrieve the newest version of the data from the list prior to transporting it.
Fig. 3. Shows how the two data identifiers Type and Filter together with the lists internal ID pointer describes exactly what parts of the list are to be extracted or overwritten. When the filter is appended on the list, only the relevant rows are accessible for extraction.

When the application layer communicates this is done through a send interface provided by the embedded application layer. To distribute information to other units and provide direct access to the units information, the embedded application layer provides three different frame types. These three types make the embedded application layer able to provide access to the local lists through the transport layer and to retrieve information for the application from other units. In Figure 4 the interfaces between the embedded application layer, the lists and application is illustrated with the three different frames used for the distributed database. This is implemented by means of two types of transport layer services, request data from an agent and send data to an agent. The flow of this data is described in Figure 4.

The network read service consist of two distinct package types, an Ireq frame that indicates to the receiver that another agent requests information from it and a Yreq type that indicates the received data has been requested. When a network read is initiated an Ireq package is sent containing the data identifiers, type, filter, and id. The frame is handled by the embedded application layer which shifts the frame type to Yreq and append it to the joblist when the job is handled.

The network write service consists of a single type called APP. When a network write is initiated an APP package is generated by the embedded application layer by reading the local list through the list interface and send it to the specified receiver. This APP package is processed by the receiver by means of the list interface and the deconstructor. The deconstructor is used to write information to the application lists whereas the constructor reads information. Note that an agent does not have write access on the network to any data that it is not currently working on, as this could result in loss of information about the common objective.

The Transport Layer: This layer has the purpose of encapsulating application data into packets suitable for transfer to the network layer. To minimize data overhead, the data transfer between agents by way of the designed protocol is of the connection less type. This means that data is transferred without first establishing an end to end connection. But as soon as the first package is received, the recipient is aware of the incoming
transmission and designates a receive buffer to the sender. When this is done, an end to end connection between the two units is established, and is first closed when the sender receives an acknowledge.

**The Network Layer:** The communication protocol has been designed in such a way, that some bytes has been reserved to include source/destination identifiers, which could be utilized by a network layer for routing purposes. The conducted tests are made using point-to-point communication, with no routing algorithm implemented. The network and transport layer constitute the second level of the framework.

**The Data Link Layer:** The services that the Date link layer provides are determined by the microprocessor board on which the framework is implemented. And is based on a protocol developed by Nordic semiconductors.

**The Physical Layer:** The physical link is based on a wireless connection that is provided by the microprocessor board on which the framework is implemented. The physical and Data link layer constitute the first layer of the framework. Some of the services provided by this layer is a cyclic redundancy check and address matching.

**Application Design**

An application regarding distributed household cleaning have been designed. This application is minded to be implemented on multiple cleaning robots in order to automate trivial cleaning tasks such as vacuum cleaning. The section to follow presents the design of the key parts of the this household cleaning application. These parts concern task assignment and cooperative completion of tasks. Furthermore two applications regarding a localisation system and an automated door have been designed. These two applications are implemented on the designed framework which verifies whether or not the Application level of the framework is exchangeable.

**Task Assignment**

A hierarchical iterative conflict resolution method has been used to solve the nonformation problem of task assignment. The procedure for this method is as follows: The system is given one or more tasks that are prioritized. Then the agents agree on who is assigned to the different tasks. The agents do this by bidding on the task, and the agent with the highest (or lowest depending on how the decision factor is defined) decision factor wins the task. The decision factor should be defined in such a way that it is unambiguous, so that only one agent can get the task. The task that is assigned first is the one with the highest priority. When the first task has been assigned the remaining agents bid on the next task. This procedure continues until all agents have been assigned to a task. In a decentralized system an agent can verify that it can assign itself to a task by communicating with all other agents. This is necessary in order to verify that only one agent is assigned to a task at once. Figure 5 shows this form for assignment structure.

![Fig. 5. Shows a general bidding structure used for assigning an agent to a task. Both agent 1 and 2 are bidding on the same task. In bidding round 2 both bidding agents have asked agent 4 and it replies to agent 1 that it should ask agent 2 next. Then agent 1 and 2 resolves who gets the task.](image-url)
have been assigned to the task. The last possible answer is a "maybe", which indicates that another agent is bidding on the same task. A "maybe" answer contains the agent ID of the agent which also is bidding on the task. The two agents that are bidding on the same task will then communicate and determine who will continue to bid on the task. The agent ID can be used as the decision factor for which agent that is assigned to a task. This is simple and perhaps not optimal since other factors, such as distance to the task, can be relevant in decision of who gets the task. An algorithm including these other factors can be implemented instead of the simple agent ID test (e.g. by a weighted sum of the decision factors). The designed framework is tested with an assignment structure where the agent ID is the only decision factor. This structure can be seen in Figure 6. It is noticed that the two agents bidding on the task do not have to complete two-way communication to determine who is assigned to the task since the agent ID is the only decision factor.

![Diagram](image)

**Fig. 6.** Shows the assignment structure used for testing the framework. This situation is the same as Figure 5 with the exception that agent 4 knows the terms on which the winner of the bidding is chosen, which is why agent 4 can inform agent 1 that it have lost the bidding.

The implemented task assignment procedure shown in Figure 6 should be understood as follows:

- **Round 1**
  - Agent 1 asks Agent 3 and receives a "yes".
  - Agent 2 asks Agent 5 and receives a "yes".

- **Round 2**
  - Agent 2 asks Agent 4 and receives a "yes".
  - Agent 1 asks Agent 4 and receives a "maybe" including the ID of Agent 2.
  - Agent 1 verifies who has the highest ID.
  - Agent 2 has the highest ID so Agent 1 loses the bidding.

- **Round 3**
  - Agent 2 asks Agent 3 and receives a "maybe" including the ID of Agent 1.
  - Agent 2 verifies who has the highest ID.
  - Agent 2 wins the bidding and continues.

- **Round 4**
  - Agent 2 asks Agent 1 and receives a "yes".
  - Agent 2 assign itself to task 2.

**Cooperative Completion Of Tasks**

Agents can complete tasks cooperatively in a centralized or decentralized manner. It is chosen to use a centralized cooperative structure which is seen in Figure 7.

![Diagram](image)

**Fig. 7.** Shows the centralized cooperative structure of the system. The master distributes parts of a task to the slaves after which the slaves starts to carry out the task.

The first agent which is assigned to a task becomes the master (the central unit) associated with that task. According to the task assignment structure presented previously there cannot be two or more masters associated with the same task. An agent becomes a slave when it offers its cooperation to a master of a task and gets assigned to the task. A master is responsible to acknowledge which agent gets assigned to a task.

The master divides the task into equal parts and distributes the information concerning each part to its workers. By doing this the workers will see their part of the task as one unique task. The division of the task is done by letting an algorithm divide
the area belonging to the task into equal squares if possible. The total amount of squares is then distributed equally between the agents. An example of how an area is divided is seen in Figure 8. The distribution of squares is done as if all squares are of equal size. This way of grid based room division produces an accurate mapping of the room but when the room increases the complexity rises with it [10].

![Figure 8](image.png)

**Fig. 8.** Shows how an area is divided into minor squares in order to distribute the task between cooperating agents.

When agents cooperate they may finish their subtasks asynchronous. When a worker is done with its subtask it notifies the master to secure that the master is able to divide the task if another agent is to cooperate. When a master finishes first it collects its workers task information and merge them. Then the workers and master leaves the task and a new bidding round is initiated. The new master on the task will get the information regarding the task from the previous master.

### III. Results

The developed framework has been tested through implementation of the task assignment algorithm and the applications regarding the localisation system and automated door. The first level of the framework is provided by the nRF905 chip which provides the wireless technology. The two remaining levels of the framework are implemented on Open Sensor Boards, which are microprocessor boards based on a dspic30f3013 with 2kB RAM and 16kB programmable memory. Furthermore the cooperative capabilities of the overall household cleaning application is verified using UPPAAL. The remaining part of the section presents the conducted test regarding the implementation of the protocol, the task assignment algorithm and the peripheral units.

**Test of Protocol**

The developed protocol for the software framework is tested by measuring how the transmission time increase as more data is sent. To emulate the data flow of a typical application of the framework, it is tested with packages constructed with the list interface. These packages consists of 3 packets and emulates task information. The results will be presented in two cases; a case when one agent is sending multiple tasks to multiple agents and a case when multiple agents is sending multiple tasks at the same time to multiple agents.

Figure 9 presents a plot of the time it takes for one agent to send tasks (1 to 6 tasks) to multiple agents (1 to 5 agents).

![Figure 9](image.png)

**Fig. 9.** Shows a plot of how long it takes one agent to send 1 to 6 tasks to 1 to 5 agents.

It can be seen that the time it takes to send all tasks increase linearly with the number of tasks and the number of recipients. This result is expected since no other unit is sending at the same time and therefore no collisions occur.

Another test is conducted where multiple agents is sending at the same time. This is done to examine how the protocol fares under circumstances with high network load. The results of this test can be seen in Figure 10. It shows the time taken for all agents to distribute all tasks (1 to 6 tasks) between all agents (2 to 5 agents). It is seen that the time taken to distribute all tasks between the agents increase rapidly as the number of agents
increase. This indicates that when more agents are transmitting at the same time collisions occur.

Figure 11 shows a plot of how many retransmissions per packet there would occur during transmission if the amount of retransmissions are unlimited. It is notable that the number of retransmissions do not vary significantly with the number of sent tasks. On the other hand when the number of agents grow the number of retransmissions per packet increase approximately linearly with the amount of agents. The peak collisions per packet is approximately 1.6 which indicates that on average every packet is going to be retransmitted more than once. The main reason for this is that the hardware which the protocol has been implemented on has not been set up to use collision avoidance. As the protocol is intended for multi agent systems with numerous agents, this result indicates that collision avoidance should be implemented if the designed protocol is to be implemented in a fully developed distributed system.

**Task Assignment Algorithm**

The test of the task assignment algorithm is conducted as a whitebox test. The whitebox test emulates an agent with a list containing 4 tasks with given priorities and 4 agents to ask. The three possible responses to this bidding (Yes, No and maybe) is fed to the agent and its response is verified. The whitebox test shows that the agent is able to respond accordingly to the responses received and assign itself to the task with the highest priority. This decision is made on the basis on a simple decision factor which is the agent ID.

**Ubisense Application**

The conducted tests of the application regarding the localisation system Ubisense are based on the setup seen in Figure 12.

The Ubisense system contains a number of sensors (at least 2), which calculate the location of tags. The location of the tags can be read out on a PC. In our test setup the location is sent from the PC to the Open Sensor Board (1). The receiving Open sensor board (2) is also connected to a PC.
which is used for reading the locations of the tags.

The conducted tests are made with between two and five tags that are moved simultaneously. It have been verified that the locations of the tags are updated when their locations change, and that these locations are transmitted to the receiving Open Sensor Boards. The conducted tests show that the Ubisense application have been successfully implemented in the designed software framework.

Automated Door Application

The conducted tests of the application regarding the automated door application are based on the setup seen in Figure 13.

![Fig. 13. Shows the setup for testing the automated door application. The door is connected to an Open Sensor Board (1), which has a wireless connection to two other Open Sensor Boards (2 + 3). Sensor Board 2 and 3 simulates agents who wish to pass through the door.](image)

The tests showed that the door opened when one of the two Open Sensor Boards requested to open the door. When the door received a close request it closed. If the door did not receive a close request it closed after its timeout time was exceeded. Furthermore it was shown that when multiple Open Sensor Boards requested the door to open it opened and stayed so until all agents had passed through the door. If one agent were to pass through the door while another agent requested to close it, the door stayed open until the agent had passed through the door. From the tests it can be concluded that the application works as intended.

Simulation of Cooperative Capabilities Using UPPAAL

The designed algorithm for cooperative completion of tasks have been simulated and tested using UPPAAL, which is a tool for test and simulation of real time systems. It has been jointly developed by researchers at Uppsala University and Aalborg University. The simulation shows that all states of the designed algorithm are reachable and it contains no deadlocks.

IV. DISCUSSION

This paper brought the development of a generic software framework for distributed coordination and control in multiagent systems into focus. Through tests it was shown that parts of an application for cleaning tasks was successfully implemented on the software framework. The implementation of the localisation system and the automated door proved that level 3 (the Application level) of the framework is generic. In order to verify that the two remaining levels (Wireless topology and Communication) of the framework is generic more tests would have to be conducted. These test should verify that other protocols can constitute level 2 of the framework and that another wireless technologies can constitute level 1.

The designed protocol is minded for multiagent systems and it has been proven that information can be distributed among multiple agents by means of the designed protocol. The tests of the protocol show that optimization regarding collision avoidance could improve the performance of the protocol. This optimization would have to be made in order to make the protocol usable.

The implementation of the designed application and protocol on the software framework constitute the main parts of a platform for distributed household cleaning. In order to complete this platform a method for cooperative completion of task should be implemented. The proposed method for cooperative completions of tasks is based on a centralized structure. The method has been tested in UPPAAL. From this it is concluded that all states of the designed algorithm are reachable and that it contains no deadlocks. To verify that the method actually enable multiple agents to cooperate requires that the method must be implemented on the designed framework and tested. If these test showed that the method enables multiple agents to cooperate, the implementation of the designed application on the framework could be used as a platform for distributed household cleaning. The automation of cleaning tasks by means of this
platform could make it possible for staff at nursing homes to use more time on solicitude towards the elder instead of trivial cleaning tasks.

REFERENCES


