Multi-Agent Robotic System Architecture for Effective Task Allocation and Management

Egons Lavendelis, Aleksis Liekna, Agris Nikitenko, Arvids Grabovskis and Janis Grundspenkis
Department of Systems Theory and Design
Riga Technical University
1 Kalku Street, LV-1658, Riga
LATVIA
{egons.lavendelis, aleksis.liekna, agris.nikitenko, arvids.grabovskis, janis.grundspenkis}@rtu.lv,
http://stpk.cs.rtu.lv/

Abstract: - Large number of autonomous robot solutions exists for various missions and domains. These robots are sufficient for the missions they are built for. At the same time each of them has limited functional and physical capabilities. Multi robot systems can be used to remove these limits. However it is true only in case when the system ensures effective interaction among the robots i.e. enables their social behaviour. Usually it is hard to implement such capabilities directly into robots due to functional and physical limitations and heterogeneity of the team. One of possible solutions is to implement management systems outside the robots. It should decompose tasks, allocate subtasks to specific robots and monitor the execution of the assigned tasks. In order to avoid inherent drawback of fully centralized systems a significant level of autonomy has to be preserved. Intelligent agents fulfil these requirements. Therefore we propose a general multi-agent system's architecture for multi-robot management system for applications with a common goal. It can be used to implement various purpose multi-robot systems from existing single operating robots without major changes in their software and hardware. A vacuum cleaning problem of a large area is considered as an application case.

Key-Words: - Multi-Robot System, Intelligent Robotics, Multi-Agent System, Multi-Agent Architecture, Multi-Robot Task Allocation

1 Introduction

Various robots for different purposes exist. Autonomous vacuum cleaning robots clean various premises [21], agricultural robots do many jobs in precise agriculture [18], etc. Usually these robots include algorithms for autonomous execution of their missions. For example, a vacuum cleaning robot is intelligent enough to autonomously clean a certain area in sufficient quality. Unfortunately these robots still have significant limitations. For instance, a single robot cannot be effectively used for larger (more efforts requiring) missions, where it is not capable to accomplish the whole mission. Example of such mission is a large area that cannot be cleaned by a single vacuum cleaning robot because of time and resource considerations. One of possible solutions is to use multiple robots simultaneously for a given mission.

At the moment widely used robots lack social capabilities to interact with each other in any forms. As a consequence no cooperation is possible among them. Direct implementation of social capabilities, like team formation, negotiations, task assignment and distributed planning into robots would affect both software and hardware of robots. The software of available robots would be changed and made more complex, because additional intelligent components would be needed to implement abovementioned social capabilities. It may demand more complex hardware and some physical means of communication among all robots. As a consequence, significant changes in the widely used robots are mandatory to implement social behaviour into robots. Such solution is expensive and lacks the necessary agility. Therefore we see a need for software that ensures cooperation among robots without specific requirements to them. We propose to address the problem via developing a robot management system that uses data received from robots and performs all the necessary interactions among them. Team formation functionality is part of the management system. The management system decomposes tasks, assigns (and reassigns in case of failures) them to the most suitable robots thereby forming the team and monitors task execution. The goal of the management system is to keep the software and hardware of robots unchanged or change it as minimal as possible. Robots should keep their internal algorithms for task execution and only communication mechanisms to the management system should be added.

We see that the management system has to be characterized by the following fundamental features. It has to be intelligent enough to form effective teams of robots. It has to be reactive in the sense that it has to follow the execution process and react on events arising during the execution indicating wrong actions, malfunctioning, etc. of robots. It has to monitor the status of the mission and pass it to the user whenever it is necessary. It should not completely take away robot's autonomy.

Characteristics of intelligent agents like autonomy, social capabilities and reactivity [17] fit the abovementioned demands of the management system. Thus, we propose to use intelligent agents in the robot management system as the core paradigm. The role of agents is to implement social behaviour and team formation. Moreover, agents carry out reactive behaviour by monitoring robots during the task execution. As of authors' knowledge there is no general architecture that explicitly specifies agents and their interactions for various purpose agent based robot management systems. In order to fulfil the above defined requirements the paper describes architecture of the robot management system whose main part is built as a multi-agent system, where agents represent robots and the user during the team formation, task allocation and execution. The remainder of the paper is organized as follows. The Section 2 describes logical architecture of the system. The Section 3 describes the proposed architecture at the level of physical components. The Section 4 blueprints the case study of the architecture for vacuum cleaning robot domain. The Section 5 compares the proposed architecture to the related works. The Section 6 concludes the paper and gives an insight of the future work.

2 Logical Architecture

The paper proposes general architecture for robot management systems of different domains that have existing single operating robots, which are capable to execute standard missions, but cannot carry out more labour consuming ones. The architecture allows building multi-robot systems from existing standard robots without making major changes in them. It is suitable for multi-robot systems with the following functionality. To achieve the goal of building the multi-robot systems – to accomplish large missions, the management system has to decompose the task specified by a user until the subtasks can be accomplished by a single robot. The decomposed subtasks must be assigned to particular robots. The allocation should make the overall performance of the system efficient. After tasks are allocated to the robots, the system has to monitor the execution for two main reasons. First, it has to report mission's status to a user and second, various failures may occur during the execution. If a robot fails to complete its task for some reason, this task must be reallocated to other robots.

The proposed architecture has the following main characteristics. The core of the management system is built as a multi-agent system. All interactions among robots are carried out by agents in the robot management system. The robots only have to communicate with the management system or to pass the information to the agents. Each robot has a corresponding (mapped) agent in the system. These agents are named robot agents and represent robots in the management system. Each robot together with its agent makes an autonomous component. Robot agents monitor their robots. They know actual states of the robots, including tasks allocated to the robots and progress of these tasks. Miscellaneous data about each robot, like battery level, errors occurred and robot's pose and location are monitored, too. Still, during the execution of the task each robot uses built in algorithms to execute its tasks autonomously. The robots do not have to implement interactions to other robots and do not have to implement any task allocation mechanisms that are implemented exclusively by the management system. It allows keeping the software and hardware of robots as simple as possible.

Similarly to robots, the user is represented by an agent, named manager agent. It receives requests for task execution from the user. The agent is responsible for task decomposition into a set of optimal subtasks that could be efficiently executed by individual robots. It has to assign subtasks to the robot agents and reallocate them in case of any failures in their execution by using some kind of negotiation protocol. The architecture is general in the sense that it does not specify the negotiation protocols. It allows using different protocols for task allocation and reallocation. For example, widely applied option of negotiation protocols is the Contract Net protocol [4]. Subtasks can be sequentially allocated using separate Contract Net protocols. Such mechanism is sufficient for the management system's functionality. Still it may lead to suboptimal task assignments. More complex option is the TraderBots approach [3]. It adds market based mechanisms to exchange tasks allocated to the robots if it leads to more optimal allocation. The exchange is done based on some virtual currency. Other options of negotiation protocols are Murdoch and protocols based on beliefs desires and intensions [5]. The generality of

architecture is based on the fact that irrespective to the task allocation protocol the agents have to do the same tasks. Firstly the manager agent has to define the task and initiate the negotiation. The robot agents have to evaluate their abilities to carry out the task and calculate costs (or some number with similar semantics) to indicate how appropriate the robot is for the task. The most important difference in negotiation protocols is existence of direct interactions among robot agents if the TraderBots or any similar approach is used. So, the architecture defines the agents used in the management system, their functionality and general interactions among them irrespectively of the task allocation mechanism used. Actually, the architecture allows realizing various negotiation protocols in the same system.

After the task is allocated to the robot its agent gives appropriate command to the robot. During the execution of the task by the robot, its agent follows the execution and reports the progress to the manager agent. Moreover, in case of any failure during the execution of the task, robot agent reports it to the manager agent, which initiates new negotiation to reassign the task to another robot. The logical architecture of the robot management system is shown in Figure 1. Optional interactions among robot agents (used only in some negotiation protocols) are denoted with dashed lines.

Communications among the components of the logical architecture are performed in the following way. A user communicates only with the manager agent. He/she specifies a task for the system and receives information about the progress of the task and different status information about robots, like, location and pose of each robot, tasks assigned to robots, status of their batteries, etc. Needed status information may vary depending on the domain. Manager agent uses a negotiation protocol to interact with robot agents. Robot agents report progress of tasks and statuses of their robots to the manager agent. Robot agents give commands to the corresponding robots and receive status of the robot, including the task being executed by the robot, error reports and various data from robot's sensors.

3 Physical Architecture

The proposed logical architecture does not include any implementation details. One solution could be to implement it directly i.e. each agent of the logical architecture then is implemented as an agent in some agent development platform, like JADE, JADEX or JACK [1], [13].

Still, if the logical architecture is implemented without any additional components, all interactions among agents and robots must be enclosed in agents. The main problem of such solution is usage of different data structures in the robots and agents. While robots use simple bytes of information, the agents use various knowledge representation structures like concepts and predicates. Mapping among these two encodings must be done. Additionally, interfaces with robots are usually programmed in different technologies. Agents as a rule are implemented in Java based platforms [10], [11], [13] while interfaces with robots are implemented using other languages and platforms, for example MS Robotics Developer Studio [23].

The middleware component is introduced to deal with these problems. So the physical architecture consists of three layers. The higher layer or deliberative layer is realized as a multi-agent system that is responsible for high level decisions like task decomposition and assignment to robots. The second layer or the middleware layer consists of the middleware server that can be implemented in any technology. The main function of this layer is to carry out communications between robots and agents, including the mapping among the data structures used in the higher and lower levels. There are no direct physical communications among agents and robots. All communications among agents and robots are dispatched by the middleware server. The third or lower layer includes robots. The overall physical architecture of the system consisting of three layers is shown in the Figure 2. The two higher levels that correspond to the management system and interfaces among layers are described in detail in the following subsections.

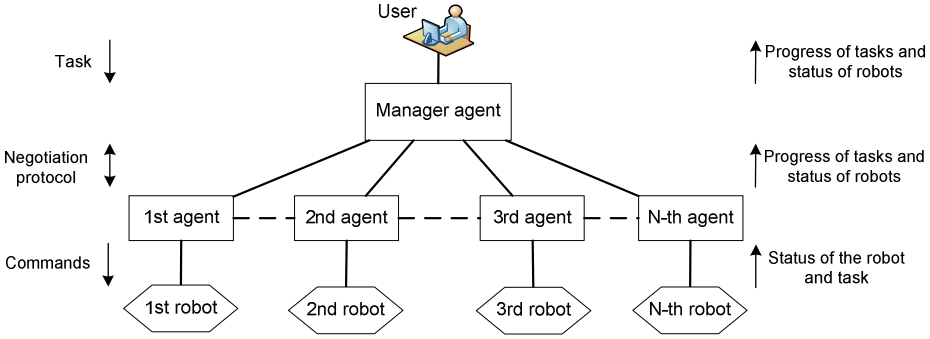

Fig. 1. Logical architecture of the management system

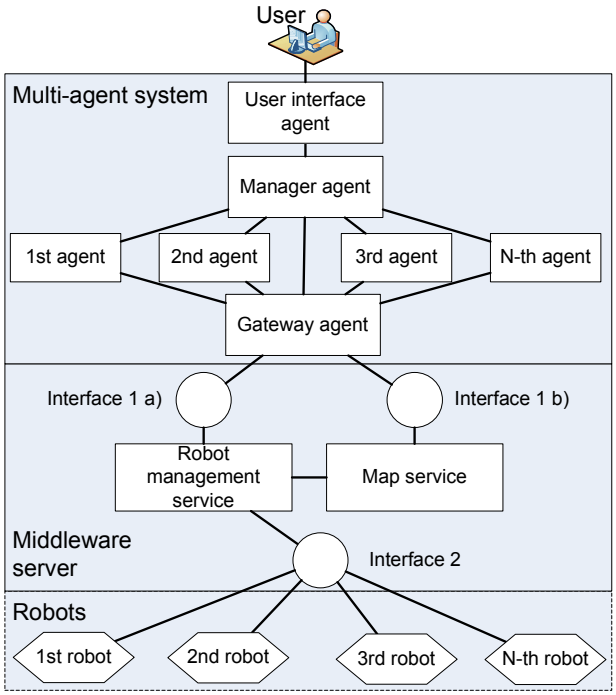

Fig. 2. The physical architecture

3.1 The multi-agent system

The main idea of the multi-agent system in the physical architecture remains unchanged from the logical architecture given in Figure 1. Still, the user interface and the interface with the middleware server must be implemented. Two more agents are introduced in order to separate these interfaces from other components. These agents are introduced because of the following reasons.

Firstly, in case of direct implementation of the logical architecture all agents have to access services from the middleware server. So, all of them would have to implement the interface with the middleware server. Additionally, all agents must be notified about changes in the robots' states, because robot agents need to react to these changes and the manager agent needs to show all changes in the robots' states to the user. Therefore, the agents have to realize not only their responsibilities, but also interactions with the middleware server. To logically separate responsibilities we introduce a gateway agent. It is the only agent communicating with the middleware server. So, it implements interface with the server. All other agents instead of making direct requests to the middleware server send messages to the gateway agent that makes requests to the middleware server. This agent also is the only one that listens to the events that are posted by the robot management service. It informs the corresponding agents about the events. In case of large number of robots and agents the gateway agent may become a bottleneck in the communications among agents and middleware server, because it dispatches all communications. The architecture allows introducing more than one gateway agent in case if such a bottleneck appears.

Secondly, the manager agent of the logical architecture fulfils functions that can be grouped into two independent groups, namely the management of the user interface and task assignment. To keep agents responsible for single type of functions or one role we introduce the user interface agent. The manager agent keeps only the second group of functions (task decomposition and allocation). The user interface agent manages the user interface including the monitoring system that graphically shows statuses of all tasks and robots. It allows the user to follow the execution of tasks. Functions of agents are summarized in the Table 1.

Table 1. Agents' functions in the multi-agent system

Agent	Functions
The user interface agent	Receive commands from the user and forward them to the manager agent. Monitoring system.
The manager agent	Task decomposition. Management of task allocation process. Task replanning in case of any failures. Reporting monitoring information to the user interface agent.
Robot agents	Task evaluation, to be able to generate proposals for the tasks. Participation in task allocation process. Creating commands for the robots. Monitoring the task execution and reacting on state changes. Reporting status of the task to the manager agent.
The gateway agent	Implementation of the service contracts. Responding to the events generated by the services of the middleware server. Calling services of the middleware server upon requests by other agents.

3.2. Middleware server

The main function of the middleware server is to provide means for communications among the multi-agent system and robots. Additionally, in order to improve the system's overall performance it is beneficial to collect incoming sensor data from all robots on the same map and ensure data availability to all robots. Here with term "map" we mean shared data structure among robots. Depending on actual mission it can be topological or metric data structure that stores mission specific data [22]. In any case it has to be up-to-date and available for all agents.

The middleware server is implemented in a service-oriented way [9]. The middleware functions of the server are included in one service, named

robot management service, while all operations with the map are included in the second service, named map service. The latter creates the map using sensor data from robots. It must be capable of providing actual version of the map upon the request of agents. Additionally, it has to calculate distance and route between given locations.

Interface 1 shown in Figure 2 consists of the service contracts. Interface 1 a) is the interface of the robot management service, while Interface 1 b) is the interface of the map service. Mainly interactions between agents and services are done in the way that agents call service methods, when they need some operations to be executed. Additionally the robot management service raises events when there are any changes in the state of the robots. The interface agent is notified about all significant changes in robots' states. Event handling in the service environment is done as described in [23].

In order to monitor the robots' status they need to have reporting functionality. One of our main goals is to create a multi robot system that is capable to accomplish its tasks using as little as possible of any additional infrastructure except the robots themselves. It implies two requirements. First, all data that are acquired about the environment come exclusively from the robots. Second, while the robot sensors are limited in the available information due to physical or functional limitations of the used sensors, the system needs to cope with limited data about the actual state of the environment. Under these assumptions we need the following information reported by robots using the Interface 2: (1) Command executed by the robot or idle state indication to be aware what tasks are being executed and progress of their execution. (2) Position and pose of the robot in terms of its coordinates and heading in any coordinate system that gives possibility to calculate the robot's actual location in the environment. Here authors abstract from the selflocalization and positioning problems being irrelevant in the paper's context. (3) Error report indicating if the assignment accomplishment is being threatened by any detected circumstances. (4) Percepts of the robot that are used to build the map. For example, if the robot senses obstacle by its bumper or distance sensor it reports it together with its location and the map is updated accordingly. (5) Various robot and mission dependant data, like battery level, charging report, etc.

Robots report their statuses to the server at regular time intervals that may depend on the nature of the mission. The robot management service has to process data reported by the robots and forward percept information coupled with the position of the robot to the map building service. If there are significant changes in the position of the robot or any other information has been changed since the last report from the robot, its management service has to post an event to the multi agent system. The exact changes in the status of robots that need to be reported depend on the mission of the system. Additionally, the server has to identify loss of communications to the robots and report possible failure of the robot to the robot agent to deal with physical failures. Commands are sent directly to robots by the robot management service and contain the command and any arguments needed for it. Alternatively, plans can be sent to the robot. In this case every command contains a list of actions.

The proposed middleware server with the specified interfaces separates the higher layer from the lower one. It allows using the same management system with any robots that have the same Interface 2. Moreover, the robots can be substituted with some simulation environment that uses the same interface enabling experiments with virtual environment instead of real robots.

4 Case Study

Example of a mission where the proposed architecture can be used is cleaning an area by vacuum cleaning robots. Such robots are industrially widely used as single-operating devices to clean various premises [21]. Regardless of their wide applications, usage of a single robot has at least one significant limitation: it can effectively clean only limited area. As a consequence, its application in large territories is ineffective. The goal is to develop a management system that simultaneously uses multiple vacuum cleaning robots to clean the same large area. The section outlines the prototype built using the proposed architecture.

At the beginning of a mission the management system receives a task (the area to clean) from a user. The system has to decompose it into smaller subareas and assign cleaning of each subarea to the particular robot. The allocation should be done so that the overall cleaning time is minimal.

The management system has to build at least rough map of the environment to allow finding exact positions of robots in the environment and to provide visualisation of the environment and monitoring data to the user. It allows user to input the area he/she intends to clean just by drawing it on the map. Two phases have been defined, namely an exploration phase and a working phase. The goal of the exploration phase is to build an initial map of the environment just after the system is installed in a

new unexplored environment. The system sends robots to explore the area. Robots report their locations coupled with the sensor data. Collected data are used to form an occupancy grid widely used in mobile robotics [6]. The initial map built during the exploration phase can be modified later on both to make it more precise and include any changes that have taken place after the exploration phase.

After building the initial map the system is ready to start the working phase and to perform its mission using the following scenario for task execution:

1. User chooses the area for the system to clean by pointing it on the map.
2. The area is passed to the manager agent that decomposes it into subareas until each subarea can be cleaned up by the particular robot.
3. Subtasks are allocated to robot agents using a negotiation protocol.
4. Agent whose proposal was accepted adds the area processing task to its task list.
5. If there is a subtask that is not already assigned to any agent, the manager agent initiates new negotiation and the scenario starts over from the Step 3. This step can be started also earlier if the task allocation is not done strictly sequentially.

In the current version of the prototype we sequentially use the Contract Net negotiation protocol. Each subtask is allocated in the following way. The protocol is initiated by the manager agent, which sends the call for proposals with the details of the task. If a robot can execute the task, its agent submits the proposal that indicates how appropriate its robot is for the given task. We use the cleaning time estimate as a measurement. It includes all main factors: workload of the robot, distance from the robot to the area, battery level and a cleaning time evaluation. After receiving responses (proposals or refusals) from all agents the manager agent chooses the most appropriate robot and informs the corresponding agent that it has the task assigned.

Robots execute subtasks assigned to them in a completely autonomous mode using their built in algorithms [16] i.e. they do not receive any other commands except the ones simulating a virtual wall around the area that robot needs to clean up, as it is important to keep the robot in the area it is responsible for in the terms of current task. Here virtual wall is a special signal that simulates a signal sent by the robot sensor when it bumps in a real wall. So we do not have to change anything inside the standard robots widely used as single-operating devices to use them as a part of the team.

The current version of the prototype consists of the multi-agent system and the simulator of the middleware layer, as our first priority was to test the multi-agent system in action before we proceed to the middleware and the physical robots. The multi-agent system is implemented in JADE [1]. It contains one agent for each physical robot, the user interface agent, the manager agent and the gateway agent as specified in the proposed architecture.

The gateway agent has one additional responsibility compared to the general architecture. It creates robot agents when a status message arrives from a robot that does not have a corresponding agent. So there is no need to manually specify the number of robots and their mappings. This is achieved automatically– as soon as a robot sends a status message, the corresponding agent is created.

The manager agent receives tasks in the form of coordinates of area that needs to be cleaned from the user interface agent. It then splits the area into smaller ones using geometrical transformations (we experiment on a few) according to the predefined robot size and capabilities. Next it uses a negotiation protocol to allocate each of the sub-areas to one of the robots. Currently a Contract NET protocol is used, but the implementation of the manager agent allows changing the task allocation method relatively easy giving a test-bed bed to analyze the efficiency of task allocation protocols and compare their results. Analysis of the implemented Contract Net protocol is given in [12] and is omitted here due to the scope of the paper. Further experiments are planned to find the most suitable protocol.

The user interface agent is only needed to issue the commands specified by the user and to display the tasks executed by the system. Moreover, it is needed only in certain times. This is why it is hosted in a separate JADE container – a slave container, which can be started and stopped independently of the master container with all other agents. So, the interface agent can be connected from anywhere and at any time. The user agent which is implemented as an Eclipse plug-in is shown in Figure 3. In this example one can observe four robots processing corresponding sub-areas of the initial task. Further details of the user interface agent are given in [8].

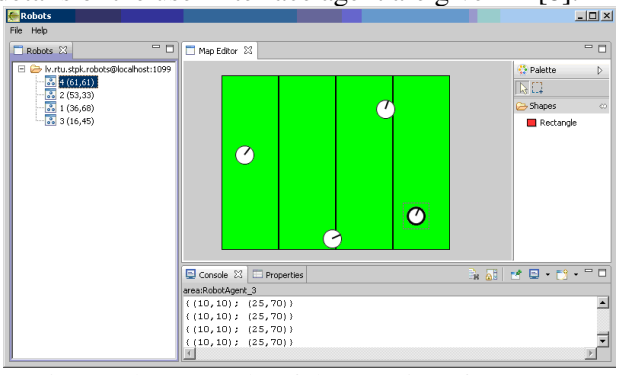

Fig. 3. The example of the user interface agent

5 Related Work

The proposed architecture is multilevel, where components at lower level tend to be more reactive and exhibit swarm-like behaviour, while higher level components tend to be more deliberative. In the same time there is a middle layer, which connects the deliberative and reactive layers. At high level this architecture is similar to hybrid deliberate architecture described in [15]. In fact most of architectures analysed by Godwin [7] may be classified under the same abstract architecture. Difference of the architecture proposed in the paper lies in emphasis on controlling multiple robots, while most of other architectures analysed by Goodwin are concentrating on controlling multiple parts of a single robot. Currently two kinds of contributions in the multi-robot systems are considered related to the paper, namely multi-robot system architectures and multi-robot task allocation.

The following efforts in building loose multi-robot system architectures are identified: STEAM [20], ALLIANCE [19] and CENTIBOTS [24]. Some other known multi-robot system architectures are intended for use in tasks where tight coordination must be employed (for example, to lift and move a box like described in [2]).

The main difference of the proposed architecture from STEAM is in the task allocation mechanism. While STEAM uses the main task coordinator (team leader) that issues orders to team members, the proposed architecture focuses on individual agents being able to apply for tasks, therefore increasing their autonomy. This allows to do separate tasks at different time intervals thereby forming highly dynamic teams. Also task reassignment is simplified if considering failure of some agent, because new process of negotiation protocol is initiated with proposition to finish the started job. Since one area can be cleaned repeatedly (unlike painting) agent does not need to be aware of previous failure and no state transition is required from agent to agent.

ALLIANCE lacks the middle layer. That makes it harder to replace actual robots with a software simulator, because single point of interface between deliberative and reactive layers makes it easier to test application. Main difference from CENTIBOTS is in complexity of each robot – CENTIBOT system uses robots with complex sensors and processing units while the proposed architecture focuses on use of low-cost hardware and in order to achieve sufficient environmental data quality, supplements deliberative level with appropriate algorithms and intelligence.

The multi-robot system architectures are used together with the negotiation protocol or mechanism used for task allocation. Examples of multi-robot task allocation mechanisms are TraderBots [3] and Murdoch [5]. The TraderBots approach is economics based. Initially the tasks are allocated using greedy task allocation. Afterwards robots may trade tasks to each other, both for other tasks or for reward enabling the system to change the allocation if it is not optimal. The Murdoch approach uses message addressing by topic not by receiver, allowing sending the proposal to robots that are capable to execute exactly the needed task. These mechanisms have their advantages and disadvantages for each application. The proposed architecture is designed to allow usage of various negotiation protocols to implement mechanisms of task allocation. It is an advantage comparing to other architectures analysed in the section.

6 Conclusions

A multi-agent based architecture is proposed for the multi-robot management system. It might be applied in domains where the management system has to decompose tasks, assign them to robots by some particular criteria and monitor execution of the tasks. The architecture includes corresponding agents for both robots and the user. It allows maintaining complete autonomy of robots, because each robot coupled with its agent might be considered as an autonomous entity during the task assignment. The robot keeps the same level of autonomy as it would have as a single-operating device, because it keeps all its original algorithms during the execution of the task. Thereby the layered control approach allows decreasing the overall workload of the control system while preserving the most significant control data flows.

The proposed architecture facilitates usage of different multi-robot task allocation methods in the same system just by realizing additional communication protocols used by agents. It allows comparison and analysis of various task allocation mechanisms in the virtual test bed that is a direction of the future work in order to find the most suitable multi-robot task allocation method for the domain of vacuum cleaning robots.

The proposed approach allows to build systems consisting of multiple robots with very little changes in the robots' hardware. The management system can be used with various robots, like vacuum cleaning robots, robots working in precision agriculture, and other application areas, where a team of robots doing the same task must be built to implement a system that is capable to accomplish larger missions than a single robot. We see that the

proposed architecture might be applied in many other areas where single robot systems are not feasible to use due to their complexity or costs.

The main direction of our future work is further evolution of the management system for the vacuum cleaning robots and testing it in both virtual and real environments. The virtual environment due to strictly practical reasons is used as a test bed for experimental purposes. To implement the fully functional management system that decomposes and assigns tasks optimally, the following research has to be carried out. We need to identify practically applicable optimality criteria and use them in order to implement multi-criteria optimization as an essential part of the system for task assignment. While we intend to build a practically applicable system, it is necessary to develop selflocalization and positioning methods that could cope with data dynamism of the whole system. This work has already been started.

Acknowledgement:
The work has been partly supported by ERAF European Regional Development Fund project 2010/0258/2DP/2.1.1.1.0/10/APIA/VIAA/005 Development of Intelligent Multiagent Robotics System Technology.

References:

[1] Bellifemine F. et al, *Developing Multi-Agent Systems With JADE*, Wiley, 2004. 286 p.

[2] Chaimowicz, L. et al, A paradigm for dynamic coordination of multiple robots. *Autonomous Robots,* Vol. 17(1), Springer, 2004, pp. 7-21.

[3] Dias, M.B., *TraderBots: A New Paradigm for Robust and Efficient Multirobot Coordination in Dynamic Environments*. Doctoral dissertation, Carnegie Mellon University, 2004.

[4] *FIPA Contract Net Interaction Protocol Specification.* Foundation for Intelligent Physical Agents, Geneva, Switzerland, 2002.

[5] Gerkey B.P. *On Multi-Robot Task Allocation*. PhD Dissertation. University of Southern California, 2003, p. 126.

[6] Gonzalez R., et al, Comparative Study of Localization Techniques for Mobile Robots based on Indirect Kalman Filter, *Proceedings of IFR Int. Symposium on Robotics*, Barcelona, Spain, 2009. pp. 253 - 258.

[7] Goodwin, J.R., *A Unified Design Framework for Mobile Robot Systems.* PhD Thesis. University of the West of England, 2008.

[8] Grabovskis A., Concept of Generic Map Visualization Framework. *In Scientific Jouranal of RTU* (accepted), 2012.

[9] Josuttis, N. *SOA in Practice. The Art of Distributed System Design*. O'Reilly, 2007.

[10] Lavendelis E. *Open multi-agent architecture and methodology for intelligent tutoring system development*. Summary of Doctoral Thesis. Riga, RTU, 49 p., 2009.

[11] Lavendelis E., Grundspenkis J. Multi-Agent Based Intelligent Tutoring System Source Code Generation Using MASITS Tool. *Scientific Journal of RTU.* Vol. 43, pp 27-36, 2010.

[12] Liekna A., et al, Analysis of Contract NET Protocol in Multi-Robot Task Allocation. *Scientific Journal of RTU* (accepted), 2012.

[13] Luck M. et al., *Agent Based Software Development*. Artech House, 208 p., 2004.

[14] Bordini R. H. et al, *Multi-Agent Programming: Languages, Platforms and Applications.* Springer, 296 p., 2010.

[15] Orebäck, A. and Christensen, H.I. Evaluation of architectures for mobile robotics. *Autonomous robots, Vol. 14(1),* 2003 pp. 33-49.

[16] Ribes, M.T. Optimization of Floor Cleaning Coverage Performance of a Random Path-Planning Mobile Robot. *Universitat de Lleida. Escola Politècnica Superior. Enginyeria en Informàtica*. p. 31. 2007.

[17] Russell, S. and Norvig, P. *Artificial Intelligence. A Modern Approach.* Pearson Education, 2nd edition, 2003.

[18] Satish Kumar, K.N. and Sudeep, C.S. Robots for Precision Agriculture. *Electronic Proc. of 13th National Conference on Mechanisms and Machines*, Bangalore, India, December, 2007.

[19] Simmons, R.G., Structured control for autonomous robots. *IEEE Transactions on Robotics and Automation, Vol. 10(1),* IEEE, 1994, pp. 34-43.

[20] Tambe, M., Agent architectures for flexible, practical teamwork. *Proceedings of the National Conference on AI,* Providence, Rhode Island, USA, July 27–28 1997, pp. 22-28.

[21] Tribelhorn, B. and Dodds, Z., Evaluating the Roomba: A low-cost, ubiquitous platform for robotics research and education. *Proceedings of Int. Conference on Robotics and Automation*, Rome, Italy, 2007, 2007, pp. 1393-1399.

[22] Thrun S. et al. *Probabilistic Robotics*, MIT Press, 667 p., 2005.

[23] Vanags M., et al. Service oriented mine hunting classroom simulation system. *Applied ICT,* April 2010, Latvia, Jelgava, pp. 95-101, 2010.

[24] Vincent, R. et al. Distributed multirobot exploration, mapping, and task allocation. *Annals of Mathematics and AI,* Vol. 52, No. 2-4., 2008, .pp. 229-255.