A Multi-agent System Architecture for Modular Robotic Mobility Aids

Georgios Lidoris and Martin Buss

Institute of Automatic Control Engineering
Technische Universität München
D-80290 Munich, Germany
Georgios.Lidoris@tum.de

Abstract

In this paper a multi-agent system architecture for a modular mobility enhancement system is presented. The system consists of one or multiple mobile robotic platforms and a set of user defined application modules, such as chairs, tables multifunctional chairs etc. All these modules can be inexpensive everyday life items, that become functional when a mobile platform is properly attached to them. This way the system can act as a wheelchair, as a carrier of objects, or even as a walker. *

1 Introduction

With the increase in the number of senior citizens, there is a growing demand for mobility aids. Older people have problems moving themselves, handling or moving objects and at the same time operating complicated devices. Therefore mobility aids with intelligent capabilities become a necessity. Several robotic, intelligent wheelchairs have been proposed demonstrating navigation, manipulation and transportation capabilities. Examples include Rolland [1] , TAO [2], OMNI [3], NavChair [4] and many other. However existing systems are still difficult to operate, to adjust to the needs of the individual user and are very costly.

At the same time research on multi-agent systems has provided both principles for the construction of complex systems involving multiple agents and mechanisms for the coordination of independent agent behaviours.

In this paper we intend to apply well-analysed concepts from multi-agent and multi-robot systems research, to create a system architecture for a modular mobility enhancement aid. This is a new and very demanding application

* This work was partially supported by the Specific Targeted Research Project MOVEMENT co-funded by INFSO DG of the European Commission (contract number 511670)

H.I. Christensen (Ed.): European Robotics Symposium 2006, STAR 22, pp. 15–26, 2006.

domain for multi-agent systems and besides presenting the proposed approach we intend to expose its specific requirements.

The system can consist of one or multiple mobile robotic platforms and a set of user defined application modules, such as chairs, tables, multifunctional chairs etc. All these modules can be inexpensive everyday life items, that become functional when a robotic mobile platform is properly attached to them. This way the system can act as a wheelchair, as a carrier of objects, or even as a walker. Also multiple levels of autonomy are supported by the system architecture, from manual operation to autonomous functionality. Such a modular system, addresses many diverse needs and can be used by people with different mobility problems. Its scalable multi-agent architecture allows it to be deployed for the home care of an individual, as well as in large health care institutions.

2 Related Work

Several approaches have been proposed in the multi-agent community to coordinate agent actions. Distributed behaviour-based architectures have been developed, like ALLIANCE [5] and [6], which were capable of dealing with complex tasks but lacked on scalability and robustness. Other approaches address multi-robot cooperation through negotiation, such as M+[7], Traderbots [8], MURDOCH [9] and numerous others. Market-based architectures are winning popularity and already a large amount of literature relevant to the field exists. An interesting survey of market based multi-robot coordination can be found in [10].

Our approach uses a negotiation mechanism for task allocation, as well as a behavioural-based layer for the execution of complex tasks. This way the robustness of market-based mechanisms is combined with the ability of behaviour-based approaches to deal with tasks of increased complexity.

A contribution to the challenges of multi-agent systems research as described in [10] is made, by introducing a new real-world application domain where long-term, reliable and robust operation of human, robot teams can be demonstrated.

3 Concept of System Architecture

As we mentioned above our application domain introduces several special requirements. We are dealing with a safety critical domain, where we have to ensure the continuous functionality of our system. Direct task allocation and supervision of system's decisions by the user must be supported. Tasks arrive in an unpredicted manner, from several user interfaces. Finally new modules can be inserted to and removed from the system at any time.

The system architecture is responsible for the coordination of modules consisting the system. It shall enable the user dependent usage of the modules and has the task of controlling the interconnection of modules and their behaviour. More specifically we present a Multi-Agent System architecture which is responsible for receiving high-level commands from the user interface, decomposes them and delegates sub-tasks to the appropriate application modules. These modules are dynamically selected according to utility functions, which are created using performance indicators and task requirements. In the formal multi-robot task allocation (MRTA) framework introduced in [11], our system addresses "Single-task robots, single robot tasks, instantaneous assignment" (ST-SR-IA). Furthermore our system continuously runs background tasks such as managing module power levels, performing conflict resolution between system modules during task executions and finally enabling the system's autonomous functionalities by making use of a behaviour-based task model.

Component selection for task allocation is performed by using a simple distibuted first-price one round auction protocol, similar to the Contract-Net protocol [12]. The progress of task execution is monitored by the winner and if no progress is made, then the task is re-assigned through a new negotiation. This is a very important fault-prevention functionality of the system.

3.1 Agent Model

Taking into account the distributed nature and complexity of the application domain, we decided to split the control of our system amongst a number of deliberative agents. Each agent is capable of reasoning, according to the classical hybrid architecture paradigm [13]. At the same time, by distributing control to a number of agents, we achieve robustness and scalability gains. Therefore our system can be considered as an extension of the layered approach [14], [15], [16] to a multi-agent domain, exploiting advantages of both centralised and distributed approaches.

The general internal structure of a system agent is illustrated in Figure 1. It consists of three parts. The first one is responsible for communicating with other agents and handling communication requests. The second part is a scheduler, which receives task requests, creates a queue and schedules their execution from the third layer. The task execution layer is responsible for the agent's actions, which according to each agent's role can vary from simple communication acts to movements, for our physical agents.

By using such an internal organization, we are able to deal with coordination and cooperation issues that arise between our agents. Our system becomes capable of having physical agents that respond quickly to dynamic events e.g. collisions, while at the same time producing and executing complex strategies for achieving multiple goals. This is due to the fact that each system component is able to schedule its own actions according to its internal capabilities and state.

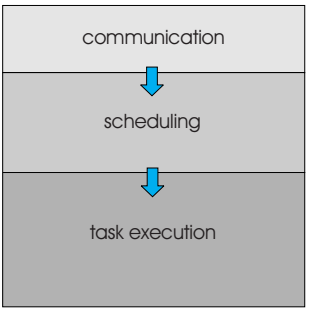

Fig. 1. Internal structure of a system agent

3.2 Overview of the Proposed Approach

The proposed system architecture consists of several software agents that are responsible for performing task decomposition, task allocation through negotiations as well as system monitoring and module management. It also contains physical agents which represent the modules that are available to the system at each time point.

No limitations exist to the number of possible modules that can be inserted to the system, enabling it to be used in small setups with only one mobile base and a few application modules, as well as in large care institutions with an arbitrary number of mobile bases and application modules. In Figure 2 we present an overview of our architectural concept, for a setup where the available modules that can be docked with the platforms are chairs, tables, multifunctional chairs and walker modules.

When the user issues a command via the input hardware of the User Interface, this command is integrated and preprocessed and then sent to a link layer. This link layer between the Multi-Agent System and the User Interface is responsible for transforming the issued command, into an appropriate form in order to be transmitted to the control architecture. Important information being encapsuled are:

- Level of autonomy in which the system is currently operating
- Type of operation to be performed by the system
- Which application module issues the command
- Command specific parameters e.g. goal position coordinates, in the autonomous functionality or speeds to be executed by the mobile platform, read by the input device for the manual functionality

This high-level command is transmitted from the user interface link layer, to the system.

In order to make clear how a command is handled by the system, we have to describe each agent's role in more detail.

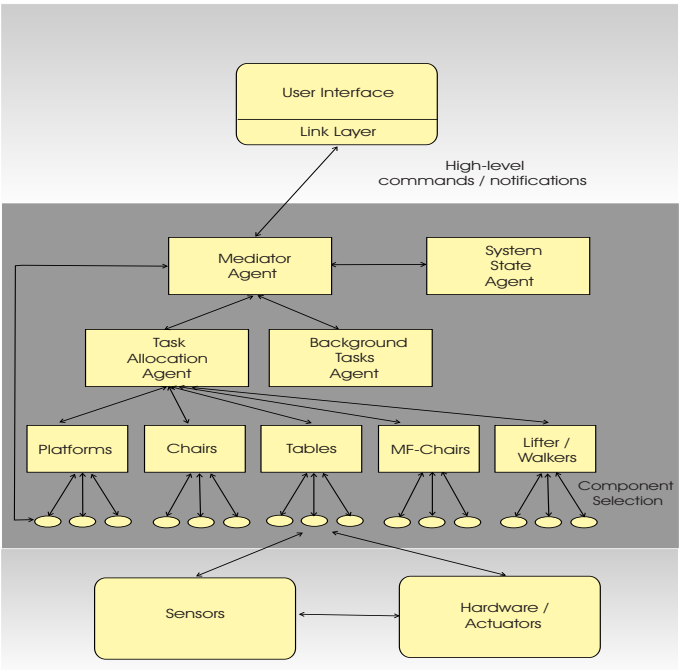

Fig. 2. Overview of the proposed approach

Mediator Agent This multi-purpose agent is responsible for accepting commands from the user interface's Link Layer, initiating task execution procedures and querying for specific module information. Finally, after the task is executed, it informs the interface for changes. It also receives safety notifications from the Background Tasks Agent and initiates actions accordingly, giving them priority.

System State Agent It is responsible for providing management and naming services for the rest of the system and monitoring the state of the system's application modules. It informs, upon request, the rest of the system about the available modules and their state. The possible states of each application module depend on its functionality. For each robotic platform its state is a tuple $< d, t >$ showing if its docked with an application module and which task it is currently undertaking. For an application module its state shows if it is docked with a platform.

Task Allocation Agent If it is assigned a task by the Mediator Agent, then decomposes it, so that the necessary components are chosen and initiates a negotiation procedure similar to Contract Net Protocol, in order to find the best module to perform the task. After it receives bids from the Component Agents it chooses a winner (the highest bidder) and assigns it the task. When it receives information from the Component Agent that the task was performed

it informs the Mediator. In the case of passive modules like tables and chairs we don't have a task assignment but the winner is the module that is more appropriate for a given task. For example if we want a given platform to go and bring a table, then the negotiation finds the most suitable one (e.g. the one that is closer) and the Mediator is informed, so that it can assign the task to the platform.

Background Tasks Agent Monitors system components and issues commands to the Mediator Agent, in order to deal with safety issues. For example, we want to avoid the risk of power failure of our mobile robotic platforms during task execution. Therefore the Background Tasks Agent constantly checks the power level of all mobile platforms. If a platform's power level is found less than an acceptable minimum, then a Charge command is issued to the Mediator and the platform in question charges its batteries. It also can be used for setting out alerts for pre-programmed system maintenance operations, module failures etc. Generally the commands issued by this agent to the Mediator Agent have higher-priority as normal commands.

Component Agents Each of these agents is created when a new hardware module is inserted to the system and it represents this module to the MAS. They are involved in the component selection negotiation procedure. If they are assigned a task they initiate a pre-programmed set of behaviours in order to accomplish it. The task description must contain information about its type and also other specific information regarding possible parts of the task. These agents have control over the hardware they embody and pass appropriate commands to it, by sequencing high level movement descriptions into low level commands. When the level of autonomy is set to manual, they just pass low level commands that are received from the UI to the hardware.

Passive modules such as chairs, tables, etc, do not perform tasks themselves but need to be docked with a platform. They take part to the negotiations initiated by the Task Allocation Agent when we need to find the most suitable module, in order to minimize resource usage. For example let's suppose that the user wants a given platform to bring a table. Table Component Agents compute their utilities based on their distance from the platform. This is useful especially for home use scenarios, where only one platform exists but many tables and chairs, which are really inexpensive.

Platform Component Agents rely on a behaviour based system. Behaviours are seen as condition / action pairs where conditions are logical expressions over the inputs and actions are themselves behaviours. In both cases, a behaviour is a directed acyclic graph of arbitrary depth. The leaves of the graphs are the behaviour type's respective outputs.

This notion of behaviour is also consistent with that laid out in [17] and [18]. Behaviours are nested at different levels: selection among lower-level behaviours can be considered a higher-level behaviour, with the overall agent behaviour considered a single "do-the-task" behaviour. These resulting high-level tasks are also interconnected with each other and all together consist

our architecture's task model. Figure 3 illustrates the interconnection between these tasks and how they are decomposed to simpler behaviours.

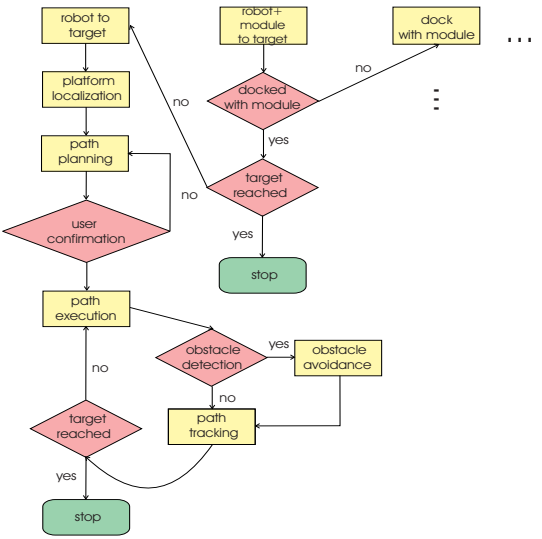

Fig. 3. Task model

When a robotic platform is assigned a task, it becomes unavailable for further tasks. This means that it does not take part in negotiations. Instead it monitors task execution and if no progress is made, it contacts the Task Allocation Agent to re-assign the task. Monitoring of task execution can simply be made by recalculating the utility of the task, in a given time space. If the utility remains unchanged or decreases below a given threshold, task execution must be terminated and the task has to be re-allocated. In the following section we will give a definition of task utility.

4 Utility for Task Assignment

Component selection based on performance measures and task requirements is of great importance for our system. Imagine use cases where multiple modules of the same functionality exist. For example a user may have more than one table modules, which are really inexpensive, in his house. The system must then be in position to choose the best module for a given task e.g. in a fetch and carry scenario, where the mobile platform has to dock with a table, reach a goal location and return to the user, the best module in order to minimize resource usage would be the one closer to the goal location. We are dealing here with an instance of the Optimal Assignment Problem (OAP) [19] that was

originally studied in game theory. A formulation of this problem for robotics can be found in [20].

To address this problem we use the concept of utility. The idea is that each individual can internally estimate the value of executing an action. This estimation results a utility function which is highly domain and task dependent. An instructive definition of utility follows [11]. It is assumed that each agent / hardware module is capable of estimating its fitness for every task of which it is capable. This estimation takes into account two task- and robot-dependent factors:

- expected quality of task execution, given the method and equipment to be used
- expected resource cost, given the spatio-temporal requirements of the task (e.g. the length of the path to a target etc)

So given a module M and a task T, if M is capable of executing T, then we can define $Q_{MT}(p)$ and $C_{MT}(d)$ as the quality and cost, respectively, expected to result from the execution of T by M. Quality of task execution can be seen as a function of application module specific parameters, denoted by p. An example can be the power level of the battery of a robotic mobile platform. If the power levels are low then the platform will have to charge soon, therefore becoming temporary unavailable to execute tasks. Cost can be seen as a function of the path length from the goal position. A non-negative utility measure can now be defined.

$$U_{MT} = \begin{cases} Q_{MT}(p) - C_{MT}(d) & \text{if M is capable of executing T and } Q_{MT}(p) > C_{MT}(d) \\ 0 & \text{otherwise} \end{cases}$$

Each agent's objective is to maximize utility. Utility maximization is equivalent in our case with effective resource usage and therefore better system performance. We can incorporate learning into utility estimation, therefore adapting the system's performance to the user behaviour and needs.

5 Experimental Design

To validate our system's functionality and take some first simulation results, we have created an experimental testbed. For the implementation of our agents we decided to use JADE (Java Agent Development Framework) as a middleware. JADE is a software development framework aimed at developing multi-agent systems and applications conforming to FIPA (Foundation for Intelligent Physical Agents) standards for intelligent agents. Using JADE as a middleware basically means that we take advantage of its communication structures and naming directory services, in order to develop our experimental set-ups effectively and with respect to the standards of the agent research community. This way we are able to analyse interactions between agents, as well as decision-making procedures. For the simulations we have used the

Player/Stage simulator. The structure of our experimental implementation is illustrated in Figure 4.

From the discussion above it has been made evident that a large number of possible system states and setups can exist, making it almost impossible to simulate all of them. For that reason we had to choose some scenarios to be simulated which would test the task allocation and fault prevention mechanisms of our architecture. The following three were chosen:

- one mobile robotic platform, two different types of application modules and multiple tasks to be performed autonomously
- multiple mobile robotic platforms, two different types of application modules and one single task to be performed autonomously, with robot malfunction and task reallocation
- multiple mobile robotic platforms, two different types of application modules and multiple tasks to be performed autonomously

Results from the simulations are very encouraging, with the system assinging tasks to modules in an efficient manner. Even when task execution was interrupted on purpose by us, tasks were re-allocated succesfully. Agent communication is optimal posing almost no communication overhead. More specif-

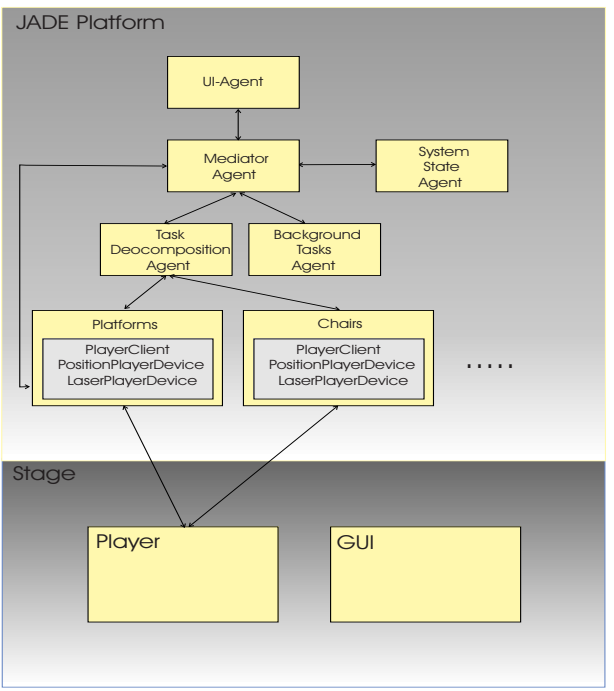

Fig. 4. Experimental implementation

ically, agent interactions are shown in Figure 5 for the task of driving an initially undocked chair ($ch1$) to a given destination ($target$). FIPA Agent Communication Language performatives are used.

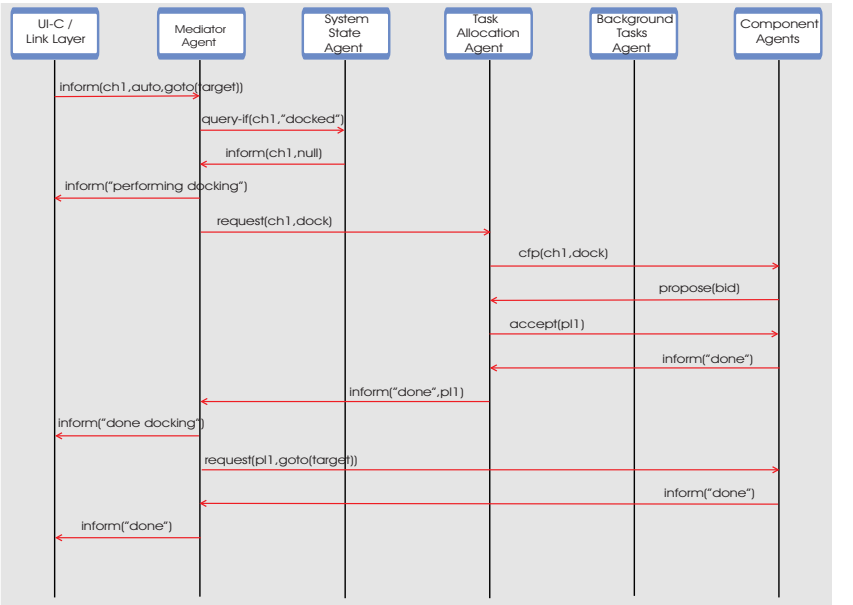

Fig. 5. Agent interactions while performing a complex task

At first Mediator Agent checks with the System State Agent whether the requested chair is docked with a mobile platform or not. The answer is negative, so a module selection procedure is iniated by the Task Allocation Agent in order to find the most suitable platform ($pl1$ in Figure). When it is found, it is ordered to dock with the chair and then reach the defined destination. The User Interface is appropriately updated by the Mediator while task execution is progressing. We can see that only few messages are used to accomplish a complicated task.

6 Conclusions

We have defined a Multi-Agent System architecture for modular mobility aids in terms of software architecture, modelling and requirements. We have shown how components interact with each other in order to achieve effective functionality of the whole system. A quantification of each module's ability to undertake a task was also presented and used in a negotiation based module selection mechanism. Throughout this effort, our main concern was simplicity

and functionality, given the safety critical nature of the application domain. Finally, a complete experimental testbed has been presented which has been used to validate our concepts.

Since this is a new and demanding application domain much work remains on the architecture. We need to develop complete performance measures and enhance the system with learning mechanisms that will allow it to adapt its performance to the user's behaviour. We also have to look deeper into mechanisms for fault prevention and system recovery. Finally, tests on physically embodied robots will be made.

References

1. Mandel C, Huebner K, Vierhuff T (2005) Towards an Autonomous Wheelchair: Cognitive Aspects in Service Robotics. In Proceedings of Towards Autonomous Robotic Systems (TAROS 2005): 165-172
2. Gomi T, Griffith A (1998) Developing intelligent wheelchairs for the handicapped. In: Assistive Technology and Artificial Intelligence: Applications in Robotics, User Interfaces and Natural Language Processing
3. Hoyer H (1999) The OMNI-Wheelchair - State of the Art. In: Proceedings of Conference on Technology and Persons with Disabilities, Los Angeles
4. Simpson R et al (1998) Navchair: An assistive wheelchair navigation system with automatic adaptation. In: Assistive Technology and Artificial Intelligence: Applications in Robotics, User Interfaces and Natural Language Processing
5. Parker L (1998) Alliance: An architecture for fault tolerant multirobot cooperation. In: IEEE Transactions on Robotics and Automation, 14(2): 220-240
6. Simmons R, Smith T, Dias M B, Goldberg D, Hershberger D, Stentz A, Zlot R (2002) A layered architecture for coordination of mobile robots. In: Multi-Robot Systems: From Swarms to Intelligent Automata
7. Botelho S, Alami R (1999) M+: A schema for multi-robot cooperation through negotiated task allocation and achievement. In: Proceedings of the IEEE International Conference on Robotics and Automation: 1234-1239
8. Zlot R, Stentz A, Dias M B, Thayer S (2002) Multi-robot exploration controlled by a market economy. In: Proceedings of the IEEE International Conference on Robotics and Automation: 3016-3023
9. Gerkey B, Mataric M J (2002) Sold!: Auction methods for multi-robot coordination. In: IEEE Transactions on Robotics and Automation, 16(5):758-768
10. Dias M B, Zlot R M, Kalra N, Stentz A (2005) Market-Based Multirobot Coordination: A Survey and Analysis. Tech report CMU-RI-TR-05-14, Robotics Institute, Carnegie Mellon University
11. Gerkey B, Mataric M J (2004) A formal analysis and taxonomy of task allocation in multi-robot systems. In: Intl. Journal of Robotics Research, 23(9):939-954
12. Smith R G (1980) The Contract Net Protocol: high level communication and control in a distributed problem solver. In: IEEE Transactions on Computers, C-29(12)
13. Gat E (1997)On three-layer architectures. In: Artificial Intelligence and Mobile Robots. MIT AAAI Press

14. Bonasso P, Kortenkamp D (1995) Characterizing an architecture for intelligent, reactive agents. In: Working Notes, AAAI Spring Symposium on Lessons Learned from Implemented Software Architectures for Physical Agents: 29-34
15. Gat E. (1991) Integrating planning and reacting in a heterogeneous asynchronous architecture for mobile robots. In: SIGART Bulletin 2
16. Connell J (1992) A hybrid architecture applied to robot navigation. In: Proceedings of IEEE ICRA '92, Nice France
17. Stone P (1998) Layered learning in multi-agent systems. Phd Thesis, Carnegie Mellon University
18. Mataric M (1994) Interaction and intelligent behavior. Phd Thesis, MIT AI Lab
19. Gale D (1960) The theory of linear economic models. McGraw-Hill Book Company, New York
20. Gerkey B (2003) On multi-robot task allocation. Phd Thesis, University of South California, Los Angeles