BULLETIN OF THE POLISH ACADEMY OF SCIENCES
TECHNICAL SCIENCES, Vol. 58, No. 1, 2010
DOI: 10.2478/v10175-010-0001-y

Multirobot system architecture: environment representation and protocols

S. AMBROSZKIEWICZ[1,2]*, W. BARTYNA[1,2], M. FADEREWSKI[1], and G. TERLIKOWSKI[2]

[1] Institute of Computer Science, Polish Academy of Sciences, 21 Ordona Ave, 01-237 Warsaw, Poland
[2] Institute of Computer Science, University of Podlasie, 51 Sienkiewicza St., 08-110 Siedlce, Poland

Abstract. An approach to the problem of interoperability in open and heterogeneous multirobot system is presented. It is based on the paradigm of Service Oriented Architecture (SOA) and a generic representation of the environment. A robot, and generally a cognitive and intelligent device, is seen as a collection of its capabilities exposed as services. Several experimental protocols (for publishing, discovering, arranging, and executing the composite services) are proposed in order to assure the interoperability in the system. The environment representation, the description language for tasks and service interfaces definition, as well as the protocols constitute together the proposed information technology for automatic task accomplishment in an open heterogeneous multirobot system.

Key words: multirobot system, interoperability, environment representation, SOA.

1. Introduction

Rapid development and ubiquitous use of intelligent devices (equipped with sensors, microcontrollers, and connected to a network) pose new possibilities and challenges in the Robotics and Information Technology. One of them is creating large open distributed systems consisting of heterogeneous devices that can interoperate in order to accomplish complex tasks. An example is Ambient Intelligence (AmI) that is currently one of the most explored research areas; see [1] and [2]. The general idea of AmI is that intelligent cognitive devices, in a human environment, are able to interoperate to perform a complex task delegated by people. Ubiquitous robotics is a similar idea resulting from development in the ubiquitous computing and the network technology. It claims that in the near future humans will live in a world where all devices are fully networked, so that any desired service can be provided at any place at any time.

These ideas require new information technologies for developing distributed systems that allow defining tasks in a declarative way by human users and automatic task accomplishing by the system. Openness and heterogeneity of the system are essential here because they enable extensibility and scalability, that is, heterogeneous devices may be added to (or removed from) the system, in the plug and play manner, so that the system may grow to a large size and still keep its basic functionality, (i.e. an automatic task accomplishing), if there are still sufficient services in the system. The key problem here is a representation of the environment common for people and other system components, that is, the devices.

Existing approaches to such systems are based either on a direct controlling the devices via network (tele-operating), or by using distributed objects technologies (like CORBA or Java RMI). These systems are closed and dedicated to a fixed collection of devices and a class of task to be performed there. The examples include systems consisting of homogenous robots executing similar actions (called swarm robotics, see [3]) and systems consisting of heterogonous robots which can share mutually data from their sensors, see [4]. There are also some attempts to apply Semantic Web and Web Services technologies to multirobot systems, see [5] or directly based on SOA architecture [6]. The first one, the URSF project, tries to directly apply technologies from the Web Services family (OWL-S, WSDL, SOAP, and BPEL stack) in the domain of Robotics. That kind of approach leads to problems like: the necessity to build a separate knowledge, base for every location where the services can be provided, lack of the possibility to define tasks in a declarative way, and impossibility to compose services provided in different locations. The PalCom project [6] offers a solution to easy integration of small digital devices (seen as services) in groups called assemblies. However, it is designed only for cognitive and computing services so that the exchange of information is simply based on appropriate data types. This is the reason that this approach cannot be applied in multirobot systems consisting of physical services that can change the state of the environment, where the local states also must be described, and the description exchanged between system components. Although these approaches are based on SOA (as the one presented in this article) they lack common and generic structure of representation of the environment of multirobot systems, and the language describing the representation. The language is necessary for the task specification and communication between heterogeneous devices. It seems that this very representation and the description language are crucial for achieving interoperability in multirobot systems. The interoperability presupposes a communication (defined by protocols) between heterogeneous components of the system concerning task delega-

*e-mail: sambrosz@ipipan.waw.pl

tion, planning, and its joint realization. The communication, in turn, presupposes a common formal language describing the environment. However, the language (understood as its syntax only) is not sufficient; it must be grounded on the environment, that is, it must have precisely defined semantics. The environment representation mentioned above serves as that semantics. In order to assure the understanding between the communication partners, the representation must be common for them, as well as for people who delegate tasks to the system to be executed. Hence, a universal representation of the environment and the corresponding description language are the basis for defining protocols needed for achieving interoperability in multirobot systems.

In this paper a new experimental information technology for distributed systems based on SOA paradigm is proposed. Service Oriented Architecture is a modern approach to designing flexible, open, and scalable distributed systems. The proposed technology, (consisting of a common environment representation, a common language for environment description, and communicational protocols), is a basis for developing multirobot systems based on services.

2. General architecture of service oriented multirobot system (SOMRS)

In the proposed architecture, capabilities of devices are considered as services. Each service is able to perform some action or function, like the transport service which moves an object from one place to another, or search service performed by a sensor network (or a mobile robot equipped with sensors) that can recognize and localize an object based on given set of its attributes. Another service example is a software application performing special data processing.

Services should publish their ability in the form of the type of operation they perform, preconditions necessary for operation execution, and effects (postcondition) of its execution. The service ability, defined in this way, is called service interface. In the case of a transport service provided by a mobile robot, the precondition is a description of places where an object may be initially, whereas the postcondition describes the class of situations where the object may be after the service execution. Each service interface is specified in the language describing the common representation of the environment. For a given interface, there may be many different implementation of the operation specified by the interface. Hence, each interface is independent from the implementation of the operation it specifies.

The classic version of the SOA paradigm, see [7], may be summarized as follows.

SOA provides a standard programming model that allows self-contained, modular software components residing on any network to be published, discovered, and invoked by each other as services. There are essentially three components of SOA: Service Provider, Service Requester (or Client), and Service Registry. The provider hosts the service and controls access to it, and is responsible for publishing a description of its service to a service registry. The requester (client) is a software component in a search of a component to invoke in order to realize a request. The service registry is a central repository that facilitates service discovery by the requesters.

In the multirobot setting, a service, (performed by an intelligent device), is identified with the special dedicated software component residing on the device microcontroller (or somewhere in the Internet) that can control the service performance.

The general purpose of a service-oriented multirobot system (SOMRS for short) is to realize client's intentions by appropriately changing situation in the physical environment, i.e., by executing a collection of services. Usually the client is a human user, however sometimes it may be a software application. In the proposed SOMRS architecture, see Fig. 1, Task Manager (TM) is responsible for system interactions with the user. This component provides a graphical user interface (GUI) for defining a user's intention, translates it to a common communication language, and delegates it (as a task) to an Agent to realize. In the final phase, TM notifies the user on the situations emerged during the task realization. After receiving a task, the Agent discovers services (via Service Registry (SR)) that (when composed) could jointly perform the task. Then the Agent arranges (with the discovered services) conditions of their execution and composes them into a process of task realization.

A new service becomes available in the system, if it publishes its interface to the Service Registry.

Representation of the physical environment (in the form of object maps) is stored in the Repository. It provides the maps to the Task Manager (for intention formulation), to the Service Registry (for determining services requested by an Agent for task realization), and to some services that need them for localization and navigation.

Figure 1 presents the general schema of the proposed multirobot system architecture, its components, and interactions between them.

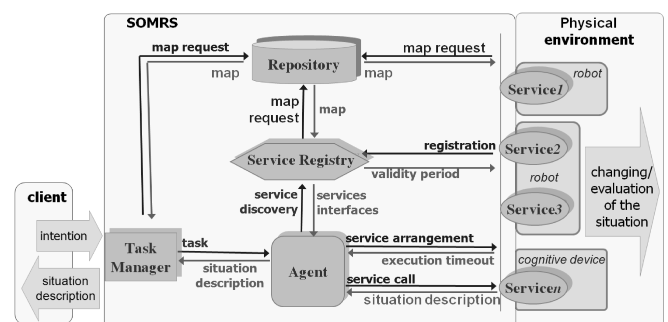

Fig. 1. The SOMRS system architecture

3. Environment representation

The basis for providing interoperability of heterogeneous devices is universal common representation of the environment for humans and devices. Classic representations in robotics (see e.g. [8]) are based on metric and topological approaches dedicated mostly to tasks related to navigation. Another approach, Spatial Semantic Hierarchy (SSH) [9], is based on

the concept of cognitive map and hierarchical representation of spatial environment structure. Recently, there have been object-based approaches (see [10]) where the environment is represented as a map of places connected by passages. Places are probabilistic graphs encoding objects and relations between them. The main problem here is an automation of the process of creating a map by recognition and classification of objects applying probabilistic methods. For example, an object, recognized (with some probability) as a refrigerator, is supposed to belong to the class of kitchen. In the paper [11], the environment representation is composed of two object hierarchies; the first one (called spatial) related to sensor data in the form of object images or occupancy grid, and the second one (called conceptual) related to some abstract notions of the representation. The recognition of places and objects consists in matching sensor data against the abstract notions. A variation of this approach is presented in [12], where the probabilistic methods are used for object recognition.

Yet another approach was created for the needs of civil engineering; it is called Building Information Model (BIM), see [13]. It is dedicated to represent physical and functional characteristics of buildings. BIM can be potentially used for robot navigation, however it is complex. The approach proposed in this paper is closely related to BIM, however, it is much simpler because it is based on the concept of cognitive maps that neglects a lot of technical details important for civil engineering, but not for people in their everyday life.

Special attention deserves the approach proposed 20 years ago by C. Zieliński, see [14–15], and [16]. It is based on the notion of object defined by attributes, and relations between the objects. The representation was created mainly for an environment of robot manipulators, and it was the semantics of the robot programming language TORBOL.

Actually, the approach presented in the paper is based on the general idea proposed by C. Zielinski. However, it introduces additional hierarchy between objects, and abstract objects like a space in a domestic premise.

The representation, proposed in the paper, is expressed in XML; that is, attributes, relations, object types, and objects are XML-structures. There is also the formal language, called Entish, see [17] for describing local situations in the environment.

4. The concept of object maps

In the Computer Science related to Robotics, the term "ontology" is equivalent to the "general structure of the representation of a multirobot system environment". The most popular definition of ontology was given by Tom Gruber in 1993 (see [18]) in the following way: ontology is a specification of a conceptualization. Conceptualization is understood here as an abstract and a simplified model (representation) of the real environment. It is a formal description of concepts (objects) and relations between them. Since the model is supposed to serve the interoperability, it must be common and formally specified, i.e., the definitions of objects and relations must be unambiguous in order to be processed automatically.

In the proposed representation, it is supposed that each object is of some predefined type. Object is determined by a collection of attributes, and optionally its internal hierarchical structure consisting of sub-objects and relations between these sub-objects. The objects that do not have internal structure are called elementary, and their types are called elementary types. The object of the WALL type may serve as an example of elementary object. It is determined by the attributes: width, height and colour. The type ROOM may be an example of a complex type; its internal structure is composed of elementary object such like walls, floor, ceiling, windows, and doors, as well as the relations between these objects.

The general structure of the proposed representation of the environment of a multirobot system is defined as a hierarchical collection of object types. An elementary type is defined as a collection of attributes with restricted ranges, whereas a complex type is defined by previously defined elementary types of its sub-objects, and relations between the sub-objects. The type BUILDING may serve as an example of a complex type; its general internal structure consists of several storeys, stairs, lifts, rooms, halls, passages, and so on.

Hence, the primitive attributes and primitive relations are the basic elements for building representation, i.e., construction of object types. A particular object (as an instance of its type) is defined by specifying concrete values of its attributes, and (if it is of complex type) also by specifying its sub-objects. An instance of the general structure (called also a map of the environment) is defined as a specification of an object of a complex type, for example, of the type BUILDING. In order to support a possibility of automatic map building and updating by mobile robots, appropriate primitive attributes and relations must be measurable and recognizable by means of sensors which the robots are equipped with.

5. Specification of service interface

In multirobot systems, the robotic devices can provide different kinds of services:

1. Physical services – changing situation in the physical environment.
2. Cognitive services – that can recognize a situations (e.g. perceive the location of a given object in relation to other objects), or evaluate a situation described in the common language, e.g. check if object having some features is in a given place.
3. Software services – that process data.

An interface of a service is defined as a formula of the language Entish [17]. Although the syntax of Entish is expressed in XML, for the sake of presentation, semiformal version of the syntax will be used. The language is a simplified version (without quantifiers) of first order logic, that is, it has logical operators (*and*, *or*, *implies*), names of relations (e.g., preconditions, postconditions), names of functions (e.g., action, range), and variables. An interface formula consists of the description of the initial situation, the final situation, the name of the abstract action that the service realizes, and the range of the service.

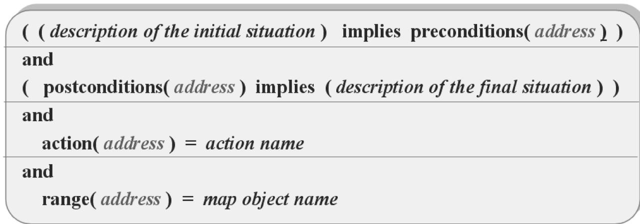

Fig. 2. Schema of a physical service interface

In Fig. 2, the interface formula is the conjunction of four sub-formulas, where a service is identified by its network address (*address*), that is, an IP address combined with a port number. In the first two sub-formulas are implications. In the first one, *"description of the initial situation"*, is the formula which implies the entry condition (*preconditions*) of the service necessary for its invocation. In other words, if the formula "*description of the initial situation*" is satisfied, then the service can be executed. The second implication means that after the service performance, the formula "*description of the final situation*" becomes true. The third sub-formula identifies the name of the type of action (*action*) realized by the service, e.g. in the case of a transport service performed by a mobile robot, the action may consist in pushing an object, or griping an object, going to a place, and then lowering it. An action type is added to the interface to simplify the reasoning to be done during search for services that can realize a task.

The last sub-formula contains information about the range of the service. It is a place where the device (providing the service) operates. The range is defined as an identifier of an object in the map of the environment, e.g., it may be a fixed room in a building. The service range is crucial for planning a task accomplishing.

6. Task definition

General schema of a task specified in the common language is presented in Fig. 3.

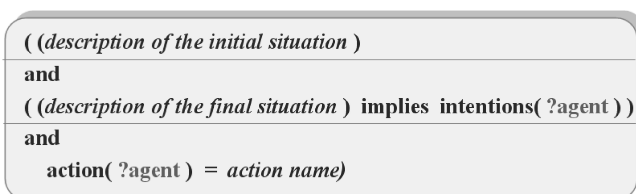

Fig. 3. Schema of a task

A task formula is a conjunction of the three sub-formulas. The first one is optional and is a description of the initial situation related to the task. The second one describes, in a declarative way, the intended final situation in the environment by an agent denoted here by "*?agent*". In other words, the intention of the agent will be realized if the formula *"(description of the final situation)"* becomes true. The last formula specifies (for an agent) the type of an action to be performed in order to accomplish the task. The parameter "*?agent*" is a variable and all variables in the Entish language are preceded by a question mark.

The main purpose of the multirobot system is to realize user's intention. In order to do so in an automatic way (without the need to specify the concrete plan by the user) the appropriate plan must be generated.

Generally the planning problem is specified as follows. „Given a set of actions, their preconditions and positive and negative effects, a complete description of the initial state and a user goal, find a sequence of actions achieving the goal". In other words, a typical planner takes three inputs: a description of the initial state, a description of the desired goal, and a set of possible actions, all encoded in a formal language. The planner produces a sequence of actions that lead from the initial state to a state meeting the goal.

The most famous planner (actually the first one) is STRIPS (Stanford Research Institute Problem Solver [18]), where state descriptions are restricted to the no complex formula, that is, conjunctions of (variable-free) positive propositions or First-Order Logic. Additionally "The Closed World Assumption" is used, that is, which is not given explicitly as true is false.

The action description language (ADL) is an automated planning and scheduling system in particular for robots. It is an advancement of STRIPS. The sense of a planning language is to describe certain conditions in the environment, and, based on these, automatically generate a chain of actions which lead to a desired goal. A goal is a partially specified condition. Before an action can be executed, its preconditions must have beeen fulfilled. The action yields effects satisfying the goal. The environment is described by means of predicates, which are either fulfilled or not. Contrary to STRIPS, the principle of the open world applies to ADL: everything not occurring in the conditions is unknown (instead of being assumed false). In addition, while in STRIPS only positive literals and conjunctions are permitted, ADL allows negative literals and disjunctions as well.

PDDL (Planning Domain Definition Language [19]) is considered now, as "the standard" language for planning problems. It is an extension of STRIPS and ADL, where typing, durative actions, and hierarchical planning were added.

The planning algorithms, designed for these languages (created especially for the planning domain), can also be applied to the language used in the SOMRS system. The Entish language, used to describe the environment representation, is also based on the descriptions of the situations (spatiotemporal states) in the environment. Service interfaces (excluding the information about how to communicate with the service) can be treated as PDDL's operators, which also have precondition and effect (*postcondition*). The services (like the operators) can be appropriately sequenced in a plan leading to the realization of the client's intention. The intention of the client can be treated as a goal for a planning algorithm, although the forward planning may not always be possible because the specification of the initial situation is not obligatory.

The Entish language allows describing the features (attributes) of the objects. So that one can define a task like: move all objects of type T and of weight w and color c from location x to location y. This feature is also used when defin-

ing a service interface to include information about the service restrictions, e.g. the transport service $S1$ cannot move objects of width greater than x centimetres (this restriction may be caused by the physical features of the gripper that the device is equipped with). This enables choosing services based on the concrete task and restrictions of the services during service discovery phase.

An important property of the proposed approach stems from introducing the arrangement phase during the process of realization of an intention. Every step of a plan of the intention realization must have been arranged before the plan can be executed. The arrangement allows for:

- Checking if the plan can be executed in the currently available set of services.
- Choosing the best service for each of the plan steps.
- Agreeing on the condition of the service execution (e.g. time requirements).

Because of this, the initial plan must be abstract, operate on services types, not on the concrete services. The abstract plans can be generated in the similar way as concrete plans. The difference is that the planning algorithm should be based on abstract interfaces of service types. An advantage of this approach is that the abstract plans can be reused for each of the client's intentions of the same class, e.g. transport tasks. Other solution to planning problem is to generate "manually" such abstract plans, e.g. for classes of intentions that are used most often. A manual preparation of the plan allows for including rules defining the behaviour in the case of occurrences of exceptional situations. This cannot be achieved by using known planning algorithms. One way of solving this problem automatically, with the exceptions, is to repeat a planning process where the current exceptional situation is treated as the initial state of the task.

7. Protocols

In distributed systems, the classic definition of a *communication protocol* specifies the format of messages exchanged between two or more communicating parties, their order, as well as the actions taken when a message is sent or received. As usually, the message format consists of the header and the body. The header includes information about the sender, the recipient, the message type, the session identifier, and the name and version of the protocol. The body contains formulas of the common description language. The interpretation of the formulas depends on the type of the message. In the proposed protocol, there are ten message types that are grouped into five request-response pairs. Each pair is related to a different phase of the protocol:

1. The message types of *service registration phase*: *register-request* (*R-RQ*) and *register-response* (*R-RS*).
2. The types of *task delegation phase*: *task-request* (*T-RQ*) and *task-response* (*T-RS*).
3. The types of *service discovery phase*: *info-request* (*I-RQ*) and *info-response* (*I-RS*).
4. The types of *service arrangement phase*: *service-request* (*S-RQ*) and *service-response* (*S-RS*).
5. The types of *service execution phase*: *execute-request* (*E-RQ*) and *execute-response* (*E-RS*).

The event of sending a message of the *type* type is denoted by S_{type}, while receiving by R_{type}. The term "negative message" (used in the sequel) means that the message contains only one formula *false* in its body. Particular phases of the proposed protocol, related to the components of the SOMRS, are presented in the form of finite automata in the following sections.

7.1. Task manager. Task Manager is responsible for realizing intentions received from a client. The client can be a user or a software application. An intention is a situation description specified, by the client, in the common language and based on the common environment representation. The intention defines, in a declarative way, the situation in the environment required by the client; e.g. an object is placed in a fixed location. The realization of the intention becomes the goal of the Task Manager that takes appropriate actions, which are shown in Fig. 4 as a finite automaton.

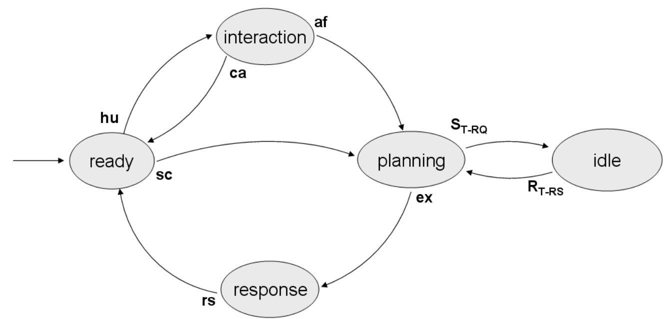

Fig. 4. Finite automaton of the Task Manager

The automaton represents states of the Task Manager and their transition as the responses to events. The states are explained below.

- *ready*: In this state TM waits for an intention from software client (event *sc*) or for human user (event *hu*).
- *interaction*: In this state human user specifies its intention. TM should provide an appropriate graphical user interface (GUI). In the case of cancellation (event *ca*) TM returns to the *ready* state.
- *planning*: After user's affirmation (event *af*), or receiving an intention directly from software client (*sc*), TM performs planning, and then follows the plan to realize the intention. The plan consists of a collection of tasks to be sent to an Agent in messages of type *task-request* (event S_{T-RQ}). Each task is sent separately. After receiving the response (R_{T-RS}), TM analyses current situation after the task execution and decides what to do next.
- *idle*: In this state TM waits for the response from Agent.
- *response*: In this state TM informs the client on the situation after tasks execution. TM moves to this state if the

event (*ex*) occurs, that is, either the plan was realized successfully, or it is not possible to appropriately modify the plan after occurrence of an exceptional situation. After responding to the client, event (*rs*), TM comes back to the *ready* state.

After receiving an intention, and having the object maps of the physical environment from the Repository, Task Manager generates a plan to realize the intention. The maps contain the names of the objects, values of their attributes, relations between these objects, and structures of complex objects. The automatic plan generation, which relies on reasoning in the common language, is based on the initial and final situation of the intention and the abstract interfaces of the available services types.

The generated plan is a collection of abstract tasks (that should be specified, arranged, and executed in a specified order), and a collection of rules defining TM behaviour in the exceptional situations. An abstract task is a formula describing only the action realized by an appropriate type of service without restrictions, specific for the concrete services of this type. Then an abstract task with the detailed description of the initial and final situation is sent to the Agent for arrangement and execution.

An exceptional situation takes place after unsuccessful task execution, that is, if that situation is different from the intended final situation. A formula describing an exceptional situation is sent to TM by the Agent. Handling of the exceptional situations is done by applying rules being actually predefined emergency plans. For example, a rule may require repeating the same task or modifying the initial and/or the final situation of the task. The level of complexity of the exceptional situation, handling, depends on assumptions made by the TM designer.

The tasks are sent by TM to the Agent, in the order specified in the plan, as the content of a message of the type *task-request*. The Agent responds with a message of the type *task-response* which contains a description of situation related to a given task. After receiving results of the task execution, TM may analyze it and send to Agent the next task to be executed, according to the plan, or apply one of the exceptional situation rules, if necessary.

For example, let a situation, where an object (having some fixed features) is placed in some location, be the client's intention. This intention can be realized by a plan consisting of a search task and a transport task. The search task will be executed when the current location of the object is not known; i.e. the client has not specified the initial situation of the intention. In this case, the description of the situation (related to the object) obtained after executing the search task becomes the initial situation of the transport task (if the object was found). An exceptional situation may occur, if after the execution of the transport task, the object is not in the desired location. A rule, that can be applied here, is to execute a new transport task in which the current situation of the object becomes the initial situation, whereas the final situation remains the same.

7.2. Service Registry. The Service Registry (SR) is a system component which gathers, stores, and provides information about services available in the system. It can be represented as a finite automaton (see. Fig. 5) having the following states.

- *ready*: In this state the Service Registry is ready to work, and waits for a message.
- *registering*: After receiving a registration request from a service (R_{R-RQ}), SR stores the service interface, responds with the message having validity time of the service registration entry (S_{R-RS}), and goes back to the state of readiness.
- *searching*: After receiving an information request from the Agent (R_{I-RQ}) containing a task formula, SR searches (in its registration database) for an interface formula that corresponds to the task formula. Appropriate interface formulas are sent back to the Agent as the contents of the response message. Then SR goes back to the state of readiness.

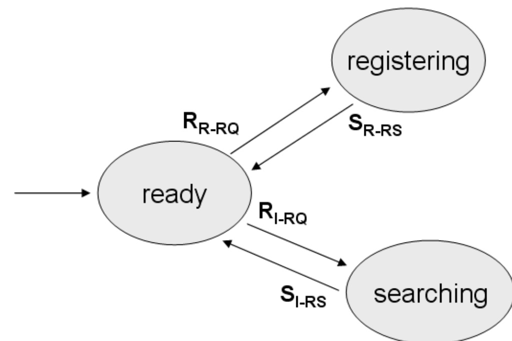

Fig. 5. The automaton for Service Registry

A service (that wants to be discovered and used by the SOMRS system) sends its interface formula to the Service Registry in the message of the type *register-request*. The initial situation of the formula may contain restrictions concerning the operation performed by the service. In the case of a transport service performed by a mobile robot, (that moves objects from one location to another), the restrictions may concern (due to the limitation of the robot gripper) the size of moved objects and their weight.

In the state *searching*, SR matches task formula (sent by the Agent in a message of the type *info-request*) against the interface formulas stored (registered) in its database. This matching may be quite complex, because of the ranges of the services, and the fact that the task may require several services to be realized. Hence, SR must perform a planning using an object map of the environment where the task is supposed to be realized. Finally, SR sends to the Agent a message of the type *info-response* containing the plan of the task execution. If there are no services that can realize the task, the response contains *false* formula

The plan consists of interface formulas of the services for each step of the plan (called subtasks), and a set of relations describing the dependencies between the subtasks, e.g., the execution order. So a subtask is an element in the plan of the task execution generated by the Service Registry. It is defined

in the same way as a usual task. The difference is that all the subtasks in a plan of the main task execution realize the same type of action as the main task. This is because the plan is generated in cases where there is no single service that can execute the main task but there exists a way to do it by a collection of services with appropriate ranges. For example, the main task of moving an object from one location to another (that can not be executed by a single transport service) may be divided into several transport subtasks.

7.3. Agent. The Agent is a system component that executes tasks in a universal way independent from the type of the tasks. The tasks are sent to the Agent by Task Manager. They can also be sent by services, for example, a transport service that wants the door (on its route) to be opened. In this case, the service becomes a task manager from "the Agent's point of view". The Agent executes each task according to the finite automaton shown in Fig. 6. Its states are as follows:

- *ready*: In this state the Agent waits for a task.
- *initialization*: In this state the Agent initializes the task data structure based on the received task in a message from TM, (event R_{T-RQ}).
- *discovery*: After the initialization (*in*), the Agent requests information (S_{I-RQ}) from the Service Registry about services that can execute the task. If the response (R_{I-RS}) from SR contains only *false* formula (that is, there are no such services), the Agent sends negative response (S_{T-RS-F}) to the TM, and then goes to the state of readiness.
- *arranging*: After the successful service discovery (*ssd*), the Agent arranges execution of the task by sending subtasks to appropriate services (event S_{S-RQ}). If one (or more) of the services responses (event R_{S-RS}) are negative, the Agent cancels already arranged services, sends the negative response to the TM (event S_{T-RS-F}), and returns to the state of readiness.
- *executing*: After the services have been successfully arranged (*sa*), the Agent executes them in the specified order by sending execution requests (S_{E-RQ}). Based on their responses (R_{E-RS}), the Agent constructs the response to the TM (S_{T-RS}) containing the description of the current situation related to the task, and then returns to the state of readiness.
- *idle*: In this state the Agent waits for the responses related to its requests.

A task data structure consists of the session identifier (the same for every message during the task execution and generated by the task sender); the address of the task sender; the timeout for task execution; the task formula; and the plan of the task execution. A task data structure is initialized after receiving message of the type *task-request*. The plan of the task execution is taken from the response message of the Service Registry. The plan is in the form of a partial order defined in a set of subtasks; the order corresponds to the execution order of the subtasks. Each of the subtasks consists of the subtask formula, a set of addresses of services that can execute the subtask, description of the situation after the task execution, and the current state of the subtask (that is "*unarranged*", "*arranged*", "*executing*", "*executed*", or "*cancelled*"). Initially, all the states are set to "unarranged".

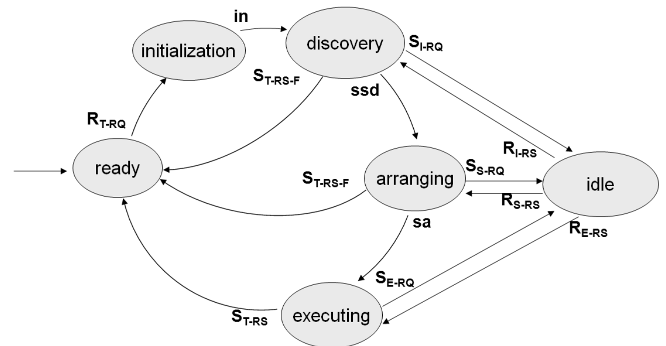

Fig. 6. The automaton for Agent

At the beginning of the arranging phase the Agent sends messages of the type *service-request* to the services that, according to the plan, are to realize the first subtasks in the execution order. The message contains the task specification and the expected time of the start of the subtask execution. The requested services may respond with the positive message of the type *service-response* containing the estimated time of the end of the task execution, or negative message meaning that it is not able to execute the subtask. In the case of multiple requests to different services that can execute the same subtask, the Agent chooses the one that commits to execute the subtask at the earliest, and sets the state of the subtask as "arranged". Then, the Agent proceeds to the next subtasks in the plan execution order (if there are any) and arranges them. If all the subtasks are arranged, and the time of the estimated execution of the last subtask does not exceed the task timeout, the Agent goes to the executing phase.

The Agent sends the execution request (messages of the type *execute-request* with the relation *true*) to arranged services in the same order as they were arranged, and sets their state as "executing". Each of the services responds with messages of the type *execute-response* with description of the situation after the successful subtask execution (its state is changed to "executed"), or description of the exceptional situation that occurred during the subtask execution (its state is changed to "cancelled"). In case of successful execution of the entire plan, the Agent replies to the TM with message of the type *task-response* containing description of the current situation form the last subtasks in the execution order. Otherwise, that is, at least one of the subtasks is cancelled, the Agent sends to the TM descriptions of the situations from the executed subtasks, and descriptions of exceptional situations from the cancelled subtasks. On the basis of its predefined rules for handling exceptions, the Task Manager decides what to do next. If the TM cancels the main task execution (by sending negative message of the type *task-request*), the Agent cancels the already made arrangements and currently executing services by sending negative messages of the

type *service-request* and the type *execute-request* respectively.

7.4. Service. Any device (including mobile robots) is seen by the SOMRS system as a collection of services that they provide. A service represents any action (function) realized by the device for which the interface was defined in the common language and published (registered) in the Service Registry. A service must also use the common environment representation and communication protocols. A service, from the system point of view, is seen as a finite automaton shown in Fig. 7, and has the following states:

- *ready*: In this state the service is ready to execute.
- *registering*: The service goes to this state for its first registration, or when the validity time of the current registration entry has expired (event *ex*). The service sends to the Service Registry its interface formula in the registration request (S_{R-RQ}). After receiving the response (R_{R-RS}), it updates the validity time (event *up*).
- *idle*: In this state the service waits for the response from the Service Register.
- *arranging*: The service goes to this state after receiving the message of the type *service-request* (R_{S-RQ}) with the task and the time of the start of the service execution. If the service can execute the task, it sends to the Agent response (message of the type *service-response*) with the estimated time of the end of the task execution (S_{S-RS}) , and then goes to the *waiting* state. Otherwise, it sends to the Agent negative message of the same type (S_{S-RS-F}) and returns to the state of readiness.
- *waiting*: In this state the service waits for a message from Agent of the type *execute-request*. After receiving it (R_{E-RQ}), it goes to the *executing* state. Receiving of the negative message of the type *service-request* (R_{S-RQ-F}) means that the service arrangement was cancelled by the Agent. The service deletes the appropriate arrangement and returns to the state of readiness.
- *executing*: In this state the service executes the arranged task. After the execution or in case of occurrence of an exceptional situation it sends message of type *execute-response* to the Agent (S_{E-RS}) with the description of the current situation related to the task. If the Agent sent the negative message of type *execute-request* (R_{E-RQ-F}), the service stops the task execution. After both of the described events, the service returns to the state of readiness.

Each service stores a list of arrangements made in the *arranging* phase. An arrangement consists of the session identifier, address of the Agent, the task formula, and estimated time of the beginning and the end of the task execution. An arrangement is deleted from the list when the service returns to the state of readiness. The service goes to this state when (including the cases mentioned before) the arranged time limits are exceeded, i.e. when the Agent has not requested the task execution, or the service has not executed the task on the time.

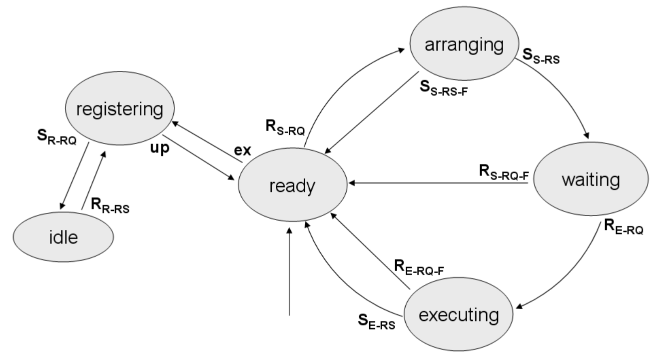

Fig. 7. The automaton of a service

Execution of some types of services (e.g. physical services) takes a longer time than in the case of a simple cognitive service, or a data-processing service. Such service has the queue of the tasks to be executed. During the arrangement, the service can add the arranged task to the existing queue based on the estimated time of the beginning and end of the task execution. Cancelled tasks are removed from the queue and other tasks can be arranged in emptied time slots. Thus, the time, returned to the Agent in the response to its service request in the arrangement phase, depends on the estimated time of the given task execution and on the first free time slot with the appropriate length in the queue. It allows the service to arrange tasks during execution of a previously arranged task and to arrange more than one task ahead.

To sum up the presentation of the proposed protocol, the order of messages exchanged between the components of the SOMRS system and phases of task execution are shown in Fig. 8.

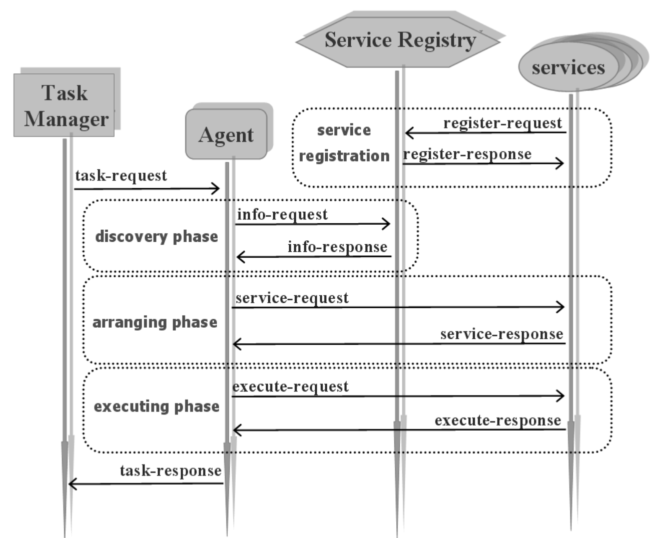

Fig. 8. The order and types of messages in the protocol

The general external behaviours (roles) of the components in the system are designed in the way that ensures clear and strict separation between sets of functions that each of them provides. A service represents (through its interface) specific action or function executed by the device to which it belongs.

Internally, it is a software application (running on the device) responsible for execution of the action and communication with the SOMRS system. Introduction of the Service Registry is an intuitive solution for the problem of service publishing and discovery, and the system openness (easy service adding and removing).

The Task Manager provides: the means for the client communication with the system, generating or choosing an existing abstract plan of the realization of the client intention, arranging and following that plan (which includes handling of the exceptional situations). The TM may be dedicated for realization of a number of selected classes of intentions, for which it can provide GUI and predefined abstract plans (or is able to generate them). The role of the Agent is the execution of a given task (which can be seen as a step in a plan of an intention realization), in a universal way that does not depend on the class of the task. Thus, there may exists more than one Task Manager in the system. Each of them can provide different functionality and different range of possible classes of intentions that they can handle. On the other hand, there is only one type of Agent in the system (although it can have many instances), because it always provides the same functionality. Any attempt to combine these two components (i.e. Task Manager and Agent) and their functionalities into one component would lead to unnecessary redundancy.

In case of several types of task managers and instances of agents present in the system, each of task managers sends tasks to an agent based on the agent address set in its configuration. An agent always responds to the known address of the task manager that has sent the task. A task manager can also have a list of addresses of available agents. Because all agents provide the same functionality, the criteria of choosing an agent can be based on a network delay or the load of the server where an agent is running as a software application.

The concept of Repository of object maps deserves a special attention. In the SOMRS system there may be one Repository with large and rich maps of physical environments. In this case there are no problems with synchronization and the integrity of stored information. On the other hand, if a map of a particular environment is intensively used (very often queried and/or updated) it may be stored in a local Repository appropriately synchronized with the main Repository.

8. Experiments

The first prototype implementation of the proposed SOMRS system was realized in the Java programming language in version 1.6. The Task Manager, the Agent and the Service Registry were implemented as servlets (Java Servlet Technology) and run on the Apache Tomcat 6.0 application server. Services were provided by two mobile robots Pioneer 3 (P3-DX). The communication between the robots and PC work stations was based on a local wireless network. Two different test environments, consisting of a room and an adjacent corridor, were used to carry out the experiments.

Because of the available devices (two mobile robots equipped with web cameras and grippers) the possible services that they can provide include searching and moving objects. This small set of services allows, however, for realization of standard and more complex intentions like moving objects and inspecting a given set of locations. The abstract plans (used by the Task Manager) for realization of these intentions were created manually. The Service Registry can automatically decompose, if needed, the search and transport tasks into subtasks based on hierarchical routing in the object maps of the test environments.

The experiments, carried out in the physical environment shown in Fig. 9, consisted in realizations of intentions having the following description of the final situation: (*smallBox isAdjacentTo woodenCloset*). The relation *isAdjacentTo* means that two given objects are placed close to each other. The initial situation was also specified by the client: (*smallBox isAdjacentTo bathroomDoor*). So, the Task Manager could define a transport task directly from the intention formulas. The Agent, after receiving this task, sent it to the Service Registry in order to obtain information about services that could execute it.

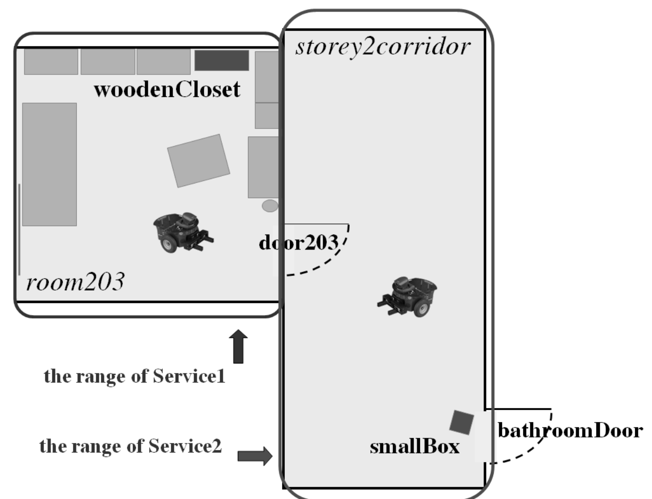

Fig. 9. Schema of the physical test bed environment

The Service Registry determined the task location. It is the lowest (according to the object hierarchy) object in the map to which the initial and final locations of the task belong, that is, the locations are its direct or indirect sub-objects. Because *bathroomDoor* is located in the corridor *storey2Corridor* and *woodenCloset* is located in the room *room203* the task location is *storey2* to which these two objects directly belong. The two registered transport services (provided by the two mobile robots) operate in the corridor *storey2Corridor* and the room *room203* respectively. Thus there was no single service operating in the task location. So the Service Registry had to decompose the task into subtasks by determining a route between the initial and final location. Since the two locations are placed next to each other, the route consisted of the two elements: *storey2Corridor* and *room203* for which appropriate transport services were available. The door *door203* was the object connecting the two elements of the route. So the final situation of the first subtask and the initial situation of the sec-

ond subtask were the same: (*smallBox isAdjacentTo door203*). The execution plan consisting of the two subtasks and the relations of succession (the subtask in the *storey2Corridor* has to be executed first) was sent to the Agent.

In the arranging phase, the Agent sent the two subtasks to appropriate services (under the addresses specified in the subtasks). The Service Managers of the two robots responded positively. After the successful arrangement, the Agent sent the execution request to the service realized by the robot operating in the *storey2Corridor*.

The robot searched the region near the *bathroomDoor* (Fig. 10a) and found the *smallBox*, what verified the initial situation of its task. Then, it started the process of positioning in order to grasp the box (Fig. 10b).

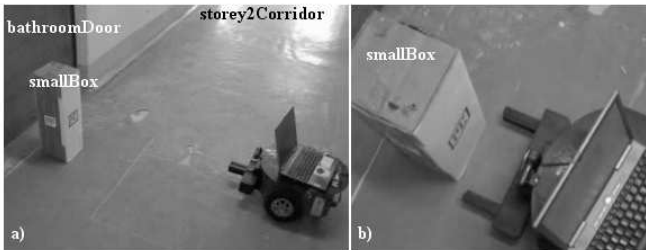

Fig. 10. a) A robot searching for the box *smallBox* near the door *bathroomDoor*; b) the robot grasping the box

In the next step, the destination location (the door *door203*) is determined based on the object map of the test environment and the description of the final situation of the first subtask. The robot computed the route to the object in an occupancy grid, (its own low level representation of the local environment for navigation), and goes along the route (Fig. 11a). After reaching the destination the robot put away the *smallBox* (Fig. 11b) and sent to the Agent a message with a description of current situation related to the task, which in this case was identical with the final situation of the first subtask.

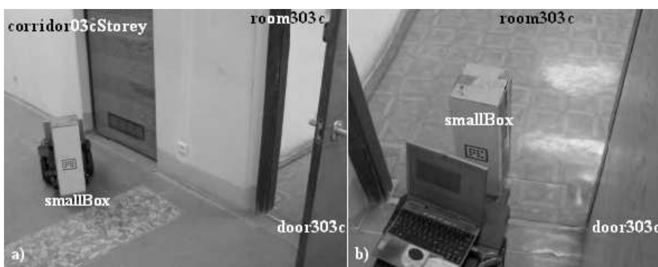

Fig. 11. a) the robot navigates in the direction of the door *door203*; b) the robot puts away the box *smallBox*

After receiving the response, the Agent sent the execution request to the second service provided by the robot operating in the room *room203*. The robot executed the arranged task in the same way as the first robot and replied to the Agent. Then, the Agent sent to the Task Manager the message with the description of the current situation described by the formula (*smallBox isAdjacentTo woodenDoor*).

9. Conclusions

The goal of the proposed information technology is to provide an infrastructure for building multirobot systems based on new ideas like ambient intelligence or ubiquitous robotics. The use of the SOA paradigm allows not only for supporting system openness and heterogeneity, but also for easy integration with existing solutions in Robotics. Dedicated and specialized multirobot systems (even *swarms*) can be added to the system as services which provide functionality described in their interfaces.

Interoperability between such heterogeneous entities (services) is achieved by defining ontology (common structure of environment representation), common language for describing the environment, defining tasks and service interfaces, and protocols for the communication between the system components. The system components and their functionalities are defined in a way that ensures extensibility and scalability of the system by enabling easy addition of new heterogeneous services and new instances of other components. The system allows for automatic and dynamic task allocation through the Service Registry and service discovery. A new feature of the system, in comparison to existing solution based on SOA, is that it also ensures the optimal task allocation in a given time and location by introducing the arrangement phase into the process of the client's intention realization.

The experiments were limited by the number of services, in this case, the number of robots. However, the main claim of the work, that is, that the proposed information technology can be applied to achieve interoperability between heterogeneous devices in SOMRS, seems to be verified. In the near future more sophisticated experiments will be carried out.

Acknowledgements. The presented work was done in the framework of the research project supported by the Polish Government Grant MNiSzW 3 11C 38 29. The authors would like to thank Prof. C. Zieliński for inspiration and constructive criticism.

REFERENCES

[1] J. Hertzberg and A. Saffiotti, "Workshop on semantic information in Robotics", *ICRA-07 Proc.* 1, CD-ROM (2007).

[2] M.E. Jefferies and W. Yeap, "Robotics and cognitive approaches to spatial mapping", in: *Tracts in Advanced Robotics Springer*, vol. 38, Springer, Berlin, 2008.

[3] G. Beni, "From swarm intelligence to swarm robotics", *Swarm Robotics: Lecture Notes in Computer Science* 3342, 1–9 (2005).

[4] L. E. Parker, and F. Tang, "Building multi-robot coalitions through automated task solution synthesis", *Proc. IEEE, Special Issue on Multi-Robot Systems* 94, 1289–1305 (2006).

[5] Y. Ha, J. Sohn, Y. Cho, and H. Yoon, "A robotic service framework supporting automated integration of ubiquitous sensors and devices", *Information Science* 177, 657–679 (2007).

[6] PALCOM Project Home page: http://www.ist-palcom.org/.

[7] IBM Services Architecture Team, "Web services architecture overview: the next stage of evolution for e-business", http://www.ibm.com/developerworks/webservices/library/w-ovr/, (2000).

[8] S. Thrun, „Robotic mapping: a survey", in *Exploring Artificial Intelligence in the New Millenium*, eds. G. Lakemeyer and B. Nebel, Morgan Kaufmann, Massachusetts, 2002.

[9] J. Kuipers, "The spatial semantic hierarchy", *Artificial Intelligence* 119, 191–233 (2000).

[10] S. Vasudevan, S. Gächter, V. Nguyen, and R. Siegwart, "Cognitive maps for mobile robots – an object based approach", *Robotics and Autonomous Systems* 55 (5), 359–371 (2007).

[11] D. Anguelov, R. Biswas, D. Koller., B. Limketkai, S. Sanner, and S. Thrun, "Learning hierarchical object maps of non-stationary environments with mobile robots", *Proc. 17th Annual Conf. Uncertainty* AI 1, CD-ROM (2002).

[12] C. Galindo, A. Saffiotti, S. Coradeschi., P. Buschka, J.A. Fernández-Madrigal, and J. González, "Multi-hierarchical semantic maps for mobile robotics", *Proc IEEE / RSJ Int. Conf. on Intelligent Robots and Systems* 1, 3492–3497 (2005).

[13] C.P. Eastman, P. Teicholz, R. Sacks, and K. Liston, *BIM Handbook. A Guide to Building Information Modeling for Owners, Managers, Designers, Engineers, and Contractors*, Wiley, Hoboken, New Jersey, 2008.

[14] C. Zieliński, "Description of command semantics of programming languages of robots", *Archives of Automatics and Telemechanics of Warsaw University of Technology* 35 (1–2), 15–45 (1990), (in Polish).

[15] C. Zieliński, "TORBOL: an object level robot programming language", *Mechatronics* 1 (4), 469–485 (1991).

[16] C. Zieliński, "Description of semantics of robot programming languages", *Mechatronics* 2 (2), 171–198 (1992).

[17] S. Ambroszkiewicz, "Entish: a language for describing data processing in open distributed systems", *Fundamenta Informaticae* 60 (1–4), 41–66 (2004).

[18] T.R. Gruber, "A translation approach to portable ontology specifications", *Knowledge Acquisition* 5 (2), 99–220 (1993).

[19] R. Fikes and N. Nilsson, "STRIPS: a new approach to the application of theorem proving to problem solving", *Artificial Intelligence* 2, 189–208 (1971).

[20] *PDDL – Planning Domain Definition Language*, Drew V. McDermott web site: http://cs-www.cs.yale.edu/homes/dvm/