# Interoperability in Open Heterogeneous Multirobot Systems

**S. Ambroszkiewicz** and **W. Bartyna**
**M. Faderewski** and **G. Terlikowski**
Institute of Computer Science, Polish Academy of Science
Al. Ordona 21, PL-01-237 Warsaw
and Institute of Computer Science, University of Podlasie
Al. Sienkiewicza 51, PL-08-110 Siedlce, Poland
Email: sambrosz@ipipan.waw.pl

**K. Cetnarowicz**
Institute of Computer Science
University of Mining and Metallurgy
Al. Mickiewicza 30, 30-059 Krakow, Poland

**Abstract**

An approach to the problem of interoperability in open and heterogeneous multirobot systems is presented. It is based on the paradigm of Service Oriented Architecture (SOA), and a generic representation of the environment. Robot, and generally device, is seen as a collection of its capabilities exposed as services. The representation of environment is used to define tasks, and service interfaces. Several protocols are proposed to enable interoperability among the services in order to publish, discover, compose services, and execute the composite services. The representation, language for task definition, and language for service interface definition, as well as the protocols constitute together an information technology for automatic task accomplishment in open system consisting of heterogeneous devices (robots).

## Introduction

The paper (Parker & Tang 2006) is the pioneer work considering the problem of interoperability in open heterogeneous multirobot systems, that is, *to enable a group of heterogeneous robots to form coalitions to accomplish a multirobot task.* Although this paper is an inspiration and the main reference of our research, the proposed approach to the problem of interoperability is based rather on information technology point of view than on the pure robotics one. The two corner-stones of the approach are: the paradigm of Service Oriented Architecture (SOA), and a generic representation of the environment.

SOA provides a standard programming model that allows software components (as self-contained, modular applications) to be described, published, located over a network, and then invoked by each other as services. There are essentially four components of SOA: Service Provider, Service Requester (also called Client), Service Registry, and Broker. The provider hosts the service, 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 search of a component to invoke in order to realize a request. The service registry is a repository that facilitates service discovery by the requesters. The broker acts as complex service towards a requester and as client towards simple services; the complex service is composed by the broker from these simple services.



Generally, the term *robot* in multirobot systems should be replaced by *device* equipped with microcomputer (microcontroller) and connected to a network, i.e., being able to communicate meaningfully with software applications running on computers, and other *devices.*

In order to apply SOA to robotics we must revise a bit the classic concept of multirobot system as consisting of autonomous robots, and assume that *service* is the basic component of the system. There are three general kinds of such *services* in multirobot system:

- The first one may be viewed as an ability to perform some kind of physical action by a robot (its actuator) or device; it may be called *physical service.*
- The second one as an ability to perceive the environment via a sensor; it may be called *cognitive service.*
- The third one is just a software application processing data; it may be called *software service.*

Hence, the notion of service from SOA may be extended to the ability to perceive and to change the environment. However, for any of such services there must be software application (or application hardcoded into the device) that controls it. Any of these very applications may be viewed as ordinary service in SOA representing appropriate physical or cognitive service.

Operation performed by service and a way to invoke the service must be described and published so that they can be understood, and the service can be accessed automatically by a requester (client) being also a software application. It means that the semantics of this description must be formal and explicit. Such description is called *service interface.* There are several approaches to define service interfaces, for example, (WebServices 2002), and (SemanticWebServices 2004). However, we propose new interface definition language based on a generic representation of the environment. Service interface is defined in the terms of changes in the environment caused by the service operation. It means that the interface is defined as precondition and postcondition (effect) of the service operation.

Also tasks to be accomplished in the systems must be formulated in a way that can be understood automatically by client application. Task may be defined, in the natural

way, as a change of local situation in the environment; it is a declarative way to define tasks. The procedural (imperative) way to define task specifies a sequence of actions to be realized. Since the basic components of SOA architectures are software applications, tasks must be performed automatically by discovering appropriate services, composing them into workflow, and then by controlling the workflow execution.

The processes implementing the discovering, composition and execution control may be called *software agents.* These agents communicate using *conversation protocols.* Once it is supposed that all services are connected to the network (based on Bluetooth/IEEE 802.11/Ethernet /TCP/UDP/IP), these agents may be situated anywhere in the Internet, e.g., on a server or on an autonomous robot, or they may be distributed over network and form a multiagent system, see (Ambroszkiewicz & Cetnarowicz 2006).

A generic representation of the environment is the basis for defining service interfaces, and in this way for creating infrastructure that enables automatic task accomplishment in open system consisting of heterogeneous devices. From the point of view of this infrastructure any device is seen as its service interfaces. Since the interfaces are defined in terms of environment changes, the infrastructure is independent from the physical devices that are in the system. This independence allows creating the infrastructure as implementation of a pure information technology. Modern information technologies are built at the level of protocol specifications to assure interoperability when the protocols are implemented by different vendors in different devices and operating systems.

The proposed technology (actually, its first experimental version) is also created at the level of specification of a generic representation of the environment, a language for defining service interfaces and tasks, and communication protocols for automatic task accomplishment in open heterogeneous multirobot systems. The goal of the paper is merely to introduce the idea of the technology, and its prototype implementation for relatively small multirobot system consisting of two mobile robots Pioneer 3-DX, and several additional devises like cameras, and laser scanner SICK LD-OEM1000. The system is too small to verify the proposed technology; however, it may serve as its explanation. For a demo of the impementation see the movie at (Robo-enT 2007).

The next section is devoted to sketch a robot and multirobot architecture based on SOA. In the Section 3, the idea of a generic representation of the environment is presented. In the Section 4 a working example, i.e., prototype implementation of the infrastructure for the small system is presented. The last section is devoted to a discussion on the limitation of the proposed approach and related work. More details on our ongoing project will be available shortly at (Robo-enT 2007).

## Architecture of robot and multi-robot system based on SOA

There are four kinds of software components, that is, service, client, registry, and broker in a system built according to the SOA paradigm. Each of them is autonomous so that it may be called agent. Hence, there are four kinds of agents: *service-agent* representing service, *task-agent* representing client having a (complex) task to perform, *info-agent* representing registry, and *broker-agent* representing broker. Since the interactions between agents are only roughly described in SOA, there must be a communication protocols that implement the interactions.

Usually, robot is considered as an autonomous system that can play the following roles:

- A robot may be seen as a collection of service-agents corresponding to its internal services. Some of these services may be exposed to other robots. Since they are controlled by the robot, it must act as a special broker-agent, that is, from the internal point of view it acts as a task-agent towards its services. From the external point of view (i.e., in the multi-robot system) the robot is viewed as a collection of service-agents performing complex services. Robot autonomy is realized by managing these service-agents.
- A robot is responsible to accomplish a task. Then it acts as a client-agent towards services implemented on other robots and devices in order to perform the task.
- If a robot has sufficient computing capabilities, it may act as a typical broker-agent performing a special complex service composed from services available in the multirobot system. It may act even as an info-agent.

One single robot may play all the above roles, however, only the first role is essential for it. The remaining roles require rather computing capabilities and connection to the network, than physical services installed on the robot. Since the system components are supposed to be connected into network, it is reasonable to perform these remaining roles on dedicated computers.

Architecture of open system consisting of heterogeneous devices (also robots) is defined as a dynamic collection of service-agents (corresponding to the services performed by these devices), info-agents, broker-agents, and task-agents. Each of the service-agents exposes the interface of its service to an info-agent, and is obliged to converse (according to the common protocol) with any task-agent interested in using the service. A task-agent discovers appropriate service interfaces by communication with an info-agent. Broker-agent is viewed as a service-agent by task-agent, and as a task-agent by the service-agents. In order to define precisely the above architecture, the protocols, the language for defining service interfaces and the tasks must be specified. The basis for these specifications is the representation of environment presented below.

## A generic representation of environment

The basic component of the proposed representation is object. An instance of the representation is a collection of objects having a hierarchical structure of the form of tree. The

leaves of the tree are *elementary objects* that can be directly perceived via their physical features (attributes) like shape, color, size, etc. The tree nodes correspond to *complex objects* having internal structure consisting of subobjects (immediate child nodes in the tree) related in some way to each other. A room may serve as an example of a complex object. Its subobjects are walls, ceiling, floor, windows, door, and some equipment like chairs, table, computers etc. These subobjects are in special relations to each other, for example, the relations of adjacency between the walls, floor and ceiling. Only elementary object (e.g., a door in a building) can be perceived directly by recognizing, for example, its shape, and location relatively to walls. Actually, a complex object is an abstract entity, and can be identified only by its elementary subobjects and their mutual relations. Although a complex object may also have some features like shape, this feature is composed from the shapes of its subobjects and their mutual relations. The extensive example in the next Section illustrates our concept of object hierarchy.

The crucial notion of the proposed representation is *object type.* It defines *the generic structure* of all the objects of this very type. Except the types of elementary objects, this structure consists of the tree expressing subobject hierarchy, the relations between subobjects at the same level of hierarchy, and object attributes that can be directly perceived or composed from attributes of the subobjects. The definition of type must be recursive because the subobjects are also of some types, however, at lower level of the hierarchy. Object is an instance of its type, so that the types determine the general structure of the objects and in this way the general structure of environment representation. It is important to notice that the relations and attributes used to define a particular type are not specific and dedicated only to this very type. Instead, they are generic and determine the representation, because they are the basic components of the type definitions. They are the most primitive notion in the proposed representation, and correspond to the perception capabilities of the system, that is, to the capabilities of cognitive services that can recognize local situations of the environment. It is important to note that the types that correspond to the elementary objects are defined using only attributes; relations are not used because elementary object does not have subobjects that may be related to each other.

The attributes like color, weight, size, and shape, as well as the relations between objects like being close, being adjacent, and being subobject may serve as example of the primitive notions. Complex notions like shape of building may be defined from the primitive ones. They are the basis for defining object types, as well as for recognizing objects; it means that the object definition and its recognition are inseparable. Hence, the primitive attributes and primitive relations constitute the basis for environment representation and at the same time the basis for recognition in the environment.

A complex object may represent a part of the environment, for example, a room, storey of a building, building, street, and even a town. The type of this object determines only the general structure of the corresponding part of the environment, e.g., general structure of the type of building consists of storeys connected by stairs and possible a lift, however without specifying the number of storeys, rooms in each of the storeys, objects in rooms like tables, chairs, computers as well as and their relations to each other.

In order to define the type of a complex object, the types of all its subobjects and possible relations between these subobjects must be already defined. Some of the subobjects may be again complex objects so that the types of their subobjects must be defined first, and so on down to the elementary types. To define a type also some of the generic attributes must be assigned to this type, for example, high, width, shape, and color as well as the range of each of these attributes.

Object is an instance of its type, so that values of its attributes, and explicit relations between its subobjects should be specified in the object. Sometimes there is no need (sometimes it is impossible) to specify them completely like in the case of a closed room with no access. Moreover, some subobjects positions and their relations may be temporary, like the position of a chair in a room. Hence, the type of an object is recognized by the types of its subobjects, their mutual relations, and attributes. It is important that this type recognition must be computable from data coming from sensors of the cognitive services.

It seems that the proposed representation is different than the classic metric as well as topological representation and its variations, see (Thrun 1998), and (Yeap & Jefferies 1999). Each of them has its own fixed general structure; it is either the binary (occupancy) grid, or position of geometric primitives in a global coordinate system, or Voronoi diagram expressing connectivity (neighborhood) of places, or a hybrid like a global topological map formed of local metric maps. Actually, all of them are dedicated rather to the problem of robot navigation which is usually seen only as a subtask to be performed by a robot.

There is no global coordinate system in our representation. A local coordinate system is also not necessary; it is important for robot navigation itself, so that a coordinate system and a corresponding map (e.g., binary grid) can be created dynamically (using methods like SLAM) by the robot. Sometimes the precise position of the robot in the map is not essential for task accomplishment; it is rather its relative position to the object that is (for example) supposed to be gripped. Although local coordinate system is not necessary in our representation, it can be often created for a complex object (like a room or corridor) on the basis of the object features (like shape and size) and mutual relations between its subobjects; actually this is done in our implementation.

Recently, an interesting approach to environment representation was proposed by (Mozos *et al.* 2007) and (Kruijff *et al.* 2007); it is called *multi-layered conceptual mapping* and consists of the four layers specified as metric map, navigation map, topological, map, and conceptual map. The conceptual map is strongly related to our approach; especially the relation *has-a* corresponds to our relations *subobject* and *is-in* defined in the next Section.

It seems that the main advantage of the proposed representation over the classic ones as well as the multi-layer conceptual mapping is the variety of generic relations and

attributes that can be defined in the representation, implemented as cognitive services that can perceive them, and then introduced *online* to the system preserving its semantic interoperability. The following examples of the generic relations explain somewhat our claim.

The intuitive relation *being an element (subobject)* aggregates two different generic relations. The first one concerns the subobjects that are necessary elements of the object like the walls in a room; this relation is *embedded* in the generic hierarchical structure of the proposed representation. The second generic relation expresses the fact that an object A *is-in* object B, however the presence of A in B is not necessary like in the case of a chair in a room.

The next intuitive relation *adjacency of two objects* is also a conglomerate of two generic relations. The first one concerns two objects that are sufficiently close to each other like chair and table, however it is not necessary for these objects, that is, it is possible that they are far away from each other. The second generic relation expresses the fact that two objects have a common subobject. For example, a room and a corridor have a door (a wall) as its common subobject. Then this relation may be viewed also as the neighborhood, however, not always as the connectivity in the context of topological representation.

Relations between subobjects are qualitative so that it may be seen as a limitation. For example, the adjacency relation does not tell how close the two objects are. However, it is supposed that there is a cognitive service (for example, implemented on a robot) that can recognize the objects and measure the distance between them. Exact parameters and values of the environments are not always needed especially if they are subject of change. The question is how precise the map of the environment should be. If the effort to construct such precise map is large, then there is no sense to construct it unless it is necessary for task accomplishment. The concept of rough skeleton maps is reasonable, because such maps give general information about the environment usually only with some permanent details. They can be updated and detailed only if it is needed.

The approach consisting in computing exact parameters of the environment, like positions of the robot arm, and of the object to be manipulated, was inherited from the classic robotics. It is inefficient in the open and a priori unknown environment. There is alternative approach in robotics called impedance control (Hogan 1985) where robots are supposed to dynamically interact with an unknown environment. Since robots are no longer confined to the fixed environments to work on well-designed automation tasks, there is no need for precise estimation of environment parameters. Some of these parameters, like robot position relatively to other objects, can be estimated on line during dynamic interactions. It seems that the concept of the environment representation based on the hierarchy of object types, generic relations and attributes may be useful for this approach. The key problem is availability of cognitive services that are able to find out objects, and recognize their mutual relations according to the generic structure determined by the representation. The proposed representation is an extension of our previous work (on environment representation for navigation and orientation by blind people) done within the framework of the project Blind-enT (Ambroszkiewicz *et al.* 2006).

The new environment representation allows defining tasks in different manner than the classic one. Instead of specifying a sequence of actions to be performed and exact parameters of these actions, a task may be defined in declarative and/or qualitative way in terms of relations and attributes. It means that only the final effect of the task is specified, like *red small object of type Box is adjacent to a green object of type Door being a subobject of corridor1*. This idea is explored in the next Section.

## Language for task and interface definition, universal protocol for task accomplishment

As already mentioned in the Introduction there are three kinds of services: *software services* that process data, *physical services* that change a local state of the environment, and so called *cognitive services*. The cognitive services deserve special attention because they perceive local states of the environment. However, any act of perception done by a cognitive service is restricted to some fixed relations, attributes, and to some fixed places; it means that the service can evaluate (only in these fixed places) formulas containing the names of these relations and attributes. The cognitive services and physical services are strongly related to each other, because physical service changes local state to another one whereas cognitive service perceives (to some extend) these two local states.

If the system consists of only software services, then environment is called *Cyberspace*, its representation is relatively simple, and consists of data types and functions that process data. This case was extensively elaborated in our previous work resulted in the experimental technology (enTish 2003) for integrating software services in order to automatically accomplish complex tasks. Our current work is based on enTish and, in fact, extends it.

In order to describe a physical service, a representation of the real environment is needed. The initial situation (called precondition) necessary to invoke the physical service as well as the situation after invocation (usually called effect or postcondition) is expressed in terms of this representation.

The generic structure of environment representation, that is, the type hierarchy, as well as object examples are constructed in XML. A part of them is presented at www site of the (Robo-enT 2007).

Example of a simple hierarchy of types is described below in some details. The most complex type is `Building`, it has `shape`, and `size` as its attributes. The types of its subobjects are: `Storey`, `Lift`, and `Stairway`. They are also complex types, i.e., types of complex objects.

- The type `Stairway` has one attribute called `number-of-steps`, and one relation called `adjacency` between its subobjects. The types of its subobjects (actually being elementary objects) are `Stairs`, `UpStair`, and `DownStair`. These last two types correspond to the beginning of the first step and the last step of the stairs. There is permanent `adjacency`

relation between object of type `UpStair` (`DownStair`) and object of type `Stairs`.

- The type `Lift` has attribute `number-of-levels` corresponding to the number of storey in the building, and attribute `cabin-position` corresponding to the current position of the lift cabin. Its subobject types are `Cabin` and *Level*; however, there is only one object of type *Cabin* and several objects of type *Level*. The type *Cabin* has attribute `size` and subobject of type `Door` as its elementary subobject. The type `Level` has attribute `number` corresponding to the identifier of the appropriate storey, and subobject of type `Door` being also elementary subobject of object of type `Corridor` defined below. Actually, the attribute `cabin-position` is an abstraction of the `adjacency` relations between the door of the cabin and the door of a level.
- The type `Storey` has two optional subobjects of type `UpStair`, and `DownStair` being elementary objects (they are also subobjects of `Stairway`), and obligatory subobjects of type `Room`, and of type `Corridor`. The type `Corridor` has two attributes `width`, `high`, and `length`. It has also several subobjects of type `Door` being elementary objects. The type `Room` has the following attributes `number`, `width`, `length`, and `high`. It has also subobjects of type `Door`.

For the sake of presentation most of the above types are oversimplified (like `Door` that should have several attributes), or may be seen as questionable like `UpStair`, and `DownStair`. Also the types `Room`, and `Corridor` should have more subobject types like `Wall`, `Ceiling`, `Floor`, and `Window`. The objects of types `Room`, `Corridor`, and `Cabin` are places where objects of types `Box`, and `Robot` may be located; this is expressed by the relation `is-in(x,y)` with intuitive meaning that object `x` is in object `y`. The type `Box` is elementary and is defined by the attributes `color`, `size` and `shape`. It is supposed that object `y` of type `Box` can be gripped (if it is sufficiently small) by an object `x` of type `Robot`; this relation is denoted by `grip(x,y)`.

The simple environment representation presented above gives rise to construct a simple language for describing local situation in the environment. The same language is used to define service interfaces and tasks. The language consists of the following names of notions introduced in the above representation:

- names of types, variables denoting objects of a fixed type, constants denoting fixed objects;
- names of attributes (functions): *number, width, length, high, number-of-steps, number-of-levels*, and *cabin-position*,
- names of relations:
  - *neighbor(x,y)* for denoting that objects have a common elementary subobject,
  - *subobject(x,y)* to express the immediate hierarchy relation that object x is a subobject of object y.
  - *is-in(x,y)*
  - *adjacency(x,y)*
  - *grip(x,y)*

Along with standard logical operators (negation, disjunction, conjunction, implication, and equality) as well as special types *Agent* and *Service* the language is a simple version of first order logic without quantifiers. The terms and formulas are constructed in the usual way. The precise syntax of this language is expressed in XML and available in the enTish www site (enTish 2003). It is important to note that although the language (called Entish) was created for systems consisting of software services, its syntax is universal and can be applied successfully for physical and cognitive services. In this presentation a semi formal version of the language is used. Service interfaces are defined in the following way.

Interface of software service, processing data according to abstract function *f*, is defined as the pair of formulas: (precondition, postcondition). Precondition formula contains input parameters (say, variable *x* of type *Typ1*, and variable *y* of type *Typ2* with some constrains) of the function *f*, whereas postconditions contains term *f(x,y)* also with some constrains. Let the name of the service be *service1*. Example of such interface is the following:

- The precondition formula: *( is-in(x, service1) and is-in(y, ser1) and less(x,y))*; where the variables *x*, and *y* are respectively of type *Typ1* and *Typ2*. The meaning of the formula is that the data of these types are delivered to the *service1*, and these data satisfy some constrains; in this case, *x* is less than *y*.
- The postcondition formula: *( z = f(x,y) and is-in(z, v) )*. The variable *z* denotes the result of processing the input data *x* and *y* by the service *service1*. This result may be sent to any service denoted by *v*; *v* is variable of type *Service*.

Interface of physical service performing an action in the environment (that is, changing local situation (state) of the environment) is defined also as the pair of formulas: (precondition, postcondition). The service performs abstract action in order to change the local situation described by precondition formula to the situation described by the postcondition; that is, the postcondition expresses the effect of action performed by the service. The postcondition formula contains also the name of the abstract action. Simple example of such interface is as follows:

- Precondition formula: *( less(width(x), 20cm) and less(weight(x),5kg) and is-in(x, room103c) )*. It means that object *x* is small (of limited size) not too heavy and is in *romm103c*.
- Postcondition formula: *( adjacent(x,y) and subobject(y, room103c) and action(service2) = move )*. It means that the object *x* is close to any subobject of *room103c*. Although the action name seems to be not essential in the post condition formula, it specifies the way the post condition is realized. Actually, the post condition may be realized in many ways, i.e., performing different actions, one of them is the action move. Let us notice that this action can be decomposed into the sequence of more primi-

tive actions: *grip, go to, put down*; however, for the class of tasks considered in the paper, it is convenient to have such composite action.

The interface specifies that the service can move a small, not too heavy object *x* within the range of *room103c*. The pre and postcondition formulas are general so that during the negotiation phase of service composition into workflow, more specific conditions may be added to these formulas. For example, that the object is *red* and of type *Box* in the precondition, whereas in the post condition that the object is close to the door of the room.

Due to special operation performed by cognitive service, consisting in evaluation of formulas, its interface is different than the interfaces of software and physical services described above. The interface is defined as follows:

- The list of names of relations, and names of attributes that can be evaluated by this cognitive service.
- The range of the service expressed as object of some type, like `Room, Building`, etc.

Any formula containing only these relation names and attribute names can be evaluated by this cognitive service, however, only within the object determined by the range. Hence, the input of this service is a formula, whereas the output is evaluated formula in special form called *signedInfo* (see Documentation, and XML-sources of (enTish 2003)).

Let us consider a cognitive service implemented on the device consisting of a camera and 3D scanner (connected to a computer), and operating in `room103c` of the `storey1` in the building. Interface of the service of this device consists of the following items:

- List of attribute names: *color, size, shape*.
- List of relation names: *is-in, adjacent*.
- Range: *room103c*.

The three kinds of services described above may be composed in the following way:

- Cognitive service is responsible for finding object z, that is, to evaluate the formula *( is-in(z, room103c) and type(z)=Box and color(z)= red)*. The service returns that the following formula is true: *( is-in(z, room103c) and type(z)=Box and color(z)=red and adjacent(z, widnow1) and coordinate(window1) = x )*. Note that the service added one relation, and one attribute *coordinate* that corresponds to the local coordinate system of the *room103c*.
- Software service is responsible for computing direction and distance between points *x* and *y* that is, the data *(direction, distance) = f(x,y)*, where *y* is the coordinates of the current robot position.
- Physical services on a mobile robot are responsible for: go to *x*, move object z to the door of *romm103c*.
- Suppose that another physical service (on a different robot) also performs action move, however, its range is *corridor1*. Since the object *door103c* is the common elementary subobject of *corridor1* and *room103c*, the robot operating in the corridor can take the object *z* and move it to the door of the lift and in this way satisfy the formula *( adjacent(z, liftdoor1) and type(z)=Box and color(z)= red )*. Note that this very formula defines the task that has been already accomplished.

In order to do so a task-agent must be created and dedicated to this very task accomplishment. The agent is responsible for making a rough plan how the task formula can be satisfied. The plan consists of actions that once composed and executed may make the formula true. Any action corresponds to service interface. So that the agent must discover services having such interfaces, negotiate with service-agents the conditions to invoke them, arrange them into workflow, and finally execute the workflow and control the execution. In the case of a failure there must be recovery mechanisms that allow the task-agent to continue the process of task accomplishment.

It is important to note that a task is defined as an intentional formula. It means that the situation of the environment described by this formula is desired by the client who has created the task. Usually, the formula is not satisfied, i.e., no such situation occurs in the environment.

In order to automate task accomplishment by the task-agents, several linguistic constructions have been introduced the language that violate its first order property. They are necessary to express in the language the following notions: intensions of task-agent, commitments of service-agent, and pre and post condition of the interface of a service, see (enTish 2003) for more details.

The representation of the environment, language for interface and task definition constitutes the basis that allows the automation of the process of task accomplishment. To realize this automation, specifications the following protocols are needed:

- **Registration:** Service-agent registers the interface of its service to an info-agent (service registry).
- **Planning:** Task-agent constructs possible plans and adopts one of them. To simplify our implementation it is supposed that plans are given to the task-agent.
- **Discovery:** Task-agent discovers appropriate services with interfaces corresponding to the actions in the adopted plan, and checks if the ranges of these services are appropriate for the task accomplishment.
- **Service composition:** Task-agent communicates with service-agents to arrange the services into workflow. Generally, there is backward and forward method to do so. The backward method starts composing the workflow from the last service in the order of execution, whereas the forward method starts with the first services. The backward method was applied in the original protocol (called entish 1.0) for the system consisting only of software services. However, even for this case it is possible to construct different (perhaps more efficient) protocols. For the multirobot system, the forward method was applied to arrange services into workflow.
- **Workflow execution:** Task-agent starts the workflow execution, by invoking the first services, and then the consecutive services in the workflow. In the case of a failure of one of the services, the task-agent must rearrange the

workflow using a special case of service composition protocol.

The example presented above was implemented in the system consisting of 2 robots Pioneer 3- DX with grippers, 1 laser scanner SICK OEM-1000, and several cameras. The environment consists of our laboratory (romm103c) and the corridor. The robot operating in more complex environment, i.e., in the laboratory, was equipped with the scanner that allowed precise local mapping. The second robot operating in the corridor was equipped only with standard sonar suite to perform local mapping. The cameras connected to computers implemented cognitive services. Since the recognition of object shape is hard (actually we work on this subject within a separate project), the cameras recognized special landmarks (labels in the form of barcode) attached to elementary objects like box, door, wall, chair, table, and so on. Each of the robots can grip an object and move it to any place within its range; so that service corresponding to the action move was implemented on any of the robots. The range of the first robot was romm103c, whereas the second robot operated only in the corridor. Any robot has also a camera that was used to recognize object (in fact, its label), so that it has a cognitive service implemented on its board.

Since there was only two types of services in the implemented system, the class of tasks had to be limited to the following tasks:

- Searching for an object given its attributes and type.
- Moving an object from one place to another.
- Mapping that consists in updating and detailing a complex object like a building.

Since the ranges of the robots overlap at the doorway between the lab and the corridor, the physical services, as well as the cognitive services of the two robots can be composed. The mapping service deserves special attention because it concerns only cognitive services. It was implemented as a broker-agent (taking care of the map of the whole building) that evaluated formulas on the requests from the task-agent. Actually, a request was forwarded to appropriate cognitive services, the evaluations were returned to the broker-agent, and then forwarded to a task-agent. So that the broker-agent acted as a central aggregated cognitive service responsible for mapping.

## Limitations of the proposed approach and related work

The main goal of our work is to propose an experimental information technology that may allow automatic task accomplishment in the systems consisting of heterogeneous devices, robots, and computers. The technology is in the form of environment representation, language for task and service interface definition, and communication protocols. The technology should be implemented as a multi-agent system (consisting of service-agents, task-agents, info- agents, and broker-agents) on these devices and computers. In the paper only the idea of the proposed technology was introduced. The implementation was done on a small system, too small to verify the technology, however enough to explain how the main components of the technology work together.

The most related work was done by (Parker & Tang 2006). In their approach there are crucial notions that strongly correspond to the notions used in our approach. The first and most fundamental notion is *information type* that is supposed to have semantic meaning and defines the specific sensing or computational data of a schema or a sensor. However, this semantic meaning is not defined in (Parker & Tang 2006). It seems that our environment representation and the language may provide a semantic for such information types. The notions of perceptual schemas (PS), and motor schemas (MS), as well as communication schemas (CS) in (Parker & Tang 2006) correspond to our notions of cognitive services, physical services, and software services. So that it is possible to define interfaces of these schemas. It seems that the ASyMTRe approach can be reformulated in the terms of our approach.

There is an interesting approach in robotics that is also based on *software services*; it is called Decentralized Software Services Protocol (DSSP) (Nielsen & Chrysanthakopoulos 2007). It defines a software application as a composition of services that are lightweight software components created, manipulated, monitored, and destroyed repeatedly over the lifetime of the application. Most interesting features of such service are its behavior, and internal state. Any information that is to be retrieved, modified, or monitored using DSSP must be expressed as part of the service state. Service behavior specifies a way of message exchanges with other services in the composition, and related change of the service state. Hence, it determines how this very service can compose with other fixed services. Since the internal state of service may be monitored and even modified by other services, the service can not be modular and self contained. Moreover service invocation is specific and determined by the behavior, so that it is tightly coupled to the other services in the composition. Hence, DSSP does not follow the SOA paradigm. It may be viewed rather as an interesting programming platform for building complex applications from simple components.

In order to conclude the paper, let us mention once again that an approach to the problem of interoperability in open and heterogeneous multirobot systems was presented. It is based on the paradigm of Service Oriented Architecture (SOA), and a generic representation of the environment. If the approach deserves some attention, the proposed technology should be implemented (by different programmers) on a large number of heterogeneous devices providing a large variety of cognitive and physical services; then, the implemented system should be verified by automatic accomplishment of a large number of different tasks.

## Acknowledgment

The work was done within the framework of the project *Interoperability of mobile and cognitive devices* supported by the grant MNiSzW No. 3 T11C 038 29.

## References


Ambroszkiewicz, S., and Cetnarowicz, K. 2006. On the concept of agent in multi-robot environment. In Hinchey, M.; Rago, P.; Rash, J.; Rouff, C.; Sterritt, R.; and Truszkowski, W., eds., *Innovative Concepts for Autonomic and Agent-Based Systems*, volume 3825 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag.

Ambroszkiewicz, S.; Bartyna, W.; Faderewski, M.; Jakubowski, S.; Kocielinski, D.; Mikulowski, D.; and Terlikowski, G. 2006. Blind-enT: Making objects visible for blind people. In Jones, S., and MacDonald, P., eds., *Vision 2005–Proceedings of the International Congress*, volume 19. Elsevier.

enTish. 2003. enTish project. Technical report, IPI PAN and University of Podlasie, www.ipipan.waw.pl/mas/.

Hogan, N. 1985. Impedance control: An approach to manipulator, Part I - Theory, Part II - Implementation, Part III - Application. *Journal of Dynamic Systems, Measurement and Control* 107:1–24.

Kruijff, G.-J. M.; Zender, H.; Jensfelt, P.; and Christensen, H. I. 2007. Situated dialogue and spatial organization: What, where... and why? *International Journal of Advanced Robotic Systems, Special Issue on Human and Robot Interactive Communication* 4(2).

Mozos, O. M.; Jensfelt, P.; Zender, H.; Kruijff, G.-J. M.; and Burgard, W. 2007. From labels to semantics: An integrated system for conceptual spatial representations of indoor environments for mobile robots. In *ICRA-07 Workshop on Semantic Information in Robotics*.

Nielsen, H. F., and Chrysanthakopoulos, G. 2007. Decentralized Software Services Protocol - DSSP. Technical monograph, Microsoft Robotics Studio, http://msdn.microsoft.com/robotics/media/dssp.pdf.

Parker, L. E., and Tang, F. 2006. Building multirobot coalitions through automated task solution synthesis. *Proceedings of the IEEE* 94(7).

Robo-enT. 2007. Robo-enT project. Technical report, IPI PAN and University of Podlasie, ii4.ap.siedlce.pl.

SemanticWebServices. 2004. Web Services Activity. Technical report, W3C, www.w3.org/2004/WS2/.

Thrun, S. 1998. Learning Maps for Indoor Mobile Robot Navigation. *Artificial Intelligence* 99(1):21–71.

WebServices. 2002. Web Services Activity. Technical report, W3C, www.w3.org/2002/ws/.

Yeap, W. K., and Jefferies, M. E. 1999. Representation of the local environment. *Artificial Intelligence* 107(2):265–301.