PROPERTY SERVICE ARCHITECTURE FOR DISTRIBUTED ROBOTIC AND SENSOR SYSTEMS

Antti Tikanmäki and Juha Röning
Computer Engineering laboratory, University of Oulu, antti.tikanmaki@ee.oulu.fi, jjr@ee.oulu.fi

Keywords: Distributed robots and systems, multi-robot systems, sensor networks, Property Service architecture.

Abstract: This paper presents a general architecture for creating complex distributed software systems, called Property Service architecture. The system may contain resources like robots, sensors, and different kinds of system services, such as controller units, data storages, or a collective model of the environment. This architecture contains several solutions and distributed system design methods for developing advanced and complex systems. It also provides the possibility to add new resources to the system easily and during operation. Each service has a very simple generalized interface. This meets the requirements of distributed robotic applications, such as remote operation, multi-robot cooperation, and the robot's internal operation. The simplicity of the interface also provides a possibility to scale down the service even on the low-cost, low-performance microcontrollers used in small-sized robots. The main features of the architecture are the dynamic properties of the resources, automatic reconfiguration, and the high level of reusability of the implemented methods and algorithms.

1 INTRODUCTION

Creating a distributed system from several kinds of devices and functionalities is a very demanding task. Since, for example, a robot may contain many different kinds of functionalities, actuators, and sensors, the process of controlling is complex and the interface becomes very complex. Different systems cannot communicate with each other if there is no single commonly used communication standard. Figure 1 shows an example of a set of resources that a distributed system might have. The operating environment might have several sensors, lights and cameras. Different robots might need to be used, some of them might be legged, and they might contain several kinds of sensors. The human operator has a graphical user interface where tasks can be selected, and the system might have several automatic functionalities. Each device is used in different ways, and they can communicate in different ways. The lighting of the room is connected to a special control protocol, room cameras can be controlled using a serial port, and mobile robots use a wireless LAN. As the number of resources in the system increases, for example when a new kind of robot is developed to the system, the process of maintaining and updating the interfaces becomes very challenging.

For heterogeneous devices that take part in a larger system, a common problem is how to define an interface for each service in the system. This becomes even harder in multi-robot systems, where different kinds of robots with different capabilities. As the system improves, the interface must be changed and the whole system must be recompiled.

In real world applications when the communication channel between robots also unreliable, a common solution is to reduce communication and increase autonomous functionalities. Instead of sending drive commands to a robot, a target of movement is delivered to the robot. Increasing the capabilities of the robot leads to an increase in features that must be accessible through the interface. This sets either dynamic requirements for the interface or a need for continuous updating of interfaces.

To be able to achieve inter operation of a versatile set of robots and devices, very simple but flexible middleware is needed. The main requirement for the middleware is that it is possible to scale down to small-sized devices with a very limited set of capabilities. On the other hand, it must be able to provide an interface to varying features of resources in the system and expand as new resources

are added to the system. This paper proves that it is possible to implement flexible application layer that can be implemented on all kinds of transportation layers while providing all the capabilities of state-of-the-art distributed architectures.

Our solution, called Property Service architecture, has been developed to be a generalized interface for all kinds of devices or resources in a distributed system. Property Service provides the possibility to easily create and add new resources to a distributed system. The main criteria for the architecture design were simplicity, scalability, dynamics and expandability, and high reusability of system components. Dynamics and expandability make it possible to add new features and functionalities to services even during operation, as well as to integrate the results of different research topics into a larger system. This is essential to be able to build robotic systems with multiple capabilities.

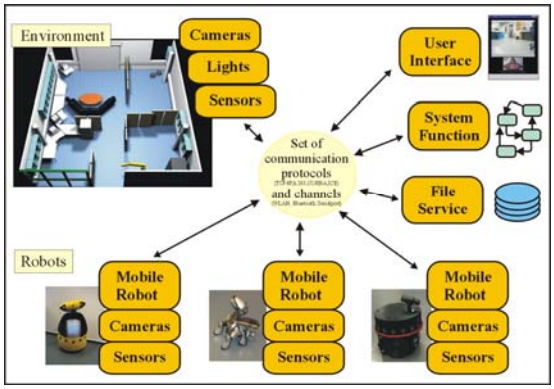

Figure 1: Set of resources that might be a part of a distributed system.

This paper describes the whole architecture that has been partly introduced on several papers previously. It will describe system-level services that improve the creation and management of distributed systems and heterogeneous multi-robot systems.

2 RELATED WORK

Many efforts have been made to create a more generally useful interface for networked robots. In many cases, communication is based on commonly used technologies like TCP/IP protocols and middleware like CORBA, Soap, etc. In multi robot architectures, several of these are based on CORBA or a real-time CORBA (OMG, 2002) extension. Examples of robotic architectures using RT-CORBA are Mobility software (Rea, 1999), the Player/Stage project (Gerkev et. al. 2003), Miro (Utz et. al. 2002) and ORCA (Brooks 2005), which is based on a CORBA component model. Later in ORCA2, CORBA has been replaced with Ice (Hemming, 2004) middleware, which provides improved features compared with CORBA

Wang et al. presented a COM-based architecture for the fusion of logical sensors (Wang et al. 2001). However their approach has some drawbacks, including platform dependency and a lack of network transparency. DDX architecture (Corke 2004) for a distributed system has been developed using UDP/IP to transfer data from remote sensors to a central data store. However their support for only lower-level data structures increase the amount of data transferred during operation and lacks expandability.

Algorithms and remote control code on heterogeneous robotics systems have also been developed with higher reusability in mind. CARMEN (Montemerlo, 2003) uses IPC to communicate between system components and provides a reusable navigation toolkit for multiple robotics. URBI (URBI www) scripting language support several kinds of robots and robot simulators. URBI also provides client/server-type networking for remote control of robots, in-language parallel processing and support for several commercial mobile robots.

Even though several possible solutions exist already, they have several disadvantages. Relying on a single communication protocol or middleware limits the possible platforms on which the service can run. As new features are added to the resource (e.g. a robot or sensor), the interface must be modified, which causes system-wide changes in the remote software that uses the interface. General middleware also requires a lot from the CPU, memory, and disc space, and it cannot be run on robots with an embedded CPU.

3 PROPERTY SERVICE

Property Service architecture provides tools for building new networked systems in just a few hours. It provides standardized data types and service types that can operate on different kinds of platforms. Property Service has been successfully used for remote operation of mobile robots (Tikanmäki, 2003), multi-robot cooperation (Mäenpää et al. 2004), remote operation of a swarm of robots (Tikanmäki, 2004), and to create a dynamic interface

for a modular robot (Tikanmäki and Vallius 2004). This paper presents the complete architecture and several system services, such as a storage service, a resource service, and grouping services, and explains how they can be used to build more complex distributed systems containing robots and other devices and system resources.

Properties are the features of each service, such as the sensors of a robot, or they are functional properties related to the operation of the service. Functional properties include, for example, the autonomous features of robots; or the tracking capabilities of a sensing device. Each property is a paired name and value and it can be set or requested. The value of each property is transferred as character arrays through a communication channel, and structured data is represented in XML format.

Each service has a special property called "*properties*", which contains a list of the properties currently available on the service. The value of "properties" changes during operation as the set of properties of the service changes, providing dynamic interface functionality.

Some properties may contain sub-properties. This hierarchic representation provides the possibility to request complex representations or, on the other hand, only special features of a property. The format is the same as that used in several programming languages for separating fields of structures. For example, by requesting the "location" property from a robot service, the client receives a 6D vector of the current location of the robot. The client can also request the "location.x" property, which returns only the x coordinate of the robot's current location.

A request for certain properties may also contain some parameters that specify, for example, the context of the return value. A good example of this is a request for robot's location, which might be requested in several coordinate systems. The client can request possible parameters using the ".parameters" extension on property name.

3.1 Service Interface

Property Service has a simple interface, which contains only two methods, "SET" and "GET", for getting and setting the properties of a service. For example, a connected client can GET the velocity of a robot through Property Service or SET the velocity of the robot. The terms client and server are misleading, as both the client side and the server side implement the Property Service interface. The term 'client service' is used when talking about user interfaces or other services that request properties from other services. Figure 2 shows the general principle of the architecture. In the Property Service architecture, user interface components can be thought of as a sensor that senses user inputs and provides these as properties. Therefore, the user interface can have properties that other services can set or get. This feature is used in the listening mechanism. A client service can request listening of a certain property of the other service. This service registers information concerning the connection of the client service to the requested property. Each time the value of the property changes in the service, an event is sent to the client service. The main advantage of this is that the client service does not need to continuously request value changes, thus reducing the amount of required communication. This is especially useful when changes in the property value occur over a long period.

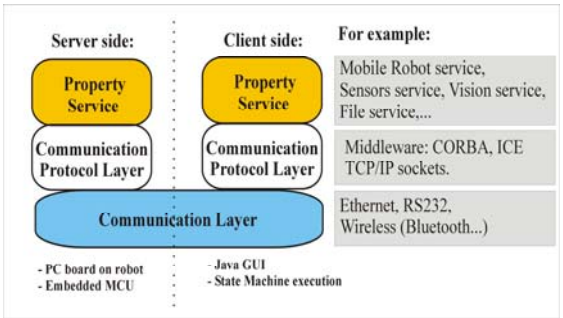

Figure 2: Principle and examples of property service layers.

3.2 Data Types

Property Service has a simple interface, which Several commonly used data types have been standardized to provide compatibility between services. The most commonly used ones are vector, list and markers. These data types are briefly introduced in the following.

A *vector* is an array of decimal numbers and its length can vary. For example, a request for a robot's location returns a 6-dimensional vector containing x, y, and z translations and a, b, and c rotations along each axis. A vector is also used to deliver measurements from various sensors, like distance sensors. A vector is also used to represent sequences, histograms, etc.

A *list* is a set of data types. It is used, for example, to list the properties that a service has. A list can wrap several kinds of data types, including a robot's path, which is presented as a list of locations (presented as vector types). The shape of an edge of an object can also be presented as a list of vectors containing image coordinates.

A *marker* is a structure that stores information about detected and tracked objects in the environment. For example, a robot's vision system provides markers for objects currently in the field of view. The marker structure contains location information and a set of detected features that the object has. For example, a ball recognized from the camera view can have recognized features like shape and color. Markers provided by different services can be collected into a model that represents current knowledge about the current environment. This provides the possibility to easily build sensor networks and swarm robotic systems.

Markers are used as input parameters for the behaviors of various services. In addition to physical objects they can also represent abstract entities that can be used to create new behaviors. For example, a target marker for a "move to" behavior that stays in front of a robot causes the robot to move forward. The measurements of each sensor can be presented as markers, which is highly useful in creating distributed sensing systems. For example, a swarm of robots produces a large amount of markers of the objects they detect. These markers are collected into one place to create an environmental model based on where the swarm operates.

Table I shows an example of each of the basic data types. As it can be seen, for example, the color of the marker is represented with a vector type. In most cases these data types are used, but each developer can also define their own data structures.

Even the interface remains the same; it is possible to make dynamic "interfaces" by changing the set of properties provided by a service. New properties can be attached to a service during operation, and they are shown when the client requests "properties" again. This feature is used in the Qutie robot (Tikanmäki and Vallius 2004), where the properties of a new attached hardware component appear on the list of properties as soon as they are available.

Table 1: Examples of commonly used data types.

Data type	**Example**
vector	(1.0 0.1 0.1)
list	((1.0 0.0) (2.0 0.1))
marker	<marker> <location>(1.0 0.1)</location> </marker>

As the value of the property in communication is an array of characters, a service can be implemented over any communication protocol containing the possibility to transfer data characters. The current implementation contains RS232, TCP/IP sockets, HTTP as well as several middleware like CORBA and ICE. To be able to communicate between various protocols, special proxies have been made to transfer a call from one media to another. As an example, the Property Service in Sony's AIBO robot communicates through a TCP/IP socket and a Wireless LAN with a PC that contains a wrapper to convert property calls to other protocols, such as ICE or CORBA calls. This provides the possibility to control AIBO from any property client without knowledge of how the AIBO is actually connected to the system. The implementation of a proxy is simple, as it simply delivers property calls from one protocol to another.

3.3 GUI as a Property

A service may contain special properties that can be used to create user interfaces automatically. Several of properties may have a ".gui" extension property that returns the preferred GUI component for the property. The service can also provide a class that contains the compiled code for the GUI component. For example, the "movement.gui.java" property returns the java class that can operate property movement on the service. If a new version of the GUI component is then developed, it can be received from the service without a need to change the client side user interface. Instead of requesting the GUI components from each service they can also be requested from a special “GUIPropertyService”, whose purpose is to provide standard GUI components for various property types

3.4 Data Transfer Principles

The ideology of Property Service is to always reduce the need for data delivery if possible. One way to do this is by implementing several automatic features for services. Instead of continuously sending moving commands to a robot, we prefer to send only target location or several route points. Instead of sending raw image data, we prefer to request only the targets that vision system can recognize from the view. If raw images are needed for remote driving of a robot, a commonly used compression format, such as JPEG, is used.

Several standard sets of properties have been used for different services. As each service delivers data in the same format, and understands the same commands, different services can be used by the client service. A good example of this is to use different robots with the same user interface.

3.5 Platforms

Currently services are implemented using C, C++, Java, PHP or Python. Services can be compiled and run with Windows and different Unix systems. A Property Service for AIBO's operating system Open-R has also been developed.

As one criterion for the architecture was simplicity, services can also be implemented on devices containing low calculation power. In many types of robotics or sensor nodes on sensor networks, it is reasonable to use embedded microcontrollers instead of a PC. Property Service has also been implemented using standard C and can be used on many embedded platforms, for example, in "Atomi" boards containing an Atmel 8-bit AVR microcontroller (Tikanmäki and Vallius 2004) and ARM-based embedded computers running embedded Linux. These two platforms are good examples of systems that cannot run complex middleware.

3.6 Implementing New Resources

The implementations of Property Service include several functions and classes that facilitate the creation of new services. Automatic features, such as a dynamic set of properties, are ready for use in the PS core library. One of the most useful tools is called Class Property Service. Class Property Service is a class that creates a service automatically from given class. The automatic service creation is available for Java and C++ classes. The main advantage of this service is that it facilitates the reuse of previously built classes. For example, researchers may want to be able to use their old manipulator remotely. The manipulator has a control box that can be used from a PC through a serial port. By using Class Property Service, a previously made control class can be used as a part of the distributed system, as the Class Property Service creates properties from methods and attributes of the class according to defined rules.

In addition to robots and sensors, Property Service can be used to control various other devices in the environment. Since the interface is simple and can also be used in simple microcontrollers, different kinds of devices, like electric locks, motion sensors or room light control units, can be included in the distributed system. This provides the possibility for a robot to use these resources, which increases the possibilities of new applications.

3.7 Standard Services

Some commonly used types of services have been standardized, so that each service in the type category provides at least a certain set of properties. The main advantage of this is the possibility to use several kinds of services (or system resources) without a need to modify the control program. A good example of this is that the user can change the robot into a different kind of robot to do the same task without modifying the control program.

3.7.1 Mobile Robot Service

The main standardized service is a mobile robot service. A mobile robot service contains properties related to movement, sensing and physical dimensions. Each robot's property service provides the same properties. Table 2 shows a basic set of properties of a mobile robot service.

Table 2: Examples of commonly used data types.

location	6D global location of the robot, containing x, y, z and rotations around the axis
velocity	6D movement of the robot, including translations and rotations
actuators	List of actuators on the robot, contains sub-properties for each actuator
sensors	List of sensors, with several subproperties
behaviors	List of the robot's primitive functionalities. All robots must have at least a "move to" behavior.

Standard properties are the same regardless of the moving mechanism of the robot. When the robot is requested to move forward one meter (by using a "*behaviors.moveto*" functional property with a marker as a parameter), a walking Aibo or a wheeled Scout robot moves one meter forward according to its moving capabilities. If a robot contains additional actuators, such as arms or legs, properties for them are added to a list of *actuators* properties, but the basic functionality remains the same. The main advantage of this is the possibility to control all kinds of mobile robots with the same control software or user interface while also providing special features of each robot.

3.7.2 Sensor Service

One main type of service is sensor. A sensor service includes different sensors, such as sonar and cameras, as well as different controlling devices, e.g. joysticks, GPS devices and touch pads. User interfaces are also like sensors, because they sense

the operational requests of the human user. The basic properties of sensor services are "location", which contains the location of the sensor in its context, and "data", which returns the raw data captured by the sensor. The default coordinate system for sensors location is relative to its base, for example, the robots origin. By using request parameters, different coordinate systems can also be used. In addition to raw sensor data, various kinds of refined information can be attached to the sensor's properties. The most advanced system is the vision sensor's property set.

Vision service is the most advanced sensor service and a good way to integrate several kinds of machine vision methods to a robotic system. Several different methods can be activated separately, and the result of image processing can be requested from the service. Results can be requested in several forms. For example, a remote client can request an edge-detected image or segments of edges from the vision service. The vision service is the interface to one or several cameras, but it can also process images sent by other services. As the interface is always the same, different kinds of cameras can be used in the same way. All targets detected by the vision sensor can be received in marker format.

3.7.3 Resource Service

To be able to find the available resources in s distributed system, a special Resource Property Service has been introduced. The properties of this service are references to services available in the system. New resources can contact it to notify of their presence. When a service contacts the resource service, it gives information how it can be contacted, like its address and protocol(s). Services are then shown as properties of Resource Property Service and each property contains the name of the service and contact information. For example, CORBA-based services' contact information is the IOR of the service and TCP/IP or ICE services indicate their IP address and the port where they can be reached.

The resource service actively checks the existence of services and removes the resources that are not available any more. Resource Property Service also contains several functional properties that can be used to search services using parameters. For example, a client might want to find all available mobile robots located in a certain room. For the search request, Resource Property Service searches for services that are mobile robots and whose "location" property matches the client's request. The resource service also starts up new system services when necessary.

3.7.4 Grouping Services

Another architectural solution is to group several services into one Property Service. A good example of this is to group a set of simple miniature robots into one Swarm Property Service (Tikanmäki 2004). Miniature robots with low computational power are commanded through a common radio channel. Each robot's properties are shown in a hierarchical set of properties of the swarm service, shown in Table 3. The properties of a single robot can be reached through these properties, and each property of the mobile robot service interface becomes a sub-property of Group Property Service with a "robots.<name>" prefix. The "<name>" parameter changes according the robot currently used. In addition to a single robot's properties, Grouping Property Service contains various functional properties that can simultaneously control a larger set of robots. For example, the client wants a swarm of robots to move to a certain destination. Grouping Property Service can create a sub set from selected robots and order each individual to move to the destination. In the case of group of mobile robots, a groups of robots are controlled by properties similar to those used to control an individual robot. For example, Swarm Property Service has "behaviors" properties that control the whole group.

Table 3: Examples of properties of swarm service.

robots	robots currently belonging to this swarm each robot has sub-properties included in robot service properties
location	location of the swarm
behaviors	List of the primitive functionalities. Similarly to single robots, group can be controlled with same behaviors like "move to" behavior.

Resource Property Service and Group Property Service have several similarities, and Resource Property Service can be expanded to act as a grouping service. In this case, the resource service reroutes the property request. Instead of giving the reference of the service to the client, it acts as a proxy and asks for the property value from the service and delivers the reply to the client. This feature is useful in, for example, a long latency system and a long request time in some cases, because the grouping service is already connected to the service, and therefore the client does not need to make a direct connection. These services also provide full transparency to the system, as the client

does not need to know the address or even the number of robots taking a part in the action.

3.7.5 State Machine Service

In addition to user interface client services, some control services have also been created. One of the main services is State Machine Service. It provides a resource that can execute RDF-formed state machines, which can control complex conditional sequences, being able to provide multiple property services at the same time. This has been described previously in (Mäenpää 2004). State Machine Service can control a large set of services simultaneously. Currently, State Machine Service also has a visual user interface, where the user can create and modify state machines graphically and during operation. By using Storage Service, introduced below, the state machine or part of it can be stored into a database for further usage or execution.

3.7.6 Storage Service

Another architectural service is Storage Service. It is a combination of file transfer systems and databases. The service can be started for a specified directory in operating systems and all (or only selected) files are shown as properties of the service. A remote client can request a property, named according to the filename, and receive the value of the property, which is the content of the file. The same procedure can be used to store new files and data into a service, by setting a new property in the service. For example, a robot can send captured images to a storage service, which stores the images in the memory (or on a hard disc) of the service. As in Resource Property Service, Storage Service may also contain several intelligent search methods, and the client can request files according to specified features. Storage Service has been used to store captured data such as images, to upload and download control sequences and state machines, and to request GUI components remotely.

3.7.7 Environment Model as a Service

To be able to operate, a distributed system must have knowledge of the environment where the services, such as robots and sensors, are operating. If the system is not designed to be fully reactive, it is reasonable to collect information received by services. The system can have one or several environment model services. If only one model is used, all the services send their measurements to this service, and the model service performs data fusion and updates the model. Each service might also have its own model, which is a local representation of the service's way of receiving information from the environment. For example, a ceiling camera might have an environment model service that provides information received using the camera. Markers are used in most communication and data storing, but other methods are also possible. For example, a model can contain a grid of temperature values gathered by a swarm of small robots or a sensor network. In this case, the temperature map is updated according to the locations of the measurements.

4 COMPARISON

The overall performance of the system depends on the computational power of the platform and the speed of the communication channel used. However, in a comparison of the development times of a new service and cooperation between different middleware, Property Service provides a great advantage. The amount of required lines of code is very small compared with, for example, CORBA-based robot middleware that provides the same features and functionalities of the resource. As the capabilities of a robot or other service increase, it becomes even more useful that the actual interface does not need to be changed.

As the code is highly reused, new services are fast to implement and all system services like state machines, and storage services are available for use immediately. For example, the "move to" behavior is common to all robots; no matter if they are legged, wheeled, or even a manipulator. System services also provide great advantages in building applications for distributed systems. This provides even more advantages compared with other systems.

Passing messages as text strings is expensive compared with other formats (like language-specific byte arrays). The performance of communication has been improved by sending only essential information and by using refined data instead of continuous control commands or raw sensor data.

Several applications and example systems have been created using Property Services. Property Service has been successfully used for remote control of several mobile robots that provide audio and video data, and receive several moving commands (Tikanmäki 2003) using CORBA middleware and a wireless LAN. It has also been used to create remote operation for a swarm of simulated robots (Tikanmäki 2004). Both are good examples of applications where quick response between services

is an essential requirement. Multi-robot cooperation and designing of operation using state machines has been demonstrated in reference (Mäenpää 2004).

5 CONCLUSION

The main advantage of Property Service in compared with competitive architectures is the ease of adding new resources to a distributed robotic system. Using Class Property Service, classes that are already available can be used remotely and connected to a larger system quickly and without a need to implement a code related to communication between services. As Property Service can be implemented on various communication channels and platforms, different kinds of new resources can be attached to the system. The usability of the architecture is not limited to robotics, it can also be used in other distributed systems, for example, in home automation systems, sensor networks, and industrial automation. As these devices become part of the architecture, they can be operated remotely or by the robot's control software, and robots easily became a part of the system. Using Resource Property Service, the robot can, for example, search for the light service of a room and switch on the light on the room upon entering the room. Complex applications built using state machines are easy to change, and the user can monitor their operation online using state machine visualization.

ACKNOWLEDGEMENTS

This work has been partly funded by Finnish Academy.

REFERENCES

OMG. *Real-Time CORBA Specification. Object Management Group, Inc.*, 1.1 edition, August 2002.

Rea 1999. *Mobility 1.1, Robot Integration Software, User's Guide*, iRobot Corporation, MobilityTM software

Gerkey, B., Vaughan, R. T. & Howard, A. 2003, *The player/stage project: Tools for multi-robot and distributed sensor systems*, in `Proceedings of the 11th International Conference on Advanced Robotics (ICAR'03)', Coimbra, Portugal, pp. 317–323.

Utz H., Sablatn¨og S., Enderle S., and Kraetzschmar G. 2002. *Miro - middleware for mobile robot applications.* IEEE Transactions on Robotics and Automation, 18(4), August 2002.

Brooks A., Kaupp T., Makarenko A., Orebäck A. and Williams S. 2005. *"Towards Component-Based Robotics"*. IEEE/RSJ International Conference on Intelligent Robots and Systems 2005.

Henning M., Spruiell M. 2004 *Distributed Programming with Ice,* ZeroC Inc., http://www.zeroc.com

Wang J., Su J., and Xi Y., 2001 *"COM-based software architecture for multisensor fusion system,"* Information Fusion, vol. 2, no. 4, pp. 261– 270,.

Corke P., Sikka P., Roberts J., E. Duff, *"DDX: A Distributed Software Architecture for Robotic Systems",* Australasian Conference on Robotics and Automation 2004

Montemerlo M., Roy N., and Thrun S.. *Perspectives on standardization in mobile robot programming: The carnegie mellon navigation (carmen) toolkit.* In IEEE/RSJ Intl. Workshop on Intelligent Robots and Systems, 2003.

URBI *http://www.urbiforge.com/*

Tikanmäki A., Röning J., Riekki J. 2003 *Remote operation of robotics systems using WLAN- and CORBA based architecture,* SPIE / Intelligent Robots and Computer Vision XXI, Oct 27 - 31, Rhode Island, Providence, RI USA, 2003

Mäenpää T., Tikanmäki A., Riekki J. and Röning J., , 2004 *A Distributed Architecture for Executing Complex Tasks with Multiple Robots,* IEEE 2004 ICRA, International Conference on Robotics and Automation,Apr 26 - May 1, New Orleans, LA, USA

Tikanmäki A., Röning J. 2004 *Advanced Remote Operation of swarms of Robots,* SPIE / Intelligent Robots and Computer Vision XXII, Philadelphia, USA

Tikanmäki A., Vallius T., Röning J., 2004 *Qutie - Modular methods for building complex mechatronic systems,* ICMA - International Conference on Machine Automation, Nov. 24.-26., Osaka, Japan