Property Service Architecture for distributed robotic and sensor systems

Antti Tikanmäki, Juha Röning,
Intelligent Systems Group, University of Oulu, Finland

Abstract - This paper presents a general architecture for creating complex distributed software systems, the Property Service architecture. The system may contain resources like robots, sensors, and different kinds of system services, such as controller units, data storages or collective model of the environment. This architecture contains several solutions and distributed system design methods to develop advanced and complex systems. It provides also a possibility to add new resources to the system easily and during the operation. Each service has 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 microcontrollers used in small-sized robots. The main features of the architecture are dynamic properties of the resources, automatic reconfiguration and the high level of reusability of the implemented methods and algorithms.

Index Terms – distributed robots and systems, multi-robot systems, sensor networks, Property Service architecture

I. INTRODUCTION

Creating a distributed system from several kinds of devices and functionalities is a very demanding task. Since, for example, a robot, may contain a lot of different kinds of functionalities, actuators and sensors, the process of controlling is complex and the interface become 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 the 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 the 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 wireless LAN. As the number of resources on the system increase, for example new kind of robot is added to the system, the process of maintaining and updating the interfaces becomes very challenging.

For heterogeneous devices that take a part on larger system it is a common problem of how to define an interface for each service on 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 whole system must be compiled again.

If communication channel between robots is unreliable, the common solution is to reduce communication and increase the autonomy. Instead of sending drive commands to robot, target for movement is delivered to the robot. Increasing the capabilities of the robot leads to increase in features that must be accessible through the interface. This sets either dynamic requirements for interface or need for continuous updating of interfaces.

To be able to achieve intersystem operation for versatile set of robots and devices, a very simple but flexible middleware is needed. Main requirement for the middleware is that it is possible to scale down to small size devices with very small amount of capabilities. On the other hand, it must be able to provide interface to varying features of resources on the system and expand as new resources are added to the system. Flexible application layer that can be implemented on all kinds of transportation layers is possible to implement while it can provide all capabilities of state of the art distributed architectures.

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

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

II. RELATED WORK

Many efforts have been made to create more generally useful interface for networked robots. Their communication bases on commonly used technologies like TCP/IP protocols, middlewares like CORBA, Soap, etc. On multi robot architectures, several of these based on CORBA or real-time CORBA [1] extension. Examples of these architectures are Mobility software [2], The Player/Stage project [3], Miro [4] and ORCA [5] that...
based on CORBA component model. Later in ORCA2, CORBA has been change to Ice [6] middleware that provides improved features in compared with CORBA.

Wang et al. presented a COM-based architecture for the fusion of logical sensors [7]. Their approach does however have some drawbacks, including platform dependency and the lack of network transparency. DDX architecture [5] for distributed system has been developed using UDP/IP to transfer data from remote sensors to a central data store. This however increase the amount of data transferred during operation.

For higher reusability for algorithms and control code on heterogeneous robotics systems has also been developed. CARMEN [9] uses IPC for communicating between system components and provides reusable navigation toolkit for robotics. URBI [10] scripting language supporting several kind of robots and robot simulators. URBI provides also client/server type networking to remote control of robots and provide parallel processing and supporting several commercial mobile robots.

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

III. PROPERTY SERVICE

Property Service architecture provides tools to build new networked systems in just a few hours. It provides standardized data types and service types that can operate in different kind of platforms. The Property Service has been successfully used for the remote operation of mobile robots [11], multi-robot cooperation [12], the remote operation of a swarm of robots [13], and for creating a dynamic interface for a modular robot [14]. In this paper the complete architecture is presented and several system services, such as file service, resource service, grouping services are shown, and how they can be used to build more complex distributed systems containing robots and other devices and system resources.

A. Properties

Properties are the features of each service, such as sensors of a robot, or they are functional properties related to the operation of the service. Functional properties include, for example, autonomous features of robots, or tracking capabilities of sensing device. Each property is a pair of 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 the XML format.

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

B. Service Interface

Property Service has a simple interface, which only 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 the robot through Property Service or SET the velocity of the robot. 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 are 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 as a sensor that sense the user input and provide them 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 a listening for a certain property of the other service. This service registers the information concerning the connection of the client service to the requested property. Each time the value of property changes on the service, an event is sent to the client service. The main advantage of this is that client service does not need to continuously request the value changes, which reduces the amount of required communication. This is especially very useful when changes in the property value occur during long period.

![Image](image)

**Fig. 2 Principle and examples of property service layers**

### C. Sub-properties

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

### D. Additional parameters for a request

The 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 of the robot’s location, which might be requested in several coordinate systems. The client can request possible parameters using the sub-property ".parameters".

### E. Data types

Several commonly useful data types have been standardized to provide compatibility between services. The most commonly used ones are vector, list and markers. The data types are briefly introduced following.

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

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

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

Markers are used as input parameters for 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 “move to” behavior that stays in front of the robot causes the robot to move forward. The measurements of each sensor can be presented as markers, and this it is highly useful for 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 in one place to create one 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 define also own data structures. Even the interface remains the same; it is possible to make dynamic “interfaces” by changing the set of properties provided by the service. New properties can be attached to a service during an operation and they are shown when the client requests “properties” again. This feature is used in the Qutie robot [14] where the properties of a new attached hardware component shows on the list of the properties as soon as they are available. |
As the value of the property in communication is an array of characters, the service can be implemented over any communication protocol containing 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 the various protocols, special proxies have been made to transfer a call from media to another. As an example, the Property Service in Sony’s AIBO robot communicates through a TCP/IP socket and Wireless LAN with a PC, which contains a wrapper to convert property calls to other protocol such as ICE or CORBA calls. This provides a possibility to control AIBO from any property client without knowledge of how the AIBO is actually connected to the system. The implementation of proxy is simple, as it simply delivers property calls from protocol to other protocol.

F. Creating a dynamic GUI using a GUI property

The service may contain special properties that can be used for creating user interfaces automatically. Several of properties may have a sub-property ”.gui” 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 property ”movement.gui.java” returns the java class that can operate property movement. 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”, which purpose is to provide standard GUI components.

G. Data transferring principles

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

Several standard set of properties has been used for different services. As each service delivers data in the same format, and understands the same commands, the different services can be use by the client service. A good example of this is to use different robots with the same user interface. Standard services are later discussed on Chapter 3.

H. Platforms

Currently the services are implemented using C, C++ or Java. Services can be compiled and run in Windows and Linux. Also property service for AIBO’s operating system Open-R has 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 PC computer. Property service has been implemented also using standard C and can be used in many embedded platforms, for example, in “Atomi”-boards containing Atmel 8-bit AVR microcontroller [14].

I. Security and authentication

In the remote operation of physical devices and other services, the security issues need to be considered seriously, especially if the resources are connected to a hostile network, such as Internet or wireless LAN. The Property Service architecture has built in an authentication system, as the IP address (or other identification of the address of the request) of the client service is received by the service. Using this address, the service can limit the access to certain properties. A client with a certain address might have administrative access to the service and therefore have more properties to set and get.

The properties of a service can also be used for simple authentication. The service might initially have only a “password” property available on the list of properties. When the client sets the “password”-property, rest of the properties become available to the requests from client’s address.

The property name and value are transferred mainly in a human-readable form. To be able to secure the content of the properties from third party listeners on hostile communication channels, Property Service requests can be encrypted using SSL. The service can also rely on other security services provided by used protocol level.

J. Implementing resources

The implementations of the 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 to use in the PS-library. One of the most useful tools is called Class Property Service. Class Property Service is class that creates Property service automatically from given class. The class here is either Java or C++ class, depending on programming language. The

<table>
<thead>
<tr>
<th>Data type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector</td>
<td>(1.2 1.3 1.5)</td>
</tr>
<tr>
<td>List</td>
<td>&lt;list&gt;(1.2 1.3 1.5) &lt;list&gt;</td>
</tr>
<tr>
<td>Marker</td>
<td>&lt;marker&gt; &lt;location&gt;(1.2 1.3 1.5) &lt;location&gt; &lt;feature&gt;color = (1.0 0.3 0.5) &lt;/feature&gt; &lt;/marker&gt;</td>
</tr>
</tbody>
</table>

TABLE I
examples of commonly used data types
main advantage of this service is that it helps to reuse the 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, previously made control class can be used as a part of the 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 controlling various other devices in the environment. Since the interface is simple and can also be used in simple microcontrollers, different kinds of devices can be included in the distributed system, for example, electrical locks, or lights. This provides a possibility to robot use these resources, which increases the possibilities of new applications.

IV. STANDARD SERVICES

Some of the 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 need for modifying 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.

K. Mobile Robot services

The main standardized service is a mobile robot service. The 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 for a mobile robot service.

The 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 “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 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 providing also special features of each robot.

L. Sensor services

One of main service type is sensor. The sensor service includes different sensors, such as sonar and cameras as well as different controlling devices, e.g. joysticks, GPS devices and touchpads. The user interfaces are also like sensors, because they sense the operational requests of the human user. The basic properties of the 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 the 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 can also process the images send 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 vision sensor can be received in marker format.

M. Resource service

To be able to find the available resources on the system, a special Resource Property Service has been introduced. The properties of this service are references to services available on the system. New resources can contact it to notify their presence. When the service contacts to resource service, it gives the information how it can be contacted, like address and protocol(s). Services are then shown as properties of Resource Property Service and each property contain the name of the service and contact information. For example, the CORBA based services contact information is IOR of the service and the TCP/IP or ICE services indicate the IP address and the port where it 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, client might want to find all available mobile robots located in a certain room. For the search request, Resource Property Service searches the services that are mobile robots and whose "location" property matches the client’s request. Resource service also starts up new system services when necessary.

N. 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 [13]. 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. Properties of single robot can be reached through these properties, each property of mobile robot service interface become sub-property of Group Property Service with a “robots.<name>” prefix. The parameter “<name>” changes according the robot used currently. In addition to a single robot's properties, Grouping Property Service contains various functional properties that can control a larger set of robots at the same time. For example, the client wants a swarm of robots to move to a certain destination. The 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, 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>
<thead>
<tr>
<th>TABLE III SOME EXAMPLES OF PROPERTIES OF SWARM PROPERTY SERVICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>behaviors</td>
</tr>
<tr>
<td>behaviors.&lt;behname&gt;.members</td>
</tr>
<tr>
<td>robots.list</td>
</tr>
<tr>
<td>robots.&lt;name&gt;.location.x</td>
</tr>
</tbody>
</table>

The 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 reference of the service to the client, it acts as a proxy and asks the property value from service and delivers the reply to the client. This feature is useful in, for example, a long latency system and the request time in some cases, because the grouping service is already connected to service, and the client does not therefore need to make a direct connection. These services provide also fully transparency to the system as client does not need to know the address or even number of robots taking a part on the action.

O. State Machine Service

In addition to user interface client services, some control services have also been created. One of the main services is the State Machine Service. It provides a resource that can execute RDF-formed state machines, which can control complex conditional sequences, able to multiple property services at the same time. This has been described previously in [12]. State Machine 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 the operation. By using the File Property Service introduced below, the state machine or part of it can be stored into a database for further usage or execution.

P. File Property Service

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

Q. Environment model service

To be able to operate, the distributed system must have knowledge of the environment where the services, such as robots and sensors, are operating. If the system has not designed to be a 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 services send their measurements to this service, and the model service does the data fusion and updates the model. Each service might also have own model; which is a local representation of service’s way to receive the information from the environment. For example, a ceiling camera might have an environment model service, which provides information received using the camera. Markers are used in most communication and data storing, but others methods are also possible. For example, model can contain
V. 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 comparison in development time of new service and cooperation between different middleware, Property Service provides great advantage. The amount of required lines of code is very small in compared with, for example, CORBA based robot middleware that provide same features and functionalities of resource. As robot’s or other service’s capabilities increase it become even more useful that 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 file services are available to use immediately. For example, behavior “move to” is common to all robots, no matter if it is legged, wheeled, or even a manipulator. System services provide also the great advantages on building applications for distributed systems. This provides even more advantages in compare with other systems.

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

Several applications have been created by 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 [11] using CORBA middleware and wireless LAN. It has also been used to create remote operation for a swarm of simulated robots [13]. Both are good examples of applications where quick response between services is essential requirement. Multi-robot cooperation and designing the operation using state-machines has been demonstrated in reference [12].

VI. CONCLUSION

The main advantage of the Property Service is the ease of adding new resources to the system. Using Class Property Service, the classes that are already available can be used remotely and connected to a larger system quickly and without need to implement a code related to communication between services. As Property Service can be implemented on various communication channels and platforms, different kind of new resources can be attached to the system. The usability of the architecture is not limited to robotics but 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 robots control software, and robots became a part of the system easily. Using a Resource Property Service, the robot can, for example, search the light service of the 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.

REFERENCES