MULTI-ROBOT SOFTWARE PLATFORM BASED ON ROBOTIC DEVICE SERVER PLAYER

Alejandro Morales, Miguel A. Gutiérrez
Depto. de Informática y Ciencias de la Computación, Universidad Católica de Ávila, Ávila, Spain

Jose A. Vicente, Vidal Moreno, Belén Curto
Depto. Informática y Automática, Universidad de Salamanca, Salamanca, Spain

Keywords: Software Platform, Architectures, Distributed System Control, Autonomous, Mobile, Robots, Environment.

Abstract: This article describes a software platform that allows to control multiple robots of any type, through wireless connections and without needing to modify its code to control each particular robot. It is a platform with an architecture in three layers, that uses the robotics device server *Player* as intermediate layer. The most abstract layer of the architecture is composed by the applications of control elaborated in any language that has socket support. These applications use the interfaces that *Player* offers to the control of the devices, so that the access to it is transparent. A server application is the most specialized layer that runs on the robot, and it manages the sensors and actuators devices of the robot at *Player*'s requests. The platform presents two interesting aspects, the first is that allows to control any robot, without having to develop specific drivers in *Player* that allow to control their devices. That is to say, it is not necessary to modify the code of the platform to integrate a new robot, simply it is necessary to adapt a model of server application, which accedes to the devices, to the robot that is wanted to integrate. The second one, is the possibility of controlling multiples robots simultaneously through wireless connections (also it admits serial connections). Also, it allows to create valid control programs for any robot, without needing to know its operation and architecture. By these reasons, it constitutes a very valid enviroment to work with multi-robot distributed systems.

1 INTRODUCTION

One of the complex aspects at the time of working with multi-robot systems is the software development. This is because in many cases robots have different devices and architectures, which forces to create specific drivers for each device and to elaborate exclusive programs, that make use of those drivers, adapted to each particular robot.

At the moment, software tools exists that try to simplify the creation of control programs for the handling of the robotic devices. Some are distributed under license and only allow to work with robots of the company that distributes them, others only allow the control of one robot simultaneously. Shapira (Saphira robot control system) and Ayllu (Werger, 2000) are two examples, that are distributed under license by *ActivMedia (*ActivMedia Robotics) and they only allow to work with robots Pioneer.

CARMEN (Carnegie Mellon Robot Navigation Toolkit) is another tool, in this case open source, composed by a set of servers applications who simplify the control and navigation of mobile robots.

Another example is *Player (Player/Stage Project)*, that tries to simplify the software development, to handling multiple robots of any type (Gerkey, 2003). It is an open source robotic device server that supplies through the network and in a transparent way for the control programs (clients) which interact with it, a control of the devices (sensors and actuators) that have the robots.

It offers a set of device interfaces that the robotic control programs use to interact, through it, with the robot devices.

Player establishes a distinction between the interface of a device and the drivers of the same one. The reason is that an only interface for all the devices of the same type exists. That is to say, *Player* implements a set of drivers, one for each

Morales A., A. Gutiérrez M., A. Vicente J., Moreno V. and Curto B. (2005).
MULTI-ROBOT SOFTWARE PLATFORM BASED ON ROBOTIC DEVICE SERVER PLAYER.
In *Proceedings of the Second International Conference on Informatics in Control, Automation and Robotics - Robotics and Automation*, pages 435-438
DOI: 10.5220/0001179604350438
Copyright © SciTePress

particular model of a certain device, and associates all of them to a same interface. In that way the control programs can interact with a device, using their interfaces, without having to know the particular characteristics of its driver. For instance: there is only one interface '*position'* that allows to control the movement devices of any robot, and it is implemented by some drivers (one by each concrete device that allows to control) adjusted to this interface.

Player provides already implemented a set of drivers that allows to control commercial devices of some robots (Pioneer 1, Pioneer 2, *AmigoBot*, RWI B-series robots...). But if you want to use it to control robots of own creation, or other ones that are not between the implemented ones, it is necessary to develop and implement a new specific driver for each device of the robot, as each robot has its own characteristics of hardware (microcontroller), system media, etc.

Player can be executed in the most of the UNIX/LINUX systems. It allows controlling multiples devices and it offers the possibility of modifies its code to add news drivers and/or device interfaces.

Control programs (clients) communicate with *Player* across a socket TCP, asking for data of the sensor devices, sending command to actuators and configuring devices. *Player* server admits multiples clients connected at the same time, one by socket, and these don't have to be in the same machine, simply they should be executed on a computer located in the same network that the server. As well as client programs can be written in any language of programming that provides socket.

A very interesting aspect is the control of multiple robots using TCP/IP wireless connections, because they offer a greater operational range and allow to control from an only PC (or robot), that is in the same network, all robots. The problem is that to do this with *Player*, we would have to execute an instance of *Player* in each robot, something that in most of robots is not possible (because they need to have installed an operating system type UNIX/LINUX). The reason is that drivers of *Player* access, normally through serial connections, directly to the devices and that is not possible using TCP/IP connections. The software platform that we propose allows to control multiple robots using only an instance of *Player*, that can be executed in a PC of the network or in a robot, and it does not need that robots has a great capacity of processing: it can be minirobots like *K-team* robots (Khepera robots).

In the following sections of this article the proposed software platform is described, its architecture, and conclusions.

2 PROPOSED SOFTWARE PLATFORM

The software platform that we propose uses *Player* as its intermediate layer and is thought to control multiple robots of any type, through wireless connections and without needing to modify its code to control each particular robot.

It tries to facilitate complex aspects such as the necessity to create a new driver for each device that you want to control. Also it allows to control the devices through TCP/IP connections.

In order to avoid having to create a new driver in *Player* for each new device that we want to control, we have developed a generic driver for each generic type of device (one to sonar, other to IR, another one for position, another one for battery...). These generic drivers don't have access directly to the devices, it is done through a server application, that is executed in the robot control module. The communication between generic drivers and the server application makes use of an own protocol and it is possible to be made through wireless connections.

This way to work is going to allow us to control, without needing to add new exclusive drivers to *Player*, any type of robot that incorporates the server application model that we propose. In this way, it is the robot that has to adapt to the software platform and not this one to each particular robot.

The platform allows to control multiple robots from an only PC (or robot), through a wireless network, being able to execute the control programs in any PC of the network.

2.1 Platform Architecture

It is an architecture in three layers (see figure 1).

The top layer is composed by the control applications elaborated in any high-level language. These applications communicate, making use of the interfaces and through TCP socket, with the intermediate layer (*Player*).

The lowest layer is composed by a server application which is executed in the control module of the robot, and has access to the robot devices. The communication between the server application and *Player* is made by TCP socket, following an own protocol (see figure 3).

Both *Player* and server application of the robot support multiple connections, so a control application can work on several robots simultaneously.

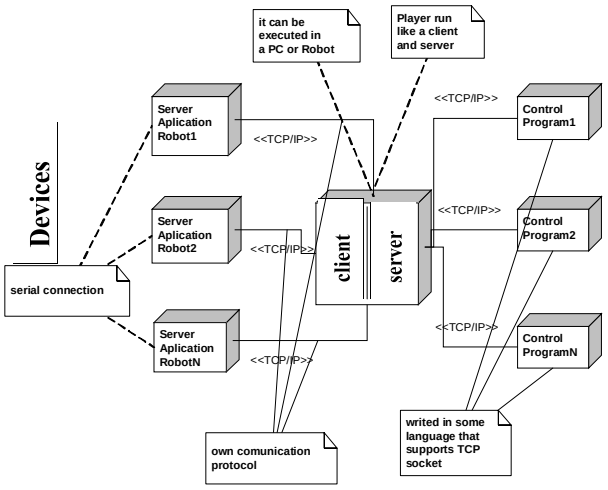

Figure 1: Layers of the software platform

2.1.1 Server Application

This server application has been designed to control a large number of devices simultaneously and allows to add new devices in a simple way. It is implemented easily to its integration in any kind of control module.

It is a matter of an application whose mission is to take care of requests of *Player* towards the devices. It is a multiprocess application because *Player* makes simultaneous requests on multiple devices, and it maintains a process taking care of the requests that *Player* makes on each one of them (see figure 2).

The different types of process are the following ones:

- a main process that is the one in charge of creating and finalizing the rest of the process, as well as taking care of the new connections made from the client part (generic drivers of *Player*).
- a process in charge of making readings of the sensorial devices and acting on the actuators devices.
- a set of process in charge of taking care of generic requests of reading and actuation made from generic drivers.

Two clearly differentiated parts can be emphasized. The first one manages the processes and takes care of requests of *Player*, following a communication protocol. The second one, makes the access to the devices or reads the value of the sensors or acts on the actuators (see figure 2). This division allows that the incorporation of new devices is as simple as it is possible.

The functions that communicate with the devices are encapsulated in a library, offering an interface to access them. In this way the server application can be adapted to any robot modifying only the functions of control and it is not almost necessary to modify the application.

In order to avoid concurrence problems when different processes accede to devices at the same time, it is necessary to use two buffers. One of them will store the reading sensors values and the other are the values for the actuators. In this way all processes will have access to sensor values, and will be able to act on actuators through buffers, without having to accede directly to the devices.

There will be an only process that is in charge to accede to the devices, through the interface, storing the reading sensors values in its buffer, and reading the buffer values of actuators to act on them. (See figure 2).

The server application communicates with *Player* through an own protocol on TCP/IP and it can interact not only with *Player* but with any another control program that respects the communication protocol established.

2.1.2 Generic Drivers Implemented in Player

As it was indicated previously in this architecture, *Player* acts as client requesting information and acting on the devices through the server application.

Drivers are the *Player* part in charge of acceding to devices. To avoid having to create a new driver for each device that we want to control, we have created a generic driver for each type of device. These generic drivers will be the clients, communicating with the server application through an own protocol (see figure 2).

In contrast with what happens with drivers of devices that *Player* possesses, these we have created serve for any device of that type. For instance: sonar's driver serves for any number and type of sonars.

This working method allows integration in *Player* robots that use the model of server application previously described and respect the communication protocol with generic drivers, without needing to modify *Player*.

2.1.3 Protocol between generic drivers and the server application

The communication between generic drivers and the server application is made following a simple protocol through TCP socket, that is described next.

In the first place generic driver sends a byte, indicating if it wants to finalize the server, the connection or to what device it wants to accede.

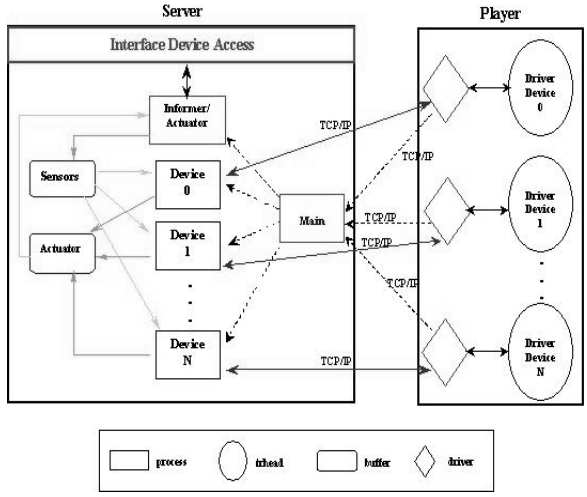

Figure 2: Architecture of the software platform

If the device is sensorial (see figure 3), the server responds with a byte indicating the number of units of that device that the robot has and next it sends the set of values of all of them, each one of those values will go in one or two bytes, depending on the data type that are (character, integer number...).

In case of being an actuator device, the generic driver sends, after the first byte, another one indicating on which units of that device wants to act and next the values of these. If an error happens while the values are sent, these are sent again.

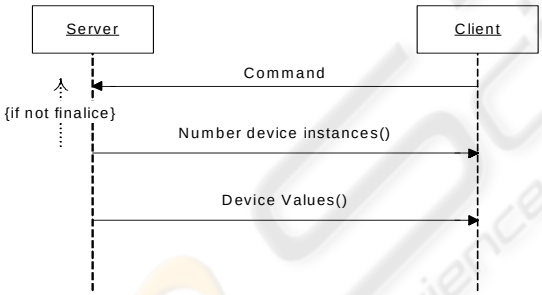

Figure 3: Protocol between server application and generic driver of a sensor device

3 CONCLUSIONS

Our intention has been to create a software platform that allows to integrate new robots in a simple way, that facilitates the creation of control programs of the robotic devices and that allow to control robots of distributed way through a wireless network.

It is thought to work on a set of robots, that can have different architectures and devices.

The platform has been designed so that integration of robots does not imply modifying its code. The creation of control programs is made using a set of simple interfaces of devices.

It allows to control, through wireless connections, the devices of any robot that adapt and execute the server application that we propose. Robots does not need to have hard processing capacity, simply must support wireless connections.

It constitutes a very useful tool to facilitate the work with multi-robot systems in distributed environments.

Making use of it, robotic control programs of a very simple way can be created, since the access to the devices of robots is transparent for the programmer. In addition, the programs can be implemented in several high-level languages and to control one or several robots simultaneously, through a network.

Also the wireless communication between its three layers improve the operational range with respect to the serial connections, and allows to control multiple robots through an only access point to the network.

The platform has been tested on a set of mobile robots, some commercial and others of own creation, developing applications that allow the cooperation of robots and the control of them simultaneously through wireless connections.

REFERENCES

ActivMedia Robotics, http://www.activmedia.com/

B. Gerkey, R. Vaughan, A. Howard, ICAR 2003.The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems.

B. B. Werger, 2000. Ayllu: Distributed port-arbitrated behavior-based control.

Carnegie Mellon Robot Navigation Toolkit, http://www-2.cs.cmu.edu/~carmen/

Khepera robots, http://www.k-team.com/robots/khepera/

Player/Stage Project, http://playerstage.sourceforge.net

Saphira robot control system, http://robots.activmedia.com/archives/saphira-users/