PERGAMON

Robotics and Computer Integrated Manufacturing 16 (2000) 29–42

Robotics and Computer Integrated Manufacturing

www.elsevier.com/locate/rcim

# Object-oriented and distributed approach for programming robotic manufacturing cells

J. Norberto Pires[a,*], J.M.G. Sá da Costa[b]

[a]Mechanical Engineering Department, University of Coimbra, 3030 Coimbra, Portugal

[b]Mechanical Engineering Department, IST - Technical University of Lisbon, Av. Rovisco Pais, 1, 1096 Lisboa Codex, Portugal

Received 15 December 1998; received in revised form 8 April 1999; accepted 23 April 1999

**Abstract**

Flexible manufacturing systems (FMS) are essential for small/medium batch and job shop manufacturing. These types of production systems are used to manufacture a considerable variety of products with medium/small production volumes. Therefore, the manufacturing platforms supporting these types of production must be flexible and organized in flexible manufacturing cells (FMC). Programming FMCs remains a difficult task and is an actual area of research and development. This paper reports an object-oriented approach developed for FMC programming. The work presented was first thought for application in industrial robot manipulators, and later extended to other FMC equipments just by putting the underlying ideas in a general framework. Initially, the motivation for this work was to develop means to add force control to a standard industrial robot manipulator. This problem requires remote access to the robot controller, remote programming and monitoring, as also is required to program and monitor any other FMC equipment. The proposed approach is distributed based on a client/server model and runs on Win32 platforms, i.e., Microsoft Windows and Windows NT. Implementation for the special case of industrial robot manipulators is presented, along with some application examples used for educational, research and industrial purposes. 

*Keywords:* Robotics; Object-oriented programming; Flexible manufacturing systems; Flexible manufacturing cells and CIM

## 1. Introduction

Actual manufacturing systems are evolving rapidly to flexible systems. Hard Automation manufacturing systems, composed by highly productive and dedicated machines, are not suitable for today's manufacturing platforms. Today the enormous diversity of products along with the requirement for better quality at lower prices makes the product life cycle very short. This is incompatible with hard automation manufacturing systems.

One of the most recent developments in the area of industrial automation is the concept of flexible manufacturing systems (FMS). These are highly computerized systems composed of several types of equipments, usually connected through a local area network (local network using MAP[1] protocols [1]) under some hierarchical computer integrated manufacturing (CIM) structure [2–4]. The factory (*shop floor*) equipments are organized in flexible manufacturing cells (FMC) with transportation devices connecting the FMCs. In some cases, functionally related FMCs are organized in flexible manufacturing lines (FML) (Fig. 1). Each FML may include several FMCs with different or equal basic capabilities. The organization proposed in Fig. 1 is a hierarchical structure [3,5] where each FMC has its own controller (cell level). Therefore, if the manufacturing process is conveniently organized in FMLs we will have several controllers in the shop floor level, e.g., one controller for each FML (process level). With this setup, an intelligent and distributed job dispatching and awarding mechanism may be implemented taking advantage of the installed industrial network [3,6–8].

The essential factor of an FMC is its flexibility, i.e., its adaptability to new manufacturing requirements that can

* Corresponding author. Tel.: +351-39-7000758; fax: +351-39-7000701.

*E-mail addresses:* norberto@dem.uc.pt (J.N. Pires), msc@gcar.ist.utl.pt (J.M.G. Sá da Costa)

[1] Manufacturing automation protocol (MAP).

0736-5845/00/$ - see front matter 
PII: S0736-5845(99)00039-3

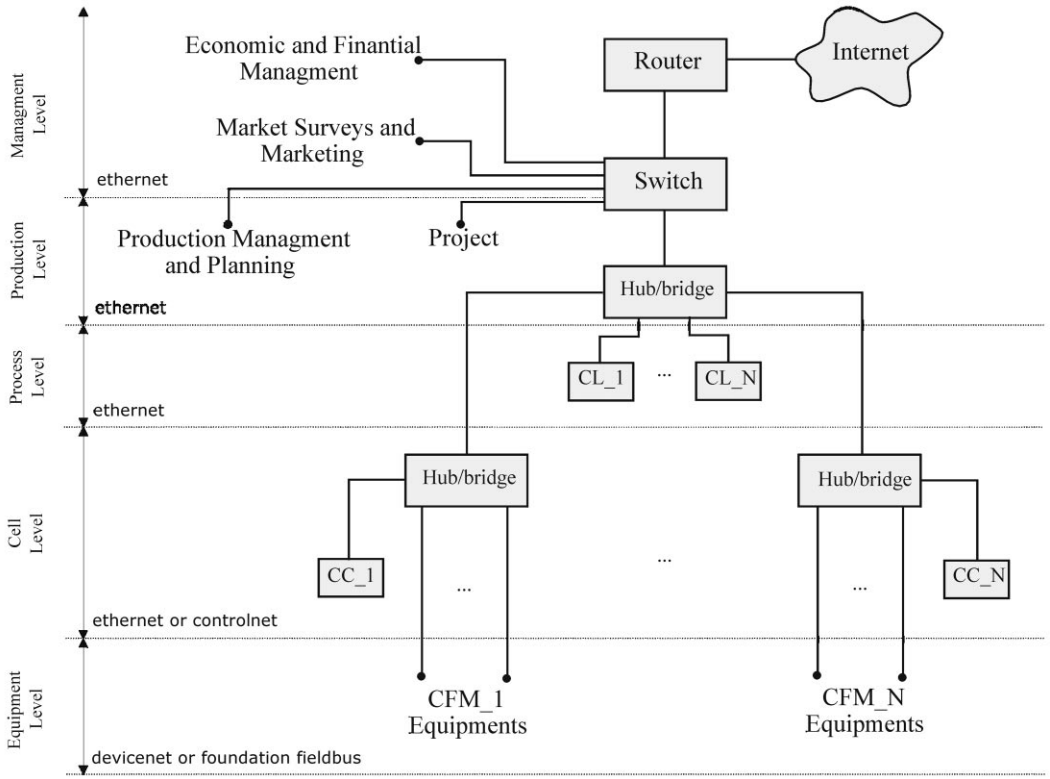



Fig. 1. Example of an industrial network organized in hierarchical levels (shop floor control structure is presented in detail). The industrial network presented is conveniently segmented (using bridge HUBs) maximizing communication throughput in each segment.

go from a modified product to a completely new product. The flexibility results from the fact that FMC equipments are programmable and easily reconfigured machines: this is the case of industrial robot manipulators, mobile robots for parts handling and transportation, programmable and logic controllers (PLC), CNC machines, vision systems, conveyors, etc.

Nevertheless, programming an FMC remains a difficult task mainly because usually FMC equipments come from different manufacturers, having their own programming language and environments. A specialist is then needed for cell programming, even if the required changes necessary to meet new manufacturing demands are only simple adaptations in the original program. This reduces significantly the FMC flexibility, i.e., its potential flexibility is only barely used. In this paper, we present an object-oriented approach for robot programming, monitoring and control. The proposed software architecture may be extended and applied to other FMC equipments, and is an alternative approach to the ones presented in the literature. These approaches usually are based on the development of a unique language for FMC programming, having specific interpreters to convert the code to each equipment native language [9]: this is complex and reduced to a limited number of manufacturers. Other object-oriented approaches showed that it is possible to have a flexible programming environment, and still program each equipment using its native language [10,11]. We consider here that for any individual equipment we can define a set of complex functions, which include all the tasks that would require the equipment when inserted into an FMC. We also consider that these functions can be requested remotely by the FMC controller, which provides also the proper parameterization.

The approach presented here is basically a collection of software controls and a communication mechanism between the host computer and the controllers of the FMC equipments. From among the several available technologies, either for software controls or communications, we selected the ones that could serve better our needs. The basic idea was to provide means for easy equipment programming and monitoring, hiding from the user all the less important details of how to address individual equipments, how to communicate, where to collect data from, etc. All of these are encapsulated into software controls (objects) that provide to the user the properties and methods necessary to do the job. For this we use actual standards, and did not tried to define a new distributed platform (as the European consortium OSACA [12], although the approach is similar), because we wanted to run the architecture from the usual PCs using standard tools.

Industrial robot manipulators are a special case of FMC equipments, because they can perform a variety of tasks, most of them in a human-like manner. As a consequence, these machines have increasing importance in today's manufacturing industry. Moreover, its importance will grow in the near future as forecasted by the United Nations and the International Federation of Robots [13]. Nevertheless, actual industrial robot manipulators remain inefficient machines when it comes to interact with the environment and deal with the contact problems. They remain position-controlled devices with no force control capabilities. Force-controlled robots are essential for industrial tasks requiring a precise control of the interacting forces generated by contact between the robot tool and the environment [14–19]. Examples of this type of tasks are polishing, deburring, grinding, etc. Adding force control to an industrial robot is one of the applications of the proposed scheme (see Section 4.1).

In the following sections of this paper, our approach for FMC programming, taking as example the case of robotic FMC equipment's, is explained in detail. Section 2 explains the basic architecture adopted which is distributed and object oriented. Section 3 details the application of the adopted architecture for the special case of the industrial robot manipulator we have in our laboratory. Special attention is also given to the integration of intelligent sensors into the FMC equipment and the connectivity of the developed programs with other commercial programs. Section 4 presents two application examples of the proposed approach. Finally conclusions are drawn in Section 5.

## 2. Basic architecture

The software architecture discussed here was developed for industrial robot manipulators, and was implemented for the special case of ABB Robots (with the new S4 control system). Nevertheless, it can be used with other equipments of an FMC, enabling fast FMC programming and monitoring. Because of this, the discussion will be kept general, aiming at application in other equipments.

To implement the ideas discussed here, FMC equipments are required to have processing capabilities (i.e., a microprocessor), some memory space and a TCP/IP interface over ethernet or serial line (SLIP). These requirements are usually available in modern FMC equipments, which means that their application is not seriously limited. The basic idea is to define for any specific equipment a library of functions that in each moment define the basic functionality of that equipment for remote use. Thus a mechanism for calling these functions from a remote computer, usually the FMC controller, should also be provided along with a message syntax protocol. This points to a distributed client/server environment, where any FMC equipment acts as a server offering a collection of services to the remote client (FMC controller). These services are general by complex tasks that cover all the functionality required for the equipment when working in a certain type of FMC. Adding other services may also be allowed and should be a simple task. Nevertheless, the library should be complete in a way that adding new services should not be necessary for usual operations. In fact, when integrated in a FMC any equipment has a well-defined number of tasks that it should perform. The only thing that is required is to identify all of them and implement the correspondent services as generally as possible; in a way that they could be used in different situations or even in different FMC's. For the special case of industrial robot manipulators, the following are examples of remote services (Table 1).

The implementation of this requires the use of three programming models:

1. Client/server – We require to have server code running on the equipment that deals with receiving calls from remote computers (clients), execute them and return the results.
2. Remote procedure calls (RPC) – This is the usual way to implement communications between the client and the server of a distributed application. The client makes what looks like a procedure call, although the resource is not local. The RPC mechanism in use translates that into network communications. The server receives the request, executes accordingly and returns the results.
3. Data sharing – We want to have services that share files, programs, databases, etc. The data-sharing services will be built on top of RPC, which provides the means for transferring data.

A messaging protocol (MMS [1,20] or other) and the definition of data structures for handling service parameters, communication and execution status must also be specified. Due to space limitations, this point will not be discussed in detail.

Another important aspect to consider is the possibility to integrate sensor data into the developed applications. The most easy and portable way to do this is to build objects for sensor access and configuration, as represented in Fig. 2. It is also very important to be able to exchange data between the developed monitoring and control applications and the commercial applications used for simulation, data analysis and representation, etc. This means that a connectivity mechanism must be available for these applications.

## 3. Application to robot manipulators

The above-explained architecture was applied to the robot/controller we have in our laboratory (ABB

Table 1
Examples of remote services

| | |
|---|---|
| Gripper services | |
| open_close_gripper | Opens/closes the gripper in use. In the case of a multi position gripper, the user should identify the final position. |
| open_close_griper_if | The same as open_close_gripper, having now the possibility to add a trigger event that can be a timer, an IO event, a variable event, etc. |
| IO services | |
| dig_in | Reads from a single or from a group of digital inputs. |
| dig_out | Writes to a single or to a group of digital outputs. |
| analog_in | Reads from a single or from a group of analog inputs. |
| analog_out | Writes to a single or to a group of signal analog outputs. |
| Motion services | |
| go_home | Sends robot to the defined home position. |
| go_position | Sends the robot to the specified position. |
| go_home_if | Waits for a condition to be met (timer, IO, variable, … ) before sending the robot home. |
| go_position_if | Waits for a condition to be met (timer, IO, variable, … ) before sending the robot to the specified position. |
| FMC specific services | |
| play_trj | Executes a specified trajectory for the specified number of times: if $-1$ is specified, the trajectory is played until a condition is met (timer, IO, variable, … ). |
| load_piece | Fetches a specified piece from a specified loading device and moves the robot to the specified position. |
| unload_piece | Unloads the piece on the specified position. |
| load_unload_piece | Fetches a specified piece from the specified loading device, goes to the specified unload position and unloads the piece. |
| production_sequence | A production_sequence is a sequence of operations that should be performed. The sequence is specified using a definition matrix and saved in the robot control system.<br>Then the sequence can be executed for a specified number of times. For example, the following specifies a very simple loading/unloading operation:<br>production_sequence(0, [100 0 320 1 200 2]);<br>production_sequence(1, [ − 1 500 10 1]);<br>The remote client sends two commands. The first command is a save sequence command (first parameter equal to zero), the second command is an execution command (first parameter equal to one). The sequences can have any number of operations. For practical use, we limited the dimension of the sequences to 1024 parameters.<br>In the first command the sequence commanded is:<br>100 – Wait for piece coming from any of the available loading devices (0). If the parameter following the 100 parameter was different than zero (0), then the system should look only to the specified loading device.<br>320 – (load_unload_piece command) loads a piece from the specified loading device and unloads it at the specified unloading device (1).<br>200 – goto_position defined by 2 (this is a safe general position to wait for new commands).<br>In the second command we command a execution:<br>− 1 – The robot plays the saved sequence until a condition is met.<br>500 – The condition is an IO Digital event. The system should wait for input 10 to become active (1). |

IRB1400/S4) [21]. Our objective was also to run the developed software on standard personal computers, which have today considerable computing power, namely the ones based on Intel Pentium microprocessors. For personal computers, the most popular operating system (OS) platforms are the Microsoft Win32-based-platforms, namely Windows NT and/or Windows 95/98. Moreover, Windows NT 4.0 is also an industrial standard, taking advantage of its compatibility with the most popular OS for desktop computers (Windows 95/98), its very good security and UNIX-like performance at a special low price. Therefore, we selected for use personal computers running Win32 OS, preferably Windows NT 4.0 or later.

ABB S4 controllers implement RPC server programs [22] with variable access services, file and program management services and system status services. To access these services the host computer (client) must implement RPC calling code through an ethernet or serial connection (both using TCP/IP protocols). ABB also developed an application specific protocol (ASP) called RAP [22] to be used with these services. The RAP messaging protocol works in the same way as MMS specified for MAP networks [1]. To implement the PC–Robot communication code based as mentioned on RPCs using RAP, we used the facility developed by the Sun Microsystems Open Computing (ONC) Group named SUN RPC 4.0. The choice of this facility was

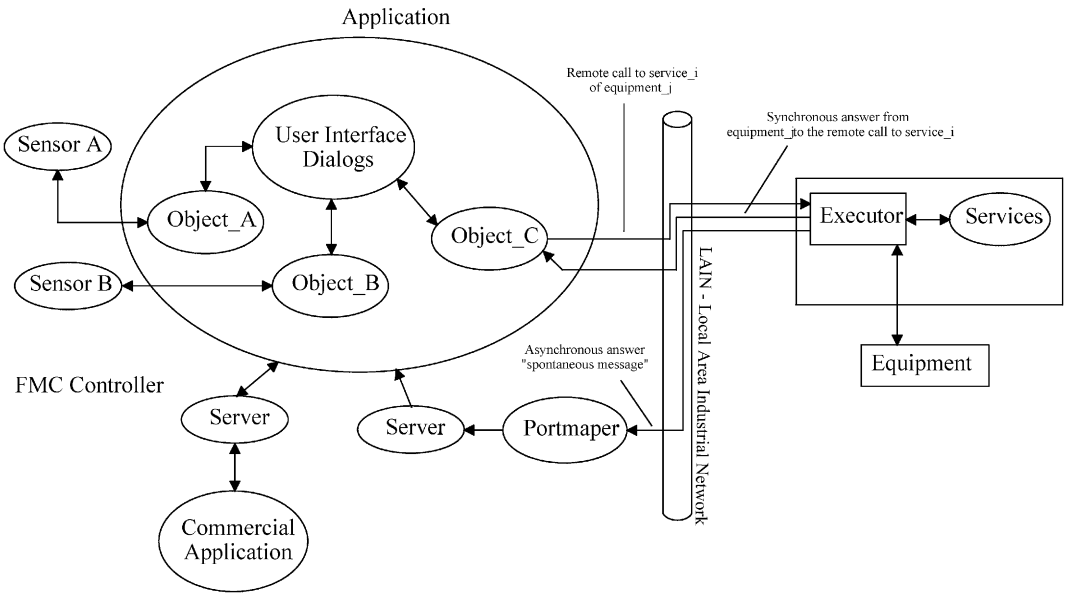



Fig. 2. Basic architecture for the distributed software.

mainly due to two reasons:

1. ABB S4 RPC server programs are SUN RPC 4.0 compatible.
2. SUN RPC 4.0 is freely available on the Internet. This facility uses the ONC-standard and includes a compiler (*rpcgen*), a *portmaper* and some other tools and utilities like the *rpcinfo*, etc. [23]. The Microsoft RPC implementation uses the OSF/DCE standard [24] which is not compatible with SUN RPC 4.0.

We need finally to select the technology that will be used to build the distributed objects. We considered the Common Object Request Broker Architecture (CORBA) technology [25] from the Object Management Group (OMG) and the Common Object Model/Object Linking and Embedding (COM/OLE/ActiveX) technology [26,27] from DEC/Microsoft. For Win32 platforms, the easier approach is the COM/OLE/ActiveX technology mainly because:

1. It is fully supported by Win32 platforms. In fact, COM is the most widely used component software model in the world, and a large majority of the new code developed for Windows and Windows NT operating systems depends on it.
2. There are very good developing tools available for those platforms, integrated into developing suits like the Microsoft Visual Studio (we use the Microsoft Visual C++ compiler as programming environment).
3. Availability of detailed documentation, easily accessible even through Internet.
4. We have some experience with this technology.

Nevertheless, OMG CORBA is a good alternative that deserves to be considered more carefully in the future if integration with Win32 platforms is provided (Microsoft is facilitating interoperability with CORBA just by licensing COM to CORBA vendors).

JAVA Beans [28] is another component technology to be considered when it is more mature. In a certain way Java Bean component technology resembles the COM/OLE/ActiveX component technology: both can be scaled to various tasks, implement properties and methods that can be used by other software components and implement events to communicate its actions to others. Nevertheless, they have different objectives. The COM/OLE/ActiveX technology uses a binary standard to implement interobject communication, focussing on language independence. Instead, Java Beans are based on the concept of Java Virtual Machines (VM) [28] and implement a Java Remote Method Invocation (RMI), which enables an object running on a Java VM to communicate with other object running on a different Java VM and invoke its methods. This focuses on platform independence, which is a very interesting and promising idea.

Finally, to exchange data between commercial applications and our own applications we selected to use the Microsoft Dynamic Data Exchange (DDE) technology. DDE performs well if the amount of data to be exchanged is not very large, is stable and is generally supported by the majority of the commercial applications available for Win32 platforms (that is the case of Matlab, Microsoft Excel, etc.)

In the next sections, the software designed to develop distributed applications will be presented and explained,

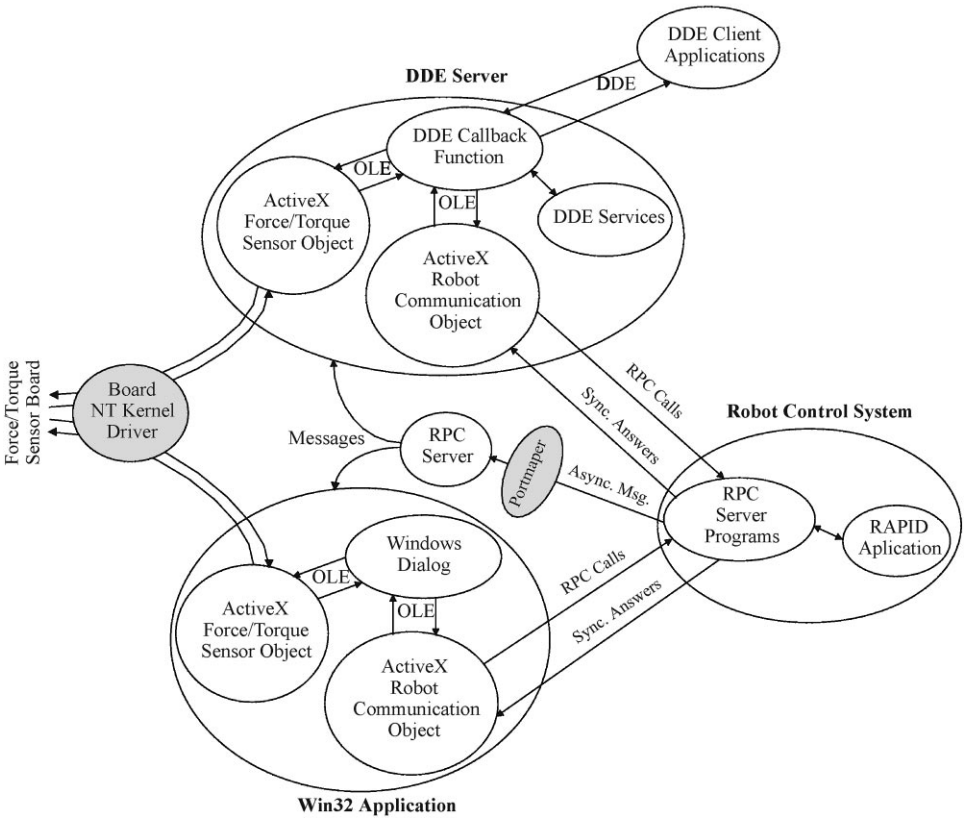



Fig. 3. Basic structure of the software for robot applications.

showing also a few examples applied to industrial, research and educational specific applications.

### 3.1. Development software

The software design to develop distributed applications is divided into three main parts (Fig. 3):

1. Robot communication software (ActiveX robot communication object).
2. Sensor access and configuration software (ActiveX F/T sensor object).
3. Connectivity to Win32 applications software (DDE server).

In the following, each of these parts will be briefly explained.

#### 3.1.1. *Robot communication software*: *ActiveX robot communication object*

This software is available as an COM/OLE/ActiveX object that implement methods and related data structures to access all the RPC services available from the ABB S4 controller. We used the SUN RPC 4.0 facility to implement the RPC calls, as explained before, recompiled using the Microsoft Visual C++ 5.0 compiler. Some of the original functions were changed to better suit our needs, without affecting the compatibility with SUN RPC compliant software.

Basically, the only services really necessary to be able to run the proposed architecture are the variable access services. Nevertheless, we did implement functions to access all the services available from the S4 controller. These services include also file and program management and control services, which are very handy for industrial applications requiring different programs and databases.

Using a service means making the respective RPC call, properly parameterized, and receiving the execution result. There are two types of services: synchronous and asynchronous. Synchronous services give the answer as the return value of the RPC call. Therefore, the prototype of a function implementing a synchronous RPC call is something like:

**short** status call_service_ *i*(**struct** parameters_*i*,

**struct** answer_*i*),

where *status* is zero if there is no error executing the call and less than zero otherwise (in this case the value returned identifies the error), *parameters_i* is a structure containing the parameters for service *i* and *answer_i* is a structure containing the execution results of service *i*. Asynchronous services, after activation, return results

asynchronously, i.e., the robot control system also makes RPC calls to the remote host to send information as soon it becomes available or when a certain event occurs (controller state change, program controller state change, IO events, etc.). These RPC calls, also named spontaneous messages, are made to all clients on the network that make a subscription to receive them (a subscription is an RPC call to the subscription service specifying the type of information the client wants to receive). The client must also implement a program to receive those messages. This program (Fig. 4) is an RPC server implementing the message receiving service, which logs on all messages and broadcasts registered Win32 messages that can be picked up by any running application by proper selection of its message queue.

The variable access services enable access to all type of data defined in the robot control system memory, requiring only each variable to be defined with a special tag (PERS instead of VAR). Using the variable access services and designing accordingly the application running on the robot controller, written using RAPID (a Pascal-like programming language from ABB Robotics), it is possible to add other services to the robot controller. In fact, with a simple switch-case-do loop switched by a variable whose value is controlled by the client (Fig. 5), we can add services to the system like the ones listed in Table 1. Furthermore, with this approach we still use the original

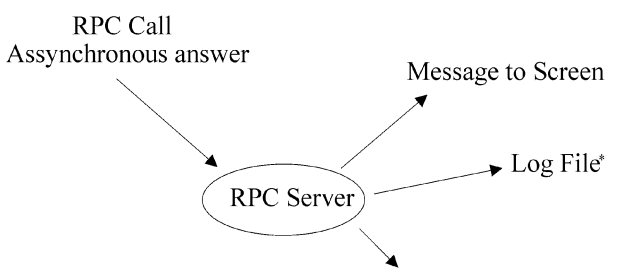



Fig. 4. RPC server diagram.

```
switch (decision_1)
{
   case 0: call service_0; break;
   case 1: call service_1; break;
   …
   case n: call service_n; break;
   default: call error_function; break;
}
```

Fig. 5. Simple switch-case-do loop used to implement new remote services.

capabilities of the robot control system, namely its advanced motion capabilities, related functions and data structures.

This procedure is not much different from that in any other RPC server program; the *svc_run* routine that is working on any of these programs is nothing more than a switch-case-do loop.

Using this structure, we built several generic and complex services for remote use. For example, a service to enable robot jogging from the remote host, using an inexpensive usual game joystick. With this service, the user can move all the axes of the robot 3-by-3 (because the joystick has only three axes). The user can select between axis 1-2-3, axes 4-5-6 and axes 7-8-9 (external axis). The robot is moved by steps. The value of each step is proportional to the position of the joystick handle multiplied by a scaling factor selectable by the user (Fig. 6).

Another example, is the *play_trajectory* service. A trajectory of up to 1000 points (position, orientation, velocity and acceleration) can be commanded to the robot. The user must first download the trajectory to the robot control system memory, using the *write_trajectory* service. Each trajectory is a structure with the following elements:

$$[n\ p_1\ v_1\ a_1\ z_1\ t_1\ p_2\ v_2\ a_2\ t_2 \cdots p_n],$$

where $n$ is the number of positions in the structure, and $p_i$ is position $i$, defined using *robtarget* structures [21].

A *robtarget* is a structure with four matrices:
$p = [x, y, z]$ is the position of the tip of the robot;
$r = [\phi_1, \phi_2, \phi_3, \phi_4]$ is the orientation expressed in quaternions;
$c = [cf_1, cf_4, cf_6, cf_x]$ is the configuration of axes 1, 4 and 6 used for inverse kinematics computations;
$e = [e_1, e_2, e_3, e_4, e_5, e_6]$ the position of the external axis (in mm or degrees, which depends on the type of external axis).
$a_i$ is the acceleration for the motion from position $i$ to position $i+1$, $v_i$ is the velocity for the motion from position $i$ to position $i+1$, $z_i$ the precision of the motion using ABB *zonedata* definition: $0 \to z_0, 1 \to z_1, \ldots, -1 \to z_{\text{user}}$. $t_i$ the definition of the tool in use using a ABB *tooldata* structure: $0 \to t_0, 1 \to t_{\text{user}}$.

We built a function in Matlab to generate these trajectories starting from the joint angle configurations. This function searches also for singular configurations and performs a graphic simulation of the all motion [29]. After passing the trajectory to the robot control system, it can be played a specified number of times, or until a condition is met, using the *play_trajectory* service. This operation can also be commanded directly from Matlab using the connectivity software that is explained below (Section 3.1.3). This function demonstrates the use of the proposed architecture to add and use new remote services to the robot control system.

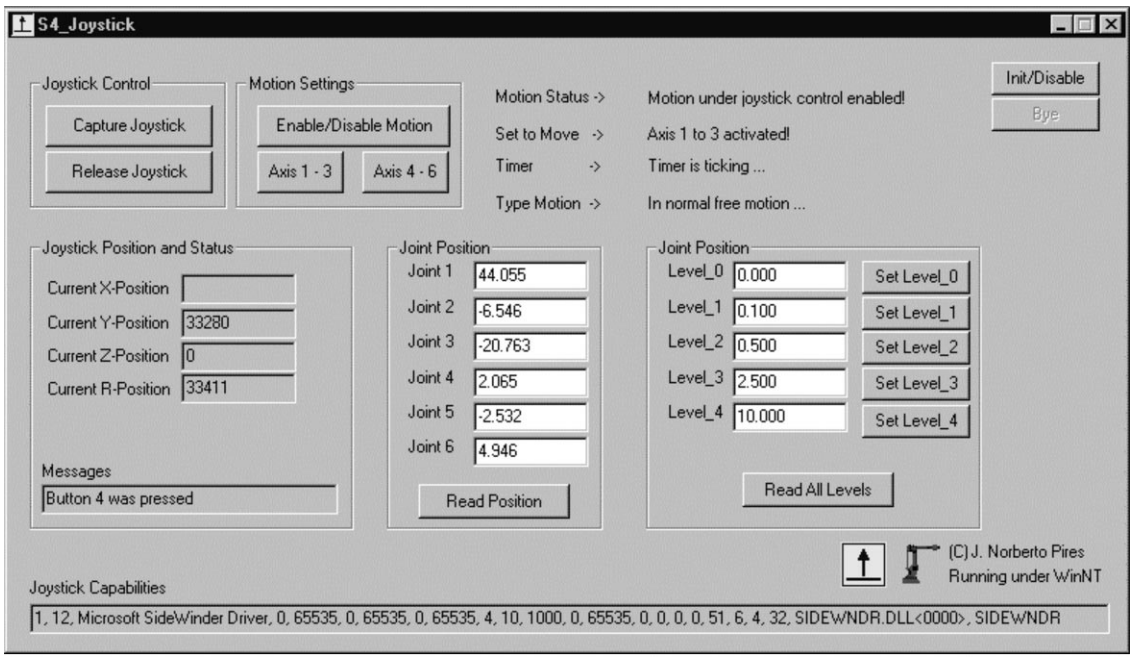



Fig. 6. Dialog developed to use the joystick remote service.

*3.1.1.1. Performance.* We made several performance tests using two robot TCP/IP connections: SLIP and ethernet. One of the tests was to write a "robtarget" structure in the robot system 1000 times, and compute the average write time. The values achieved were:

SLIP connection: ~150 ms,

Ethernet connection: ~15 ms.

Another test was to write a "robtarget" structure specifying a 0.05 mm shift of the robot in the ZZ direction (Tool reference frame) from actual position, wait until the end of the motion and then repeat 1000 times. The average cycle time was

SLIP connection: ~190ms,

Ethernet connection: ~20 ms.

*3.1.2. Sensor access and configuration software*

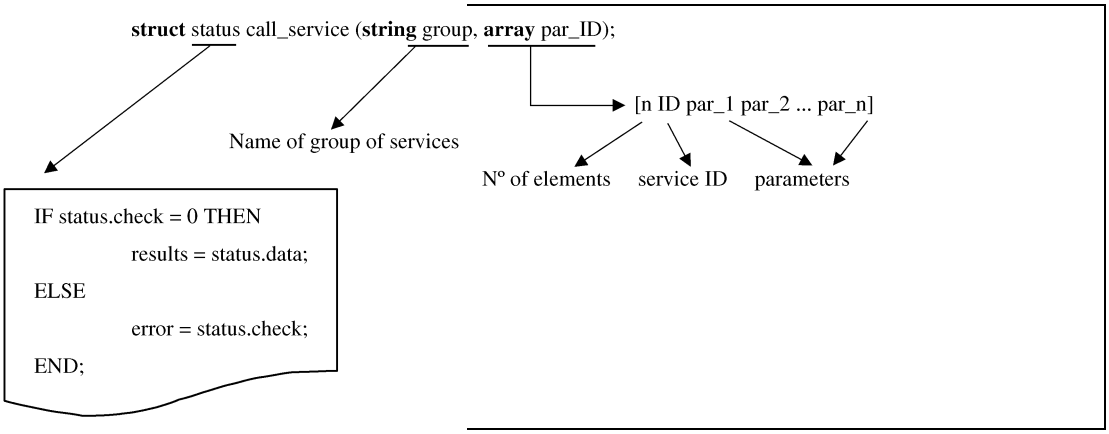
For some specific applications, the user must add intelligent sensors to its setup. By intelligent sensors, we understand sensors with some data receiving and processing hardware connected to the sensing unit through a serial/parallel high-speed interface. This is the case of force/torque sensors, vision systems, laser distance measurement systems, laser guidance systems, etc. To integrate these sensors into any application requiring its data readings, software must be made to operate with it. Again, the most easy and portable way to do this is by using objects that implement methods and related data structures to access and configure the sensor.

As an example, we built one of these objects to manage a six-axes force/torque sensor from JR3 Inc. [30] (Fig. 7). In this case, we also used the Microsoft COM/OLE/ActiveX technology to implement the object. The methods and data structures implemented enable the user to read forces and torques at a specified rate, setup the sensor (offsets, full scales, maximum and minimum allowable values, etc.) watch for warnings and errors, etc. This object can be easily integrated into Visual C++ or Visual Basic projects. Under Windows NT, a driver is needed to access any IO port. The driver acts at the NT Hardware Abstraction Layer (HAL) level to manage the hardware access [31]. We built such a driver for Windows NT 4.0, adding also a function (*init_jr3*) to the sensor object that detects the running OS and calls the driver if NT was detected (under Windows 95 the driver is not used). This means that the first thing that any application must do, before accessing sensor, would be to call *init_jr3*.

*3.1.3. Connectivity to Win32 application software: DDE server*

In several situations regarding industrial or research applications, we need to use some other programs like mathematical and/or simulation packages, spreadsheets, databases, etc. It would be very important and simplifying to have a mechanism enabling direct connection to the robot and/or to the sensors. In a general case of a FMC, the proposed mechanism should provide means to extend its functionality to the other equipments of the FMC. This points to some type of server program, built

using the above-mentioned objects, offering to its clients all the services necessary to access the above mentioned resources. The only thing that remains to be selected is the technology to be used by the server program to expose its services. As already mentioned we chose DDE, because currently almost all Win32 applications are also DDE clients. The DDE call from a client should have the following syntax:




The DDE server main dialog is represented in Fig. 8. The dialog also gives information about the state of the server, the state of the robot controller and of the robot program controller.

To access the DDE server from a client, a connection must be established with the server before trying to make any service calls. Fig. 9 shows an example of reading the force/torque sensor data from Microsoft Excel. In the following, we will demonstrate how to access the server from Matlab (see [29,32] for more details and a list of the functions currently available to access the DDE server from Matlab).

```
≫channel = dde_init('s4srv', 'matlab5');                    //Opens DDE connection.
≫abb_connect(channel, 'robot_1');                           //Connect to robot.
≫abb_mon(channel, 'robot_1');                               //Robot in ready state.
≫abb_progprep(channel, 'robot_1');                          //Robot program prepared.
≫abb_progrun(channel, 'robot_1');                           //Robot program started.
≫points                                                     //Positions to achieve.
0          0          0          0          0          0
1.0472     0          0          0          0.1745     0
1.0472     0.3491     0.5236     0          0.1745     0
1.0472     0.3491     0.5236     0.5236     0.1745     0
1.0472     0.3491     0.5236    -0.5236     0.1745     0
1.0472     0.3491     0.5236     0          0.1745     0
1.0472     0.3491     0.5236     0.5236    -1.5708     0
1.0472     0.3491     0.5236    -0.5236     1.5708     0
0          0          0          0          0.1745     0
0          0          0          0          0          0
≫abb14trj(dh,points, [30 30 30 30 30 30 30 30], 'b','d:\traj2.dat','m');
//Generates the trajectories using a 5th order polynomial [33] and saves the results in a file using the ABB robtarget format [29].
≫abb_writetrj(channel, 'robot_1');                          //Writes trajectories to robot.
≫abb_playtrj(channel, 'robot_1', 2);                        //Play trajectories 2 times.
≫jr3_resetoff(channel, 1);                                  //Reset F/T sensor offsets.
≫jr3_ftread(channel, 2)                                     //Read F/T data using Filter 2.
0          0          0          0          0          0          0          0
≫jr3_readfs(channel, 1)                                     //Reads Max. (1) Full Scales.
48         50         131        34         34         46
≫abb_proghalt(channel, 1);                                  //Stops robot program.
≫abb_moff(channel, 'robot_1');                              //Robot in standby.
≫abb_disconnect(channel, 'robot_1');                        //Disconnects from robot.
≫dde_term(channel);                                         //Terminates DDE connection.
```

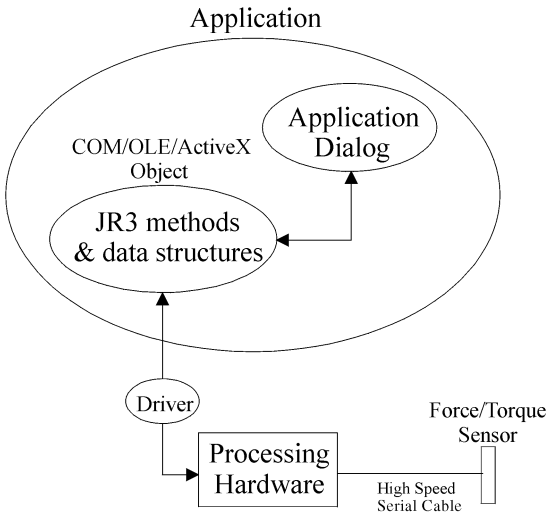



Fig. 7. Force/torque sensor integration.

If used in conjunction with a robotics toolbox [29,34,35], the developed DDE server extends significantly their application because it provides real access to the robot (which can be anywhere in the network) and sensors. Moreover, taking into consideration that this server is open, the user can add new services to the server upgrading in this way its functionality.

## 4. Test case examples

In this section two examples will be presented and briefly discussed, showing the application of the ideas presented above.

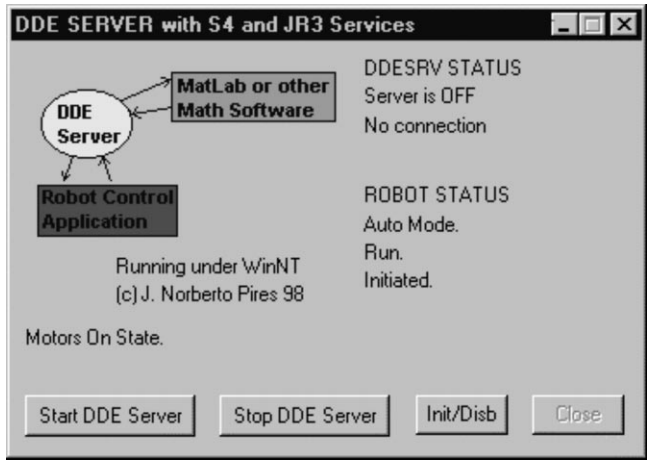



Fig. 8. DDE server dialog.

### 4.1. Force control on industrial robots

With the above-explained setup, we can add force control to standard industrial robots. Nevertheless, we are limited to indirect force control approaches [36], i.e., the force control law generates position commands to the inner position controller (robot control system). The basic scheme is represented in Fig. 10.

The force control function may implement several indirect force control approaches: classical (PID controllers), fuzzy, etc. Nevertheless, the following guidelines should be considered:

(a) *Simplicity*. The force control law must be simple and easy/faster to compute to enable real time.

(b) *PI-type control*. If a null steady-state error is to be achieved a PI-type force control law should be selected.

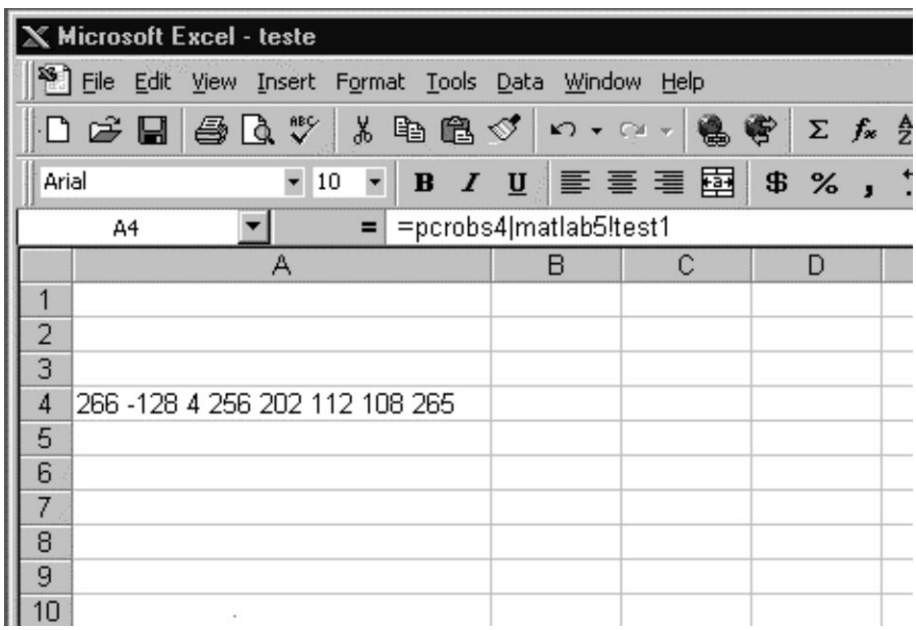



Fig. 9. Using force/torque sensor services from Excel.

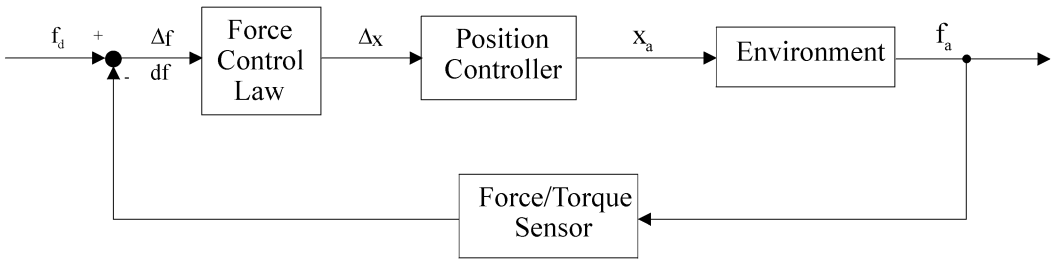



Fig. 10. Indirect force control approach.

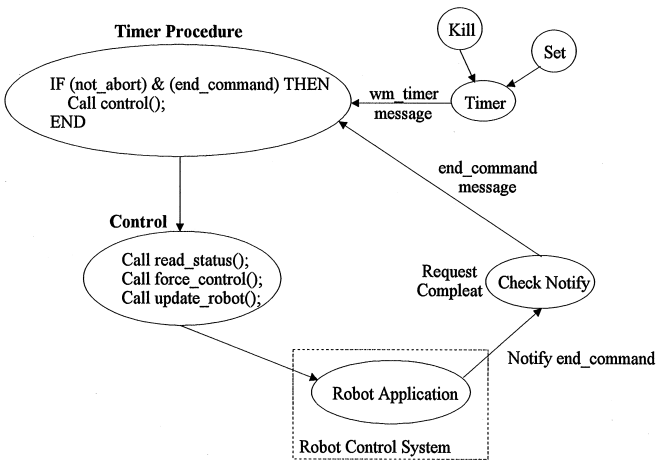



Fig. 11. Basic architecture of the force control application.

The derivative term is not desirable due to the noise associated with force readings.

(c) *Robustness to environment stiffness*. The system should work with several environment stiffness constants (the environment is modeled as a linear spring), which generally happens under industrial environments.

(d) Implementation requirements should not include significant changes to the original control system.

We built a force control testbed program for Win32 platforms, using the development software described before, namely COM/OLE/ActiveX objects used to access the robot and the force/torque sensor. Its basic architecture is represented in Fig. 11. In Fig. 12 we show the force control program, along with the DDE server, Matlab shell and a graphic representation, obtained using Matlab graphic capabilities, of the results of a force control experiment.

The application is distributed based on a client/server model as discussed earlier. The main part of the application is located in the PC, with a RAPID program running on the robot control system (along with the RPC servers used to communicate with the robot, of course). The RAPID program receives new position/orientation commands, updates the robot accordingly and fires the end_of_command message. The discussion of force control approaches and their implementation details is beyond the scope of this paper. Nevertheless, the interested reader can find some application inside and preliminary results of a fuzzy-PI controller in [36].

Direct force control approaches require direct and real time access to position readings and motor torque signals. This requires open robot control systems [37,38] which are not currently available at industry.

### *4.2. Monitoring and controlling an industrial robot in a FMC*

The application briefly described in this section (Fig. 13) is an industrial application actually working on a Portuguese company that produces glasses, especially automobile glasses. The setup presented is located at a production line that produces automobile front glasses for several automobile manufacturers from Europe and USA. The robot receives commands from the FMC controller to pick a glass pair from position AUTO_FEED or from position MANUAL_FEED, and put it on the mechanical transporter that will take it into the oven. In the oven, the glasses will gain the curvature characteristic of its model. The mechanical carriage works as a mold. Therefore, there is a carriage for each model of glass. The CCD cameras are used to position correctly the glass on the carriage. They see two special marks on each carriage and compute the corrections on the robot position/orientation necessary to correctly unload the glasses onto the carriage.

We developed the robot program to be commanded from the FMC controller (a PLC from Siemens, already present in the setup) using a parallel interface. The commands are calls to services (or complex general tasks) built in the same way as explained before, but these can be called also using a parallel IO interface (there is a variable driving the switch-case-do loop whose value can be changed using RPC calls or using the parallel IO interface).

Setting up the robot program is a difficult task. We need to adjust velocities and accelerations of all the

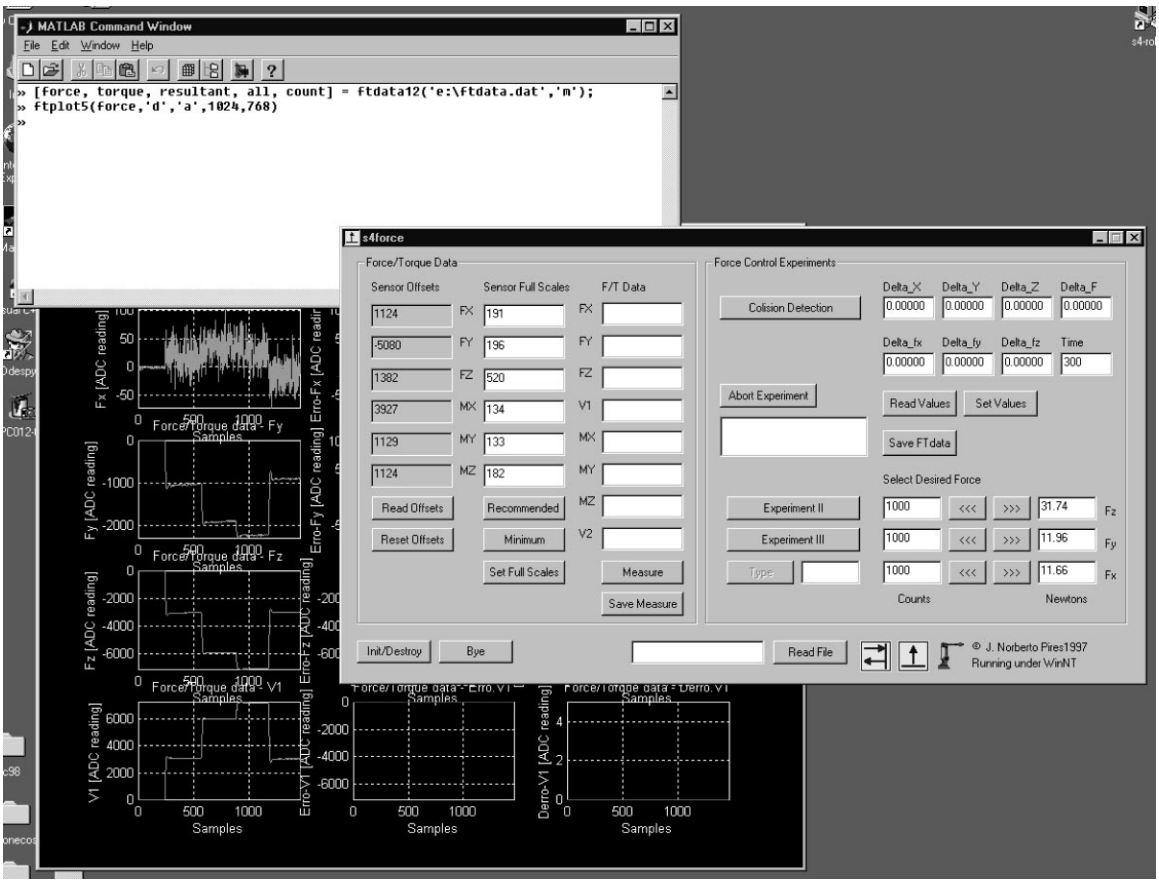



Fig. 12. Testbed for force control experiments (SLIP connection was used in this example; sampling at 190 ms; Matlab was used only to process and plot data after experiment).

programmed trajectories to not only reduce the cycle time but also to perform efficiently the required tasks. In addition, we need to teach some new positions to the robot when a new glass model is introduced. The teaching procedure includes:

1. Adjusting the position to pick the glass pair (that may be different depending on the glass thickness);
2. Adjusting the gripper positions to hold the new glass;
3. Adjusting the approach and download positions in the mechanical carriage;
4. Adjusting the unload motion to have a perfect fit between the glass and the carriage.

This means that each glass model has its own setup that is saved in a database sorted by the glass code type.

A PC was added to optimize remotely the robot program and to monitor all the operations, including productivity rates. All the system events and productivity values are logged on to the robot controller, which means that the PC program does not need to be always running. It can be started whenever we want to retrieve information on system events and/or productivity values, or perform any adjustment in the robot program. These adjustments are corrections made to any of the variables of the program: we can change safe fly trajectories, positions, velocities, and accelerations of all the axes including external axes, etc. These variables can be adjusted without stopping the robot program, until the robot performs satisfactorily. This procedure reduces significantly the setup time of the program. The teaching operation referred above can be performed on the robot console using the robot program, or using the PC application, which is far easier. We also added a robot explorer application to handle files and programs between the robot controller and the PC, which is very useful for everyday backup operations.

## 5. Conclusion

In this paper, several aspects related to FMC programming were discussed. An object-oriented approach for FMC programming and monitoring was presented. This approach is based on an earlier effort carried out to add force control to a standard industrial robot

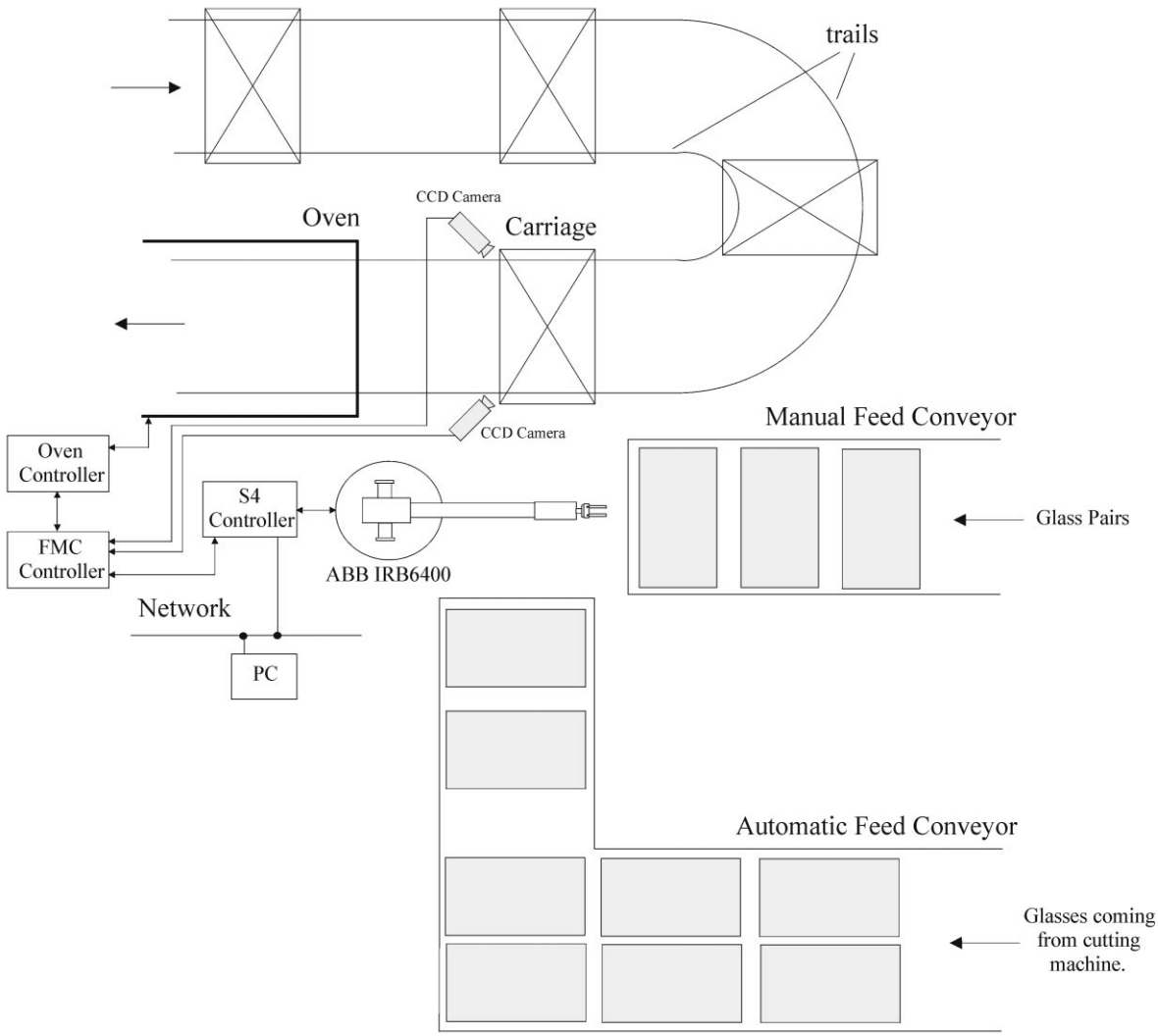



Fig. 13. Industrial FMC in the automobile glass industry.

manipulator, which is here extended to include other FMC equipments. Implementation of the proposed software architecture was presented for the special case of the robot we have at our laboratory (ABB IRB 1400). The developed solution includes robot communication and access software, sensor integration software and connectivity to Win32 applications software. Finally, application examples of the proposed software are briefly presented.

The definition of general/complex set of functions for each FMC equipment and the definition of a calling messaging protocol to access these functions remotely, facilitate the programming task of FMC equipments by regular operators. A specialist is only needed to define those functions that are available as remote services, and only the operator should know how to call and parameterize them using simple commands (Table 1).

## Acknowledgements

This work was supported by JNICT/PRAXIS XXI, under grant No. D2621/94.

## References

[1] Halsall F. Data communications, computer networks and open systems, 3rd edn. Reading, MA: Addison-Wesley, 1992.
[2] Kusiak A. Modelling and design of flexible manufacturing systems. Amsterdam: Elsevier Science Publishers, 1986.
[3] Ou-Yang C, Lin JS. The development of a hybrid hierarchical/heterarchical shop floor control system applying bidding method in job dispatching. Robotics Comput Integrated Manuf 1998;14(3):199–217.
[4] Waldner JB. CIM, principles of computer integrated manufacturing. New York: Wiley, 1992.
[5] Liang GR. A hybrid model of hierarchical control architecture in automated manufacturing systems. In Advances in factories of the

future, CIM and robotics. Amsterdam: Elsevier Science Publishers, 1993. p. 277–86.
[6] Baker AD. Complete manufacturing control using a contract net: a simulation study. Proceedings of the IEEE International Conference on Computer Integrated Manufacturing, 1988. p. 100–9.
[7] Shaw M. A distributed scheduling method for computer integrated manufacturing: the use of local area networks in cellular systems. Int J Pro Res 1987;25(9):1285–303.
[8] Zhang Y, Kameda H, Shimizu K. Adaptive bidding load balance algorithms in heterogeneous distributed systems. Proceedings of the IEEE Second International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 1994. p. 250–4.
[9] Fahim A, Choi K. The UNISET approach for the Programming of Flexible Manufacturing Cells. Robot Comput Integrated Manuf 1998;14(1):69–78.
[10] Li L. Object-oriented modeling and implementation of control software for a robotic flexible manufacturing cell. Robot Comput Integrated Manuf 1994;11(1):1–12.
[11] Magy E, Haidegger G. Object-oriented approach for analyzing, designing and controlling manufacturing cells. Proceedings of AUTOFACT'93, Chicago, (MS244), 1993. p. 1–10.
[12] Sperling W, Lutz P. Enabling open control systems – an introduction to the OSACA platform. Proceedings of the ISRAM'97, vol. 6. New York: ASME Press, 1996.
[13] United Nations and International Federation of Robots. World industrial robots 1996: statistics and forecasts. New York: ONU/IFR, 1996.
[14] Siciliano B. Parallel force/position control of robot manipulators. Proceedings of the 7th International Symposium of Robotics Research. London, UK: Springer, 1996. p. 79–89.
[15] Whitney D. Historical perspective and state of the art in robot force control. Int J Robotics Res 1987;6(1):3–14.
[16] Pires JN, Sá da Costa JMG. Real time system for position and force control of mechanical manipulators. 7th International Machine Design Conference. Ankara, Turkey, 1996. p. 707–717.
[17] Khatib O. A unified approach for motion and force control of robotic manipulators: the operational space formulation. IEEE J Robotics Automat 1987;3:43–53.
[18] Volpe R, Khosla P. A theoretical and experimental investigation of explicit force control strategies for manipulators. IEEE Trans Automat Control 1993;38(11):1634–50.
[19] Chiaverini S, Sciavicco L. The parallel approach to force/position control of robotic manipulators. IEEE Trans Robotics Automat 1993;9:361–73.
[20] Santos JP, Ferreira JJ, Mendonça JM. Distributed platforms for shop floor communications, a comparative analysis. Proceedings of the IFAC/IFIP/CTI Conference on Management and Control of Production and Logistics, Campinas, Brazil, September 1997. p. 628–34.
[21] ABB IRB1400 Users Manual, ABB Flexible Automation, 1997.
[22] RAP, Service Protocol Definition, ABB Flexible Automation, 1996.
[23] Bloomer J. Power programming with RPC. O'Reilly & Associates, Inc., 1992.
[24] OSF DCE, Introduction to OSF DCE. Open Software Foundation and Prentice Hall, 1995.
[25] Mowbray TJ, Zahavi R. The essential CORBA, system integration using distributed objects. New York: Wiley, 1995.
[26] Rogerson D. Inside COM. Microsoft Press, 1997.
[27] Box D. Essential COM. Reading, MA: Addison-Wesley, 1998.
[28] Camplone M, Walrath K. The JAVA™ Tutorial, 2nd edn. Reading, MA: Addison-Wesley, 1998.
[29] Pires JN, Sá da Costa JMG. Ferramentas de Software para Controlar, Programar e Monitorizar um Robô Manipulador Industrial usando Computadores Pessoais. Ibero-Am Mech Eng Assoc Mag 1998;2(1):43–67.
[30] JR3 Force/Torque Sensor Users Manual, JR3 Inc, 1997.
[31] Hamilton D, Williams M. Programming windows NT 4. SAMS Publishing, 1996.
[32] Pires JN. Realização de Controlo de Força em Robôs Manipuladores Industriais. Ph.D. Thesis, Mechanical Engineering Department, University of Coimbra, 1999.
[33] Craig JJ. Introduction to robotics, mechanics and control, 2nd edn.. Reading MA: Addison-Wesley, 1989.
[34] Gourdeau R. Object oriented programming for robotic manipulator simulation. IEEE Robotics Automat. Mag. 1997;4(3):21–9.
[35] Corke PI. A robotics toolbox for matlab. IEEE Robotics Automat Mag 1996;3(1):24–32.
[36] Pires JN, Sá da Costa JMG. Running an industrial robot from a typical personal computer. Proceedings of the IEEE International Conference on Electronics. Lisbon-Portugal: Circuits and Systems, 1998. p. 267–70.
[37] Pires JN, Sá da Costa JMG. Implementation of a real time system for motion and force control of mechanical manipulators. Proceedings of the IASTED International Conference on Robotics and Manufacturing. USA: Honolulu, 1996.
[38] Nilsson K. Industrial robot programming. Ph.D. Thesis, Lund Institute of Technology, Department of Automatic Control, 1996.