INTEGRATED AND DISTRIBUTED MANUFACTURING, A MULTI-AGENT PERSPECTIVE

José Barata[1], **L.M. Camarinha-Matos**[1], **Raymond Boissier**[2], **Paulo Leitão**[3], **Francisco Restivo**[4] **and Mohammed Raddadi**[2]

[1] New University of Lisbon, FST, Quinta da Torre, 2825-114 Caparica, Portugal.
email: {jab,cam}@uninova.pt

[2] Université Paris-Nord, GRPI. 93206 Saint Denis Cedex 1. France.
email: boissier@iut-stdenis.univ-paris13.fr

[3] Polytechnic Institute of Bragança, Apartado 134, 5301-857 Bragança, Portugal.
email: pleitao@ipb.pt

[4] Faculty of Engineering, Univ. Porto, Rua Dr. Roberto Frias, 4200-465 Porto, Portugal.
email: fjr@fe.up.pt

Abstract

There has been a large and rapid evolution in terns of the paradigms and technologies to support the development of distributed manufacturing systems and real-time applications. This paper starts with a survey of the main proposals in the area based on the experience gained by the authors in various fields pertaining to manufacturing device modelling, control and supervision, industrial messaging, production management and re-engineering methodology. The rational for a multi-agent approach is discussed as the available technologies to support the development of agent-based applications are summarised. Finally an implementation of controlling agents in a flexible manufacturing system is described and discussed.

1 Introduction

Nowadays, the rapid changes in the economic and technical environments associated to new manufacturing paradigms such as mass customisation require a capability for agile and fast adaptation to environment changes. Some of the social reasons for this changing scenario are the increasing level of human education in advanced economies, the new requirements in terms of quality and product diversity, the new environment regulations, and the incapability of centralised systems to efficiently master complexity. Furthermore, the wide availability of communication technologies, the potential economy reached when designing large control applications out of smaller building blocks or components, which are easier to develop and check, and the time saving when

replacing one identified component, represent the technical factors to motivate new approaches for the design and development of manufacturing systems.

Geographical distribution of responsibility and control leading to a network of collaborative manufacturing entities and the high level of autonomy of those entities is crucial to face the capability of agile and dynamic adaptation to changes. The multi-agents technology may play an important role in supporting the development of control and supervision applications for manufacturing systems that fulfil the requirements imposed by the new manufacturing environments.

This paper examines the integration and interaction between the logical supervisory part and the physical devices, using standard communication protocols, a crucial point for the successful development of a new generation of agile control and integration applications.

2 Techniques for the Distribution of Control Applications

2.1 Protocols: from packets to objects and agents

A basic condition for distributed computing to develop is the possibility of efficiently exchanging data. The packet switching technique, implemented with the now ubiquitous Ethernet, was a first step. However, for efficient and perennial programming to take place, higher levels of software protocols were necessary to reach the so called application layer from which operations usual in one's computer could as well be attainable on a remote one: managing files, executing a programs... The TCP/IP protocols at once brought network and distance independent programming opportunities. In the following sections we will give a schematic overview of the not always straightforward path towards higher and higher abstraction and standardisation levels in distributed programming.

2.1.1 The ISO/OSI stack versus Internet.

The 7 layers protocol stack of the ISO/OSI model for communication is a standard in computer education. It was to supersede the less structured UDP/TCP-IP 4 layer protocols. However the TCP/IP stack and utilities was included for free, as soon as the early 80's, in all Unix-like platforms. For that reason, ISO protocols where only accepted in large organisations, or for long distance applications. By the early 90's TCP stacks and utilities were available for DOS based PC's, letting them communicate with Unix systems. At that time we could already propose an *Open Machine-tool Controller* linked in a transparent manner to CAD/CAM sources (Raddadi & al. 1993).

2.1.2 Distributed application programming in the IP world

The Internet protocol (IP) routes packets of data up to 64 Kbytes large, and supports essentially two transport level protocols:

- TCP (Transport Control Protocol) is a reliable stream mode connection based protocol (partners negotiate to open a communication session on a particular *port number*), near to the OSI TP4 protocol, well applicable for applications such as file transfer (FTP) or remote connection (Telnet). No limit is set to size of transmitted data. Fragmentation and packet numbering take place for sizes larger than 64 Kb;
- UDP (User Datagram Protocol) is a connectionless protocol with no flow control or error recovery, applicable for local and small volume interactions such as data value reports, it is used in the trivial file transfer protocol (TFTP).

2.1.3 TCP/IP programming through the socket interface

Sockets are a de facto standard interface to the TCP and UDP protocols. Opening a socket returns a handle. Because of their very low level, programmers have to care about platform specific details (byte ordering, etc.) so that sockets should be limited to system programming. A possible improvement is to standardise the description and encoding of transmitted data. This is the case with the *Abstract Syntax Number One* (ASN.1) and the *Basic Encoding Scheme* (BER) used in the ISO protocols or in the TCP/IP Simple Network Management Protocol.

Server side		Client side	
socket( )	returns a socket reference		
bind( )	binds a local IP address and port nr		
listen( )	listens for connexion registrations		
accept( )	accepts a client connexion-		
// waiting		connect( )	establishes connexion
recv( )	receives data in stream mode	send( )	
send( )	send data in stream mode	recv( )	
closesocket( )	closes connexion	closesocket( )	

Figure 1 - An example of stream mode communication (TCP), using sockets

2.1.4 Client-server programming through Remote Procedure Calls

Remote Procedure Calls, originally developed by Sun, and available on Unix and Unix-like platforms, have been extended and standardised by the Open Group (formerly OSF-X/Open) within the *Distributed Computing Environment* frame. A slightly altered version of DCE/RPC has been integrated in the Microsoft Windows family platforms.

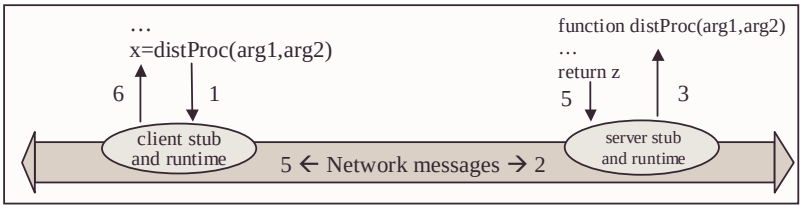

Figure 2 - Remote procedure call, a simplified representation

RPC is a high level mechanism to let a client program invoke a procedure on a distant server in a manner as transparent as possible to the programmer. To achieve such a goal, the interface (calling and return type) of the called procedures must be described in an *Interface Definition Language* (not CORBA's IDL) and then compiled so as to generate the so-called local *stubs*. At runtime the calling program actually invokes a RPC package generated image of the distant function in the local stub, the stub packs this call into network messages to the distant stub, which in turns invokes the real distant procedure, the distant procedure executes and returns its possible value to the calling program along the same path. Port mapping, conversion of arguments into messages, etc. are managed by the RPC runtime infrastructure. In order to avoid homonyms, server procedures have to register with a 128 bits *Globally Unique Identifier* (GUID or UUID).

2.1.5 Distributed Object in CORBA and DCOM

The idea common to *Object Request Brokers* (ORB), also designated as software buses or middleware, is to allow a local object invoke methods on a distant one as if it were local. In order to mask remoteness and networking details, the middleware runtime installs end points (stub, skeleton, proxy, whatever their names) on the client side and on the server side. The client object submits an invocation to the local image of the distant

server object. The signature of this call is forwarded to the distant host where it is converted into an actual method invocation on the server object; the results are returned to the client object in the same indirect way. The behaviour is very close to the RPC mechanism, however a RPC remote procedure is offered by a dedicated server, while ORB methods are attached to an object and a server can handle *many objects*. For this scheme to work several conditions are required. At least:

- a language independent Interface Definition Language (IDL) for describing interfaces and generating stubs for various target languages;
- an object registering mechanism and object locating schemes for unambiguous referencing and easy object access

Java RMI : a language specific prototyping tool

Sun's Java *Remote Method Invocation* is a lightweight solution for developing distributed object applications. The *Jini* distributed application environment brings services near to CORBA's with added dynamics. However, it is a language specific scheme, and as such better applicable for prototyping than for heterogeneous manufacturing applications. Besides Sun's JDK now includes a simple CORBA ORB, which brings us to the next paragraph.

CORBA : a global approach

The *Common Object Request Broker Architecture* is a distributed programming architecture specification corresponding to the post client-server era, set up by the *Object Management Group* (www.omg.org). CORBA allows object interaction independently of the source language. Besides the ORB specification, the CORBA architecture defines general low level object services such as Naming, Object life cycle, Event notification, Security, Persistence,… and may provide vertical CORBA application level facilities such as Manufacturing (OMG 2000). ORBs are available from many vendors, and those written in Java are platform independent. CORBA ORBs from various origins are inter-operable through the Internet Inter-ORB Protocol.

New constraints in the industrial world, such as video monitoring, real-time response or distant control introduce Quality of Service (QoS) concern. In this field, classical TCP/IP may not be adequate, this is why we have been recently experimenting with *Jonathan* (Dumant 1998), a flexible ORB complying with the ReTINA reference model for time constrained distributed processing. In Jonathan communications pass through a *binding object*, which may encapsulate various network configurations: ATM has been tested and fieldbus protocols could also be handled.

DCOM : from Clipboard to Distributed Component Object Model.

DCOM, Microsoft's *Distributed Component Object Model*, is the result of a progressive approach to offer users increasing package interoperability. It began with the simple clipboard cut and paste concept, continued with *Dynamic Data Exchange* and it experienced a great improvement with OLE-COM: at this stage the so-called *component* objects could register and later be activated from within other applications. A COM client object can request a particular interface to be passed back (or the default generic *IUnknown* interface). The Client communicates directly with a *Proxy* and the Server with a *Stub*. Communication between the Proxy and Stub is through messages: Window messages if the client and server are on the same platform, RPC when they are on separate ones. Microsoft promotes the extension of the DCOM scheme to Unix-like platforms, and CORBA to DCOM gateways do exist. Still DCOM is only a LAN protocol and Microsoft is now investigating inter-object co-operation across the Internet, using the XML based *Simple Object Access Protocol*, SOAP (W3C 2000).

2.2 Developing distributed manufacturing applications.

Whatever the approach, hierarchical or holonic, efficient functioning of distributed manufacturing applications relies on communication with or among the various manufacturing or monitoring equipment. Manufacturing applications need a reasonable level of standardisation to be accepted. Indeed standardisation guarantees relatively easy configuration and, more important, reconfiguration after a period of time. Significant example of the lack of standardisation is the number of specific drivers supervisory software vendors have to develop. On the other hand some universal standards are not accepted because they are expensive, or hard to set-up, or to far away from common programming practices. This was the case with ISO MMS, exceedingly dependent on an underlying architecture, and out of reach for small companies.

2.2.1 Lifting-up MMS with an object approach

The *Manufacturing Message Specification* (MMS) (ISO 1990) brought together many IT and Manufacturing specialists to define a common framework for developing communication support between industrial computerised equipment. The key concept of MMS is the *Virtual Manufacturing Device* associated with every real device (machine or cell). A VMD offers all services concerning itself and its related abstractions, mainly: *Domains* (which represent resources, possibly downloadable), *Named variables* (which can be of domain or VMD scope), *Program invocations* (corresponding to the execution of a machine task) and *Events* (with various associated mechanisms). Adaptations of this very general and abstract model to particular classes of devices (machine-tools, robots, logic controllers, process control) are proposed in *Companion Standards*.

Figure 3 shows the reference ISO-MMS architecture in (a), and possible adaptations : (b) is a frequently adopted solution using RFC 1006 to emulate ISO services over TCP/IP, (c) is a cumbersome unrealistic solution consisting in implementing all MMS services over TCP/IP sockets, (d)is a transitory solution using RPCs, presented in (Gressier 1995), and (e) is the present stable state of the OO-MMS approach of the CEDRIC & GRPI teams. In this approach, translation of MMS PDUs specifications from ASN.1 into CORBA IDL, allows the conversion of the conventional MMS service requests into Object-VMD method invocations.

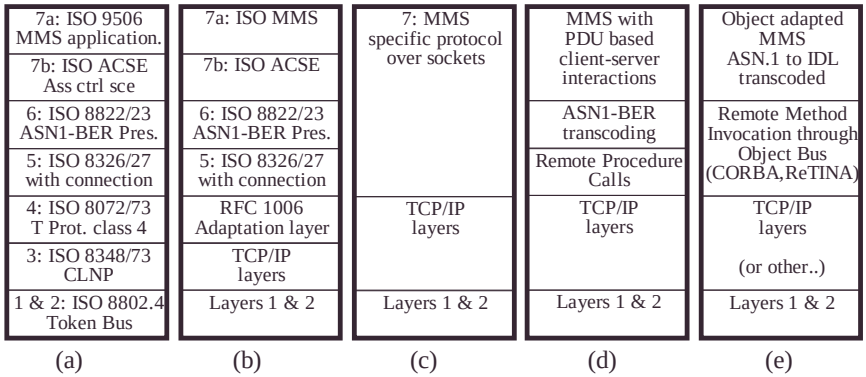

Figure 3 - Various approaches to MMS implementation

Various prototypes based on this OO-MMS concept have been developed. The first worked in the Sun/Chorus micro-kernel environment with the COOL ORB (Guyonnet & al. 1997), the following, presented at the WESIC 1998 conference used a heterogeneous system environment (Windows and Linux), the Orbacus ORB and an automatic

translation of ASN.1 PDU specification into IDL (Gressier & al. 1999). More recently the ReTINA model has been used within the *Jonathan* distributed environment (Boissier & al. 1998), a promising approach for applications subject to specific transport schemes.

2.2.2 OPC, ready to use but proprietary

OPC (OLE for Process Control) is at once a model and an implementation strategy promoted by the OPC Foundation (http://www.opcfoundation.org), a consortium of many vendors of measurement and control equipment, originally in a SCADA perspective (*Supervision, Control And Data Acquisition*). The model is based on standardised *Data Access Servers* and *Event Servers*. Data Servers contain *Groups* and Groups contain *Items*, the images of real data. Data are assigned with time stamps and quality indexes. Event servers behave as state machines, they contain OPC Conditions which control which under-conditions will send event notification to client applications. As an implementation strategy, OPC requires the servers to be OLE/DCOM compliant and as such written in C++. They can offer an Automation Interface which allows final developers or users to write Visual Basic clients and import data in standard Windows applications. A recent alternative project is Java OPC (JOPC), a Java package for implementing OPC in Java, and testing various interaction schemes : RMI, CORBA and XML (Mortensen 2000).

2.2.3 From MMS VMD to manufacturing agents

What is clear from the preceding section is the necessity of well-defined, public interfaces for technical objects. By re-lifting MMS and building it on an object-oriented basis, which can integrate real-time constraints, we believe we will get of a good framework for building agent-based architectures at workshop and company level. MMS, although structured, allows a great variety of particular VMDs. Existing object invocation scheme allow limited flexibility: dynamic invocation in CORBA, parameter naming in SOAP... It would be interesting to evaluate the integration of the MMS concepts within a Jini framework. Anticipating the next paragraph it may be suggested that in a world of many various and changing object instances only a higher level of collaboration may grant significant flexible and perennial supervisory systems. Software agents, which are expected to negotiate their interaction with their environment through *speech acts constructs*, still appear as the next step in abstraction.

3 Multi-agents in the manufacturing domain

The multi-agent system is a concept originated in the Distributed Artificial Intelligence area, and can be defined as a set of nodes, designated by agents (Ferber 1999). In spite of several existing definitions, it is possible to define an agent as a component of software and/or hardware, which is capable of acting and decision making in order to accomplish tasks. In the manufacturing systems domain, an agent is a software entity, that represents manufacturing system entities, such as physical devices and tasks.

3.1 Motivation to use multi-agents

The multi-agent systems represent a suitable technology to support the distributed manufacturing environment, since the manufacturing applications present characteristics like being modular, decentralised, changeable, ill-structured and complex, for what the agents are best suited (Parunak 1998). Analysing the benefits of multi-agent technology it is possible to conclude that it fulfils some of main requirements of the actual distributed manufacturing systems: autonomy (an agent can operate without the direct

intervention of external entities, and has some kind of control over their behaviour), co-operation (agents interact with other agents in order to achieve a common goal), reactivity and pro-activity (agents perceive their environment and respond adaptatively to changes that occur on it). Last, agents can be organised in a decentralised structure, and easily reorganised into different organisational structures (Leitão & Restivo 2001).

3.2 Platforms to support the development of multi-agents applications

Multi-agent systems can be adequately developed using usual O-O languages. Better suitability to support the development of such systems however is dependent on the provided features to deal with the basic requirements imposed by multi-agent systems, namely : concurrency, object-oriented approach, serialisation, remote access, etc. When building multi-agent systems there is a set of complex features that must be implemented, not supported by usual languages, which increases the programming effort. Communication channels at agent level, agent communication language, yellow and white pages services, ontologies for common understanding, code mobility, and agent management services are examples of features not directly implemented by programming languages.

To reduce the programming effort, it is therefore interesting to resort to multi-agent environments like AgentBuilder (AgentBuilder 1999), Concordia (Concordia 2001), JatLite (http://java.stanford.edu), IBMAglets (http://www.trl.ibm.co.jp/aglets), FIPA-OS (FIPA-OS 2001), and Jade (http://sharon.cselt.it/projects/jade/). Most of these platforms use the Java language, because of its various features to implement distributed systems. Such development environments provide some predefined features and agent models that ease the development of multi-agent systems, which can vary from platform to platform. These differences reflect the philosophy and the target problems envisioned by the platform developers. It is necessary to choose the most adequate environment for the type of problem in hands.

In the research addressed by this paper only three platforms were initially considered to be analysed: JatLite, FIPA-OS and Jade. The main reason for this choice was the need for a free platform, with good documentation and support, and previous experiences from other research groups with whom the authors have close relationship. The three platforms were analysed according to the following items: available support, ease of use, programming effort, documentation, use of standards (FIPA or others), facilities to implement rule oriented programming, and features to support the management of agent communities like white pages and/or yellow pages. The use of standards is important because it can increase the interoperability among different type of agents, and thus allow an increasing use of agents. Agent communication languages (ACL) are one of the most important aspects within the use of standards. The two current major ACLs are KQML (*Knowledge Query and Manipulation Language*) (Finn et al. 1993) and the FIPA-ACL. When dealing with an ACL integration problem, besides the syntactic translation, it will be important to verify that the semantic content is preserved during the exchange of messages. Agents need to have a common understanding of the concepts of their domain knowledge, which is known as *ontology*.

From the three platforms (JatLite, FIPA-OS, and JADE), JADE was the choice because it better responds to the mentioned requirements. In effect, JADE has very good support with a very active supporting mailing list. The use of *Behaviours* supplied by the platform, the good documentation, as well as the easy connection to Jess (*Java Expert Shell System*) (Friedman-Hill 1999) (a rule oriented programming infrastructure) helped

in reducing the programming effort. Moreover JADE, as FIPA-OS, implements the FIPA-ACL, which like KQML, is based on the speech-act theory (Searle 1969). FIPA-ACL describes every communicative act with both a narrative form and formal semantics based on modal logic, and it also includes a normative description of a set of high-level interaction protocols like requesting an action, contract-net, etc (Labrou & Al. 1999). Another interesting feature of JADE is the functionalities provided to manage the community of agents following the FIPA reference architecture for multi-agent platforms. It includes a *Remote Monitoring Agent* (RMA) tool, which is used to control the life cycle of the agent platform. RMA can be used to start and kill new agents, as well as for debugging purposes. An agent for white pages and life cycle services (Agent Management Service - AMS) is also included. AMS exerts supervisory control over access to and use of the multi-agent platform, and maintains a directory of agent identifiers and agent state. An agent for yellow pages services (Directory Finder - DF) is also included and can be federated with other DFs on other existing platforms.

FIPA-OS is very similar to JADE from the architecture point of view, because both are FIPA complaint and Open Source. With FIPA-OS it is even easier to model concurrent tasks because its task model provides better functionalities than JADE's behaviours. The main difference between the two platforms resides on the documentation supplied and on the easiness to start implementing, which favours JADE. A final word about JatLite that has poor documentation and features to support the management of agents. Moreover, the programming functions made available by the infrastructure are at lower level than JADE and FIPA-OS, which increases the programming effort.

3.3 Interaction with physical devices

The development of agent-based control and supervision applications in manufacturing presents some important delicate aspects: co-operation between agents, communication, decision-making and scheduling, self-organisation, learning and disturbance handling, interaction with legacy systems and interaction with physical devices. In the course of integrating control and supervision of distributed manufacturing systems, one of the biggest problems is the interaction with physical devices, due mainly to the following factors: proprietary communication protocols, different specifications and functionalities for each physical device, and complexity and cost of standard protocols. The agents that represents physical devices, such as sensors, robots, CNC machines, etc, should have an interface module that implements the communication between the logical part of an agent and the physical device.

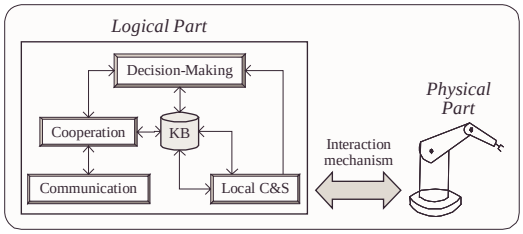

Figure 4 – Agent with logical and physical components

In order to facilitate the development of agent based control applications for distributed manufacturing systems it is important to have standard mechanisms and protocols to implement the interaction layer between the logical component and the physical device. As described in previous section, the trends for the distributed communication platform

are the use of CORBA or DCOM communication architectures with an MMS inspired manufacturing specialisation.

The solution to implement this standard interaction process is the development of re-usable libraries of objects that represents the functionality of the physical devices and implements the following basic services: *variables* (read and write), *program manipulation* (download, upload, start, stop, pause, resume), and *events* (notifications). Using these libraries of objects, the agents can easily access the physical device.

4 An agent-based control: the Novaflex flexible platform

4.1 The NovaFlex

The NovaFlex flexible assembly cell, installed in the Uninova Institute facilities, is composed of four subsystems: assembly cell Nr 1 and Nr 2, automatic warehouse, and transportation subsystem. Assembly cell 1 is built around a 6 axes ABB IRB2000 robot. The cell also comprises a tool exchange mechanism composed of a tool warehouse with four grippers and one changeover mechanism installed on the robot's wrist. Finally, a fixing device is installed on the conveyor, that is placed in front of the robot. Assembly cell 2 is built around the BOSCH SR800 SCARA robot. This cell also includes a tool exchange mechanism with 4 different tools and the fixing device is also mounted on the conveyor in front of the robot. The automatic warehouse has places to store 50 BOSCH T2 pallets loaded with either raw materials, finished or unfinished products. The warehouse loads and unloads the pallets from one of the transportation subsystem's conveyors. Raw materials and finished/unfinished products are transported to and from the different subsystems by BOSCH conveyors that are the basis of the transportation subsystem. A fake clock is the demonstration product being assembled in the NovaFlex.

Figure 5 – NovaFlex platform

4.2 An Agile NovaFlex

This complex cell, composed of several different heterogeneous controllers, is intended to be turned into an agile, easily changeable, and configurable assembly cell system, which could be used as a test-case to show how a system to support shop floor re-engineering should be designed. To reach this goal, a multi-agent approach is being used and the various manufacturing components were *agentified*. The next step is establishing the community of agents. This community supports two different types of organisational structures. In the first one, the various agents share some specific aspects related to the environment where they are installed (the NovaFlex). At this level they do not really have direct connections with the others, but with an organisation that acts as a common

shell for the whole community of agents belonging to the NovaFlex. This organisation defines: (1) the ontologies to be used, (2) the services available to the community, (3) the skills and competencies available within the community, (4) the used communication protocols, (5) mechanisms to publicise job opportunities, i.e. tasks to be served by he shop floor (manufacturing agents are always eager to find something to do), etc. This structure is called a manufacturing *cluster* in analogy to clusters of enterprises.

The second type of organisation is the *consortium*, which is a group of agents co-operating to pursue some common objective. While a cluster is a long-term organisation the consortium can be seen as short-term one because it only exists while the objective(s) that triggered the consortium's creation are still valid. The consortium can be dissolved as soon as the objectives are achieved. Consortia are dynamically created structures to serve the tasks asked by NovaFlex users.

In the shop-floor domain a cluster can be formed by a group of manufacturing components (robots, conveyors, grippers, etc), which have the potential to support the creation of certain kinds of consortia. A consortium defined in the same domain, represents manufacturing components co-operating to achieve an objective. A robot co-operating with a gripper chosen from a tools magazine illustrates a simple example of a consortium (Barata & Camarinha-Matos 2001). *Contracts* are the mechanism that regulates the behavioural relationships among consortium members both at cluster and consortium levels. The same entity regulated by different contracts can have different behaviours. When regulated by contracts (represented as declarative, configurable information structures) manufacturing components consortia lead to significantly more agile manufacturing systems because it is possible to exploit the same physical system in different ways, depending only on how consortia are organised.

4.3 Agentification

One important point when agentifying manufacturing components is the process of connecting the physical controller to the agent. This could be an easy task if every physical component was controlled directly by its own agent. However, outdated legacy controllers with close architectures control most of existing physical components. To integrate these legacy components in the agents' framework it is necessary to develop a software wrapper to hide the details of each component. The wrapper acts as an abstract machine to the agent supplying primitives that represent the functionality of the physical component and its local controller. The agents access the wrapper using a local software interface (proxy), where all the services of the wrapper are defined. Figure 6 shows a high level representation for an operative agent indicating how the wrapper integrates a manufacturing component (robot).

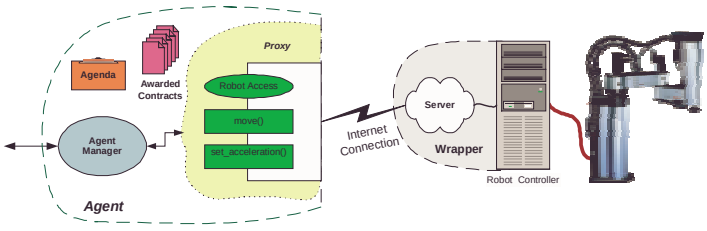

Figure 6 – Physical component integration

Wrappers for the NovaFlex's subsystems were developed using DCOM, not because of particular superiority but because (1) all the computers available to control the NovaFlex

are running Microsoft operating systems (Windows95, 98, and NT), (2) C++ Builder, the used development language, has good tools to develop DCOM applications, and (3) developers were better trained on the Microsoft environment. In effect any of the distributed object architectures could be used because the intention was to prove that distributed objects are adequate to integrate physical controllers with agents (agentification) and are even better than the old two-tier architecture, like RPCs.

Let us consider, the agentification of the IRB2000 robot. It can be controlled using either the teach-pendent or through commands sent to its serial RS232-C port. These commands can control a large number of tasks and must be issued following some low-level protocol. To control the robot, the remote host must track timeouts, error and state messages, through protocols ADLP-10 (*ABB Data Link Protocol*) and ARP 2.0 (*ABB Robot Application Protocol*). These were defined using appropiate classes. A software layer (robot control kernel) to implement robot controller's commands (move, acceleration, out, in, …) was developed on top of those two protocols, using threads. Class *TSerial* implements the methods required to control the serial port like sending and receiving bytes. On the other hand *TComunica* uses *TSerial* to send control datagrams to the robot using the ABB protocol. This class also manages Timeouts. Class *TABBLink* implements the functionalities of the robot itself. It uses *TComunica* to construct the message to be sent to the robot. This class also has a set of attributes to model some of the physical controller variables. The methods to control de robot defined here are:

1. **int** read_digital_input(**int** dig_inp_number, **bool** *dig_inp_value) – to read a digital port.
2. **int** write_digital_output(**int** number, **int** value) – to write in the digital port.
3. **int** open_grip() – to open the robot riper.
4. **int** close_grip() – to close the robot gripper.
5. **int** read_status(**int** *prog_number, **int** *inst_number, **int** *tcp, **int** *frame, **int** *inf, **double** *x, **double** *y, **double** *z, **double** *q1, **double** *q2, **double** *q3, **double** *q4) – get robot status
6. **int** move_xyz(**int** move_type, **int** orient_type, **int** coord_type, **int** veloc, **double** x, **double** y, **double** z, **double** q1, **double** q2, **double** q3, **double** q4) – move robot
7. **int** write_program_to_robot(**unsigned char** buf[], **int** prog_or_block_type, **int** prog_or_block_numb)
8. **int** read_program_from_robot(**int** prog_or_block_numb, **int** prog_or_block_type, **unsigned** char *buf[])
9. …

These methods are the most important aspect to stress because they resume the functionality of this component. To control the robot remotely it is only sufficient to include the definition of class *TABBLink* in the client application. To test this component, a client application with a graphical user interface was developed. This client allows the remote operation of the robot through the execution of all functionalities provided by the component.

5 Conclusions

This paper discussed the technological trends in distributed systems and a multi-agent based approach to implement *agile manufacturing*, which means a solution for fast and frequent reconfigurations of the production system in collaboration with possible temporary industrial partners (virtual enterprise). Such a distributed and highly dynamic situation is more and more frequent in manufacturing. Various partial developments have been realised in order to prove the feasibility of the proposed approach. Next steps requiring further research include the development of functionalities to support dynamic consortium formation/modification and cluster installation in a perspective of shopfloor re-engineering.

6 References

Agent Builder Web Site (1999), *AgentBuilder - An integrated Toolkit for Constructing Intelligence Software Agents*, (http://www.agentbuilder.com).

Bellifemine, F., Poggi, A., Rimassa, G. & Turci, P. (2000), Object Oriented Framework to Realize Agent Systems, *Proceedings of WOA2000, May*: 103-128.

Boissier, R., Epivent, M., Gressier-Soudan, E., Horn, F. Laurent, A. & Razafindramary, D. (1998) *Providing Real-Time Object Oriented Industriel Messaging Services.* Proc. ECOOP'98, Brussels, July 1998.

Camarinha-Matos, L.M. & Barata, J. (2000), *Shop-floor reengineering to support agility in virtual enterprise environments*, Proc. PRO-VE'00, Florianopolis (BR), Dec.

Camarinha-Matos, L.M. & Barata, J. (2001), Contract-based Approach for Shop-Floor Re-engineering, *Submitted to Low Cost Automation International Conference*.

Concordia Web Site (2001), (http://www.meitca.com/HSL/Projects/Concordia).

Dumant, B., Dang Tran, F., Horn, F. & Stéfani, J.-B. (1998) *An open distributed processing environment in Java.* Proc. Middleware'98, Lake District (UK), Sep.

Ferber J. (1999), *Multi-Agent Systems, An Introduction to Distributed Artificial Intelligence*, Addison-Wesley.

Finin, T., Weber, J., Wiederhold, G., Genesereth, M., Fritzson, R., McKay, D., McGuire, J., Pelavin, R., Shapiro, S. & Bech, C. (1993), *Specification of the KQML Agent-Communication Language*, The DARPA Knowledge Sharing Initiative.

FIPA-OS Web Site (2001), (http://fipa-os.sourceforge.net).

Friedman-Hill, E.J. (1999), *Jess, The Java Expert System Shell*, Sandia National Laboratories, Livermore, CA. (http: //herzberg1.ca.sandia.gov/jess/).

Gressier-Soudan, E., Epivent, M., Laurent, A., Razafindramary, D., Raddadi, M. & Boissier, R. (1999), Component oriented control architecture, the COCA project, *Microprocessors and Microsystems Journal*, **23** 2: 95-102.

Gressier-Soudan, E., Lefebvre, M. & Natkin, S. (1995*) TCP/IP in manufacturing: an experiment with MMS over RPC.* Proc. ULPAA'95. Sidney, Dec. 1995.

Guyonnet, G., Gressier, E. & Weis, F. (1997*) COOL-MMS: a CORBA approach to ISO-MMS.* Proc. ECOOP'97, Jysäskyla (SF), June 1997.

ISO (1990) ISO 9506, *Industrial Automation Systems – Manufacturing Message Specification – Part 1 to 4*, International Standard Organisation, Genova.

Labrou, Y., Finin, T. & Peng, Y. (1999), Agent Communication Languages: The Current Landscape. IEEE Intelligent Systems. Vol 14 – 2. pp 45-52.

Leitão, P. & Restivo, F. (2001), An Agile and Cooperative Architecture for Distributed Manufacturing Systems, *to appear: Proceedings of Robotics and Manufacturing'2001 International Conference*, Cancun, Mexico, 21-24 May.

Mortensen, M. (2000), *The JOPC project, introduction* (http://www.jopc.com)

OMG (1998), CORBA based Machine Control white paper. OMG Doc. mfg/98-03-10.

OPC Foundation (1998), *OLE for Process Control presentation document*, (http:// www.opcfoundation.org)

Parunak, H. Van Dyke (1998), *What can Agents do in Industry, and Why? An Overview of Industrially Oriented R&D at CEC*, Industrial Technology Institute; CIA'98.

Raddadi, M., Razafindramary, D., Boissier, R. & Ponsonnet, R. (1993), Spécification temps-réel et réalisation d'un directeur de commande numérique ouvert, évolutif pour machine-outil. *Revue RAPA*, **6** (3), Hermès Paris.

Searle, J.R. (1969), *Speech Acts: An Essay in the Philosophy of Language*, Cambridge University Press.

W3C (2000) *Simple Object Access Protocol (SOAP) 1.1*, W3C Note 08, May 2000 http://www.w3.org/TR/SOAP/