Manufacturing Entities With Incomplete Information

Paulo Sousa, Carlos Ramos
Departamento de Engenharia Informática
Instituto Superior de Engenharia do Porto (ISEP/IPP)
Rua São Tomé, s/n,
4200 Porto
PORTUGAL
E-mails: {psousa, csr}@dei.isep.ipp.pt

José Neves
Departamento de Informática
Universidade do Minho
Largo do Paço
4719 Braga Codex
PORTUGAL
E-mail: jneves@di.uminho.pt

Abstract: In the 80's, Computer Integrated Manufacturing (CIM) concept was adopted as a solution to all the problems in manufacturing. However, it was soon realized that the evolving demands for manufacturing systems were not handled by CIM. New alternatives rise, and especially Agent-Based Manufacturing and Holonic Manufacturing Systems, for coping with today's manufacturing systems requirements. From an informational point of view (i.e. their database), manufacturing systems are very complex. Thus special attention is given to the representation and handling of incomplete information. From a developer's point of view, the ability to reuse code (i.e. functionality) across several entities greatly reduces developing efforts. Thus, inheritance of behaviours is proposed as a way to achieving code reuse. This paper presents a framework for the development of intelligent manufacturing systems, which allows the modelling of incomplete information and inheritance into each agent/holon. A prototype for an already described system for dynamic scheduling of manufacturing orders is also presented.

Paulo Sousa studied Computer Science at *Instituto Politécnico do Porto* (Polytechnic Institute of Porto, Portugal) - ISEP/IPP from 1990 through 1995 with a specialisation in Industrial Informatics. In 1998, he successfully concluded a post-graduation on "Distributed Systems, Computer Architectures and Computer Communications" at *Universidade do Minho* (University of Minho, Portugal), and started his PhD on Holonic Manufacturing Systems. Paulo Sousa worked for 3 years as an application developer for a Portuguese software house in the field of electronic archives, database retrievals and component building. In 1996 he became Assistant Professor at ISEP/IPP. His main research interests are Computer Graphics and Distributed Intelligent Systems. *URL:* http://www.dei.isep.ipp.pt/~psousa

Carlos Ramos is Coordinator Professor at the Institute of Engineering - Polytechnic of Porto (ISEP/IPP). He got his Ph.D from University of Porto in 1993. He is responsible for a research group involving 30 researchers and covering such areas as Planning & Scheduling, Knowledge-based Systems, Intelligent Agents and Knowledge Discovery. This group has published more than 150 papers in Scientific Journals and Conferences over the last 6 years. He is also Director of the Computer Integrated Manufacturing Centre of ISEP/IPP.

José Neves is Cathedratic Professor of Computer Science at the University of Minho, Portugal (UM). He got his Ph. D in Computer Science from Harriet Watt University, Edinburgh, Scotland in 1984. His current research activities span the fields of Extended Logic Programming, Knowledge Representation and Reasoning systems and Multi-agent Systems, and he has more than 100 published papers in international journals, conferences, workshops and symposiums. He is the Director of the Research Group on Artificial Intelligence of UM.

1. Introduction

Manufacturing has changed a lot since its beginning, especially for the last decades. These changes are not only technology-related, but also concern the way customers interact with the manufacturing system, the environment that must be preserved, the shorter product's life-cycle, etc.

Until a few years ago, the CIM (Computer Integrated Manufacturing) concept was considered satisfactory enough for treating enterprise/manufacturing requirements. However, taking into account a new set of organisational and economic concepts, it becomes clear that the centralised CIM approach is not the answer. On the contrary, the requirements for today's and future manufacturing systems suggest autonomy, distribution, and flexibility, while stressing the need for co-ordination amongst production units[1]. It is expected that rigid, static and hierarchical manufacturing systems will give way to systems that are more adaptable to rapid change [2].

It is at this point that agent technology is adequate. An agent is an autonomous entity, maybe intelligent, that pursuits its own goals and, eventually, it may cater for the user's ones [3]. Some authors would like to augment this definition in order to incorporate social ability[4], mental behaviour (e.g. knowledge, beliefs, desires, and intentions)[5], and learning [6].

A Multi-Agent System (MAS) is a population of agents that exhibit social behaviour and are capable of interacting with each other in a co-operative fashion [7], while simultaneously each agent may pursue individual objectives[8]. This interaction assumes some kind of communication language (e.g. KQML [9]) and perception of the surrounding environment. The use of a multi-agent architecture will make the decisions be taken in a decentralised way[10].

Agent-based systems are "best suited for applications that are modular, decentralised, changeable, ill-structured and complex"[11], and with a "great number of interactions among components"[10].

On the other hand, a manufacturing system may be characterized by:

- Its functions (e.g. production planning, production scheduling, inventory, etc.), which can be seen as modules.
- As it was already referred, the traditional CIM centralised architecture is not satisfactory for today's manufacturing requirements; new manufacturing systems must be decentralised manufacturing systems.
- Physically, a manufacturing system is based on resources (e.g. numeric control machines, robots, AGVs, conveyors). The number and configuration of these resources may change during the system's life time.
- Since the manufacturing process is a dynamic one (e.g. suppliers and consumers in a supply chain may change many times) it is impossible to know the exact structure or topology of the system, thus being ill-structured.
- The number of products and orders, as well as different alternative production routes, count for the highly complex nature of the manufacturing systems.

Therefore, it seems that a framework based on the principles which the Distributed Artificial Intelligence (DAI) and Multi-Agent Systems (MAS) paradigms were built on, appears as the most natural solution for the new generation of manufacturing systems [12]. This methodology will allow the modelling of a system as a set of intelligent, autonomous and co-operative elements in order to achieve reconfigurable and extensible architectures [13].

On the manufacturing arena, these two paradigms (DAI and MAS) are best represented by the concept of Holonic Manufacturing Systems (HMSs). The idea behind HMSs is to provide a dynamic and decentralised manufacturing process, in which humans are effectively integrated, so that changes can be made dynamically and continuously.

As Agent-based Manufacturing relies on the notion of agent, HMSs are based on the notion of Holon [14]. Arthur Koestler coined the term 'Holon' from a combination of Greek *holos* (whole) with the suffix *on* which, as in prot*on* or neutr*on*, suggests a particle or part. A Holon means simultaneously a whole (to their subordinated parts) and a dependent part when seen from the inverse direction. Thus, a Holon can be made up of other holons.

This aggregation of holons inside other holons is called holarchy. An HMS is a holarchy that integrates the entire range of manufacturing activities from order booking through design, production, and marketing to achieve the agile manufacturing enterprise.

Although there is no consensus in the research community about agents and holons, as far as the scope of this paper, the two words can be used as synonyms for autonomous and intelligent entities able to co-operate with each other.

The paper is organised as follows. Section 2 presents the basic architecture for each individual agent. Section 3 introduces the issue of handling incomplete and negative information in manufacturing systems. This Section also presents the types of null values used and how to implement them. Section 4 states the problem of scheduling in manufacturing and briefly presents the proposed system's architecture. A prototype of an application for the dynamic scheduling of manufacturing orders is also presented. In Section 5 we look at future work, and present the conclusions.

2. The Holon's Archetype

Figure 1 shows a holonic manufacturing system with seven (7) holons of three different types – resources, tasks and products.

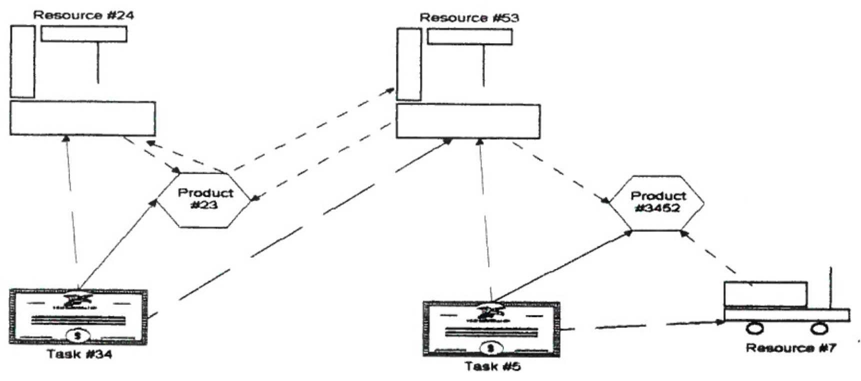


Figure 1. A Holonic Manufacturing System

Lines connecting holons represent confidence relations (or any kind of relationship, e.g. temporary clustering) between two holons. Different lines (e.g. solid; dotted) represent different relations. These relations are unidirectional and their value's meaning is entirely within the holon's responsibility, i.e. the semantics of a relation and its value are holon-dependent and not imposed by the system.

Figure 2 shows the archetype of a holon with its main building blocks.

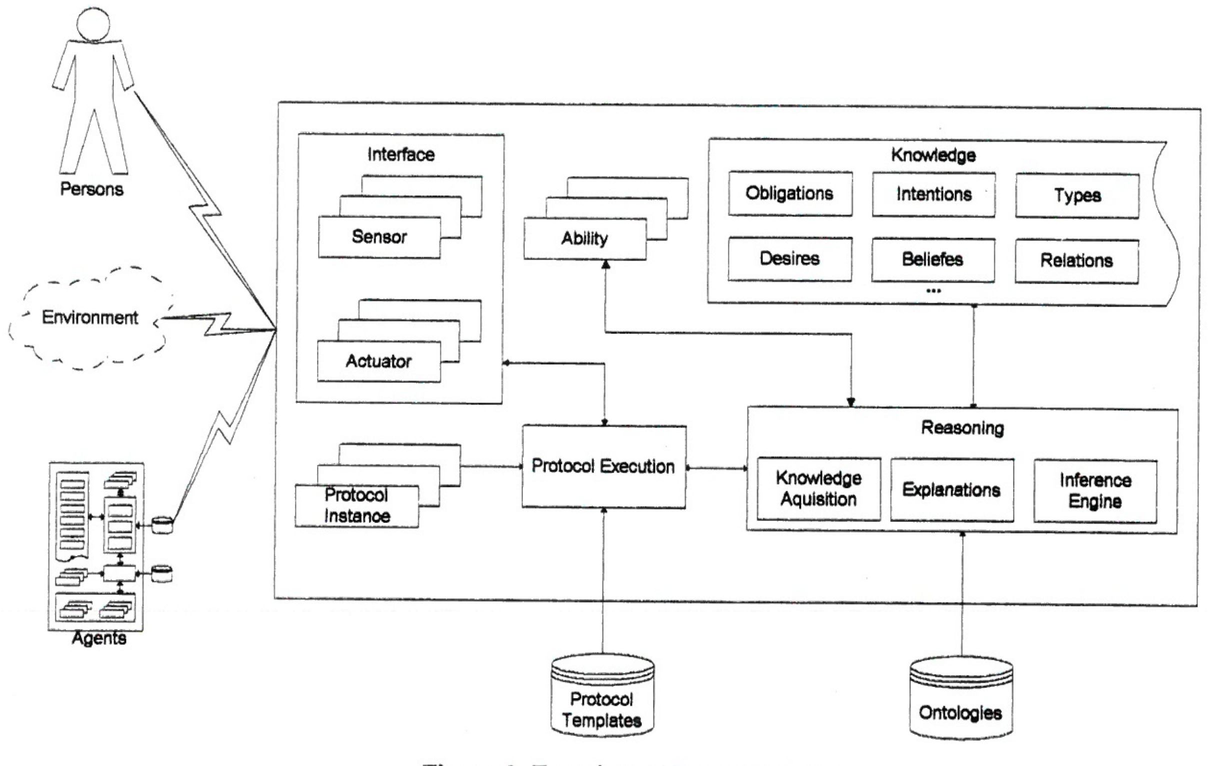


Figure 2. Functional View of A Holon

The *interface* block means both Human-Computer Interaction and the ability to interact with other holons via an agent communication language. The *Protocol Execution* block is responsible for handling conversations, for receiving messages from the environment (e.g. from other agents, or physical sensors), looking for the appropriate protocol script and executing it through passing the information to the reasoning module.

The holon's *knowledge* is defined by its beliefs, desires, intentions, obligations, etc. It also has its own objectives; therefore, the goal's contradiction must be avoided. The productions below state the mental model of holon *Resource#24*, as given in Figure 2, stating its goals (e.g. to earn a profit of 100 or more cash units), and capabilities (e.g. mill).

```
name(resource#34)

type(resource#34,            milling_machine)


goal(state(stable),             handle_goal_1)
handle_goal_1     ←state(initializing)      ∧
                       peer(D)               ∧
                       name(MySelf)          ∧
             send(msg(error), D, MySelf)     ∧
                       update(state, [stable])


handle_goal_1



goal( (profit(X), X > 100); handle_goal_2)

handle_goal_2

state(stable)

profit(30)

location('thor.cim.isep.ipp.pt')

mill(Speed, Dur, Tool, ToolPath)
             ←set_speed(Speed)            ∧
             load_tool(Tool)              ∧
              execute(Duration, ToolPath) ∧
             increment_profit(25)

cfg(Filename)        ←         update(state,
[initialising])                               ∧
                 read_cfg(Filename)           ∧
                 update(state,  [stable])     ∧
                 increment_profit(10)

increment_profit(I)      ←profit(A)           ∧
                   N   is   A   +   I         ∧
                   update(profit, [N])
```

Each holon belongs to one or more classes (i.e. a *type*) that define its basic abilities and knowledge by a mechanism of inheritance similar to the one found in the object-oriented paradigm to computing, and in [15]. The type can also be viewed as a role that the holon plays in the system (e.g. resource, task, order).

The *relations* block identifies and quantifies the relationships a holon has with other holons in the system (e.g. the confidence value in the results of some other holon). This block can be seen as a set of tuples according to the template:

(HolonId, RelationName, HolonId, Value, Parameters)

For some of agents in Figure 1, those relations could be implemented as:

```
relation(task#34, confidence, res#24,
     90,[op1])
relation(task#34, confidence, res#24,
     100,[op2])
relation(task#34, confidence, res#53,
     100,[op1,op3])
relation(task#34, make, product#23, _, [])

relation(res#24,   capability,   product#23,
_,[op1])
relation(product#23, operation, res#24,    1,
     [op1]        )
```

The holonic concept of holarchy is implemented in the *Part/Whole* block as follows:

holarchy(HolonId, HolarchyName,
PartsList, WholeList)

where *Name* is the name of the holarchy (since a holon can belong to several holarchies at the same time) and *PartsList* and *WholeList* are lists containing the identification of holons related to this one in that holarchy. For example:

holarchy(cell_1, sched_holarchy, [res_234, res_7], [scheduling])

The holon's *reasoning* abilities make it draw conclusions in the face of its knowledge, give explanations about its conclusions, and acquire new knowledge from the user, the environment, or other holons. It is also responsible for the holon's behaviour. A possible and simple solution for the previous example can be:

```
on_read ← name(MySelf) ∧
         read(msg(Q, R), Me, Sender) ∧
         assert(peer(Sender)) ∧
         ( (demo_α(Question) ∧
         send(msg(R), Sender, Me) ∧
         handle_goals
         ) ∨
         handle_goals
         ) ∧
         retract(peer(_))

handle_goals ←          goal(G, E) ∧
```

```
                         not G → E ∧
                         fail
handle_goals
```

where the predicate *on_read* is called inside a loop, and stops when a termination criterion is reached (e.g. maximum profit, user shut-down). In event -driven systems, it can be attached to the message-arrival event-handler.

The predicate *demo_α(P)* handles any event within the holon's boundaries, or tries to solve it by inheritance (i.e. looking at the holon's classes for a specific ability upon request). For example:

```
demo_α(P) ← P
demo_α(P) ← name(MySelf)               ∧
            type(MySelf, MyType)       ∧
      demo_τ(MyType, P)


demo_τ(milling_machine, load_execute(P))
←       load_CNC_code(P)               ∧
        execute_CNC_code()


demo_τ(T, P) ←       type(T,    T2)    ∧
          demo_τ(T2, P)
```

In a scenario like the one where a user's holon (called, for instance, *User#2*) wants to execute a certain Computer Numeric Control (CNC) program in resource *Resource#24*, it proceeds on sending a request to holon *Resource#34*, in the following way:

send(msg(load_execute('part1'),   R), user#2, res#24) ∧

read(msg(R), res#24, user#2)

Obtaining the result in *R*. Although holon *Resource#24* does not know how to answer to *load_execute(P),* that ability is inherited from its class *milling_machine*. The inheritance is not limited to two levels, as shown by the predicate *demo_τ*.

3. Incomplete and Negative Information

In this Section, the applicability of extended logic programming to the representation of partial information is investigated.

3.1 Introduction

Considering the Logic Programming framework, a question *?P* to the database can be proved or not (thus being *true* or *false*). However, classical logic programming is based on some assumptions that impose limitations to the kind of information processing necessary in order to handle incomplete data. Some of these assumptions are:

- *Closed World* – every data item missing from the database is considered false;
- *Closed Domain* – every object in the universe of discourse is represented in the database.

Real systems, however, do not work on these assumptions, and can largely benefit from approaches that turn around these limitations. Partial information commonly occurs in knowledge bases and negotiation procedures[16,17,18] .

The proposed object language for one's system is of familiar logic programming. I.e. it contains a two -place one-direction connective that forms rules out of literals, which endorse two kinds of negation, *weak* (also called *negation as failure*) and *strong* (also called classical or *explicit negation*). A *strong literal* is an atomic first-order formula preceded by *strong negation* '¬'. A *weak literal* is a literal of the form *not L*, where *L* is a strong literal. Informally, *not L* reads as *there is no evidence that L is the case*, while ¬*L* is to be interpreted as *L is definitely not the case.*

In other words, instead of relying on the closed-world assumption (i.e. everything not known is false), the knowledge of something being false must explicitly be represented in the Knowledge Base (KB). Thus, the KB has two parts of knowledge: what is known to be *true*, and what is known to be *false*. Everything else is *unknown*.

In Logic Programming this can be seen as:

```
demo(P, true) ←        P
demo(P, false) ←       ¬P
demo(P, unknown)
```

where P is the goal to be proven and ¬ stands for 'explicit negation'.

However, there are cases where the Closed-World Assumption (CWA) makes sense, e.g. an Order Management program can assume that if an order is not in the system, then it does not exist (i.e. it is *false* and not *unknown*). In that case, the CWA is represented as:

$\neg$P $\leftarrow$ not P

which stands for *P is false if it is not possible to prove P*.

By adding the ability to handle incomplete information, a system's database can better describe the real world. In the manufacturing case, there are several situations where the whole information needed by the system is unavailable, e.g. a customer's order does not completely state the product's attributes (colour, etc.). Instead of considering this as a malformed input and of ignoring it, the system can use it to guide its decision given the fact the information available is of confidence. I.e. in the previous example of an incomplete customer order, the customer really wanted to place the order and would define the colour and other missing attributes later on.

Although this information is not completely defined, and thus it is impossible to really use it, its existence can be of more use than if it were not there. In a scenario of manufacturing scheduling, it is not possible to allocate the task and the complete list of materials for an incomplete order. However, it is possible to make a "low-commitment" reservation for that task and materials based on average duration and statistical data, of which materials are used more often.

3.2 Adding Incomplete Information Handling

This Subsection gives examples of three types of null values following the approach described in [19]. Consider the knowledge base of a manufacturing system presented by the following productions.

order(100, manuel, [detail(300, meter, shirt_ref50)])
order(101, carlos, [detail(10, unit, pants_ref104)])
¬order(Number, Customer, Details) $\leftarrow$
not order(Number, Customer, Details)

The last production states that if an order is not represented in the knowledge base then it is considered *false*. I.e. closed-world assumption.

3.2.1 Null Values

A null value represents some missing data item. An example is an order that was placed but the operator did not know which customer had done it.

order(202, someone, [detail(20, unit, sweater_ref304)])
null(someone)
exception_null(order(Number, _, Detail))
$\leftarrow$ *order(Number, Customer, Detail)* $\wedge$ *null(Customer)*

The first clause identifies the new knowledge. The second clause actually states that *someone* is a null value. The third one is the mechanism for handling with null values for clauses of type *order/3*.

If the query *?order(202, manuel, [detail(20, unit, sweater_ref304)])* is posed to the database, the answer will be *unknown* instead of *false*.

3.2.2 Null Values of A Possible Set

Another kind of null value represents information of an enumerated set. For example, an order which the customer has not decided yet the colour of the product for.

exception_set(order(305, josé, [detail(10,meter, shirt_ref10a)]))
exception_set(order(305, josé, [detail(10,meter, shirt_ref10b)]))

This type of null value implements the logical exclusive or operator.

3.2.3 Prohibited Values

These kinds of null values specify situations not allowed in the database. An example can be the fact that orders are not accepted from "Papolians" customers (from a fictional country called "Papolia").

null_n_allowed(customer(Name, papolia))

3.3 Modifications in the Holon's Inference Engine

This Subsection presents the modifications needed in the inference engine of the Holon in order to accommodate and handle the kind of incomplete information presented in the previous Section. The predicate *demo_α* needs be modified as follows:

```
demo_α(P, true) ← P ∧ not exception_null(P)
demo_α(P, unknown) ← exception_null(P)
demo_α(P, unknown) ← exception_set(P)
demo_α(P, not_allowed) ← null_n_allowed(P)
demo_α(P, false) ← ¬P ∧ not exception_set(P)
demo_α(P, T) ← name(MySelf) ∧
               type(MySelf, MyType) ∧
                  demo_τ(MyType, P, T)
demo_α(P, unknown)
```

Obviously, the call to *demo_α* also needs be changed and the answer sent back to other agents should now contain the *true-value* returned by *demo_α*.

```
on_read ← name(MySelf) ∧
          read(msg(Q, R), Me, Sender) ∧
          assert(peer(Sender)) ∧
          ( (demo_α(Question, T) ∧
          send(msg(R, T), Sender, Me) ∧
          handle_goals
          ) ∨
          handle_goals
          ) ∧
          retract(peer(_))
```

The agent who requested the information *(Sender)* must know to act accordingly to the *true value T* that he received for his question.

3.4 Sample Questions

Some questions that can be posed to the system and the respective answers are presented next.

?- demo_α(order(100, manuel, _), T)
T = true

?- demo_α(order(105, miguel, _), T)
T = false

The previous example shows the normal functioning of the system, the order 100 is for customer *manuel*, so it is *true*. The second situation shows the CWA at work, since that order is not present in the KB, it is considered *false*.

?- demo_α(order(202, miguel, _), T)
T = unknown

The next example is a situation where a null value was declared, so the answer would be *unknown*.

?- demo_α(order(305, josé, [detail(10, meter, shirt_ref10a)]), T)
T = unknown
?- demo_α(order(305, josé, [detail(10, meter, shirt_ref47X)]), T)
T = false

The second question in the previous example shows that the answer will be *false* if the question is not one of the multiple choices that had been declared.

?- demo_α(customer(pappee, papolia), T)
T = not_allowed

In this last example, a *not allowed* value is returned when trying to prove for a "papolian" customer.

4. Case Application: Manufacturing Scheduling

Scheduling of manufacturing tasks is concerned with the allocation of task's operations to resources in order to complete a task within specified constraints. These constraints include delivery date, quality degree, etc.

The scheduling procedure must take into account things like duplicate functionality (i.e. several resources are able to perform the same operation), load balancing, production policies (e.g. JIT, make to stock, etc.), and the dynamics of the shop-floor (e.g. tool set-up times, resource unavailability – due to break-down or maintenance).

4.1 Proposed System Architecture

This Subection briefly describes a system architecture for the dynamic scheduling of manufacturing orders (a very detailed description is given in [13, 20] and is not the objective of this paper). The *Fabricare* system (http://www.dei.isep.ipp.pt/~psousa/fabricare/), as it is called, uses a scheduling procedure adapted from a centralised method described in [21], and an adaptation[22] of the Contract Net Protocol [23] .

The user interacts with the system via a special agent called *Task Manager* that is responsible for launching new *Task Agents*. For the scheduling of operations, a *Task Agent* will negotiate with *Resource Agents*, and establish a contract. The *Resource Agents* will then use constraint propagation in order to guarantee the relationships among the different operations that aim at the same task (Figure 3).

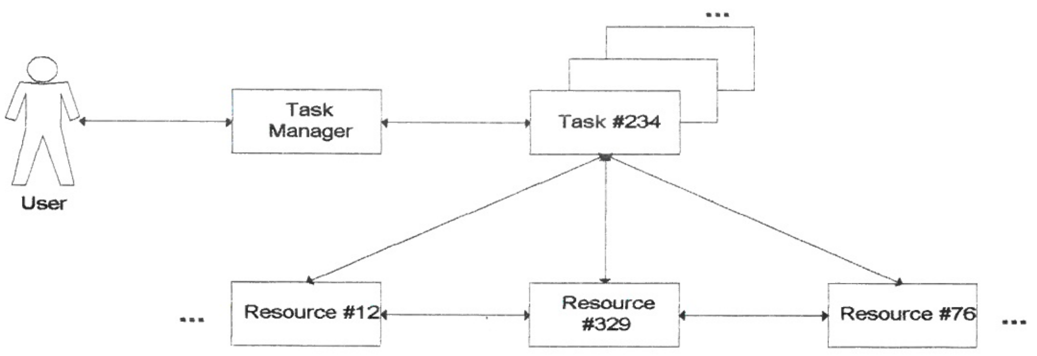
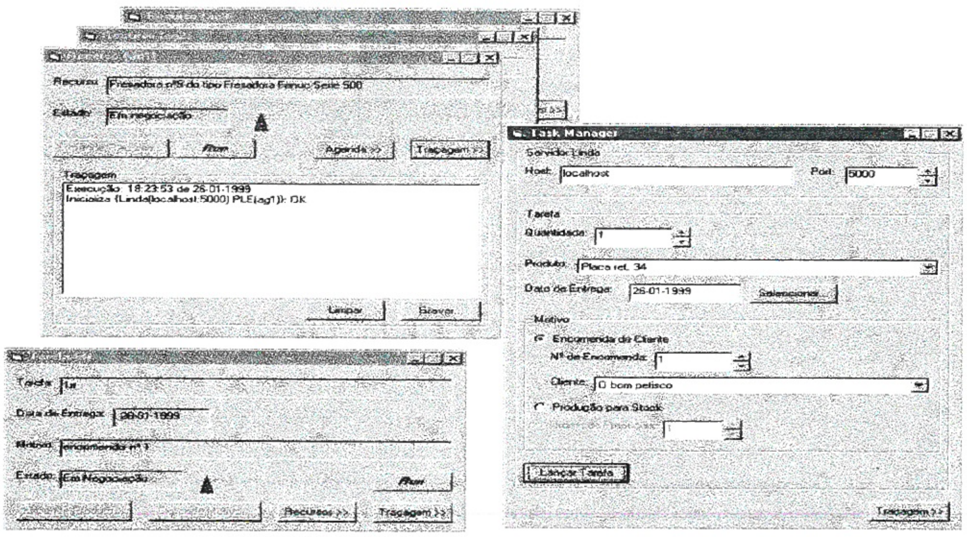


Figure 3. System Operation

A first implementation[24] of this architecture was used to test a framework for Holonic Manufacturing Systems. In that work, a C++ framework was built with classes representing agents' generic behaviour and application specific classes were used for particular agents (e.g. a resource, or a specific resource). This paper addresses a new approach to be described in the next Section.

4.2 Prototype Application

The prototype application (Figure 4), called Fabricare, was built using SICStus Prolog for the agent's part of the system, and Visual Basic for the user interface.



Recurso: Fresadora nº9 do tipo Fresadora Fanuc Serie 500
Estado: Em negociação

Traçagem
Execução: 18:23:53 de 26-01-1999
Inicializa {Linda(localhost:5000) PLE(ag1)}: OK

Task Manager
Servidor Linda
Host: localhost
Port: 5000
Tarefa
Quantidade: 1
Produto: Placa ref. 34
Data de Entrega: 26-01-1999
Motivo
Encomenda de Cliente
Nº de Encomenda: 1
Cliente: O bom petisco
Produção para Stock

Tarefa: 1ª
Data de Entrega: 26-01-1999
Motivo: encomenda nº 1
Estado: Em Negociação

Figure 4. The Prototype's GUI

At this stage, communication is done using SICStus's implementation of the Linda co-ordination technology [15] that handles all aspects of communications, by implementing a shared tuple-space (i.e. there are no directed communications between agents). The use of the Linda has significantly improved the prototype's development, but it may become a severe bottleneck [26], since it introduces a centralised element into the system.

Since the development is at an early stage, the prototype is very simple, and has been used mainly to test the correctness of the distributed version of the scheduling procedure. At this moment, *Task Agents* always choose the earliest time interval available from the several solutions bided by resources, and only consider the first resource bid for each operation (i.e. it does not take advantage of different production paths).

Although the ability of handling incomplete information is incorporated in the system, the decision process is not yet guided by that knowledge. However, the database can be queried regarding incomplete and negative information. *Resource Holons* will use this information to generate low -commitment schedules into their agendas.

Exchanged messages between task and resources are Prolog clauses written to and read from the Linda tuple-space. Each clause has identifiers which indicate the origin and the destination agent, as well as a tag that makes it possible for each party to identify related messages.

Each agent uses the relation block of the agent archetype (Figure 2), to maintain confidence information about other agents. This information can be used by the *Task Agents* to select among different *Resource Agents*, those that bided the same operation.

5. Conclusions and Future Work

Future work will include the support for an agent communication language (e.g. KQML [9]) for directed communication between agents, and the mechanisms to avoid the bottlenecks that may exist with the use of the Linda.

Some kind of a directory service is also necessary for each agent to locate other agents and advertise its own abilities.

Future Work also includes running the system with some publicly available benchmarks for a better comparison with other distributed and centralised systems.

An architecture was proposed and presented, to allow the modelling of intelligent agents with inheritance. This architecture was used to build a prototype of a manufacturing system.

The proposed architecture also allows the modelling of incomplete information in the system. Incomplete information can be used to help the decision process thus providing an added value to system instead of being treated as erroneous data.

An important aspect that allowed for a reduced development time and code-size (when compared to the C++ version) was the use of the Prolog language, due to its symbolic processing nature.

The use of a programming language more specific to the Artificial Intelligence field (i.e. Prolog) makes it possible to quickly use the existing body of research in such matters as rationality, decision support, etc.

Acknowledgment

This work has been partially supported by ESPRIT project 21955 – Intelligent Manufacturing Systems Working Group. The first author would also like to express his gratitude to the Scientific Council of ISEP/IPP for their financial support.

REFERENCES

1. SOLBERG, J. and KASHYAP, R. , **Research in Intelligent Manufacturing Systems**, Proceedings of the IEEE, Vol. 81(1), January 1993.

2. VALCKENAERS, P., BONNEVILLE, F., VAN BRUSSEL, H., BONGAERTS, L. and WYNS, J., **Results of the Holonic Control System Benchmark at K.U. Leuven**, Proceedings of the Rensselaer's 4th International Conference on Computer Integrated Manufacturing and Automation Technology, 1994.

3. GREEN, S., HURST, L., NANGLE, B., CUNNINGHAM, P., SOMERS, F. and EVANS, R., **Software Agents: A Review**, Intelligent Agents Group Report, 1997.

4. GENESERETH, M. and KETCHPEL, S., **Software Agents**, COMMUNICATIONS OF THE ACM, 37(7), 1994, pp.48-53.

5. SHOHAM, Y., **Agent-Oriented Programming**, ARTIFICIAL INTELLIGENCE, 60(1), 1993, pp.51-92.

6. SCHAERF, A., SHOHAM, Y. and TENNENHOLTZ, M., **Adaptive Load Balancing: A Study in Multi-agent Learning**, JOURNAL OF ARTIFICIAL INTELLIGENCE RESEARCH, Vol. 2, MORGAN KAUFMANN PUBLISHERS, 1995, pp. 475-500.

7. DURFEE, E. and ROSENSCHEIN, J., **Distributed Problem Solving and Multi-agent Systems: Comparisons and Examples,** Proceedings of the 13th International Distributed Artificial Intelligence Workshop, 1994, pp. 94-104.

8. FERBER, J., **Modèle de systèmes multi-agents: du réactif au cognitive**, Proceedings of INFAUTOM'93, Toulouse, 18-19 February 1993, pp. 26-56.

9. FINN, T., LABROU, Y. and MAYFIELD, J. , **KQML As An Agent Communication Language**, in J. Bradshaw (Ed.) Software Agents, AAAI PRESS/MIT PRESS, 1997.

10. KOUISS, K., PIERREVAL, H. and MEBARKI, N., **Using Multi-Agent Architecture in FMS for Dynamic Scheduling**, JOURNAL OF INTELLIGENT MANUFACTURING, Vol. 8, Chapman & Hall,1997, pp. 41-47.

11. PARUNAK, H. , **What Can Agents Do in Industry and Why?**, Proceedings of the 2nd International Conference on Cooperative Information Agents (CIA'98), Paris, 3-8 July 1998.

12. SOUSA, P. and RAMOS, C., **Proposal of A Scheduling Holon for Manufacturing**, Proceedings of the 2nd International Conference on the Practical Application of Agents and Multi-Agents Technologies (PAAM'97), London, 21-23 April 1997, pp. 255-268.

13. SOUSA, P. and RAMOS, C., **A Dynamic Scheduling Agent for Manufacturing Orders**, JOURNAL OF INTELLIGENT MANUFACTURING - Special Issue on Agent Based Manufacturing, Vol. 9(2), Chapman & Hall, 1998, pp. 107-112.

14. KOESTLER, A., **The Ghost in the Machine**, HUTCHINSON & CO, London, 1967.

15. ANALIDE, C. and NEVES, J., **Estruturas Hierárquicas com Herança**. Unidade de Ensino. Departamento de Informática. Universidade do Minho, Braga, Portugal, Janeiro 1997.

16. NEVES, J., **A Logic Interpreter To Handle Time and Negotiation in Logic Databases**, Proceedings of the ACM 1984 Annual Conference, San Francisco, CA, 1984.

17. TRAYLOR, B. and GELFOND, M., **Representing Null Values in Logic Programming**, Proceedings of the International Logic Symposium (ILPS'93), Vancouver, British Columbia, Canada, 1993.

18. NEVES, J., MACHADO, J., ANALIDE, C., NOVAIS, P. and ABELHA, A., **Extended Logic Programming Applied to the Specification of Multi-Agent Systems and Their Computing Environment**, Proceedings of the 1997 IEEE International Conference on Intelligent Processing Systems, Beijing, 1997.

19. ANALIDE, C. and NEVES, J. , **Representação de Informação Incompleta**, Unidade de Ensino, Departamento de Informática. Universidade do Minho, Braga, Portugal, Novembro 1996.

20. SOUSA, P., RAMOS, C. and NEVES, J., **Contracting Tasks Between Autonomous Resources – An Application to Dynamic Scheduling of Manufacturing Orders**, Proceedings of the 4th International Conference on the Practical Application of Agents and Multi-Agent Technologies (PAAM'99), London, 19-21 April 1999, pp. 345-362.

21. RAMOS, C., ALMEIDA, A. and VALE, Z., **Scheduling Manufacturing Tasks Considering Due Dates: A New Method Based On Behaviours and Agendas,** International. Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, Melbourne, 1995.

22. SOUSA, P. and RAMOS, C., **A Distributed Architecture and Negotiation Protocol for Scheduling in Manufacturing Systems**, COMPUTERS IN INDUSTRY - Special Issue on Life Cycle Approaches to Production Systems: Management, Control and Supervision, Vol. 38(2), ELSEVIER, March 1999, pp.103-113.

23. DAVIS, R. and SMITH, R., **Negotiation As A Metaphor for Distributed Problem Solving**, ARTIFICIAL INTELLIGENCE, Vol. 20(1), 1983, pp. 63-109.

24. SILVA, N., SOUSA, P. and RAMOS, C., **A Holonic Manufacturing System Implementation**, Proceedings of the Advanced Summer Institute (ASI'98). Bremen, 14-17 July 1998.

25. BJORNSON, R., CARRIERO, N., GELERNTER, D. and LEICHTER, J., **Linda, the Portable Parallel**, Technical Report 520, Yale University Department of Computer Science, January 1998.

26. NWANA, H., LEE, L. and JENNINGS, N., **Coordination in Software Agent Systems**, BT TECHNOLOGY JOURNAL, 14 (4), 1996, pp. 79-88.