13

CONTRACT MANAGEMENT IN AGILE MANUFACTURING SYSTEMS

José Barata, L.M. Camarinha-Matos
New University of Lisbon
Quinta da Torre – 2829 Monte Caparica, PORTUGAL
jab@uninova.pt, cam@uninova.pt

In this paper an agent-based architecture in which cooperation is regulated by contracts is proposed as a flexible approach to dynamic shop-floor re-engineering. It describes the dynamic and flexible co-operation of manufacturing agent and how they can be created from a generic agent template. First experimental results are also introduced.

1. INTRODUCTION

The capability to rapidly change the shop-floor infrastructure is a condition to allow participation of manufacturing enterprises in dynamic cooperative networks. The evolution in the market conditions, the environment and working conditions regulations, improved standards for quality, fast technological mutation, and changes of the production paradigm itself, impose agility and new ways of co-operation to manufacturing companies. Enterprise networks, virtual enterprises, advanced supply chains, etc. are examples of co-operative structures created to cope with the mentioned aspects. Manufacturing companies wishing to join these networked structures need to be highly adaptable to cope with the requirements imposed by the very dynamics and unpredictable changes. Shop-floor agility implies an increasing level of re-engineering activity. The processes of change (re-engineering/adaptation) have been addressed mostly at the level of business process re-engineering and information technology infrastructures. Little attention however has been devoted to the changes of the manufacturing system itself and yet the shop floor suffers a continuous evolution along its life cycle.

A particularly critical element in a re-engineering process is the control system. Current control/supervision architectures are not agile because any shop-floor changes require programming modifications, which imply the need for qualified programmers, usually not available in manufacturing SMEs. To worsen the situation, the changes (even small changes) might impact the global system architecture, what increases the programming effort and the potential for side-effect errors. It is therefore mandatory to develop approaches to eliminate or reduce these problems, making the process of change (re-engineering) faster and more flexible, focusing on configuration instead of codification.

The work described in this paper assumes that there is a similarity between this re-engineering process and the formation of consortia regulated by contracts in networked enterprise organisations. The problems a company faces in order to join a

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35585-6_68

L. M. Camarinha-Matos (ed.), *Collaborative Business Ecosystems and Virtual Enterprises*

consortium are similar to the shop-floor adaptation problem. The proposed approach is therefore to use the mechanisms and principles developed to support the enterprise integration into dynamic enterprise networks as inspiration for an agile shop-floor re-engineering process. This paper describes the architecture to support dynamic and flexible co-operation of manufacturing agents and how they can be created from a generic agent template, using contracts. First experimental results and current developments are also introduced.

2. ARCHITECTURE

An agent-based architecture in which cooperation is regulated by contracts is proposed as a flexible approach to dynamic shop-floor re-engineering. Manufacturing components are modelled using agents, which build up a community. Every manufacturing component e.g. robots, tools, fixing devices, is associated to an agent that represents its behaviour. These agents must interact in order to generate aggregated functionalities that in some cases are more complex than the simple addition of their individual capabilities. This is what happens, for instance, when several manufacturing components are working together on a manufacturing cell.

When forming a group of collaborative agents there are no limitations on the type of agents, but there is an important restriction that limits their co-operation capability – their spatial relationship. Manufacturing agents that are not spatially related cannot co-operate, as it is the case, for instance of a robot and a tool. If the tool is not within the reachability space of the robot it will be impossible to create a co-operation relationship.

A set of agents working together towards a common goal is called a **consortium**. The consortium – the agency equivalent to a manufacturing cell - is the basic organisational form of co-operation. A basic consortium is composed of one or more agents performing the role of *manufacturing agents* and one agent performing the role of *co-ordinator*, and can also include other consortia as members. The co-ordinator of a consortium is able to execute complex operations that are composed of simpler operations offered by the consortium members.

The better the dynamics of the consortia the better will be the agility of a manufacturing system. If agility is seen as the capability to easily change the operation of a manufacturing system as a reaction to a change in the environment, then an easy way to create and change consortia is an important support to give agility to a manufacturing system.

The important question then is how to create a consortium. When creating a consortium it is mandatory to know what are the available and willing agents to participate on it. It would be important that these agents could be grouped by their spatial relationships (or any other relevant relationship e.g. technological compatibility), i.e., manufacturing agents that could establish consortia should be grouped together because they share something when they are candidates to consortia. The structure used to group manufacturing agents that can co-operate and from which the agents share some concepts is called **cluster**.

Figure 1 shows how manufacturing agents are related to the cluster and consortium. Agentified components in the same "geographical" area of the shop-floor join the same cluster. The different consortia that can be created out of the

cluster represent the different manners to exploit/operate a manufacturing system. Adding or removing a component from the physical manufacturing system also implies that the agent must be removed from the cluster which can also have impact on the established consortia. Each consortium is formed with the help of a **broker**. The broker is an agent that interacts with a human, the cluster, and candidate members to the consortium. Consortium can be created either automatically or manually. In the beginning only the manual option is considered. In the manual option the human interacts with the broker agent to create the consortium by choosing the candidates from the cluster.

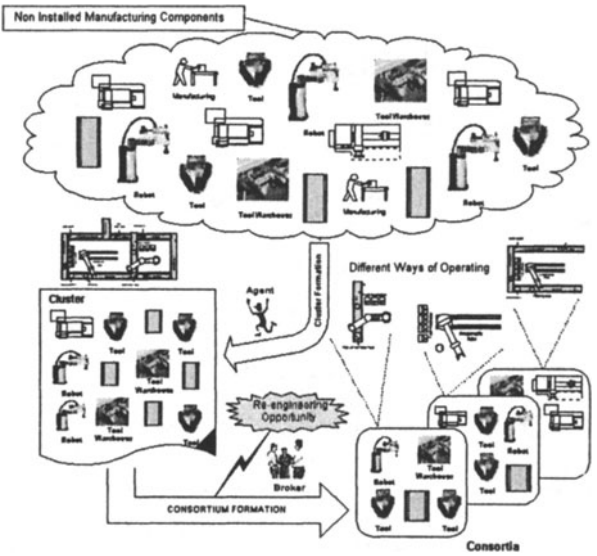

Figure 1 – Consortia Formation

The interactions between the cluster and its members are regulated by a **cluster adhesion contract**, which is a bilateral adhesion contract formed between the cluster and the candidate member. This contract establishes the terms under which the co-operation is going to be established. It includes terms such as the ontologies that must be used by the candidate, the duration, the consideration (a law term that describes what the candidate should give in turn of joining the cluster, usually the skills that the candidate is bringing to the cluster).

The behaviour of the consortium is regulated by **a multilateral consortium contract** that is "signed" by all members of the consortium. The important terms of this type of contract other the usual ones like duration, names of the members, penalties, etc. are the consideration and the individual skills that each member brings to the contract. Note that the skills involved in a specific consortium contract may be a subset of the skills offered by the involved agent when it joins the cluster. The importance of contracts as a mechanism to create/change flexible and agile control structures (consortia) lays on the fact that the generic behaviours presented by generic agents are constrained by the contract that each agent has signed. This calls forth that different consortium behaviours can be achieved by just changing the terms of the consortium contract, namely the skills brought to the consortium.

As it was mentioned above, an agent can participate on a consortium according to two different roles: 1) as member or 2) as co-ordinator. A member must execute all the operations signed by it on the consortium contract that are requested by the co-ordinator. On the other hand, the co-ordinator can create complex operations (services) by aggregation of the individual operations of the members. A co-ordinator can have more complex operations either by adding more members or through the creation of a hierarchical structure of consortia (Figure 2).

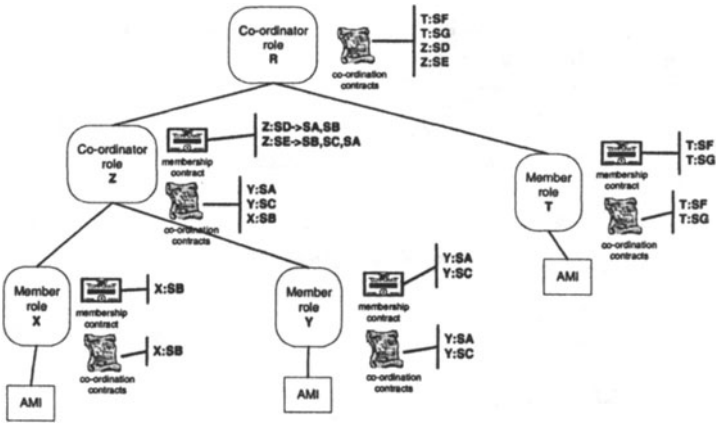

Figure 2 – Hierarchy of consortia

Figure 2 illustrates both roles an agent can play. It is interesting to note that in both roles agents might be subject to two kinds of contracts: **membership contracts**, and **co-ordination contracts**, what just expresses their dual roles, as an agent can be simultaneously member of a consortium and co-ordinator of another consortium. This is the case of the agent Z, which co-ordinates member X and Y, and simultaneously is a member of the consortium led by R. The terms to the right of each contract in figure 2 represent the skills offered to the consortium by each member. For instance, agent Z knows from its co-ordination contract that it can access the skills SA and SC, provided by agent Y, and skill SB, provided by agent X. From these skills agent Z creates the more complex skills SD and SE, which are offered to a higher-level consortium contract, which are represented in the membership contract of Z. The services (complex operations) of each level can be represented using a workflow model or even a Petri Net, and each generic agent must have an execution engine to execute these kinds of operations. In the first approach these operations will be created with the help of a human expert interacting with the broker agent, during the creation/changing of a consortium.
Figure 2 also shows that a **manufacturing agent** co-ordinates an AMI (Agent Machine Interface), which is an agent representing the controller of some specific physical manufacturing component. Separating the AMI from the agent enables that the same generic agent model can be used to play both roles: member and co-ordinator. This is an important advantage because any kind of manufacturing architecture can be implemented using only one type of agent.

Use-cases. The proposed architecture can be better clarified through the discussion of the use-cases of the most important types of involved agents: the *cluster agent*, the *broker agent*, and the *generic agent*. Please note that some agent

behaviours are not represented in the following figures, for the sake of simplicity. This is the case, for instance, of the behaviours related to the termination or breach of contracts.

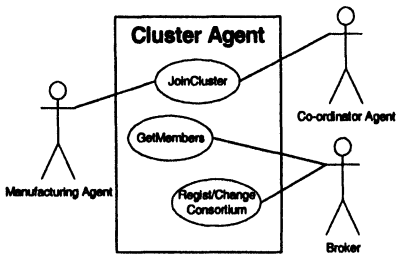

Figure 3 – Use-case of the Cluster agent

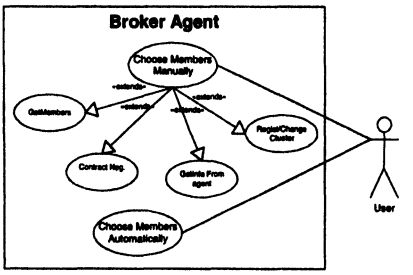

Figure 4 – Use-case of the Broker agent

The cluster agent main interactions come about with manufacturing agents, co-ordinator agents and the broker agent (Fig. 3). The interactions with the manufacturing and co-ordination agents happen in the early phase of their life cycle. This was expected because every agent willing to participate in a consortium must belong to a cluster. The broker agent needs to contact the cluster to find out about the available candidates willing to enter into partnership.

The human user is the only actor that commands the broker agent (Figure 4). Consortia can be chosen either using a manual or automatic approach. In the current state of the work only the manual approach is considered. The cases extended by the manual approach show the main interaction that must be performed by the broker with the other actors of the architecture. *GetMembers* interacts with the Cluster to receive the list of available candidates. *GetInfo* interacts with a generic agent to find out its internal details. These interactions take place only among those agents that have been selected by the user as participants in the consortium being created. *Regist/ChangeCluster* interacts with the cluster whenever a consortium has been created or changed. This guarantees that the cluster always have updated information about the current situation of all its members. *ContractNeg* interacts with the chosen agents for the consortium, and it is in this case that the consortium contract is created and negotiated with their members.

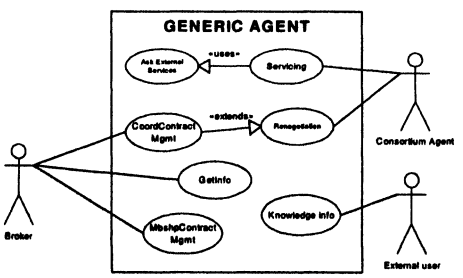

Figure 5 – Use-case of a Generic agent

The actors that act on the generic agent are the broker agent, an external user, and any agent member of a consortium (Fig. 5). The external user is a human that uses the user interface of the generic agent to update and obtain information that is relevant for re-engineering purposes. The information that is relevant here includes: 1) details about the expert that usually operates the physical component represented by the agent, 2) maintenance information, 3) skills, 4) credits, etc. The broker interacts with the behaviours related to the contract negotiation: *CoordContractMgmt* and *MbshpContractMgmt*. The former is actuated when the generic agent is playing the role of co-ordinator and the latter when it is playing the role of member. The behaviour dealing with co-ordinating contracts uses the

renegotiation case whenever the agent is participating as member in other higher-level consortium. An agent playing the co-ordination role can be also actuated by the co-ordinator of a lower level consortium. *GetInfo* is played by the broker to obtain details about the skills of the agent. The remaining behaviour is one of the most important because it is there that the operations offered by the agent are carried on. The servicing case comprehends the mentioned workflow or Petri Net engine to execute complex operations as well as the validation of the requests. Every request must be checked against the contract to see if it is valid. This case must always use the *AskExternalServices* behaviour to ask members of the co-ordinated contract to execute the operation they have offered.

3. CONTRACTS

Background. A brief presentation of the most significant points of the law of contracts will be introduced to give the reader a background of concepts that are being used in the two types of contracts introduced by the proposed architecture.

A contract is **defined** by the Merriam-Webster's Dictionary of Law as "*an agreement between two or more parties that creates in each party a duty to do or not do something and a right to performance of the other's duty or a remedy for the breach of the other's duty*". A contract is made up of a promise of one entity to do a certain thing in exchange for a promise from another entity to do another thing. Some law researchers (Almeida 2001) claim that the contractual statements (promises) are performing acts in the sense that they have effects. This means that the existence of a contract between two or more entities imposes constrains on their behaviour and can produce outcomes that were not possible without a contract, mainly due to the performing nature of the statements or promises.

There are several types of contracts, but in this work only two are considered: generic **multilateral contracts** and **adhesion contracts**. The most important difference between them is the process of formation, which in the case of the adhesion is made using standardised forms. The contract offered by the cluster agent to the candidate member agents is a typical contract of adhesion, in the sense that the cluster imposes its terms. The only thing an agent can do is not accepting it. Part of the terms of this adhesion contract, namely the consideration of the candidate agent, is left open to be filled in by the candidate, when accepting the offer. This mechanism is perfectly supported by real life law systems. **Consideration** was defined by an 1875 English decision as "some right, interest, profit or benefit accruing to the one party, or some forbearance, detriment, loss or responsibility given, suffered or undertaken by the other".

In most of the law systems to **create a contract** it is required at least two sequential statements: an offer followed by an acceptance. An offer can be followed by a counter-offer, which in turn can also be followed by another counter-offer and so on. The process terminates when one of the partners sends an acceptance. The offer and the acceptance might not be the first and second action but they will be surely the last but one, and the last. Offers may set certain conditions on acceptance and to these, the acceptor is bound. The acceptance validates and gives life to the contract. The contract starts at the moment the acceptance reaches the offeror. The

cluster, and generic agents when negotiating the cluster contract will use the offeror-acceptance protocol of real life contracts with some adaptations.

An offer, once made, can be revoked before acceptance. An offer can also expire if a deadline for acceptance passes. If there is no specified deadline, then the offer expires in a "reasonable time", depending on the subject matter of the contract (Almeida 2001). In the approach being followed an offer is made without specifying a deadline. This indicates that it must be answered in a "reasonable time", which is the normal time-out imposed to the global architecture for communication among the agents. An offer that was rejected cannot be subsequently accepted.

An alternative to reach an agreement other than the offer-acceptance protocol is using joint contractual terms, which express the agreements of the parts in only one text. This modality is specially used for creating contracts that involve more than two partners (multi-lateral contracts). In this case the parts reach agreement on the final terms of the contract using different kind of communicative acts in a preliminary phase. Afterwards, the final contract is put on a written form (final agreement) and finally all the partners must subscribe the contract. The contract turns effective when the last partner subscribes the document. The formation of the consortium contract used in the proposed architecture uses this modality with some adaptations. The human user interacting with the broker will do the agreement on the terms of the contract (preliminary phase). It is this user that chooses the skills that each agent will bring to the contract (this user is just configuring the system). The broker agent then sends the final *text* to all partners to be subscribed. When the last agent finally subscribes it, the contract is considered as valid.

Examples in the architecture. The model of the cluster adhesion contract is depicted in figure 6, and it shows the different parts involved in the contract:

(i) General Conditions. This part describes its purpose, definitions and the ontology that validates the definitions and concepts of the contract. The parties, the contract identification as well as the starting date and the duration of the contract are also included in this part.

(ii) Consideration. Both considerations from the offeror (cluster) and from the offeree (agent) are described in this part. The agent consideration is the skills that it is interested in bringing to the cluster, which might be used by future consortia. Although the consideration from the cluster side could be just to let the agent belonging to the cluster (implicit consideration) it was decided to define an explicit consideration – cluster skills are represented by some clustering functionalities.

(iii) Contract Logistics. An important parameter defined here is the *clusterWorkingPlace* that defines the physical area covered by the cluster. A candidate agent to the cluster must check the adhesion contract to see if it is installed in the area covered by the cluster it is trying to join. The addresses of the parties are also defined here. An address in this architecture is the Agent Identification Descriptor (AID) within the multiagent-supporting infrastructure, which is needed to be used to send/receive messages. The maximum number of repetitions for each message (tries) before the sender party has the right to breach the contract claiming that the receiver party is not performing well is also defined. Related to the liveliness of agents is the parameter *maximumTime*, which describes how long the sender party should wait before considering repeating the message.

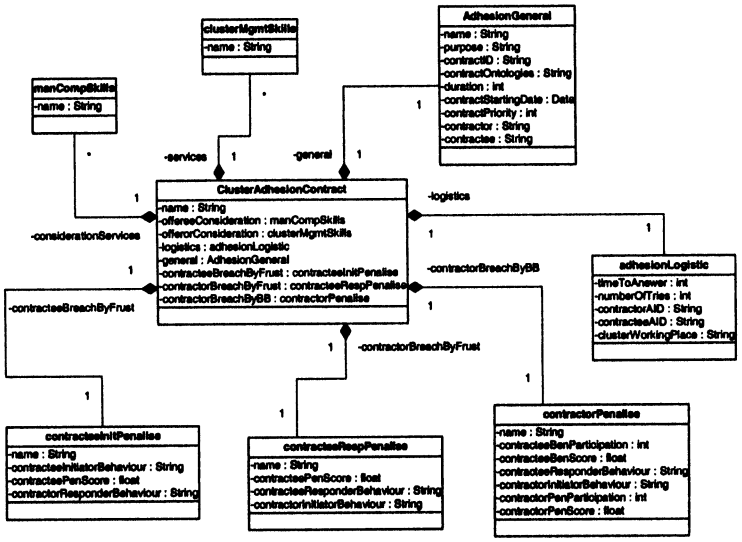

Figure 6 – UML diagram of the Cluster Adhesion Contract

(iv) Exceptions & Terminations. This part describes the behaviour that must be followed when the contract is not completely performed or when the contract reaches its end. The contract can be breached (interrupted) by three important reasons: 1) breached by frustration of the contractor (offeror), 2) breached by frustration of the contractee (offeree), and 3) breached by bad behaviour of the offeree. The breach by frustration occurs whenever a party cannot perform its obligations because of an unexpected event, from which it is not responsible. In this case the other party can only claim for light or even none remedies. The worst situation occurs when breaching by bad behaviour in this case the remedies to be claimed by the offended party can be very high. In the contract modelled it is described the behaviours that might be used for each of the cases and also some remedies that might be claimed by the offended partner.

It must be pointed out that the non-existence of a breach by bad behaviour of the offeror (the cluster) just occurs from the fact that the cluster is not expected to have bad behaviour, because when behaving improperly it is really impairing itself. On the other hand from a law point of view the dominant partner can impose an adhesion contract where it cannot be liable for its bad behaviour.

The cluster adhesion contract is defined externally to the cluster and modelled using a knowledge representation system – Protégé 2000 (Protégé 2001). The cluster agent can interact with this system to have access to the contract representation. Whenever

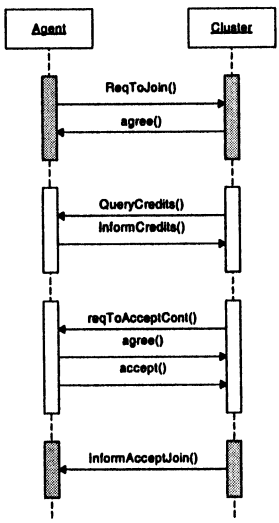

Figure 7 – Sequence of the Joining Cluster Protocol

it needs to offer an adhesion contract to an agent it just uses the form, waiting afterwards for its acceptance or refusal.

An example of the negotiation protocol used by an agent when joining a cluster is shown in figure 7. The formation of the contract starts when the cluster sends the message *reqToAcceptCont*, which contains an instance of an adhesion contract, prepared for the candidate agent. The accept message from the candidate contains the complete adhesion contract, now filled with the terms of the candidate (its skills), and when received by the cluster the contract turns to be valid. The sequence shown in figure 7 only shows the normal situation. For the sake of clarity abnormal situations were not shown. The cluster only agrees to negotiate with the candidate agent if it is not on the black list of the cluster. The cluster agent then checks for the credits of the candidate, which represents a kind of curriculum vitae. A credit is, for instance, the number of hours working properly, or a number that qualifies the global performance of the agent when working on consortia. Those agents with lower level qualification can sometimes not be accepted as members of the cluster. This is to guarantee that consortia created out of a cluster have a certain level of qualification.

The model of the consortium contract can be seen in figure 8. This contract is created by the broker agent with the help of a human expert.

It should be stressed that the human user currently does the negotiation process behind the preliminary phase of this type of contract.

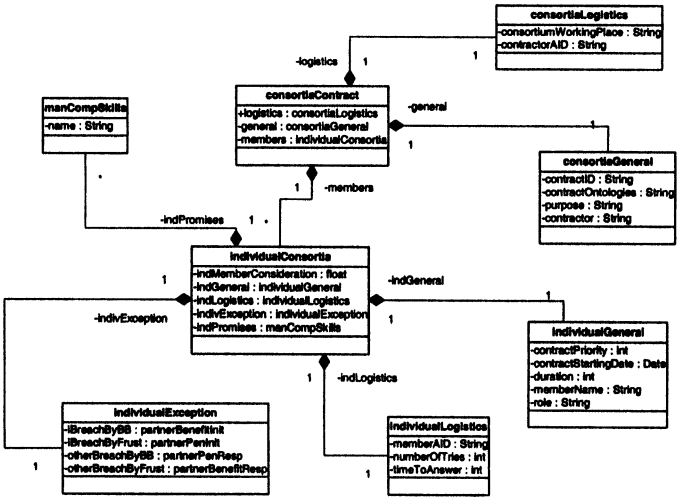

Figure 8 – UML diagram of the Consortium Contract

The model of this type of contract has many similarities with the previous one but has also some slight differences because it is a multilateral contract instead of bilateral contract. To support various members and one contractor the contract has one common part dedicated to the contractor (the agent playing the co-ordination role), and another part dedicated to each of the other members. The *members* attribute is composed of several *individualConsortia* elements that in turn describe the individual contractual terms of each member of the consortium. The *consortiaGeneral* and *consortiaLogistics* parts are similar to the adhesion contract, but applied to the co-ordinator. The *consortiumWorkingPlace* has the same role as

before but now applied at the consortium level. The **promise** (declaration or manifestation of an intention in a contract) brought to the contract by each member is a set of manufacturing skills that are represented by the class *manCompSkills*.

The consideration for each member is a kind of reward. At the end of the contract the co-ordinator awards each member with a number that represents the quality of the handed out service. This award, if added to the agent credits, can be used to improve (or even reduce) its qualification, and might be important for the future participation of the agent on consortia. The Logistic and general parts of the individual members are similar to the global part describing the co-ordinator as well as in the case of the cluster contract. The work about the consortium contracts is an undergoing activity. The breach and termination of contracts and the capability for an agent to detect that it cannot accomplish its established contract still are open questions under investigation.

Figure 9 shows the protocol of consortium contract formation. Its simplicity is mainly due to the fact that most of the negotiation work is made by the human. After choosing what skills each candidate will bring to the consortium the broker generates a pre-proposal for the contract that contains all terms of the individual members and the co-ordinator. This contract is generated using the same approach as the cluster contract.

The reference model of the contract, represented in Protégé 2000 is similar to the UML model shown in figure 8. The broker then instantiates a contract and fills the fields with the choices of the user and sends it to the parties (Figure 9), using the FIPA request protocol. The *Inform* reply of this protocol corresponds to the subscription of the contract. When all parties reply the broker then sends an *Inform* message to the co-ordinator stating that the contract is now valid, and can start to be used.

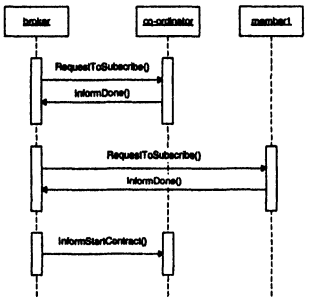

Figure 9 – Sequence of the formation of the consortium contract

4. CLUSTER AND BROKER INTERACTIONS

The most important global behaviours of the proposed architecture can be summarised as: 1) cluster registering, 2) new consortium formation, 3) consortium alteration, 4) service execution, and 5) contract termination. Cluster registering is the global behaviour of registering an agent in the cluster. This has been already discussed when the cluster contract was mentioned. The service execution behaviour is the action related to the execution of requests from higher levels and how they can be decomposed into lower level requests. Finally, the contract termination behaviour includes all the aspects related to contract termination and the breach of contract. For lack of space these two behaviours will not be described here. Consortium formation and change involves the process of creating a new consortium or the

changes that can be done to it, either by removing or adding members. These behaviours will be the focus of this section.

Figure 10 shows the activity diagram, with the boundaries where each state occurs, for the process of creating a new consortium.

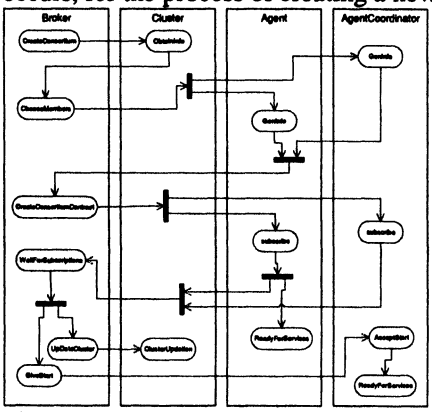

Figure 10 – The activity diagram of the process of creating a new consortium

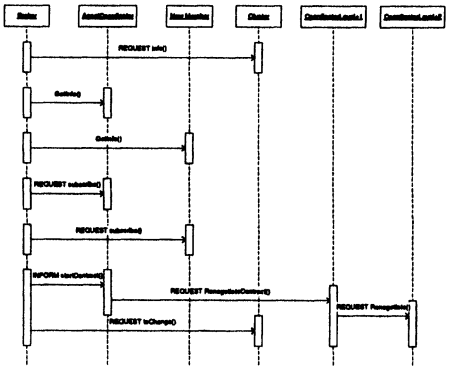

Figure 11 – The activity diagram of the process of changing a consortium

The process of changing a consortium is very similar to the creation of a new one thanks to the architecture of the generic agent. The only difference is found in the re-negotiation behaviour that must run in the co-ordinator of the consortium that is going to be changed. Only after that the co-ordinator is ready to accept services. This re-negotiation is important because in the case of a removal of an agent the consortium can be on a situation that it is no longer capable of honouring the promises made to the higher-level consortium it is a member of. To keep the system working properly it is necessary to update the promise on the higher-level consortium contract.

Figure 11 shows a simplified sequence diagram for the change of a consortium and it also indicates the different actors involved when a consortium is changed.

5. EXPERIMENTAL DEVELOPMENTS

5.1 Agent Platform

The JADE – Java Agent Development framework (JADE 2001) was chosen for the experimental work mainly because it is an open source FIPA compliant platform, provides good documentation and support, and it is also recommended by the experience of other research groups with whom the authors have close relationship. The use of *Behaviours* supplied by JADE, and the easy connection to JESS (Friedman-Hill 1999) helped in reducing the programming effort. Moreover JADE, implements the FIPA-ACL agent communication language. Another interesting feature of JADE is the functionalities provided to manage the community of agents. It includes a *Remote Monitoring Agent* (RMA) tool, which is used to control the life cycle of the agent platform, and an agent for *white pages* and life cycle services (Agent Management Service - AMS).

In figure 12 (left side) the JADE monitoring tool shows the three example agents of the architecture. The generic agent address is da0@pc-3:1099/JADE. Although all agents were running in the same platform pc-3, this is not at all mandatory. The right side of figure 12 shows the sequence of messages between the cluster and a generic agent. This specific case shows the registering sequence in the cluster of two generic agents. The informative of the message can also be seen and this sequence can be compared with the protocol of figure 9.

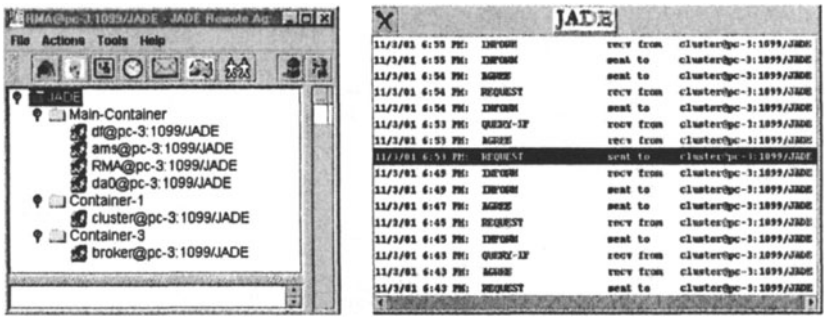

Figure 12 – JADE Monitoring tool and Messages between the Cluster and the Generic Agent

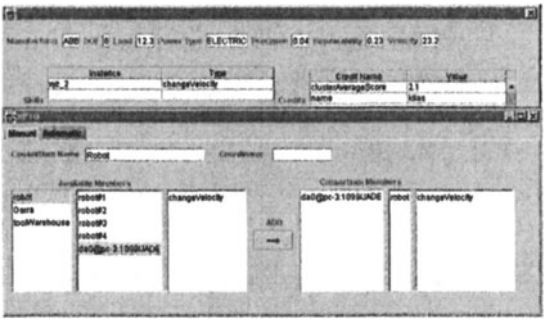

Figure 13 – Broker GUI

The lower part of figure 13 is the basic GUI of the broker. When the user chooses a candidate by selecting it (middle column of available members), the broker asks that agent and then the agent replies with its basic information to the broker, which then pops-up a window showing the basic parameters of that agent (upper window). The figure shows that the cluster has three types of manufacturing components: robots, gripper, and tool warehouse (left column). The middle column shows the agents of that type registered in the cluster. The right column shows the skills.

Due to space restrictions it is not possible to include a detailed presentation of the experimental results already achieved. This will be done in a follow-on paper.

5. 2 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 agent machine interface (AMI) accesses the wrapper using a local software interface (proxy), where all the services of the wrapper are defined. Figure 14 shows a high level representation for an operative agent indicating how the wrapper integrates a manufacturing component (robot).

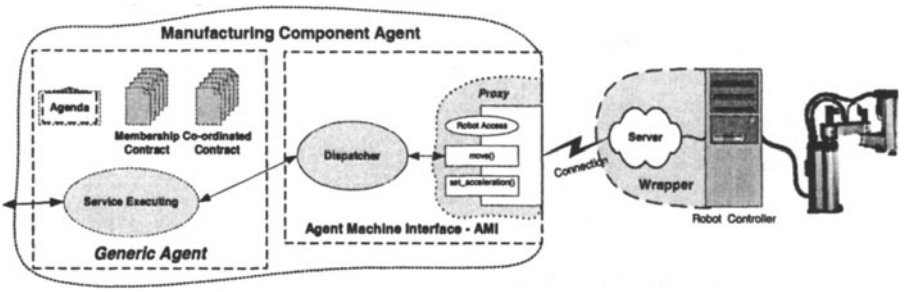

Figure 14 – Physical Component Integration

In previous works, the wrapper used to integrate physical components during the agentification process has been successfully implemented using two-tier client-server architecture (Barata, et al., 1996; Camarinha-Matos, et al., 1996; Camarinha-Matos, et al., 1997). Nevertheless, this architecture can have an important drawback, which is its limited flexibility in moving program functionality from one server to another (Schussel, 95). Three-tier architectures have emerged as a solution to this limitation, namely three-tier with ORB (Object Request Broker) architecture. The current big players in this arena are the CORBA (Common Object Request Broker Architecture), DCOM (Distributed Component Object Model), and the JRMI (Java Remote Method Invocation). Recently the wrappers for our manufacturing system were developed using DCOM 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.

Figure 14 helps to clarify the statement that a manufacturing agent is a generic agent tied to an Agent Machine Interface. Each AMI must implement the services supported by the physical component and it was also developed to be generic, i.e., an AMI can be connected to different distributed component (proxy) just by configuring what are the services of that proxy and the name/address of the component. The generic agent tied to the AMI behaves slightly different from other agents, at the initialisation phase. The former type reads from a contract representation file an instance of a consortium contract between itself and the AMI, and establishes a consortium. The member promise part (AMI) of the contract contains all the services supplied by the AMI. The agents not connected to an AMI, on the other hand, are configured not to read any contract representation file at initialisation time. This approach is very flexible because it permits to create (generate) any type of manufacturing agent just by configuring an AMI and the

consortium contract between the agent and the AMI. The only part of the system that is dependent of the physical component is of course the wrapper.

6. CONCLUSIONS

This paper has introduced a new control architecture founded on the concepts of cluster and consortium regulated by contracts. The advantages of the approach to support the shop-floor life cycle can be summarised in the following:

1. **Flexibility/Agility** – Separating the aspects purely related to agent competence (skills) from the aspects related to co-operation (regulated by contract) and co-ordination facilitates the development of much more agile structures. It is preferable to have an entity (agents) with generic capabilities (skills) and whose tasks are regulated by a contract than an entity with fixed behaviour (pre-programmed). Consortia and contracts may allow creating different structures with different objectives, with the same agents.
2. **Adaptability** – Changes made to the capability of an agent or consortium may be propagated to the rest of the architecture almost automatically or via simple re-configuration of contract clauses.
3. **Integration** – Adding new agents to the architecture becomes easier because there is no need for reprogramming the other components.
4. **Configuration** – Physical changes are essentially made based on configurations done in clusters, consortia and/or contracts.

The main behaviours associated to the cluster, broker, and generic agent concepts were described and explained. The importance of the generic agent to create manufacturing agents as well as co-operating consortia was also stressed. A first implementation of contract models and the various types of agents shows promising results towards a highly re-configurable control infrastructure within a real complex manufacturing environment. Further work is necessary on cluster management services, dynamic consortium formation and modification, and contract configuration / specification tools.

7. REFERENCES

1. Almeida, C. F.. (2000). *Contratos I – Conceitos, Fontes, Formação.* Almedina. Coimbra
2. Barata, J., W. Vieira. and L.M. Camarinha-Matos. (1996) Integration and MultiAgent Supervision of Flexible Manufacturing Systems. In: *Proceedings of Mechatronics'96 - The 5th UK Mechatronics Forum International Conference.* 18-20 Sep. Guimarães: Portugal.
3. Camarinha-Matos, L.M., L. S. Lopes and J. Barata (1996) Integration and Learning in Supervision of Flexible Assembly System. *IEEE Transactions on Robotics and Automation (special issue on Assembly and Task Planning),* **12**, 202-219.
4. Camarinha-Matos, L.M., J. Barata. and L. Flores (1997). Shopfloor Integration and Multiagent based Supervision. In: *Proceedings of INES'97 - IEEE International Conference on Intelligent Engineering Systems 1997,* ISBN 0-7803-3627-5, pp 457-462, 15-17 Sep. Budapest: Hungary.
5. Friedman-Hill, E.J. (1999), *Jess, The Java Expert System Shell,* Sandia National Laboratories, Livermore, CA. (http: //herzberg1.ca.sandia.gov/jess/).
6. JADE Web Site (2001), (http://sharon.cselt.it/projects/jade/).
7. Protégé Web Site (2001). (http://protégé.stanford.edu).
8. Schussel, George (1995). Client/Server Past, Present, and Future (http://news.dci.com/geos/dbsejava.htm).