AllianceNet: Information Sharing, Negotiation and Decision-Making for Distributed Organizations

Jean Marc Andreoli[1], Stefania Castellani[1], and Manuel Munier*[2]

[1] Xerox Research Centre Europe, Grenoble Laboratory, F-38240 Meylan, France,
{andreoli,castella}@xrce.xerox.com
[2] IUT des Pays de l'Adour, 40004 Mont de Marsan, France,
munier@marsan.univ-pau.fr

Abstract. We explore issues in providing support for information sharing, negotiations and decision-making to distributed autonomous organizations, grouped in alliances to improve their own ability to accomplish customers' requests. In particular, we consider the case of an alliance of printshops offering similar and/or complementary print competencies and capabilities, competing but also collaborating with each other to perform print jobs.
We present a typical scenario of the activities within such an alliance, where the main task of the printshop managers is to schedule their portfolio of jobs. We then introduce a multi-agent architecture, called *AllianceNet*, allowing a manager to flexibly negotiate with the allied printshops some jobs that s/he cannot or does not wish to perform locally. The purpose of the agents in AllianceNet is not to replace the printshop managers, but rather to assist them in the decision process by making available the information needed in the negotiations and by automating the tasks implementing the committed decisions. In particular, we discuss the kind of information used and shared among printshops, the support offered to printshop managers to make informed decisions and to consistently enact and monitor their execution.

Keywords: Negotiation, Protocols, Decentralized systems, Agent models and architecture, "Business-to-business" electronic commerce.

1 Introduction

We explore issues in providing support for information sharing, negotiations and decision-making to distributed autonomous organizations grouped in alliances. We consider here alliances of organizations offering similar and/or complementary competencies and capabilities, competing but also collaborating with each other to improve their own ability to accomplish customers' requests. In particular, we are interested in an alliance of printshops executing print jobs (simply

* Work performed while visiting the Xerox Research Centre Europe in Grenoble

called jobs in the sequel). Each printshop may act sometimes as an "outsourcing" entity, submitting job requests to other printshops in the alliance, and sometimes as an "insourcing" entity, accepting such requests. In fact, the interactions we consider between the printshops are very general "business-to-business" interactions, so that our approach applies to any alliance of organizations, whatever their domain (the case of printshops has been chosen mainly because of its close links to Xerox core business).

The collaborations within an alliance can be partially formalized and automated as workflows, but they cannot be satisfactorily modeled by simple activity diagrams with rigid dependencies defining only synchronizations and ordering between "blackbox"-like activities. More realistically, the printshop managers should be flexibly supported in scheduling and negotiating their portfolio of jobs. For example, a printshop manager may wish to outsource a job and then select, among the printshops making insourcing offers, those providing the best cost/color-quality performance ratio. Also, the manager of an insourcing printshop may need to re-negotiate with the outsourcing printshop the commitment for a job, e.g. for changing a deadline. To be successful, an information technology tool supporting such an alliance should satisfy two constraints: it should be non-disruptive, i.e. respect the actual work practice, and at the same time it should create new opportunities.

Printshops in an alliance are fully autonomous organizations and, as such, each of them is responsible for managing its own jobs and resources. This precludes a straightforward approach to the management of the alliance, in which each partner is requested to declare to the alliance all its available resources (both human and machines), and the alliance handles all the customer requests, splitting and dispatching them in an optimal way among the different partners, using, for example, planning and job-shop scheduling techniques. Indeed, this highly centralized approach, based on a "super-scheduler", is not adapted to the situation we consider for several reasons: (*i*) given the competitive context, the printshop managers are unlikely to give up control over their job portfolio and their resources; (*ii*) many decisions on how to best manage the jobs in a printshop must take into account information that can only be provided by the people working in that printshop itself: this information comes from their experience and the printshop local interests and is difficult to formalize and integrate in a super-scheduler; (*iii*) a "super-scheduler" solution in a distributed context is usually difficult to scale-up and to evolve dynamically.

Sec. 2 describes in more details the activities of a single printshop and its interactions with the alliance. Sec. 3 introduces a multi-agent architecture, called *AllianceNet*, supporting the negotiations occurring in the scenario previously described. In particular, we discuss the kind of information used and shared among printshops, the support offered to printshop managers to make informed decisions and to consistently enact and monitor their execution. Sec. 4 describes our current directions of investigations.

2 A Printshop Alliance Scenario

The following scenario is a simplified, though non-trivial description of the activities in a printshop including its interactions with the customers and possibly with other printshops. Building upon the description of a variety of print work processes (in [6] and [4]), we have defined a model for printshop activities which could be modified to encompass/exclude some activities and refine role assignments, but which attempts to model significant kinds of behaviors.

A printshop may receive print requests from both customers and other printshops (outsourcing requests in the latter case). Conversely, the printshop manager may wish to outsource some of her/his jobs, totally or in part, to the alliance. When a print request reaches the printshop with given parameters (e.g. deadline), a first estimation is established. The manager analyses the job description to understand how it can be accomplished, taking into account the current job schedule, the availability of the resources, and trying to optimize the global cost. Existing ad-hoc scheduling tools can be used here. Based on the results of this evaluation, the manager decides either to reject or to accept the print request. In the former case, the negative decision is communicated to the requester. In the latter case, the job has to be allocated. If the current schedule of the printshop allows inclusion of the new job, the manager may decide to perform it locally. However, it may be possible that the job cannot be locally performed (at least not as a block), given the requirements, the printshop resource availability and technical capabilities. For example, if the request includes a color print and the printshop has only black and white printers, then at least the part requiring a color printer should be outsourced. Moreover, even if the execution of the job is consistent with the printshop schedule and equipment, the manager might still decide to outsource (part of) the job, for example, in order to save some of the available resources for a job currently under negotiation with a major customer.

If the manager decides that (part of) the job has to be performed remotely, (s)he will start a negotiation with the partner printshops. The outcome of a negotiation can be "success" (the job was fully outsourced), "failure" (no outsourcing agreement could be reached) or "partial" (only part of the job could be outsourced). An elementary negotiation scheme relies on an "invitation to tender". The manager decides if and how to split the job into slots and notifies the other printshops in the alliance about the outsourcing requests for the different slots. The manager collects quotations from partner printshops, evaluates them and chooses a solution. The outsourced job (or slots) is (are) then sent to the selected insourcer(s). If no "good" solution is found, the manager may accept a sub-optimal offer anyway (and possibly face delays), or re-allocate local resources in order to perform the job locally, or revise the splitting of the job. In any case, the process of choosing a solution is far from trivial and we do not try here to automate it (using so called "intelligent" agents). Once a solution is adopted, it must be implemented and monitored.

The main requirement for the architecture is that it must offer a lot of flexibility in the negotiations occurring in the scenario described above. The manager of a printshop is responsible for issuing the quotations and for allocating the jobs

of that printshop. So, (s)he needs to make informed decisions based upon the estimations, the job schedule of the printshop and the knowledge (s)he has about the other printshops technical capabilities and actual resources availabilities.

The manager should be able to negotiate jobs in several ways, and choose a negotiation model on a case by case basis. For example, if a job cannot be performed as a unique block, the manager must split the job and make outsourcing requests of (some of) the pieces. The splitting may be decided a priori, on the basis of the structure of the job, and thus entirely precede the submission of the outsourcing requests, but, more realistically, the two processes have to occur in parallel, the splitting being revised as potential insourcers produce offers.

The manager may also need to re-negotiate the commitment for a job, e.g. when a customer changes requirements and the manager has already committed with an insourcing printshop, or when an insourcing printshop is unable to respect a deadline.

3 An Architecture for AllianceNet

The architecture described here mainly focus on the distributed negotiation aspects of the previous scenario.

3.1 The Agent Infrastructure

We consider a distributed architecture (Figure 1) where each printshop is a site at a node of the network, allowing collaboration among them. The alliance itself may have resources of its own, to store the state of the negotiations, which may be dispatched on some of the partners' sites, or at some distinct alliance specific sites. Other partners in the alliance, which are not printshops but offer complementary services, are also represented as different sites.

Several kinds of existing tools can support a printshop manager when issuing job quotation and managing the job schedules. For example, in a UK commercial printshop, the manager and her/his collaborators make use of a forward loading board, as reported in the case study in [4]. Another possibility could be to adopt a simulation tool like Zippin [2], that allows to simulate job schedules with alternative configurations for jobs and resources and to evaluate benefits and drawbacks. In all cases, the architecture must be able to integrate existing tools which may not have been developed according to its own model. A coordination infrastructure, with wrapping capabilities, is thus needed.

In the case study cited above, negotiations among printshops are performed by non computer supported means, e.g. through telephone calls between printshop managers. On the contrary, our architecture seeks to provide the managers with a computer support for flexible negotiations. The idea is to benefit from the distributed setting to create a computer supported "market" inside the alliance. In *AllianceNet*, this market is modeled by business objects and business rules implemented on an agent based virtual enterprise development platform called

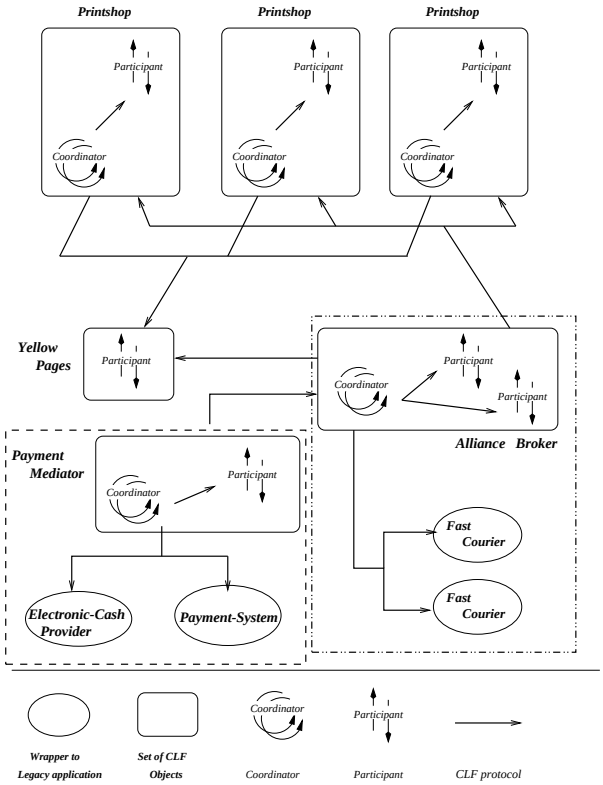

Fig. 1. Overall architecture

CLF [1]. CLF offers two major features that are adapted to our needs: a library of business objects and a portable scripting language.

CLF (and in particular its subsystem called Mekano [1]) offers a library of ready-made customizable business objects, as well as facilities to devise new objects without starting from scratch. These new objects can then be integrated into Mekano for later re-use. Compared to other platforms with similar goals, such as Jini, Enterprise Java Beans or Corba, the main characteristic of CLF is that it is built around a rich object model ("objects as resource managers") in which basic features enabling negotiation are accounted for at the lowest interaction protocol level, through eight interaction "verbs" (*a la* KQML) similar to speech acts as found in many agent models. There are two main classes of verbs, allowing respectively to search for distributed resources and to consistently enact distributed resource manipulations. The sequencing of the verb occurrences must conform to a correctness criterion, and a state diagram captures the possible states of the participants in an interaction and their transitions (a classical technique to specify negotiation protocols, e.g. see [7, 9]). The complexity of the

diagram is hidden to the programmer by a set of tools provided by the Mekano library.

We make use of the following CLF objects. For each printshop, a CLF *Printshop* object (new to Mekano) manages job descriptions and time slots, held as resources. A general description of each partner printshop in the alliance is made available through a CLF *Yellow pages* object (customized from an existing Mekano component), which is an alliance-level object. It is thus possible to search for and establish connections with any printshop in the alliance on the basis of its capabilities (e.g. its equipments). Dynamic registration or deregistration of printshops in/from the alliance is directly accounted for by the CLF object model and does not require special treatment. CLF "Fast Delivery" objects offer distributed delivery services and a CLF "Payment Mediator" object provides payment facilities [1]. These two kinds of objects are provided by Mekano (but not yet integrated in our prototype).

3.2 Negotiation using CLF Scripts

One of the most salient feature of CLF is its powerful coordination facilities, provided by a highly portable scripting language, adapted to describe business rules. Interpreters of this language are CLF objects called coordinators which manipulate rule-based scripts as resources, allowing reflexivity and a lot of flexibility in the organization of the interactions between the partners of the alliance (for outsourcing, insourcing, job splitting and services combinations, e.g. printing and delivering). We outline here the scripts used in our prototype to support some forms of negotiation. For lack of space, we cannot describe in detail the general behavior of CLF scripts. The interested reader is referred to [1]. Basically, CLF scripts are made of rules which have a purely declarative interpretation in terms of resource manipulations (see the examples below), together with a corresponding operational interpretation that makes use of the resource oriented primitives offered by the CLF protocol (the eight verbs). Not surprisingly, rules have often been used for the flexible coordination of tasks in workflow management or transactions in federated databases.

For example, a simple outsourcing mechanism is implemented in CLF by the following rule:

```
JobRequest(job) @ Partner(job,dest) @
Offer(dest, job, offer) @ Accept(job, dest, offer)
   <>- OutJob(job, dest, offer)
```

This rule asynchronously builds a search-tree of all the possibilities to outsource a job request, and enacts one of them. The left-hand side of the rule (left of `<>-`) specifies the resources needed (in combination) to achieve outsourcing: (*i*) a job to outsource: `JobRequest` provided by a *PrintShop* agent, (*ii*) a partner printshop that could potentially insource the job: `Partner` provided by the *Yellow pages* agent, (*iii*) an offer: `Offer` made by that partner for that job, and (*iv*) an acceptance of that offer: `Accept` generated by the partner who initiated the outsourcing. All these resources are searched in their respective agents using the

search capabilities of the CLF protocol. During this search phase, the agents may asynchronously provide an unbounded number of resources to match the tokens in the rule: for example, the *Yellow page* agent may return a stream of potential partners that are a-priori suitable for the job, and may asynchronously fill the stream as new partners join the alliance or change their description. Hence, the execution of the rule builds a search tree, with new branches being created each time a new matching resource is found. When one branch is complete, the enactment capabilities of the CLF protocol are used to ensure the atomic consumption of the resources. Each agent is asked to reserve (if possible) the resource it returned in this branch of the search phase, and if all the reservations succeed, they are all confirmed, otherwise the transaction (for that branch) is aborted, but the other branches are still concurrently active. This allows to account for missing resources, which are available at search time but not anymore at enactment time, e.g. a job request which was finally allocated locally instead of outsourced, or an offer which relied on a machine which then became out-of-order, or a printshop that retracted from the alliance, etc. Other verbs of the CLF protocol are used to propagate failure information in the construction of the search tree (and thus avoid developing branches that are doomed to fail), and to notify the successful enactment of a branch (insertion of the `OutJob` resource on the right-hand side of the rule).

A printshop manager may also wish to split jobs into slots and outsource them separately. This could be done by a new rule for the splitting and the rule above for the outsourcing. However, that would force the decision to split or not to be taken a-priori. Instead, a manager may wish to try at the same time to outsource the job as one block and by pieces, making sure of course that in the end, only one of the two solutions is actually adopted. This is realized by the following CLF script, which fully exploits the transactional facilities of the CLF protocol.

```
JobRequest(job) @ SplitJob(job, part1, part2) @
Partner(part1, dest1) @ Offer(dest1, part1, offer1) @
Partner(part2, dest2) @ Offer(dest2, part2, offer2) @
Accept(part1, dest1, offer1) @ Accept(part2, dest2, offer2)
  <>- OutJob(part1, dest1, offer1) @ OutJob(part2, dest2, offer2)
```

This rule can (and will) safely run in parallel with the previous rule for the non-splitting case. Indeed, the splitting rule will be effectively enacted for a given job only if insourcers for both pieces of the split job are found and accepted by the outsourcer. If the splitting rule is finally enacted, the `JobRequest` resource is removed, thus disabling the non-splitting rule for that job (and vice-versa). Also, several resources may match the `SplitJob` token in the rule, corresponding to different ways to split the job, e.g. by the bulk, or according to the structure of the job (coversheet vs content or color pictures vs black and white text). All these possibilities will be explored in parallel in the construction of the search tree, but again, in the end, at most one possibility will be effectively enacted.

Other mechanisms for outsourcing jobs can as easily be implemented using CLF scripts. For example, the "Dutch auction" mechanism can be captured by a slight modification of the above scripts. Consider the case without splitting.

In a Dutch auction session, the outsourcer publishes the information concerning the job to be outsourced, as in the previous case, but instead of waiting for insourcing offers, it also publishes a proposed bargain for the job. Potential insourcers may then accept the bargain as such, and the first one to do so gets it. If the bargain is not taken by anyone, the outsourcer may then revise it and propose a new bargain (eventually with more appealing conditions). The following script implements such a behavior.

```
JobRequestDA(job,bargain) @ Partner(job,dest) @
@ AcceptDA(job, dest, bargain)
   <>- OutJob(job, dest, offer)
```

Here, the resources involved are: (*i*) a job to outsource under the Dutch auction mechanism, together with a bargain: `JobRequestDA` provided by a *PrintShop* agent, (*ii*) a partner printshop that could potentially insource the job: `Partner` provided by the *Yellow pages* agent, (*iii*) an acceptance of the bargain attached to the initial request: `AcceptDA` generated by the partner who potentially wishes to insource the job.

The CLF scripts given above can be refined, customized at each partner site. That can even be done dynamically while the system is running, using the reflective features of CLF that allow to manipulate scripts as resources. It is thus very easy to de-activate a rule and activate a new, replacement rule. For instance, the splitting rule could thus be dynamically replaced by a variant in which a control token is inserted in the left-hand side, making some basic automatic cross-checks on the offers made by the partners. Again, the resource-oriented transactional semantics of the CLF ensures that the de-activation of a rule does not generate inconsistencies: de-activation of a rule is treated as a missing resource.

Rules can also be combined in various ways, to achieve more complex behaviors, e.g. heterogeneous splitting where a job is split and one part is outsourced under the Dutch auction mechanism while the other part is outsourced by the usual mechanism; or arbitrary joining of jobs, where two, *a priori* independent jobs are joined together to be considered as one outsourcing job request.

3.3 The Printshop Manager Interface

Figure 2 shows the interface for a printshop manager in the alliance. It shows the local jobs ("List of my jobs") and the job requests issued by remote printshops ("Job requests from partners"). Jobs that are intended to be done locally or jobs for which job requests have not yet been defined, are shown as type "J" in the "List of my jobs". Jobs for which the manager has issued a job request are shown as type "JR". Upon selection of a job from the list, the interface displays details about that job.

In the case of a "Job request" (type `JR`), the detailed view includes the list of the offers for that request (if any) made by the alliance partners. For example, in Figure 2 the job `job01` has been selected and the corresponding detailed view has been displayed. This view allows the manager to: (1) accept an offer; or (2) retrieve the job; or (3) split it.

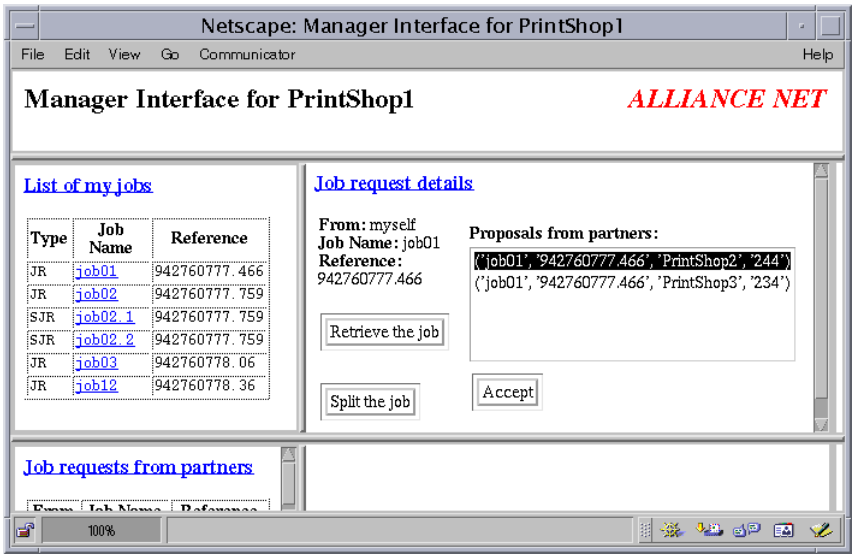

Fig. 2. The User Interface: Viewing partners' proposals for an outsourced job

If the manager *accepts* an offer, the job is assigned to the partner who made that offer (the job entry in the "List of my jobs" view is removed and the job appears in the partner's "List of my jobs" view). If the manager *retrieves* the job then the corresponding job request is removed and the job becomes a local job (the job status in the "List of my jobs" changes to "J"). Finally, if the manager *splits* the job in two slots, two new corresponding requests are created (see `job02` in Figure 2). The non-split job request is not removed, so that the manager can still choose between splitting or non-splitting offers. Of course, if one of the two alternatives is accepted and succeeds, the other one disappears from the view. In that case, if the manager mistakenly accepts the other alternative before it disappears (which may happen given the asynchronous infrastructure), (s)he will anyway be notified of the error since the transaction will abort (the job request is already consumed).

In the case of a local job (type `J`), the detailed view shows a description of the selected job and allows the manager to turn it into a job request for outsourcing.

Finally, the "Job request from partners" view shows information about the job requests sent by the other printshops in the alliance (e.g. the printshop that made the request). Upon selection of a remote request from this list, a detailed view of the job is displayed that allows the manager to: (1) see the offers (s)he already made (if any) for that request (but not proposals from other partners, thus preserving confidentiality); (2) make an offer or a new offer; and (3) delete a previous offer.

4 Perspectives

The basic negotiation mechanism illustrated in the previous section is essentially an extension of the Contract Net protocol [12] (or similar mechanisms such as the Dutch Auction) with transactional facilities, which enable the coordinated execution of a collection of concurrent, interdependent Contract Nets (e.g., in the case of splitting, there are Contract Nets for the job request as a whole and for the pieces).

However, each Contract Net, in the system presented here, is rather rudimentary, in that the end decision in the protocol is made on the basis of the offers that have been received, with a priori no possibility to request revisions of the offers (e.g. to make counter-offers, as in [8]), except of course by outside means, such as direct (e.g. phone) conversations between the printshop managers. The first step to overcome this limitation is to refine the structure of the offers and replace them by *negotiation objects*. The most simple negotiation object is a price: an outsourcing request specifies the description of a job and the insourcing offers specify a price. But negotiation objects may be more sophisticated. They may include refinements of the specification wherever it was left free (e.g. color quality or price range), thus allowing unbounded chains of successive refinements. The refinement process is multi-phase and based, again, on speech acts [5]. At each stage, the negotiation may progress in two ways: either by refining the negotiation object, or by simply giving it up and replacing it by several alternative negotiation objects (e.g. black-and-white in two days or color in three) which may then be negotiated concurrently. Thus, we still have a search tree, developed asynchronously, whose branches represent the different ways to proceed. The transactional semantics ensures that only one solution will be selected at the end. Inactive branches of negotiation may be saved, so as to be re-activated in case of re-negotiation. Thus, we move from a uni-directional "announce/collect/decide" paradigm to a multi-directional "announce/refine/decide" paradigm.

Note that the refinement process for negotiation objects is similar to propagation in distributed constraint satisfaction [13, 11]. No deep assumption is made here about the nature of the propagated information. In usual DCSP, it may be choices of value or no-goods, propagated according to a static or dynamic prioritization on the agents [3]. Here, the propagated information, held by the negotiation objects, is defined in a negotiation *language*, known to all the partners in the alliance, e.g. capable of constraining prices, print quality, deadlines, delivering conditions, etc. The negotiation object could also include a history of its evolution, and not only static attribute-value informations. This would allow to constrain not only the values of the attributes concerning the job at hand, but also the ordering in which decisions about that job have been taken. Finally, the ordering of the propagations itself could be constrained by some negotiation *protocols* known to all the partners in the alliance (e.g. turn-taking or master-slave propagation; see [10] for a detailed investigation of these schemas), and which could themselves be negotiated.

5 Conclusion

In this paper, we have presented an infrastructure providing support for information sharing, negotiations and decision-making to distributed autonomous organizations, grouped in alliances. Flexibility, an absolute requirement in this context, is achieved by combining techniques coming mainly from three different domains: (*i*) multi-agent systems, (*ii*) (relaxed) transaction models and workflows, (*iii*) (distributed) constraint satisfaction.

Acknowledgement We are grateful to François Pacull, Jean-Luc Meunier and Christer Fernstrom for helpful comments on this paper.

References

1. J-M. Andreoli, D. Arregui, F. Pacull, M. Riviere, J-Y. Vion-Dury, and J. Willamowski. CLF/Mekano: a framework for building virtual-enterprise applications. In *Proc. of EDOC'99*, Manheim, Germany, 1999.
2. J-M. Andreoli, S. Castellani, U. Borghoff, R. Pareschi, and G. Teege. Agent-based decision support for managing print tasks. In *Proc. of PAAM'98*, London, U.K., 1998.
3. A. Armstrong and E. Durfee. Dynamic prioritization of complex agents in distributed constraint satisfaction problems. In *Proc. of IJCAI'97*, 1997.
4. G. Button and W. Sharrock. The production of order and the order of production. In *Proc. of ECSCW'97*, Lancaster, U.K., 1997.
5. M. Chang and C. Woo. A speech act based negotiation protocol: Design, implementation and test use. *ACM Transactions on Information Systems*, 12(4):360–382, 1994.
6. C. Eliezer and D. Zwang. Print production workflow: Defining the issues. *Patricia Seybold Report on Publishing Systems*, 27(3), 1997.
7. J-L. Koning, G. Francois, and Y. Demazeau. An approach for designing negotiation protocols in a multi-agent system. In *Proc. of IFIP'98*, pages 333–346, Wien, Austria, 1998.
8. C. Koo. A commitment-based communication model for distributed office environments. In *Proc. of the Conference on Office Information System*, New-York, NY, U.S.A., 1988.
9. S. McConnell. Negotiation facility, 1999. OMG final revised submission for a Corba service.
10. M. Munier. *Une Architecture pour Intégrer des Composants de Controle de la Coopération dans un Atelier Distribué.* PhD thesis, Université Henri Poincaré, Nancy, France, 1999.
11. C. Petrie, H. Jeon, and M. Cutkosky. Combining constraint propagation and backtracking for distributed engineering. In *Proc. of the AAAI'97 workshop on Constraints and Agents*, Providence, RI, U.S.A., 1997.
12. R.G. Smith. The contract net protocol: High level communication and control in a distributed problem solver. *IEEE Transactions on Computing*, 29(12):1104–1113, 1980.
13. M. Yokoo, E. Durfee, T. Ishida, and K. Kuwabara. Distributed constraint satisfaction for formalizing distributed problem solving. In *Proc. of the 12th Int'l Conference on Distributed Computing Systems*, pages 614–621, 1992.