Self-Regulation in Self-Organising Multi-Agent Systems for Adaptive and Intelligent Manufacturing Control

Gaël Clair, Elsy Kaddoum, Marie-Pierre Gleizes, Gauthier Picard

Self-Regulation in Self-Organising Multi-Agent Systems for Adaptive and Intelligent Manufacturing Control

Gaël Clair[†], Elsy Kaddoum[‡], Marie-Pierre Gleizes[‡], Gauthier Picard[†]
[†] *SMA@G2I, École Nationale Supérieure des Mines de Saint-Étienne, France*
{clair,picard}@emse.fr
[‡] *IRIT, Université de Toulouse III, France*
{gleizes,kaddoum}@irit.fr

Abstract

In this paper, we explore the potential of distributed satisfaction techniques as to provide self-regulated manufacturing control. This work relies on a DisCSP-based modeling distributed among agents (e.g. machines) having enough and reasoning capabilities to cooperate and negotiate for a committed schedule. This approach is used to dynamically regulate the system (the network of machines) when perturbations occur (machine break-out, operator or container unavailability, or even priority command). Thus, for these machines, embodied intelligence and autonomy are a mean to provide a more flexible and adaptive manufacturing network. In this paper, we present two different multi-agent models and two extensions of well-known DisCSP solvers. Experiments using a dedicated simulation platform, MASC, are presented and discussed.

1. Introduction

The concept of Intelligent Factory has increased in influence these past years, and represents one of the scientific European priorities (see MANUFUTURE[1] technology platform, for instance). Nowadays, this concept is integrated at all the levels, from the design of production systems to and the deployment of logistics systems, from the management of supply chain to the deliverance of finalised products. Its objective is clear: producing faster, with a better accuracy, with minimum cost, and with more flexibility. Building such factories implies using new Information and Communication Technologies (ICT) and becomes notably possible thanks to new computational paradigms such as ubiquitous computing, pervasive systemics, or ambient intelligence. In fact, numerous technologies, supporting this development, have been the focus of several projects and technology platforms (e.g. ARTEMIS[2], EPoSS[3]) and become enough mature (e.g. Sunspot, RFID and dedicated middlewares). As to design such factories, in an ambient intelligence context, multi-agent systems seems to be a promising approach; this context being strongly distributed and dynamic. More precisely, self-organising multi-agent systems, able of self-regulation consecutive to exogenous perturbations or endogenous dysfunctions, can be used to answer to these non functional requirements. This is in this context that we propose to use adaptive multi-agent systems (AMAS) [6], based on DisCSP modelling (*Distributed Constraint Satisfaction Problem*) [16], to develop an intelligent and adaptive distributed manufacturing control system.

Section 2 expounds the DisCSP framework, and the multi-agent approaches to tackle these problems. Section 3 describes the manufacturing control problem. Sections 4 and 5 present the multi-agent models and the distributed algorithms to solve task allocation problems for manufacturing control. Section 6 shows results and analysis from simulations on the MASC platform, and some comparisons with classical algorithms for solving DisCSP. These results are discussed in section 7 before concluding in section 8.

2. DisCSP and Multi-Agent Systems

Constraint Satisfaction Problems (CSP) framework is a means to model problem to solve in Artificial Intelligence or Operations Research. It consists in a set of *variables* to assign ($\mathcal{X} = \{x_1, \ldots, x_n\}$), which take values that are in specific *domains* ($\mathcal{D} = \{D(x_1), \ldots, D(x_n)\}$) and that are restricted by several *constraints* ($C = \{c_1, \ldots, x_k\}$).

Numerous algorithms exist to solve this kind of problems, having to trade off between completeness and efficiency [4]. As an extension to this framework, Distributed CSP (DisCSP) framework proposes to spread the decision

[1] www.manufuture.org/

[2] www.artemis-office.org

[3] www.smart-systems-integration.org

for assigning variables to a set of agents [16]. Several works propose distributed and asynchronous extensions of classical algorithms (backtracking, breakout, etc.), or more original solutions, such ERA, which utilises situated agents, interacting via a shared environment, to solve combinatorial problems [8].

Nevertheless, facing dynamic environments (constraint changes, adding or deleting variables), these algorithms may not be adequate; either because they rely on constructive algorithms (every variable has not necessarily a value at each time *t*) or because they extend *anytime* algorithms having difficulties to answer to dynamic perturbations. One answer to this dynamic is the solution repairing by using constraint propagation, which is generally performed by centralised solutions as sub-procedures [4].

Distributing constraint propagation can be viewed as a self-regulating mechanism, as soon as agents are autonomous concerning their decision to propagate constraints, with their only acquaintances being the agents situated in their direct neighbourhood. Such self-organising solutions have been applied with success to classical CSPs such as frequency assignment problem [12, 11] or to continuous problems such as preliminary aircraft design [14].

3. Manufacturing Control Problems

3.1. Description

"*Production planning and control is concerned with the application of scientific methodologies to problems faced by production management, where materials and/or other items, flowing within a manufacturing system, are combined and converted in an organised manner to add value in accordance with the production control policies of management*" [15]. Production planning and control can be decomposed into several elements. Our study concerns one of them, the manufacturing control or production activity control, defined by [5] as "*principles and techniques used by the management to plan in the short term, control and evaluate the production activities of the manufacturing organisation*". To sum up, manufacturing control is a means to plan the use of resources for production by respecting predefined constraints (delivery delays, operators schedule, etc.). All this is deployed in a dynamic environment, in which unpredictable perturbations may occur (breakdowns, strikes, sick leave, etc.), generating new constraints at runtime. Multi-agent approaches dedicated to tackle this problem exist and essentially focus on modelling (e.g. using holonic infrastructure) or methodology, and not necessarily on solving mechanisms, which are often delegated to Operations Research [1, 9]. Nevertheless, some approaches are based on self-organising mechanisms like stigmergy [7].

3.2. MASC Platform

In the framework of the Colline[4] working group of the French Association for Artificial Intelligence (AFIA), a simulation platform for manufacturing control has been developed. The goal of this platform is to encourage different research groups to propose and implement their multi-agent solutions, as to compare them on common scenarios. Here, the issue is three-fold:

- implement new algorithms,
- define new scenarios for comparison,
- define new metrics and criteria for comparison.

This working group essentially interests in the implementation of techniques for self-adaptation based on multi-agent systems, as to compare and characterise them. It is also interesting to compare these different approaches to more classical ones such as ABT or AWCS [16], which seem adequate distributed solutions for manufacturing control, expressed as a constraint-based problem. MASC platform concerns manufacturing control and for this it simulates the functioning of a factory following a requirements set dressed up by the Technical Forum Group on Self-Organisation[5] (TFG SO). In MASC, we consider a set of operators, a set of stations (machines), and a set of containers to process. Stations are equipped for transforming the content of containers, but only with the control of an operator with required skills (or qualifications). Containers must be processed by several stations as to be completely transformed and released, before a given deadline.

Algorithm 1: Main simulation loop in MASC

```
Build operators
Build stations
Build containers
while not (all containers are completed) do
    Scheduler.executeStep(Available Stations,
    AvailableOperators, AvailableContainers)
end
```

Thus, at the beginning of each fabrication cycle, the simulator provides:

- available stations with their qualifications (or transformation capabilities),
- available operators with their qualifications (or skills),
- containers to process,

[4] Collective, Interaction, Emergence: `www.irit.fr/COLLINE`
[5] `http://www.irit.fr/TFGSO`

- potential perturbations that occur (breakdowns, unavailability, etc.).

Qualifications correspond to what an operator or a station can process. For a container, qualifications are the required qualifications for both operators and stations to process it.

MASC is provided with a benchmark greedy algorithm, AmasCOP, that uses a heuristic based on a cooperation measure [2].

3.3. CSP Specification

Let $M = \{m_1, \ldots, m_n\}$ be the set of stations. We have the following objective variables:

- $\forall i \in [1, n], o_i^t$ is the operator assigned to the station m_i at time t,
- $\forall i \in [1, n], c_i^t$ is the container assigned to the station m_i at time t.

So, the problem proposed in MASC can be modelled as a CSP at each time t:

- $\mathcal{X} = \mathcal{X}_o^t \cup \mathcal{X}_c^t$
  where $\mathcal{X}_o^t = \{o_i^t | i \in [1, n]\}$ and $\mathcal{X}_c^t = \{c_i^t | i \in [1, n]\}$,
- $\mathcal{D} = \{D(o_1), \ldots, D(o_m), D(c_1), \ldots, D(c_p)\}$
  where m (resp. p) is the number of operators (resp. containers) provided as input at time t,
- $C = C_{\text{diff}} \cup C_{\text{qual}} \cup C_{\text{time}}$
  where $C_{\text{diff}} = \{\text{alldiff}(\mathcal{X}_o^t), \text{alldiff}(\mathcal{X}_c^t)\}$, $C_{\text{qual}} = \{(q(o_i^t) = q(c_i^t) = q(m_i)) | i \in [1, n]\}$ and $C_{\text{time}} = \{(\text{end}(c_i) < t_{max}^{c_i}) \, | i \in [1, p]\}$.

$\mathcal{X}$ is here composed of $2n$ variables (2 per station) to assign. We have: $\forall i, D(o_i) \subset [1, n]$ and $D(c_i) \subset [1, n]$. This means that operators can be assigned to certain stations, and that containers must be processed by certain stations: these are *qualifications* for operators and containers that force these assignments. Constraints in C_{qual} mean that an operator must have the same qualification than a container on a same station at same time t. *alldiff* constraints mean that an operator or a container cannot be on two different stations at time t, and $<$ constraints mean that a container must be processed before the deadline t_{max}.

The main specificities of this CSP are:

- *alldiff* constraints imply a complete constraint network: all the variables for a same type of entity (operator or container) are linked by binary relation like $\forall i, j, o_i^t \neq o_j^t$ and $\forall i, j, c_i^t \neq c_j^t$. This implies that classical distributed algorithms (e.g. ABT or AWCS) will connect each agent with all the other agents.
- *alldiff* constraints are only applicable to assigned variables (e.g. if there are more stations than necessary, there is no solution), which will require to filter before solving processing by classical approaches.
- Temporal constraints only specify end-time ($\text{end}(c_i)$) but not start-time. This is due to the real time dimension of the problem: the platform provides new information at each time t.

Several solutions can be used for the multi-agent modelling of this problem. We will here distinguish two families of methods which corresponds on one hand to distributed and asynchronous algorithms for solving DisCSP (ABT and AWCS) and on the other hand to models based on the adaptive multi-agent theory (AMAS) [3].

4. Agentifying Variables

Some approaches propose a simple distribution of CSPs: an agent is responsible for the assignment of one variable (or more) [16]. In our problem, according to the relation between variables in $\mathcal{X}_o^t$ and variables in $\mathcal{X}_c^t$, it is obvious that an agent will encapsulate two variables for one station. So there is one agent per station. For more details concerning these approaches, we redirect the reader to [16]. Algorithms based on such models are only solving algorithms (no repair nor *anytime* properties) since they do not provide complete assignment at any time.

4.1. Asynchronous Backtracking (ABT)

Yokoo introduced ABT in which agents are completely ordered and own input and output links with other agents [16]. The order is determined with different methods: domain sizes, number of constraints, etc. A link between two agents represents a constraint between two variables owned by these agents. It functions as follows.

An agent chooses a value and informs following (output) agents, by using a *ok?* message. A recipient of such a message will try to find its value, knowing its predecessor's tentative value. If it manages to assign its variables to values respecting the constraints, it repeats the same process with its successors. However, if it does not manage to find a correct value it send back to its predecessor a *nogood* message to inform it that the proposed tentative value is a nogood value, which cannot lead to a solution. Receiving a *nogood* message causes a tentative for a new chosen value.

ABT has properties of completeness and correctness, but on large problems, it shows an exponential solving time. Another negative characteristic of ABT is the fact that agents needs to be totally ordered, which can be restrictive in dynamic environments, in which agents can appear

or disappear during the solving process –shattering the total order.

4.2. Asynchronous Weak-Commitment Search (AWCS)

[16] also proposed the AWCS algorithm in which priorities are attached to agents (initially set to 0) and dynamically changed during search. Like ABT, *ok?* and *nogood* messages are used by agents to interact. During the search, agents try to minimise the number of violated constraints with agents with lower priority. If an agent has no possible value, it increases its priority and initiates a search for a new value.

Here again, there is a total order on the set of agents, but unlike ABT, it is dynamic. Actually, it changes during run-time, thanks to priority updates.

4.3 Heuristic for ABT and AWCS

To fit to the manufacturing control problem, we consider a heuristic inspired from the AmasCOP approach [2], shown in formula 1 where t_r is the total remaining time, n_s (resp. n_o) the number of stations (resp. operators), $t_r(op)$ the remaining time to process operation op, $n_{cs}(op)$ (resp. $n_{co}(op)$) the number of stations (resp. operators) compatibles for operation op. This formula is used to compute a cooperation degree as to determine the order to choose containers, station and operators.

$$NC = \frac{t_r * n_s * n_o}{t_r(op) * n_{cs}(op) * n_{co}(op)} \quad (1)$$

Algorithm 2: Heuristic for ABT/AWCS

```
for compatible container c do
  | c.val ← compute heuristic value NC
end
for compatible operator o do
  | o.val ← max{values of containers compatible with o}
end
s.val ← max{values of operators compatible with s}
```

Algorithm 2 shows the method to compute the degree for a station (agent). First the non cooperation degree is computed (NC) using formula 1, for every container compatible with the station. Next, the algorithm maximize the values for each operators, as to sort them. Therefore, this heuristic orders agents and domains.

5. Agentifying Domain Entities

Other approaches consist in modelling agents from the domain description, independently from any CSP formalisation [1, 7, 10]. Software agents represent entities of the problem to solve (stations, operators, containers) or tools for coordination. In this framework, we present two multi-agent systems (ETTO4MC and DAmasCOP) based on the Adaptive Multi-Agent Systems theory [3, 6]. This theory relies on the principle that if every agent in a system is cooperative, then the system is functionally adequate and therefore adapted to its environment. So, they distinguish three key notions in this theory: cooperation, self-organisation and emergence. An agent is considered as being cooperative if, at each instant, it is capable of detecting, processing and using its skills from a received environmental signal. As to face the dynamics of their environment, these agents are equipped with mechanisms enabling them to modify their organisation. This modification can only concerns parameters of interaction (self-regulation) or topology of the organisation. This is generally called self-organisation, as soon as agents are autonomous in their decision to act on the organisation. Following modifications of the organisation and interactions, agents make the global function of the system emerge.

In the case of manufacturing control, this approach seems to be easily applicable: booking agents for operators and containers (mutually needing each other) explore stations, looking for partnerships and location with respect to their constraints and qualifications.

5.1. ETTO4MC Approach

The model we propose in this section, ETTO4MC (*Emergent Timetabling Organisation for Manufacturing Control*) is an asynchronous extension of ETTO [10] for the manufacturing control, developed with JADE[6]. So, here all domain entities are agentified: stations, operators and containers. Two kinds of agents are considered. *BookingAgents* (BA) represents operators and containers having to reserve a station and to sign a partnership with another BA, for a single step. BAs memorise their partners (past and current), and the encountered *ResourceAgents* (RA) as to suggest them to other agents they encounter during the solving process. RAs represent stations. Contrary to ETTO, resources are agents (and not only a cell in a grid representing the schedule). RAs memorise agent that reserve them. Table 1 shows the messages that agents send and receive. Initially, RAs are attached to stations and BAs are deployed within the network to find stations and partners. This deployment can be guided (e.g. pre-filtering the domains) or random, since this approach consists in self-regulating the current state of the system, which adapts to modifications from the environment (constraints or actors).

This paragraph presents how the multi-agent system perform the solving process, with a sample execution. A BA proposes to reserve a RA it knows (*okForBooking?*) for the

[6] http://jade.tilab.com/

Agents	Messages	Description
BA→RA	*okForBooking?()*	Ask the RA's availability, its state (reserved or not) and qualifications
	book(ba)	Reservation of the RA by a BA
	cancel(ba)	Canceling of a reservation by a BA
RA→BA	*inform?(q,s,a)*	Send information about a resource (qualification, state, reserving agents)
	partner(a,s)	Confirmation for the partnership
	cancel(a)	Canceling of a partnership, with a proposal of other potential partners
BA→BA	*negotiate?(v)*	Negotiation for a partnership with a cooperation value v
	result(a)	Information concerning the agent that won the negotiation

Table 1. Messages in ETTO4MC

current step. This last one sends back a message *inform?* containing its qualifications, its reservation state (full reserved by two BA, half reserved by one BA, or free), and waits for the BA's answer. Depending on the returned qualifications, the BA may either renounce (*cancel*) if they are not compatible or try to reserve. Reservation can be done either directly (*book*) if the BA lacks one partner of this type (container or operator) or indirectly by negotiating with the agent of the same type that reserves this resource (*negotiate?*). The agents that initiates the negotiation is informed of the result of the negotiation (*result*) as to reserve if it is a success (*book*). In the case of a reservation, the partners of this reservation are informed (*partner*). During a negotiation, the agent having the highest cooperation value reserves the RA at the expense of those having the lowest one, which will have to search for another RA.

The cooperation value for containers is inspired by the ones used in ETTO [10] and AmasCOP [2]. Cooperation is pointed up during conflict between two BA for reserving a RA. It is based on different situations depending on the state of the examined station, container and operator. Evaluation of the level of cooperativeness and the priority of a station over another is done on s_q, the number of shared qualifications (operator and container), and on c_q, the number of consecutive qualifications that can be processed (container):

$$\mathcal{V}^s_{coop} = s_q c_q$$

Concerning negotiation between containers, the breathing space of containers is a function of t_r, the remaining time before the deadline for the container, and δ_c the required time to completely process the container. $t_r - \delta_c$ (the absolute deadline for the container process) has priority over t_r so it is multiplied by a constant μ:

$$\mathcal{V}^c_{coop} = \mu \frac{1}{t_r - \delta_c} + t_r \qquad \text{with } \mu > t_r$$

For a negotiation between operators, or even different entities (operator vs. container), this is the difficulty to find a station, with respect to the available stations, that is taken into account in the measure of cooperation. It depends on the inverse of $\frac{m_s}{a_s}$, the potential of the agent to find a partner (the ratio between the number of compatible stations m_s and the total number of available stations a_s), v_s, the number of visited stations and the inverse of $\frac{p}{r}$, the ratio between the number of previous successful partnerships (p) and the number of previously requested partnerships (r), which measures the efficiency of the agent to find partners. As used above, the constant μ is used to put priority of a criterion over others:

$$\mathcal{V}^o_{coop} = \mu^2 \frac{a_s}{m_s} + \mu \frac{1}{v_s} + \frac{r}{p} \qquad \text{with } \mu > \max\{\frac{1}{v_s}, \frac{r}{p}\}$$

Even if they have found a solution, agents still try to find better partnerships, with better cooperation value. The only termination criteria is time, which constitutes the main flaw of ETTO(4MC). Nevertheless, following a modification in the network, agents are capable of self-adapt and regulate conflicts step by step. Although it is very different from previously presented algorithms for solving DisCSP, CSP constraints and variables are however encapsulated in the model. Variables from $\mathcal{X}_o$ (resp. $\mathcal{X}_c$) are distributed among BAs representing operators (resp. containers) and RAs. In fact, a BA owns a variable that indicates the reserved RA and reciprocally. The consistency between these two values is ensured by the RA. Constraints from $\mathcal{C}_{\text{diff}}$ are managed by BAs that reserve only one RA at the same time, only once (otherwise they cancel a reservation before reserving another RA). Constraints from $\mathcal{C}_{\text{qual}}$ are managed by BAs and RAs during reservation or partnership phases. Finally, constraints from $\mathcal{C}_{\text{time}}$ are managed by BAs when the compute their value for negotiation or reservation.

5.2. DAmasCOP Approach

In DAmasCOP (*Decentralised AmasCOP*) developed with JAVAct[7] containers explore stations as to process their pieces. Stations search for operators in adequacy with the qualifications of the current assigned container. At the beginning, containers are provided with the name and the qualification of one station of the factory. Each station has a given neighbourhood, that regroups a set of stations, and a list of operators having a least one qualification in common with the station.

[7] http://www.irit.fr/PERSONNEL/SMAC/arcangeli/JavAct_fr.html

Agents	Messages	Description
Container	*askToUseStation(q, p)*	Request for using a station with a qualification q and a priority p
	askForStationNeighborhood()	Request for the neighbourhood of a station
Station	*acceptContainer()*	Acceptance of a request from a container c
	rejectContainer(n)	Rebuttal of a request from a container and sending the neighbourhood n of the station
	stationNeighborhood(n)	Sending the neighbourhood n of the station
	cancel()	Cancelling the reservation of the container
	askToUseoperator(q,p)	Request for using an operator with a qualification q and a priority p
	cancel()	Cancelling the reservation of the operator
	confirmOperator()	Confirmation of the reservation of the operator
Operator	*acceptStation()*	Acceptance of the request of the station
	acceptStationIfNoOneElse()	Acceptance of the request of the station, if it does not find another operator
	rejectStation()	Rebuttal of the station
	cancel()	Cancelling of the reservation of the station

Table 2. Messages in DAmasCOP

Before each fabrication cycle (corresponding to the *while* loop in algorithm 1), each container computes a priority level as a function of its own knowledge, qualifications to process and the remaining time before the deadline. During exploration, a container memorises encountered stations to optimise its further searches. Thus, it may ask, to the station it evaluates as being the most relevant with respect to its needs, either its neighbourhood (if the station does not have the required qualification) or its usage state. When a container asks for using a station, two cases can occur:

- The container is less priority than the other containers that have requested the station, therefore the station sends to this agent its neighbourhood as to continue its search with new acquaintance.
- The container is more priority, and in this case, the station tries to find a free operator capable of processing container's contents. If an operator is found, the station accepts the container, else it rejects the container and send its neighbourhood to it.

Priority (or cooperation value) is computed as in AmasCOP but without taking into account the number of available or matching operators (only the stations) [2]:

$$\mathcal{V}^{c}_{coop} = t_r a_s - (t_r - \delta_c) m_s$$

where t_r is the time before the deadline for the container, a_s is the number of available stations, δ_c is the time to completely process the container, m_s is the number of stations matching with the constraints of the container.

Concerning operators, they try to favour highest priority requests. Thus, once connected with a station, there can be several possible reactions, depending on its state:

- *free*: it accepts the job,
- *reserved by a more priority station*, *perturbed* or *working on another station*: it rejects the job,
- *reserved by a less priority station*: the operator informs the new station that it can break free if no other operator is available; next, it waits an acknowledgement from the station requesting it before cancelling its current reservation.

Thereby an operator that has accepted to work on a station may cancel this job as to help a more priority station. In the same way, a station that has accepted to process a container may cancel this reservation and process high priority containers. Containers continue their search for other stations.

To halt this search process, measures determining stability of system must be injected at macro level. Currently, agents only reason on local representations, and therefore inserting such measures is difficult. In the context of this paper, we propose an *anytime* algorithm that can be halted after a given deadline, expressed in a number of agents' life cycle (perception-decision-action). In the continuation of this work, we will introduce an observing agent capable of exploiting macro-level information as to detect the stability and equilibrium, and to decide to stop the search process.

6 Results and Analysis

In this section, we present results for the resolution of an task assignment problem, in an unpredictable context: at each fabrication cycle t, agents only know current factory orders, and not those to be done after this cycle. A fabrication cycle corresponds to the execution of the procedure *executeStep* (cf. Algorithm 1).

6.1. Experimental Setup

The scenario we consider concerns:

- 10 qualifications with duration between 1 and 6,
- 5 perturbations, occurring at different cycles and with a duration between 10 and 30 fabrication cycles,
- 25 operators with 2 or 3 qualifications (except 3 operators with only 1 qualification),

- 30 stations with 2 or 3 qualifications (except 4 stations with only 1 qualification),
- 100 containers:
  - 50 at time $t = 0$ (deadline at $t = 200$),
  - 25 at time $t = 5$ (deadline at $t = 300$),
  - 25 at time $t = 10$ (deadline at $t = 150$).

Presented algorithms have been implemented using the JADE platform (except DAmasCOP, developed using JAVAct), and plugged to the MASC platform. ABT and AWCS have been extended using the cooperation measure used in the AmasCOP greedy algorithm, provided by default in MASC [2], for ordering domains for stations, operators and containers. To sum up, the methods experimented (and their implementation framework in brackets) using the presented scenario are:

- AmasCOP, the MASC greedy algorithm (Java),
- ABT, ABTCOP (ABT with the cooperation measure from AmasCOP), AWCS, AWCSCOP (AWCS with the cooperation measure from AmasCOP) and ETTO4MC (JADE),
- DAmasCOP (JAVAct).

In spite of the fact that most of these approaches are developed using distributable frameworks (JADE, JAVAct), we only proceeded to experiments on a single computer.

We next present results (over 100 simulations) conforming to the following metrics:

- resolution time in milliseconds, by fabrication cycle and total,
- number of exchanged messages, by fabrication cycle and total,
- number of containers, processed and remaining.

6.2. Resolution Time

Figure 1 shows the time as a function of fabrication cycles, needed by each presented method. Table 3 sums up the total resolution time for each of the methods. This metric is not the most relevant concerning adaptation. Nevertheless, it enables us to verify whether self-organising approaches are realistic concerning execution time. One can note that limit time let to ETTO4MC is about 300 ms/cycle, which partly explains its very high resolution time, compared to other algorithms.

Resolution times are very disparate. Algorithms based on variable agentification are very efficient, and their extensions using a cooperation measure decrease the number of fabrication cycles. However, ETTO4MC approach is not efficient at all. This is partly due to the intuitive choice made for partnerships, and to the JADE platform which more resource consuming than JAVAct, concerning the management of messages, which are far more numerous than in ABT or AWCS approaches. Finally, DAmasCOP is very efficient, and it seems due to the lightness of the JAVAct middleware.

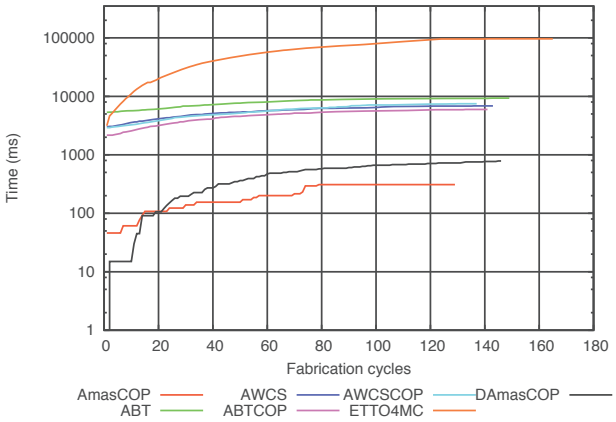

Figure 1. Resolution time as function of fabrication cycles

	Time (ms)	Cycles	Messages
AmasCOP	310	128	n/a
ABT	9325	145	14764
AWCS	6875	143	7147
ABTCOP	5965	141	5613
AWCSCOP	7470	137	7848
ETTO4MC	96468	165	147900
DAmasCOP	785	146	355906

Table 3. Total resolution time and number of exchanged messages

6.3. Message Traffic

Figure 2 show the number of exchanged messages (within JADE and JAVAct frameworks), whatever the type of message (cf. Table 1 for ETTO4MC, Table 2 for DAmasCOP and Section 4 for ABT and AWCS), according to fabrication cycles for each approach. Table 3 also sums up the total number of sent messages for each approach. This represents a relevant indicator of the network load implied by the different methods we presented.

Variable agentification approaches required few messages, and the extensions we suggest decrease significantly

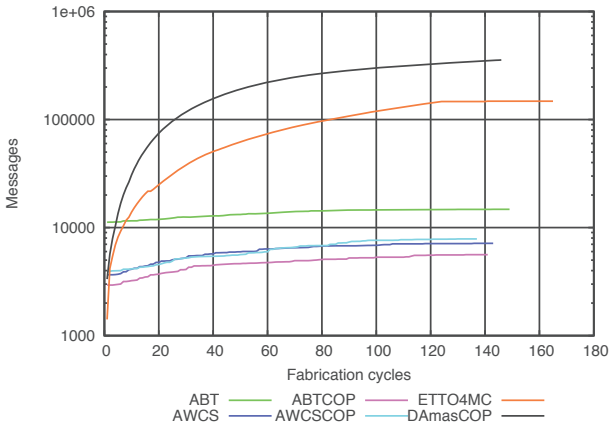

Figure 2. Number of exchanged messages as a function of fabrication cycles

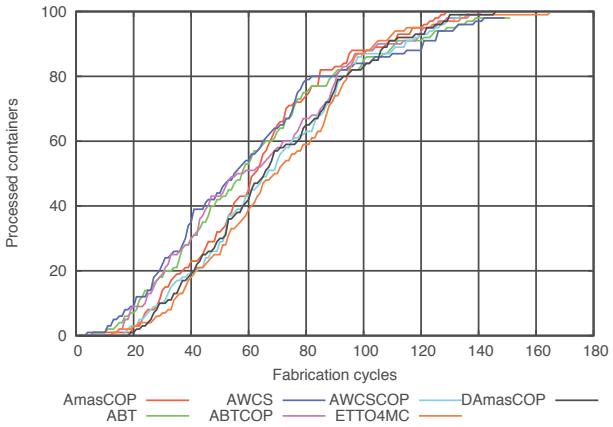

Figure 3. Number of processed containers as a function of fabrication cycles

this load. ETTO4MC and DAmasCOP hugely generates messages between agents, that can be a real obstacle for deploying these systems in ambient environments. AmasCOP does not generate any message (*n/a* value) because it is only a Java implementation of a centralised greedy algorithm led by the cooperation measure. One can also note that even if DAmasCOP generates more messages than all the other experimented approaches its implementation using JAVAct is very efficient.

6.4. Processed Containers

Figure 3 shows the evolution of container processing during the simulation. Table 4 presents the total number of processed containers at the end of the simulation. This is useful to determine whether methods are efficient from the production viewpoint.

We can observe that variable agentification approaches are less efficient from this point of view than entity agentification ones. Nevertheless, by adding the cooperation measure in the decision for choosing containers, operators and stations, ABT and AWCS are enhanced; besides ABTCOP and AWCSCOP process all the containers. We can also notice that all the algorithms are equivalent considering the number of fabrication cycles excepting ETTO4MC which processes far less containers at the beginning of the simulation than the other algorithms, but still manages to process all the containers with a reasonable number of fabrication cycles ($t = 165$); whereas ABT, at the contrary, processes a lot of containers at the beginning but ends the process before ($t = 145$).

	Fabrications	Remaining
AmasCOP	100	0
ABT	98	2
AWCS	98	2
ABTCOP	100	0
AWCSCOP	100	0
ETTO4MC	100	0
DAmasCOP	100	0

Table 4. Number of processed and remaining containers at the end of the simulation

7 Discussion

7.1. Dynamics

The proposed scenario presents interesting dynamics with massive incoming commands during simulation. It is also interesting to note that methods reasoning with the cooperation measure (AWCSCOP, DAmasCOP and ETTO4MC) manage to process all the requested containers. This adaptation to dynamics can be explained thank to the notion of cooperation which is at the center of peer-to-peer negotiations, that enable agents to solve local conflicts; whereas global approaches (e.g. ABT) can output solutions very far from the preceding solution, following a perturbation. However, this has a cost for self-organising approaches (DAmasCOP and ETTO4MC): very important message traffic.

Nevertheless, the MASC platform limited our study of dynamics. In fact, the only dynamics injected in the platform comes by the concept of fabrication cycle. At each cycle, MASC determines whether a container appears or an operator is no more available, for instance. There is no real

time dimension: an algorithm plugged to MASC can provide results for each cycle without any time constraint. It may be interesting to fix a time limit for each cycle as to test the robustness of the methods, facing response time requirements. In this direction, we have experimented ETTO4MC, for which such a time limit can be set up (e.g. 300 ms) for every fabrication cycles. However, this reflection has only sense for *anytime* algorithms, so not for ABT(COP) nor AWCS(COP), which do not provide any results (even if not a complete solution) before the end of the solving process, if there is not enough time.

7.2. Distribution and Decentralisation

All the expounded approaches do not display the same level of distribution. In fact, AmasCOP is not distributed at all (nor directly distributable for that matter since it requires global knowledge to function), for example. Distributed approaches (all the others), which are not based on the same agentification, displays different distribution too. Indeed, ABT and AWCS (and their extensions) distribute decision among as many agents as stations (that is 30 stations in the scenario presented in Section 6), while DAmasCOP and ETTO4MC requires as many agents as entities in the domain (that is 30 + 25 + 100 = 155). This notably explains the increase in number of messages and therefore of the time to process these coordination and cooperation messages. This counterbalances the bad results concerning resolution time obtained by ETTO4MC, which is developed using JADE and is therefore potentially distributable, with minimum implementation effort, at a larger scale than ABT for instance, since it deploys more agents.

While ABT and AWCS are based on a predefined order for responsibilities between agents (an agent is responsible for another agent in the sense that it checks the consistency of its assignments), self-organising approaches do not rely on any preset hierarchy. In fact, ETTO4MC and DAmasCOP maximise the decision distribution. Nevertheless, this requires an important coordination, but takes advantage by not introducing any breaking point in the system, due to a hierarchy. In ABT, if a high priority agent disappears, the resolution concerning the low level agent is reconsidered. Since responsibilities are diffuse in self-organising approaches, the danger of a breaking point is pushed aside.

To sum up, we can classify the different approaches by using two criteria, distribution (utilisation of a distributed infrastructure like JADE) and decentralisation (less hierarchy between agents):

- AmasCOP is centralised and non distributed,
- ABT(COP), AWCS(COP) are centralised and distributed,
- ETTO4MC and DAmasCOP are decentralised and distributed.

Among this sample group of methods, one can choose depending on the deployment infrastructure (distributed or not, robust network, etc.) and on the adaptation needs (more decentralisation implies more flexibility, but more coordination time). These trade-offs are a good way the position within the spectrum of possible applications.

7.3. Self-Organisation and Self-Regulation

Generally, multi-agent systems using cooperative local decision (as those based on the AMAS theory) are qualified as self-organising. Indeed, each agent has capabilities for changing its interactions with other agents. For example, in ETTO4MC or DAmasCOP, agents can change their partnerships depending on their qualifications or their priority. This changes are only consequence of the local interactions they have with their neighbourhood. In the manufacturing control problem, this neighbourhood is strongly dependant from the structure of the constraint network. As we underlined in section 3.3, this network is complete (each variable is connected to all the others), contrary to other problems, such frequency assignment in which locality principle (the fact that agents have a limited neighbourhood) is more easy to preserve [11]. This implies to artificially limit the neighbourhood of agents in DAmasCOP and ETTO4MC: limited number of peers, randomly chosen or agents only know other agents in connection with their current partner.

However, the current model only handle the self-regulation dimension of self-organising systems. In fact, agents are only able to regulate the system by changing some peer-to-peer interactions and then by reducing global constraint violation. We can imagine that completely self-organising agents would be able to add/remove agents by themselves, which can be interpreted, at the constraint network abstraction level, as a change in the organisation of the constraints.

7.4. Comparing Different Approaches

This work allows us to draw the limits of the metrics and the platform we use. Although the primary goal of MASC is to propose an environment for testing multi-agent approaches on a difficult problem, the manufacturing control problem, it is fussy to compare them. Indeed, all these approaches do not have the same level of distribution, do not provide results with same level of quality, in an reasonable time, or require numerous data exchanges. Moreover, some approaches have been implemented with distributed framework (JADE or JAVAct) whereas others only propose multi-agent at the modeling phase (AmasCOP). In addi-

tion, even distributed frameworks display great discrepancies concerning performances and services. The methodological dimension (advantages of an approach from a development viewpoint) for adaptive systems remains difficult to evaluate. As we have previously underlined, MASC platform is limited concerning dynamics and real-time measurements. This narrows scenarios and tests that can highlight the benefits of deploying self-organising multi-agent systems.

This justifies our perspective to develop a real world ambient environment for tests, based on a network of devices. It is also one of our perspectives to extend the comparison to other approaches proposed in the Colline working group of AFIA, and to define new scenarios and relative metrics putting the stress on disturbances and adaptation.

8 Conclusion

In this paper we have presented several approaches for making manufacturing control more adaptive, in a distributed context: variable agentification based approaches (ABT and AWCS) and self-organising approaches by entity agentification (ETTO4MC and DAmasCOP). These approaches differ in their way to explore the search space, in their communication requirements, and in their capabilities for answering to dynamics. We have also extended ABT and AWCS with a cooperation measure between variables, resulting on ABTCOP and AWCSCOP.

From the experiments we have made on MASC, it is clear that classical DisCSP solvers remain relevant, but they are improvable, as we have done by proposing ABTCOP and AWCSCOP, which are more efficient, generate less messages and process more containers. However, approaches agentifying entities process all the containers but with far more time and communication charge, since they required far more messages to coordinate the decentralised decision process.

Therefore, thanks to MASC platform, flaws and advantages of these methods have been identified. Nevertheless, other algorithms can still be implemented (notably, parallel local search [13], decentralised local search [16] or stigmergic ones [7]). Likewise, future works will consist in deploying these systems on pervasive networks (composed of Sunspots) as to evaluate their relevance in a real context, not simulated. This stage will be eased thanks to the use we made of distributed frameworks like JADE or JAVAct, in this current work.

Acknowledgements

We would like to thank Pierre Glize (IRIT), responsible for the Colline working group of the AFIA, and André Machonin (IRIT), maintainer of the MASC platform for his answers to our numerous questions and requirements.

References

[1] S. Bussmann, N. R. Jennings, and M. Wooldridge. *Multiagent Systems for Manufacturing Control*. Springer, 2004.

[2] D. Capera, C. Bernon, and P. Glize. Étude d'un processus d'allocation coopératif de ressources entre agents pour la gestion de production . In *7e Congrès de la ROADEF*, pages 369–383. PUV, 2006.

[3] D. Capera, J. Georgé, M.-P. Gleizes, and P. Glize. The AMAS theory for complex problem solving based on self-organizing cooperative agents. In *1st Int. TAPOCS Workshop at 12th IEEE WETICE*, pages 383–388. IEEE, 2003.

[4] R. Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.

[5] J. Duggan and J. Browne. Production Activity Control: A Practical Approach to Scheduling. *International Journal of Flexible Manufacturing Systems*, 4(1):79–103, 1991.

[6] J.-P. Georgé, B. Edmonds, and P. Glize. Making Self-Organizing Adaptive Multi-Agent Systems Work - Towards the engineering of emergent multi-agent systems (chapter 8). In *Methodologies and Software Engineering for Agent Systems*, pages 319–338. Kluwer, 2004.

[7] H. Karuna, P. Valckenaers, B. Saint-Germain, P. Verstraete, C. B. Zamfirescu, and H. Van Brussek. Emergent Forecasting Using Stigmergy Approach in Manufacturing Coordination and Control. In *Engineering Self-Organizing Applications (ESOA'04)*, pages 210–226. Springer, 2004.

[8] J. Liu, H. Jing, and Y. Y. Tang. Multi-agent Oriented Constraint Satisfaction. *Artificial Intelligence*, 136(1):101–144, 2002.

[9] V. Mavrík, V. Vyatkin, and A. W. Colombo, editors. *Holonic and Multi-Agent Systems for Manufacturing, 3rd International Conference on Industrial Applications of Holonic and Multi-Agent Systems (HoloMAS'07)*, volume 4659 of *LNCS*. Springer, 2007.

[10] G. Picard, C. Bernon, and M.-P. Gleizes. Emergent Timetabling Organization. In *Multi-Agent Systems and Applications IV - 4th International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS'05)*, volume 3690 of *LNAI*, pages 440–449. Springer, 2005.

[11] G. Picard, M.-P. Gleizes, and P. Glize. Distributed Frequency Assignment Using Cooperative Self-Organization. In *First IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO'07), Boston, Mass., USA, July 9-11, 2007*, pages 183–192. IEEE Computer Society, 2007.

[12] G. Picard and P. Glize. Model and Analysis of Local Decision Based on Cooperative Self-Organization for Problem Solving. *Multiagent and Grid Systems*, 2(3):253–265, 2006.

[13] E. Talbi, editor. *Parallel Combinatorial Optimization*. Wiley, 2006.

[14] J.-B. Welcomme, M.-P. Gleizes, and R. Redon. Adaptive Multi-Agent Systems for Multidisciplinary Design Optimisation. In *16th International Conference on Engineering Design (ICED'07), Paris*. The Design Society, 2007.

[15] B. Wu. *Manufacturing Systems Design & Analysis: Context & Techniques*. Springer, 1994.

[16] M. Yokoo. *Distributed Constraint Satisfaction: Foundations of Cooperation in Multi-Agent Systems*. Springer, 2001.