Proceedings of the Third International Symposium on Swarm Intelligence Algorithms and Applications Symposium, A symposium at the AISB 2010 Convention

Cyrille Bertelle, Gérard H.E. Duchamp, Rawan Ghnemat

Proceedings of the Third International Symposium on

Swarm Intelligence Algorithms and Applications Symposium

A symposium at the AISB 2010 Convention
29 March – 1 April 2010
De Montfort University, Leicester, UK

Editors
Cyrille Bertelle, Gérard H.E. Duchamp and Rawan Ghnemat

Published by SSAISB: The Society for the Study of Artificial Intelligence and the Simulation of Behaviour
http://www.aisb.org.uk/

ISBN: 1902956966

SIAAS 2010

Swarm Intelligence Algorithms and Applications Symposium

within

AISB 2010 Convention

March 29 – April 01 2010

De Montfort University, Leicester, UK

Cyrille Bertelle, Gérard H.E. Duchamp and Rawan Ghnemat (eds)

Program Committee:

- Aladdin Ayesh, De Montfort University, Leceister, UK
- Habib Abdulrab, INSA Rouen University, France
- Eduard Babkin, Higher School of Economics, Russia
- Cyrille Bertelle, University of Le Havre, France
- Gérard H.E. Duchamp, University of Paris XIII, France
- Rawan Ghnemat, German-Jordanian University, Amman, Jordan
- Laszlo Gulyas, Eotvos University, Budapest, Hungary
- Colin G. Johnson, University of Kent, UK
- Alaa Sheta, Taif University, KSA
- Eric Alfaro, IPN-UPIICSA-SEPI, Mexico
- Tim Blackwell, Goldsmith, University of London, UK
- Frank Guerin, Department of Computing Science, University of Aberdeen, UK

General Chairs:

- Cyrille Bertelle, University of Le Havre, France
- Gérard H.E. Duchamp, University of Paris XIII, France
- Rawan Ghnemat, German-Jordanian University, Amman, Jordan

Preface

The increasing complexity of the current world can be observed each day. Sustainable development for example consists of economical and social systems management within natural environment. The understanding of the whole leads to what we call territorial intelligence. The way of modelling these complex systems is often based on interactive networks, dealing with the interconnection between all of the system components. Decision making on this complex world, need tools able to detect and manage emergent organizations through these networks. Distributed Artificial Intelligence (DAI) is the adapted conceptual trend which allows the proposal of some relevant solutions by relying on social and physical sciences models exhibited and observed in nature (e.g. ant colonies, molecular crystallisation, etc.). In this search and management of emerging organization, swarm intelligence algorithms proved to be popular and effective methods to use. On the technological front, the increasing number of robotic systems, advances in nano technology, and the sheer complexity of modern enterprise systems, especially those boosting high degree of autonomy, makes the development of swarm intelligence timely and needed.

Le Havre, Paris & Amman
March 15th 2010

Cyrille Bertelle
Gérard H.E. Duchamp
Rawan Ghnemat

Contents

Exactly Solved Models for Collective Behaviour and Complex Systems

Gérard H.E. Duchamp [1]

Abstract

Complex systems is a set of theories which serves to model emergent behaviours or emergent phenomena in systems which are not reducible to the sum of their parts. Among these theories is the modelisation of collective phenomena. As miracles of regularity in nature appears the exceptional family of "exactly solved model" as the Ohm formula $U = RI$ which is sol simple but after all is a relation between statistics of electrons.

This talk begins by recalling the "triple birth of quantum mechanics" and the history of ideas leading to the fundamental commutation relation $AB - BA = 1$. We show how these operators can be found in the macroscopic word as simple statistical "death and birth" processes and gain avery natural and easy description of the traditional Fock space then explaining the necessity of the normal form by simple spectral arguments.

Then comes the evolution groups and two aspects of the product formula applied to combinatorial field theories. Firstly, we remark that the case when the functions involved in the product formula have a constant term is of special interest as often these functions give rise to substitutional groups. The groups arising from the normal ordering problem of boson strings are naturally associated with explicit vector fields, or their conjugates, in the case when there is only one annihilation operator. We also consider one-parameter groups of operators when several annihilators are present. Secondly, we discuss the Feynman-type graph representation resulting from the product formula and show that there is a correspondence between the packed integer matrices of the theory of noncommutative symmetric functions and these Feynman-type graphs.

[1] LIPN, University of Paris XIII, France, email: ghed@lipn.univ-paris13.fr

From Ants To Robots: A Decentralised Task Allocation Model For A Swarm of Robots

Sifat Momen[*] and Amanda J.C. Sharkey[*]

Abstract. This paper presents a task allocation and task switching model for a decentralised group of mobile robots. The model is strongly inspired by the behaviour of social insects, typically ants, which exhibit remarkable techniques for allocating tasks and adapting to the changing environment. The agents in the model allocate tasks based on the local cues they perceive and without the need of any centralised controller. The paper compares two communication techniques inspired by the ant colony behaviour. The objective of this paper is fourfold: **(1)** to design and **(2)** implement a task allocation model for a swarm of mobile agents, **(3)** to develop two communication techniques (one in which agents communicate directly along with indirect communication and in the other they only communicate indirectly), and **(4)** empirically investigate the impact of the two techniques on the performance of the system. The agents are assumed to have limited perception and to only be able to interact locally. Experimental results reveal that the model is highly adaptive and efficient whichever communication technique is used. Furthermore, the results also show that the incorporation of explicit communication improves the performance of the system significantly over that of indirect communication. The model offers benefits to heterogeneous mobile robot systems to dynamically allocate tasks among themselves without the need of any centralised controller in such a way so as to improve the performance of the overall system.

1 INTRODUCTION

Flocks of birds meandering in the evening light, armies of ants marching for foraging, herds of buffalos congregating to avoid predators, synchronised flashes from male fireflies attempting to attract the female ones or even the pods of dolphins dancing up and down in unison are some of the spectacular examples of collective behaviours [1,2] that animals display. Their behaviours are not only enthralling to watch (figure 1) but are also some of the finest examples of how individuals form groups which enable the group as a whole to carry out tasks that could not be accomplished by a single individual with the same efficiency. It is now well established that animals self organise [2] by repeatedly interacting with the neighbouring individuals and the environment in the vicinity resulting in the emergence of such collective behaviours. Individual agents neither have any global templates of the environment nor follow any particular leader. Instead, they behave as purely reactive individuals trying to synchronise with the immediate neighbours through some simple local interactions. Such local cohesion among the agents facilitate the tendency to become a part of a group which consequently benefits them in numerous ways including the possibility of minimizing danger of an individual from a potential predator [3], accomplishing tasks that are otherwise difficult to carry out and also in transferring vital information within the group quickly [4]. Each individual does not have enough intelligence to carry out its job optimally rather as a group such intelligence emerges (through local interactions) which is often referred to as swarm intelligence (SI). Intriguingly, such group behaviour and self organised systems are not limited to nature, but are also extremely prevalent within the very society we live in: pedestrians travelling [5] like that of the flocking of birds, spreading of rumour [6], rhythmic applause after a good concert [7, 8], traffic flow [9] in a busy road and even the evolution of a new sign language that emerged from mere interactions between the school children in Nicaragua [10] are just some of the examples of such self organized systems within our society.

Recent time has witnessed huge interest in the field of swarm intelligence and swarm robotics (SR) [14; for a brief history, see 11 – 13] among researchers in areas as different as biology and engineering. The concept of *swarm robotics* is strongly inspired from biology and especially from the behaviour of eusocial insects [15] like that of ants, bees, termites and wasps that show some remarkable examples of how a large number of simple individuals can use extremely simple rules and local communication to result in a collectively intelligent system. For engineers and roboticists, the field provides a number of key advantages (including robustness, flexibility and scalability) over the traditional deliberative based system for a wide number of practical applications whereas for the biologists, it provides a novel platform to analyse the mechanism underlying the principles of collective behaviour within animal groups.

Our work is strongly inspired by the behaviour of eusocial insects especially from that of the ants which provide us with a number of keen techniques for dividing tasks among the individuals. The individuals not only carry out a single task but also have the ability to switch between tasks in response to the changing demand or any other external perturbation. The decision to switch task does not arise from any global knowledge of the environment whatsoever but is rather taken autonomously based on the local interactions between the individuals and the stimuli received.

This paper addresses the issue of division of labour (DOL), inspired from the ant colony behaviour, within the realms of heterogeneous groups of robots (agents). The aim of this paper is to present an agent based model (ABM) (an extended version of that presented in [16, 17]) that would enable us to explore the dynamics of making decisions within groups of agents/robots. The model embraces threshold based approach [34] and presents two techniques (one in which agents use explicit and another

[*] Neurocomputing and Robotics Group, Dept. of Computer Science, Univ. of Sheffield, Regent Court, Portobello, Sheffield S1 4DP, UK. Email: {s.momen, amanda}@dcs.shef.ac.uk

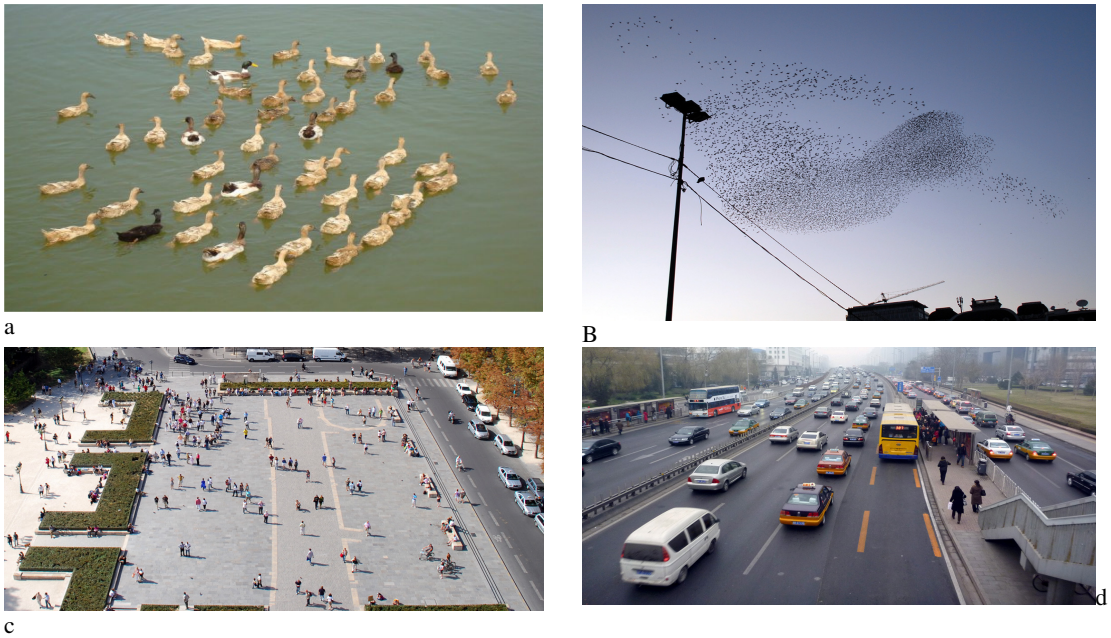

Figure 1: Examples of self-organised collective behaviour. a) a team of ducks (provided by Nafi Ahmed), b) a flock of birds in Milan, c) a crowd of people in front of Notre Dame Cathedral, Paris, d) traffic flow in busy Beijing. Photos (b) – (d) are provided by Lei Ye. All Photos used with permission.

in which agents use indirect communication), inspired by the ant colony behaviour for autonomous division of labour based on the environmental and social cues they receive. Furthermore, we show that such techniques give rise to an extremely adaptive and efficient system. The system is inherently scalable owing to its decentralised nature. Here we pose and empirically investigate two relevant questions: (1) whether the behavioural rules incorporated in this model are sufficient to produce an adaptive system and (2) whether the additional use of explicit communication has any advantage over the indirect communication for the performance of the system? Answers to these questions would provide us with a better understanding of the impact of different types of interactions between the agents on the performance on the system. The rest of the paper is organised as follows: Section 2 discusses the mechanisms of DOL in social insects such as ants. Our model is proposed and described in Section 3 followed by the experiments and results in Section 4. Finally, Section 5 concludes the paper with a remark on our future work.

2 DIVISION OF LABOUR IN EUSOCIAL INSECTS

Ants are referred to as eusocial insects (insects exhibiting social behaviours including that of cooperative brood care, overlapping between generations and division of labour), a term first coined by E.O. Wilson in 1971 [15; also see 18] while classifying insects in terms of the social behaviours they exhibit, belonging to the family of *Formicidae* of the order *Hymenoptera*. There are currently over 12,000 known species of ants having colony size ranging from a few individuals to over millions. Ants are known to use simple yet extremely sophisticated communication mechanisms ranging from recruitment techniques via pheromones, tandem learning, antennal contact, and even stridulation [19]. They are extremely small in size (individuals weigh as little as 5 mg); however their social behaviours allow them to live at large.

So, what makes these tiny creatures so successful in effectively running and maintaining colonies some of which are as big as that of London city (by population)? What strategies have they embraced that led them to be socially so successful? Recent researchers [20, 21, 22, 23, 24] have pointed out that it is their embracement of effective DOL that has allowed them to be socially so successful.

A. Division of Labour in Ants

Ants are perhaps best known for their ability to effectively divide a wide range of tasks among the workers. This phenomenon of dividing tasks among workers is what is termed as *division of labour*; a term first introduced by Adam Smith, the father of modern economics, in his influential book "The Wealth of Nations" [25]. However, ants are known to have been using more effective mechanisms of dividing labour for millions of years. Not only can they divide tasks among groups, they can rearrange the distribution of workers depending on the need of the colony or any perturbation caused.

Ants exhibit a wide range of techniques for dividing labour. These mechanisms can be broadly categorised in three groups [26]: worker polymorphism, age polyethism and individual variability.

Worker polymorphism (also called physical castes) arises in ant colonies (e.g. *Atta colombica*) that have distinguishable subcastes within the worker ants [27]. One subcaste differs from the other one in terms of the size/morphology of the worker ants, which in turn influence the type of task chosen. For instance in many ant species major workers, with large head and sharp mandibles, specialise in tasks that require physical strengths like guarding nests and transporting food items whereas minor

workers tend to specialise in lighter tasks such as cleaning the nest and brood caring.

Some ant species such as *Pogonomyrmex barbatus, Cataglyphis bicolour* and *Oecophylla smaragdina* [24, 28, 29] tend to carry out tasks depending on their age (age polyethism). They tend to follow some kind of centrifugal tendency in selecting the task they carry out with the younger ones working closer to the centre of the nest and the older ones working more distant from the nest.

However most of the ant genera carry out tasks depending on the local cues they receive. Local communications between the ants influence individuals of which task to be selected.

B. Models of Division of Labour in Ants and Robots

The last two decades have witnessed the development of a number of models trying to establish the mechanism of the selection of tasks in social insects such as ants. These models differ from each other in many aspects including worker-worker interactions, genetic basis of task selection, motivational state of the worker, spatial arrangement of the workers in the nest and also the learning parameters [30].

Out of these models, one of the most important is the response threshold model, where agents have different threshold for different tasks. The agents can also update the values of the threshold/stimulus to respond to changing demand. Various versions of response threshold model has been embraced and developed by the researchers [for further details see 31, 32, 33, 34, 35].

Other promising models in DOL include that of the foraging for work model (FFW) [36, 37] and that of the learning models [38, 39, 40] in deciding what task to execute. Recently there has been enormous interest in using swarms of robots to model and carry out different tasks. Using SR for such purposes not only facilitates solving various practical solutions for engineering problems, it also provides a novel platform for the biologists to understand animal behaviour better. Biologist Robert Full, professor at UC Berkeley, describes this association between biology and another discipline as biomutualism [41].

One of the earliest works in this field was carried out by Krieger and Billeter in 2000 [42]. They used up to twelve mobile robots to make autonomous decisions of whether to forage (collect items from the environment) or rest depending on the nest energy level which was periodically echoed to the robots. Wenguo Liu and his colleagues [43, 44] used a threshold based approach in simulated robots to develop an adaptive threshold based mechanism to divide the number of foragers and resters so as to optimise the net energy level of the system. Momen and Sharkey [17] used ABM to simulate ant colony behaviour within the realms of heterogeneous groups of robots. They further investigated the advantages of task switching mechanisms, exhibited in various colonies of ants, in terms of the net energy gained by the swarm.

3 Proposed Model

The model proposed in this paper places three types of agents (mobile foragers, mobile brood carers, and static brood members) within a 71 × 51 grid in a 2D environment. The model contains a nest consisting of four separate chambers with brood surrounded by brood carers whereas foragers mostly reside outside in the environment (figure 2). Each of the chambers has its own odour that is spread over the environment.

The nest chamber odours are modelled as falling linearly from its respective centre of the chamber; thus each of the four types of smells creates a potential gradient uphill towards the centre of the chamber. The use of such artificial potential field technique for navigation has been an extremely popular approach [45 – 47] in mobile robotics typically in the case of avoiding obstacles or moving towards a target.

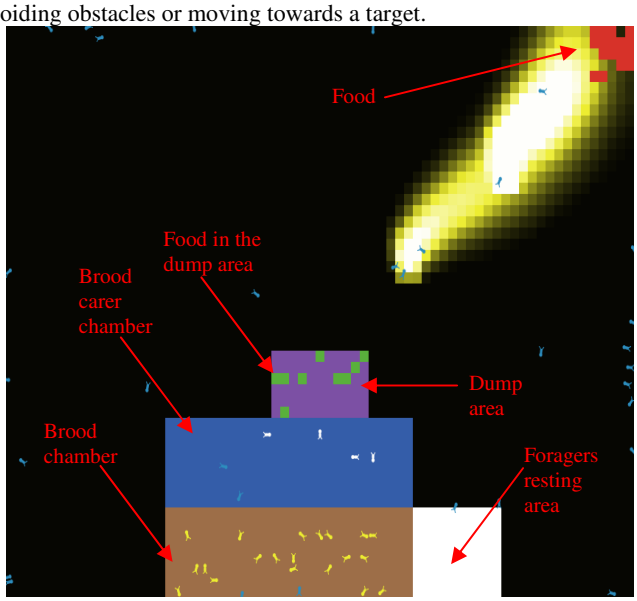

Figure 2. Snapshot of the simulation

In this model, brood reside at the lowest chamber of the nest. There is a dump area (DA) located at the entrance of the nest for the foragers to drop off food and leave it for the brood carers to pick them up for feeding the hungry brood. Similar spatial distribution is observed in various ant species [48, 24].

3.1 Behavioural Rules

Different types of agents follow different rules to accomplish their tasks. There are principally three types of tasks present in this scenario; foraging, resting and brood caring. Foraging involves foragers exploring the environment in order to find a food source, followed by picking up a piece of food, finding its path back to the dump area of the nest and finally dropping the food item there. Brood caring involves brood carers picking up a piece of food from the dump area, locating a hungry brood member, and feeding it. The resting task involves the agents (foragers and brood carers) resting within their respective chambers.

The behavioural rules of the agents are described below:

A. Brood

Each brood member can be in one of the two states: *hungry* or *non-hungry*. Initially all the brood are in the non-hungry state having a randomised hunger level. At every simulation time step, the hunger level of each brood member increases by its hunger

rate (eq. 1) which is selected randomly (Fig. 3). When the hunger level of a brood member exceeds some threshold (th_h), it switches its state to hungry, and seeks the attention of the brood carers by emitting a chemical signal (called shouting chemical here). The strength of the shouting chemical is modelled to fall linearly with the distance from the hungry brood member in such a way that it is at maximum at the location of the hungry brood member and minimum at the shouting-radius. The strength of the chemical is zero if the distance from the hungry brood member is more than the shouting-radius (eq. 3). However, if a hungry brood member is fed by a brood carer, the hunger level of the brood decreases by some constant value. If the hunger level at any time, t, is below the threshold parameter, the brood member switches its state back to the non-hungry state (eq. 2) (Fig. 4).

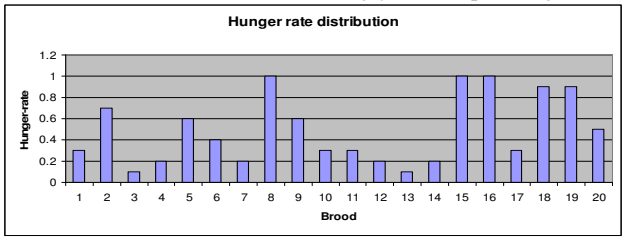

Figure 3. Hunger rate distribution across brood members of the brood in one of the runs

$$HL_{new} = HL_{old} + HR \quad (1)$$

where $0 \le HR \le 1$

$$HS = \begin{cases} 1, & HL_t \ge th_h \\ 0, & HL_t < th_h \end{cases} \quad (2)$$

Where,

HL_{new} is the new hunger level of the brood member,

HL_{old} is the previous hunger level of the brood,

HR is the hunger rate of the brood,

HS is the hunger state of the brood; 1 = hungry state and 0 = non-hungry state,

HL_t is the hunger level of the brood at time step t, and

th_h is the threshold parameter of the hunger level.

$$C_{SC} = \begin{cases} A - Bx, & x \le sr \\ 0, & x > sr \end{cases} \quad (3)$$

where,

C_{SC} is the concentration level of the shouting chemical,

x is the distance from the centre of the hungry brood, and

sr is the shouting radius.

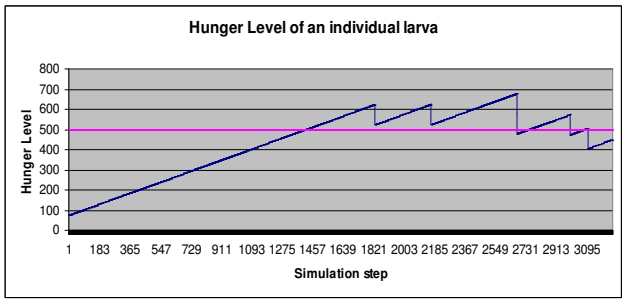

Figure 4. Hunger level of a brood member as a function of time (the reduction of the hunger level is due to being fed by brood carers)

B. Brood Carers

Brood carers respond to the need of the brood by adjusting their threshold values depending on stimuli (in this case, the shouting chemical) they receive (See section 3.2 for detailed description). Once a brood carer makes the decision to feed a hungry brood member, it goes to the dump area (DA) of the nest in search of food. It uses its local sensing to smell the scent of the dump area at its immediate patch ahead, patch left and ahead and patch right and ahead. Then the brood carer compares the relative strength of the scents in the three directions and moves in the direction of the strongest scent. This simple local interaction with the environment allows the brood carer to easily locate the DA. Once it reaches the dump area, it moves randomly within the DA to find a piece of food and as it finds a piece of food, it picks up the food item and travels towards the brood chamber in search of a hungry brood member.

The brood carer finds the brood chamber using the same technique as that of finding the dump area of the nest. Once a brood carer reaches the brood chamber, it uses the potential gradient of the shouting chemical to go uphill in order to locate a hungry brood member. Once the brood carer successfully locates a hungry brood member, it feeds it which causes the brood member's hunger level to be reduced by the energy provided by the food item (the simulations presented in this paper assumes that all the food items provide the same energy).

C. Foragers

Foragers can either forage or rest depending on what is required. A forager starts its journey by exiting the nest and travelling in a random direction in search of food. If it finds a food item in the environment, the forager picks the piece of food up, becomes laden and goes back to the DA of the nest by following the potential gradient of the scent of the DA. As it travels back to the nest, the forager leaves a simulated trail of chemical markers called pheromones in the environment. Once the forager reaches the DA, it leaves the food item there, decides if it needs to forage further or not and if so turns 180^0 and starts foraging again. The decision of whether to further forage or switch to a different task is explained in Section 3.2.

The simulated pheromone both diffuses and evaporates at constant rates. When unladen foragers looking for food encounter pheromones, they use the pheromone trails to go uphill towards the food source (See figure 2). The concentration level of pheromone at any of the patches of the environment can be modelled as in equation 4.

$$\frac{dC}{dt} = \alpha n_L + \frac{1}{8}\beta C' - (\varepsilon + \phi)C \quad (4)$$

where,

C is the concentration level of the pheromone at the patch concerned,

n_L is the number of laden ants passing the patch concerned,

C' is the cumulative concentration level of the pheromones of the nearby eight patches,

C is the concentration of the chemical at the patch concerned,

ε is the evaporation rate of the chemical,

ϕ is the diffusion rate of the chemical, and

α and β are some constants such that $\alpha > 0, \beta > 0$.

Each of the foragers also maintains two types of clocks: a searching-clock and a resting-clock to track how long it has been searching for food and resting within its chamber. Every time a forager switches its task it resets all these clock values so that the clock values in the previous state do not affect the present state.

D. Obstacle Avoidance by Foragers

Foragers when foraging also actively prevent collision between each other by turning away from their nearest neighbour when required. The dynamic obstacle avoidance algorithm is inspired from the Craig Reynold's model of the flocking of birds [49; also see 50]. When an agent gets too close to another agent, it uses the headings of its nearest neighbour and its own heading to calculate the angle and the direction to turn in order to prevent any collision.

3.2 Task Allocation and Task Switching Algorithm

In this model, the brood create a task demand by emitting the shouting chemical whereas brood carers and foragers carry out their tasks, or sometimes switch task, in response to the changing demand. The environment is extremely dynamic and therefore the agents need to be flexible enough to meet the changing demand. For instance if there is no food available in the environment, the foragers would need to abandon their foraging task and rest in their designated area to save their energy [17]. Similarly, if the amount of hungry brood increases, some foragers might need to switch their task and become brood carers to meet the changing demand. The need for switching in the other direction might also arise if there is not enough food in the DA when the brood carers need to feed the brood. In that case, some brood carers might decide to change their task to foraging to assist the existing foragers. However, it is also important that the agents are not too flexible. Otherwise a slight change in the demand might cause all the agents to switch to the same task which would not be a desirable outcome. In order to prevent this, there is a need for some sort of "natural queue" that would allow appropriate task switching, and avoid the problem of all the foragers and brood carers switching to the same task. A threshold based mechanism [34] facilitates such a "natural queue", and provides the key motivation for embracing this technique.

Foragers and brood carers between them carry out three tasks namely **(1)** foraging, **(2)** brood caring and **(3)** resting as well as deciding which task to carry out next (Figure 5).

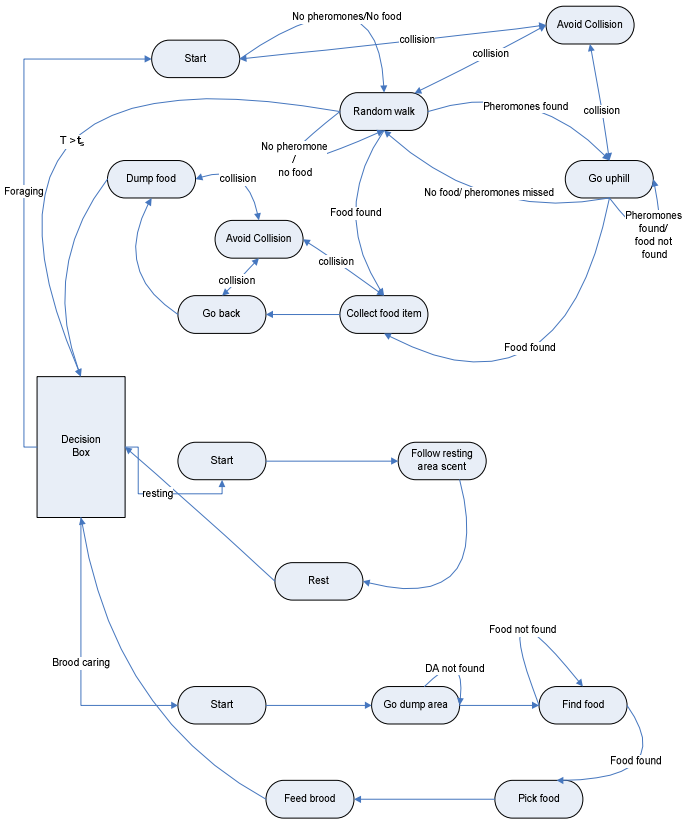

Figure 5. Schematic Diagram of the task allocation model of an agent

Every mobile agent maintains three types of thresholds: threshold for foraging (t_f), threshold for resting (t_r) and threshold for brood caring (t_{bc}) and updates them on the basis of events encountered. In a threshold based mechanism, the threshold value for a particular task of an agent is decreased when the agent encounters a stimulus for the task. Thus, over the course of time the threshold value for a task decreases significantly if exposed to greater stimuli for the task. The probability of an agent carrying out a particular task is modelled to vary inversely with the threshold value of the task and the threshold values are updated based on the events encountered; thus enabling agents to adapt to the changing demand.

We use a simple but effective principle for updating the thresholds. The threshold value for a particular task is decreased if the agent has either successfully completed the task or has received a stimulus for that task. On the other hand, the threshold value for the task is increased if either it has been unsuccessful in carrying out the task or hasn't experienced a stimulus for a long time (equation 5). This allows an agent to adapt to the dynamic system and react accordingly. Table 1 illustrates the adaptation rules for the agents.

$$t_{f,r,bc} = t_{f,r,bc} \pm \Delta \quad (5)$$

where Δ is the adaptation rate i.e. the rate at which the agents update their threshold values.

The threshold values are bound between two threshold bounds: an upper threshold bound and a lower threshold bound. If the threshold value becomes more than the upper threshold bound, it resets the threshold value of the task equal to the upper threshold bound.

Event	Which Agent?	Action	Remarks
Shouting chemical perceived	ALL except brood	t_{bc} decreases	There is a demand for feeding the brood
Food at DA < lower threshold of food in DA	ALL except brood	t_f decreases, t_r increases	There is less food in DA. Therefore more food is Required
Food at DA > upper threshold of food in DA	ALL except brood	t_f increases, t_r decreases, t_{bc} decreases	There is sufficient food in DA.
Brood carers looking for food in DA too long	Brood carers	t_f decreases, t_{bc} increases	There is insufficient food in DA
Foragers searching too long for food	Foragers	t_f increases, t_r decreases	There might not be enough food in the environment
Resting too long inside The chamber	Foragers	t_f decreases, t_r increases	There might be a need for foraging instead
Resting too long inside the chamber	Brood Carers	t_r increases	Should look out for other tasks
Successful retrieval of food	Foragers	t_f decreases	There might be some more food out in the environment

Table 1. Adaptation Rules for the agents

Similarly, if the threshold value falls below the lower threshold bound, the threshold for the task is reset to be equal to the lower threshold bound. The task with lowest threshold is selected with some probability.

3.3 Indirect versus Explicit Communication

The effect of two different forms of communication, (i) indirect, and (ii) explicit, are compared here. In the indirect case, the agents carry out tasks and update their thresholds as outlined in Table 1. They do not explicitly communicate among themselves. Such communication techniques are what we call indirect communication. In explicit communication, ants briefly communicate about their status when they are close (either the distance between the agents is less than one body length or they are present at the DA to sense how much food is available there). This is inspired by the observation that in many ant species (e.g. the red harvester ants *Pogonomyrmex barbatus*) where when two ants are extremely close, they communicate with each other using their antennae to determine what task the other ant is carrying out by accessing their cuticular hydrocarbon profile [51].

In our model, if agents are communicating explicitly, not only do they follow the rules outlined in Table 1., they also follow two further rules: (1) If agents perceive the shouting chemical, then for a brief period of time, they explicitly pass on information about the shouting chemical to other agents they encounter with, causing those agents to reduce their t_{bc}., (2) Similarly, if the amount of food in the dump area becomes less than the lower threshold of food in DA (the agents perceive the amount of food in DA by the chemical strength of the food in DA), the agents who perceive this would pass on information (for a brief period of time) to other agents they encounter about this incident which causes the agents to reduce their t_f.

4 Experiments and Results

Each simulation runs 20 times using the behavioural rules discussed in section 3 for 5000 simulation time steps. For every run, the following readings are recorded after the 5000th simulation time step: the number of agents (1) foraging, (2) brood caring and (3) resting, (4) the number of food items left at the DA, (5) the initial hunger level i.e. the hunger level at $t=0$ and (6) the final hunger level i.e. the hunger level at $t=5000$ and their mean is evaluated after the 20 runs. The hunger level, here, is the cumulative hunger level per unit brood member, which is used to measure the performance of the system and is calculated using the equation 6.

$$hunger\,level=\frac{\sum_{\forall brood} HL}{A\times n_b} \qquad (6)$$

Where, HL is the hunger level of each brood member (as mentioned in Section 3), A is some constant (for experimental purposes $A=th_h=500$ [also refer to equation (2)]) and n_b is the number of brood. Figure 6 shows a typical curve for HL with respect to time.

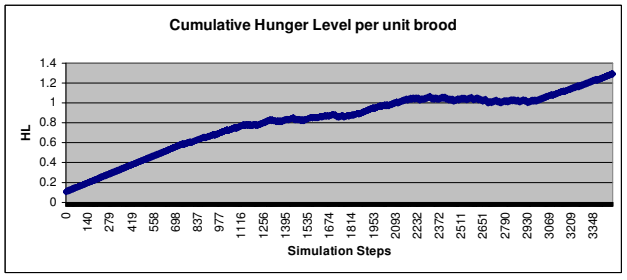

Figure 6. A typical HL curve as a function of time

The following analyses the two questions posed earlier i.e. (1) whether the behavioural rules of the agents mentioned here result in a system that can adapt to the changing demand?, and (2) whether the employment of explicit communication results in an improvement over the indirect communication with respect to the performance of the system?

A. Is the system adaptive?

One of the key motivations for taking inspiration from living swarm systems is that these systems are highly adaptive and flexible. In SR and other engineering applications of swarm intelligence, the design of such adaptive system is extremely crucial.

In this situation, it is necessary that foragers and brood carers continuously change their tasks in order to meet the changing demand[17,18]. This can be best investigated by analysing how the ratio of the two types of workers varies with the change in the demand. Figure 7 shows three curves plotted over the simulation time steps; red line at the top indicating the amount of food present in the dump area, violet line the number of hungry brood while the black line draws the ratio of the number of foragers to that of the brood carers. Initially there was no hungry brood and the amount of food in the DA was adequate; hence no demand was there. As a result, the ratio of the two types of workers did not need to be adjusted and hence remained constant. However, a demand for food by the brood instigates some foragers to switch their tasks to brood carers and assist other brood carers in feeding the brood. As the number of hungry brood members continues to increase and the amount of food in the DA continues falling it triggers more mobile agents to take up the foraging task in order to meet the demand.

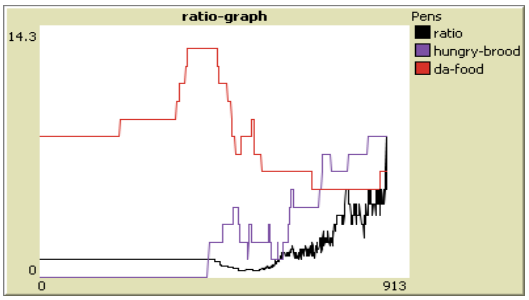

Figure 7: Switching of tasks between foragers and brood carers

B. Explicit Communication versus Indirect Communication

To compare and investigate the impact of using explicit communication and that of the indirect communication on the performance of the system and also to investigate whether explicit communication has any significant improvement in terms of the performance of the system over that of the indirect communication, the number of brood members is varied between 5 and 20 and for each of these cases, the experiment is repeated 20 times. The mean hunger level is then calculated. The experiment is repeated for both the conditions (i.e. with explicit and indirect communication). Table 2 and figure 8 depict the results obtained.

Number of brood	Mean Hunger Level (with Explicit comm.)	Mean Hunger Level (with Indirect comm.)
5	1.031596	1.155916
10	1.245766	1.680475
15	1.470009	2.128418
20	1.796184	2.305359

Table 2. Mean Hunger level

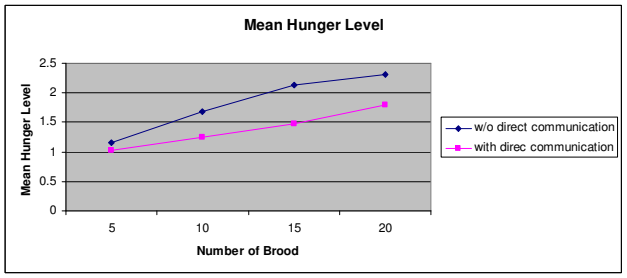

Figure 8. Variation of mean hunger level with the number of brood

Figure 8 shows that as the size of the brood increases, the mean hunger level also increases. More intriguingly, the mean hunger level in the explicit communication in general tends to be lower than the one where indirect communication has been used indicating that the system performance is improved by the inclusion of explicit communication. Regression analysis reveals that the slope of the graph (i.e. the rate of the mean hunger level with respect to the size of the brood) in indirect communication is 1.4 times greater than that in the explicit communication indicating that the performance keeps on improving with explicit communication with the size of the brood. Furthermore, a statistical t test with 95% confidence reveals that the dataset with explicit communication is significantly different from that with indirect communication except when the number of brood = 5 where the mean hunger levels for the two sets of experiment are not significantly different.

C. Performance of the System

Performance is measured in terms of the mean hunger level of the brood members. The higher the mean hunger level is, the lower is the performance of the system and vice versa. If the simulation is run for considerable time, the mean hunger level stays greater than 1. The performance of the system in percentage (%) is calculated using equation (7) [Also see equation (6)].

$$P = \frac{100}{hunger\,level} = \frac{100 \times A \times n_b}{\sum_{\forall brood-members} HL} \qquad (7)$$

Figure 9 shows the performance of the system for both explicit and indirect communication and indicates that the performance of the system is always better with the explicit communication than that of indirect communication.

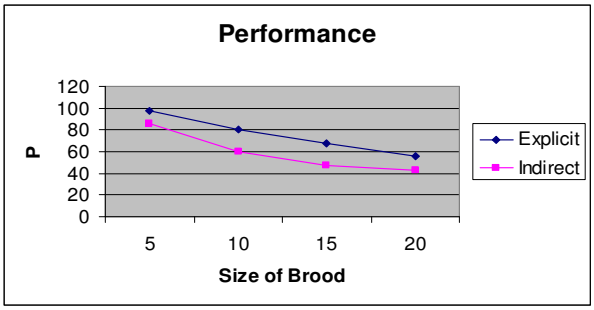

Figure 9: Performance of the system

5. Conclusions and Future Work

We have presented a task allocation and task switching model for a simulated swarm of autonomous mobile robots. The model employs a threshold based approach for adapting to changing demands and is strongly inspired by the behaviour of eusocial insects. Simple local rules have been developed for the agents that allow them to self organise and adapt to the changing environment as needed by the colony. Furthermore two communication techniques (indirect and explicit communications) have been presented and compared. Both the techniques work well in terms of responding to the changing demand; however empirical investigation shows that agents engaged in explicit communication work better than that of the ones engaged with indirect communication. Future work will be based on carrying out experiments to see the impact of the food availability and the swarm size on the performance as well as the robustness of the system. We also plan to develop a hybrid system of specialised and generalised agents to see if such system has any advantage(s) over the ones presented here.

REFERENCES

[1] D.J.T. Sumpter. The principles of collective animal behaviour. *Phil. Trans. R. Soc.* 361:5-22 (2006).

[2] S.Camazine, J.L.Deneubourg, N.R.Franks, J.Sneyd, G.Theraulaz, E.Bonabeau. Self-organization in biological systems. *Princeton University Press* (2001).

[3] W.Hamilton. Geometry of the selfish herd. *Journal of theoretical biology*. 31:295-311 (1971).

[4] J. Krause and G.D. Ruxton. Living in groups. *Oxford University Press* (2002)

[5] D.Helbing, P.Molnár, I.J.Farkas, K.Bolay. Self-organizing pedestrian movement. *Environment and Planning B: Planning and Design.* 28:361-383 (2001).

[6] D.J. Daley and D.G. Kendal. Stochastic Rumours. *J.Inst.Maths.Applics*. 1:42-55 (1965).

[7] Z.Neda, E.Ravasz, Y.Brechet, T.Vicsek, A.L.Barabasi. The sound of many hands clapping. *Nature*. 403:849 (2000).

[8] Z.Neda, E.Ravasz, T.Vicsek, Y.Brechet, A.L.Barabasi. Physics of the rhythmic applause. *Phys. Rev. E* 61:6987-6992 (2000).

[9] M.Resnick. Turtles, termites and traffic jams: exploration in massively parallel microworlds. *MIT Press* Paperback edition (1997).

[10] A.Senghas and M.Coppola. Children creating language: How Nicaraguan sign language acquired a spatial grammar. *Psychological Science*. 12:323-328 (2001).

[11] A.J.C. Sharkey. Swarm Robotics and Minimalism. *Connection Science*. 19(3):245-260 (2007)

[12] A.J.C. Sharkey. Robots, Insects and Swarm Intelligence. *Artificial Intelligence Review*. 26:255-268 (2006).

[13] G.Beni. From Swarm Intelligence to Swarm Robotics. In Swarm Robotics (Eds) Erol Şahin and W.M. Spears. LNCS 3342. 10-20 (2005)

[14] G.Beni and J.Wang. Swarm intelligence in cellular robotic systems. Proceeding of NATO Advanced, Workshops on Robots and Biological Systems (1989).

[15] E.O. Wilson. The Insect Societies. *Belknap Press of Harvard University Press*. Cambridge, Massachusetts London, England (1971).

[16] S. Momen and A.J.C. Sharkey. An ant-like task allocation model for heterogeneous groups of robots. *IUSSI 2008: 4th European Meeting of the International Union for the Study of Social Insects*. La Roche-en-Ardenne, Belgium 117 (2008).

[17] S.Momen and A.J.C. Sharkey. An ant-like task allocation model for a swarm of heterogeneous robots. *SIAAS 2009, AISB convention.* Edinburgh, Scotland 31-38 (2009).

[18] S.Momen and A.J.C. Sharkey. Strategies of division of labour for improving task efficiency in multi-robot systems. *IEEE World Congress on Nature and Biologically Inspired Computing (NABIC'09)*, Coimbatore, India 672 – 677 (2009).

[19] D.E. Jackson and F.L.W. Ratnieks. Communication in ants. *Current Biology*. 16:570-574 (2006).

[20] R.Jeanson, J.F.Fewell, R.Gorelick, S.M.Bertam. Emergence of increased division of labour as a function of group size. *Behav. Ecol. Sociobiol*. 62:289-298 (2007)

[21] C.Anderson and D.W. McShea. Individual *versus* social complexity, with particular reference to ant colonies. *Biol. Rev*. 76:211-237 (2001).

[22] B. Hölldobler and E.O.Wilson. the ANTS. *Belknap Press of Harvard University Press*. Cambridge, Massachusetts London, England (1990).

[23] B. Hölldobler and E.O.Wilson. The Super-organism: The Beauty, Elegance and Strangeness of Insect Societies. *W.W. Norton* (2008).

[24] A.F.G. Bourke and N.R. Franks. Social Evolution in Ants. *Princeton University Press*. (1995).

[25] A. Smith. The Wealth of Nations. Books I-III. Reprinted: 1986. Penguin, Harmondsworth, UK (1776)

[26] E. Bonabeau, M.Dorigo, G.Theraulaz. Swarm Intelligence: From Natural to Artificial Systems. *Oxford University Press*. (1999).

[27] G.F. Oster and E.O.Wilson. Castes and ecology in social insects. *Princeton University Press*, Princeton. (1978).

[28] D.M. Gordon and B. Hölldobler. Worker longevity in harvester ants. *Psyche*. 94:341-346 (1987).

[29] K.K. Ingram, P.Oefner, D.M. Gordon. Task-specific expression of the foraging gene in harvester ants. *Molecular Biology*. 14:813-818 (2005).

[30] S.N. Beshers. Models of division of labour in social insects. *Annu. Rev. Entomol*. 46:413-440 (2001).

[31] R.E. Page and S.D. Mitchell. Self organization and adaptation in insect societies. In PSA vol. 2 (Eds) A. Fine, M.Forbes, L.Wessels. East Lansing MI: *Philos. Sci. Assoc.* 289-298 (1991).

[32] R.E. Page and S.D. Mitchell. Self organization and the evolution of division of labour. *Apidologie.* 29: 171-190 (1998).

[33] G.E. Robinson and R.E. Page. Genetic basis for division of labour in an insect society. In *The Genetics of Social Evolution* (Eds) M.E. Breed and R.E. Page. 61-80 (1989).

[34] E. Bonabeau, G. Theraulaz, J.L. Deneubourg, Quantitative study of fixed threshold model for the regulation of division of labour in insect societies. *Proceedings of the Royal Society of London* B. 263:1565-1569 (1996).

[35] E.Bonabeau, G.Theraulaz, J.L.Deneubourg. Fixed response threshold and regulation of division of labour in insect societies. *Bull. Math. Biol.* 60:753-807 (1998).

[36] C.Tofts. Algorithms for task allocation in ants (a study of temporal polyethism: theory). *Bull Math. Biol.* 55:891-918 (1992).

[37] C.Tofts and N.R. Franks. Doing the right thing – ants, honeybees and naked mole-rats. *Trends Ecol. Evol.* 7:346-349 (1993).

[38] T.H. Labella. Division of labour in groups of robots. PhD thesis. Universite Libre de Bruxelles (2007).

[39] T.H. Labella, M.Dorigo, J.L. Deneubourg. Efficiency and task allocation in prey retrieval. *BioADIT* 2004 (Eds) A.J. Ijspeert et.al. LNCS 3141 274-289 (2004).
[40] J.L.Deneubourg, S.Goss, J.M.Pasteels, D.Fresneau, J.P. Lachaud. Self-organization in ant societies (III): Learning in foraging and division of labour. In *From individual to collective behaviour in social insects* (Eds) J.M. Pasteels and J.L. Deneubourg. Experentia Supplementum. 54: 177-196 (1987).
[41] R. Full. Learning from the Gecko's tail. TED talk. http://www.youtube.com/watch?v=d3syTrElgcg&feature=channel. (2009).
[42] M.J.B. Kreiger and J.B. Billeter. The call of duty: Self organised task allocation in a population of up to twelve mobile robots. *Robotics and Autonomous Systems*. 30:65-84 (2000).
[43] W. Liu, A.F.T. Winfield, J.Sa, J.Chen, L.Dou. Towards energy optimisation: Emergent task allocation in a swarm of foraging robots. *Adaptive Behaviour*. 15(3):289-305 (2007).
[44] W. Liu, A.F.T. Winfield, J.Sa, J.Chen, L.Dou. Strategies for energy optimisation in a swarm of foraging robots. (Eds) E. Şahin, W. Spears and A.F.T. Winfield. *Swarm Robotics 2006*. LNCS 4433. 14-26 (2007).
[45] J.H. Chuang. Potential based modelling of three-dimensional workspace for obstacle avoidance. *IEEE Transactions on Robotics and Automation*. 14(5): 778-785 (1998).
[46] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *Proc. IEEE ICRA*. 500-505 (1985).
[47] I.Mir, B.P. Amavasai and S. Meikle. Incremental perception in robotic swarms. *IEEE INMIC '06*. 427-432 (2006).
[48] D.M. Gordon. Ants at work: how an insect society is organized. *The Free Press* (1999).
[49] C.W.Reynolds. Flocks, herds and schools: a distributed behavioural model. *Computer Graphics*. 21:25-33 (1987).
[50] S.Momen, B.P. Amavasai, N.H.Siddique. Mixed species flocking for heterogeneous robotic swarms. *IEEE Eurocon 2007: The International Conference on Computer as a tool*. 2329-2336 (2007).
[51] M.J. Greeene and D.M. Gordon. Social Insects: Cuticular hydrocarbons inform task decisions. *Nature*. 423:32 (2003).

Application of CACS approach for distributed logistic systems

Sami AL-MAQTARI[1], **Habib ABDULRAB**[1], **Eduard BABKIN**[1 2]

Abstract. The article offers original approach which is called Controller Agent for Constraints Satisfaction (CACS). That approach combines multi-agent architecture with constraint solvers in the unified framework which expresses major features of Swarm Intelligence approach and replaces traditional stochastic adaptation of the swarm of the autonomous agents by constraint-driven adaptation. We describe major theoretic, methodological and software engineering principles of composition of constraints and agents in the framework of one multi-agent system, as well as application of our approach for modelling of particular logistic problem.

1. INTRODUCTION

Simultaneous rapid grow of logistics market in different regions of the world [1, 2], and its important role in modern economy require wide application of logistics information and management systems for coordinated planning and control. Distributed organizational structure and application of holonic management principles in modern organizations inevitably determine distributed and autonomous features of information systems supporting logistic operations [5]. In such kinds of the systems it is very difficult to apply usual centralized approaches and algorithms for decision support and optimization.

Swarm Intelligence [3, 4] represents one of the interesting paradigm for maintaining self-organization and control in the distributed systems. One of the principal aspect of the swarm-oriented distributed intelligent systems is presence of multiple intellectual and autonomous particles which interact with each other in some way. As it is started in [4]: "Swarm is a population of interacting elements that is able to optimize some global objectives thought collaborative search in space".

Different projects offered approaches for practical application of Swarm Intelligence paradigm in the form of multi-agent systems [6, 28, 30]. Although some of them (i.e. [28]) offer a formal framework for declarative expression and analysis, researchers and practitioners still lack proper generic methods for engineering of the multi-agent systems which have such properties of Swarm Intelligence as emergent behavior, peer-to-peer communication, etc.

Analysis of known logistic problems and algorithms shows that in the domain of applied logistics and optimization general principles of swarm-oriented organization may be realized using proper combination of multi-agent systems (MAS) and constraints satisfaction approach (CSP). So, in this research we pursue the goal to offer a new mechanism of emergent multi-agent behaviour for collaborative search of some feasible solution in accordance with certain inter-agent constraints. In terms of Swarm Intelligence research we replace stochastic adaptation of the swarm of the autonomous agents by constraint-driven adaptation.

In our research we try to satisfy such important requirements of Swarm Intelligence as self-organization and dynamic adaptation to evolving internal or external conditions. Existing approaches to combination of MAS and CSP like [16, 17, 32] do not provide much flexibility and support of dynamic modification of the combined structure of agents and constraints. That's why in this article we propose an original approach which offers a solution for dynamic modification of the combined structure of agents and constraints. Our approach, which was called CACS (Controller Agent for Constraints Satisfaction), allows for joint exploitation of attractive features of the paradigm of multi-agent systems (MAS) and the paradigm of distributed constraint satisfaction (DCSP).

This paper extends and combines our earlier work on joint application of MAS and DCSP paradigms [33, 34]. We describe major theoretic, methodological and software engineering principles of composition of constraints and agents in the framework of one multi-agent system, as well as application of our approach for modelling of particular logistic problem.

The paper is organized as follows. In Section 2 we give background information about MAS and DCSP for better understanding of scientific and technological foundations of our research. In Section 3 we describe main principles of CACS approach. Section 4 contains description of software architecture and implementation principles for software prototype which supports proposed CACS approach. The same section contains overview of used 3d party software platforms. Section 5 describes proposed methodology of practical application of CACS during design and development of DSS. In Section 6 we give overview of the application in ship loading logistics based on CACS prototype. We discuss the achieved results and provide directions for future work in Section 7.

2. FOUNDATIONS OF MAS AND DCSP

Paradigm of swarm intelligence is very often and naturally implemented on the basis of multi-agent systems. These systems express major features of collective intelligence [7, 8, 9] and represent the model of problem in terms of autonomous entities that live in a common environment and who share certain resources. The interactions between these individual entities induce cognitive abilities of the whole. Despite multiple-domain-oriented peculiarities majority of multi-agent systems has several significant common features:

[1] LITIS Laboratory, INSA Rouen, France. Email: {**almaqtari**, **abdulrab**}**@insa-rouen.fr**.

[2] TAPRADESS Laboraotry, State University – Higher School of Economics, Nizhny Novgorod, Russia. Email: **eababkin@hse.ru**.

• A limited and local view: every entity has a partial and local knowledge of its environment.
• A set of simple rules: each entity follows a set of simple rules.
• The interactions are manifold: each individual entity has a relationship with one or more other individuals in the group.
•The emerging structure is useful to the community: different entities are a benefit to work (sometimes instinctively) and their performance is better than if they had been alone.

From these points of view, the paradigm of multi-agent systems seek to simulate the coordination of autonomous entities called agents that represent individuals in their community. An agent is an entity that can be viewed as perceiving and acting independently in its environment. According to J. Ferber [10] "One agent called a physical or virtual:

1) which can act in an environment,
2) that can communicate directly with other agents,
3) which is driven by a set of trends (in the form of individual objectives or function of satisfaction and even survival, it seeks to optimize),
4) which has its own resources,
5) which is able to collect (but limited) its environment,
6) which has only a partial representation of this environment (and possibly none),
7) has expertise and provides services,
8) which may be repeated,
9) whose behavior tends to meet its objectives, taking into account the resources and skills available to it and according to its perception, its representations and the communications it receives. "

Given such definition of the agent, we can define a multi-agent system as a set of agents located in a certain environment. They share some common resources, and they interact with each other either directly or indirectly (via their effects on the environment). They seek to achieve the goals of individual agents in the interest of all. The multi-agent systems have applications in the field of artificial intelligence, where they reduce the complexity of solving a problem by dividing the necessary knowledge into sub-units, involving an intelligent agent independent at each of these sub - sets and coordinating the activity of these agents [10].

Because general definitions of inter-agent interaction are too vague we need to apply more strict and formal conventions to express allowable methods of communication between agents. Paradigm of constraints satisfaction, particularly distributed constraints satisfactions, offers flexible and convenient foundations to do this.

The paradigm of constraints satisfaction provides a generic method for declarative description of complex constrained or optimization problems in terms of variables and constraints [12, 13]. Formally, a Constraint Satisfaction Problem (CSP) is a triple (V, D, C) where:

There is $V = \{v_1, \ldots, v_n\}$ is a set of n variables,

a corresponding set $D = \{D(v_1), \ldots, D(v_n)\}$ of n domains from which each variable can take its values from,

and $C = \{c_1, \ldots, c_m\}$ is a set of m constraints over the values of the variables in V. Each constraint $c_i = C(V_i)$ is a logical predicate over subset of variables $V_i \subseteq V$ with an arbitrary arity k : $c_i(v_a, \ldots, v_k)$ that maps the Cartesian product $D(v_a) \times \ldots \times D(v_k)$ to $\{0, 1\}$. As usual the value 1 means that the value combination for $v_a, \ldots, v_k$ is allowed, and 0 otherwise. Constraints involving only two variables are called *binary constraints* [14]. A binary constraint between x_i and x_j can be denoted as c_{ij}. Although most of real world problems are represented by non-binary constraints, most of them can be transformed into binary ones using some techniques such as the dual graph method and hidden variable method [15]. Translating non-binary constraints into binary ones allows processing the CSP using efficient techniques adapted only for binary constraints. However, this translation implies normally an increase in number of constraints.

A solution for a CSP is an assignment of values for each variable in V such that all the constraints in C are satisfied. A single solver supports the tasks of collecting all data of the problem: variables, domains and constraints. It treats all such information in a centralized manner.

A Distributed Constraint Satisfaction Problem (DCSP) is a CSP where the variables are distributed among agents in a Multi-Agent System and the agents are connected by relationships that represent constraints. DCSP is a suitable abstraction to solve constrained problems without global control during per—to-peer agent communication and cooperation [16]. A DCSP can be formalized as a combination of (V, D, C, A, ∂) described as follows:

V, D, C are the same as explained for an original CSP,

$A = \{a_1, \ldots, a_p\}$ is a set of p agents,

and $\partial : V \rightarrow A$ is a function used to map each variable v_j to its owner agent a_i.

Each variable belongs to only one agent, i.e. $\forall\, v_1, \ldots, v_k \in V_i \Leftrightarrow \partial(v_1) = \ldots = \partial(v_k)$ where $V_i \subset V$ represents the subset of variables that belong to agent a_i. These subsets are distinct, i.e. $V_1 \cap \ldots \cap V_p = \varnothing$ and the union of all subsets represents the set of all variables, i.e. $V_1 \cup \ldots \cup V_p = V$. The distribution of variables among agents divides the set of constraints C into two subsets according to the variables involved within the constraint. The first set is the one of intra-agent constraints C_{intra} that represent the constraints over the variables owned by the same agent $C_{intra} = \{C(V_i) \mid \partial(v_1) = \ldots = \partial(v_k), v_1, \ldots, v_k \in V_i\}$.

The second set is the one of inter-agent constraints C_{inter} that represents the constraints over the variables owned by two or more agents. Obviously, these two subsets are distinct $C_{intra} \cap C_{inter} = \varnothing$ and complementary $C_{intra} \cup C_{inter} = C$.

The variables involved within inter-agent constraints C_{inter} are denoted as *interface variables* $V_{interface}$. Assigning values to a variable in a constraint that belongs to C_{inter} has a direct effect on all the agents which have variables involved in the same constraint. The interface variables should take values before the rest of the variables in the system in order to satisfy the constraints inside C_{inter} firstly. Then, the satisfaction of internal constraints in C_{intra} becomes an internal problem that can be treated separately inside each agent independently of other agents. If the agent cannot find a solution for its intra-agent constraints, it fails and requests another value proposition for its interface variables. To simplify things, we will assume that there are no intra-agent constraints, i.e. $C_{intra} = \varnothing$. Therefore, all variables in V are interface variables $V = V_{interface}$.

Many techniques are used to solve DCSPs. In general the technique proposes a distributed algorithm which is executed by

agents that communicate by sending and receiving messages. In general, the messages contain information about assignments of values to variables and rebuttals trust by employees who have no purpose compatible with their own variables. Mainly we mention the Asynchronous Backtracking (ABT) algorithm that was proposed by M. Yokoo [17] and some of its alternatives [18, 19, 20]. These approaches are designed mainly for the treatment of non-binary constraints, however most systems of real constraints are non-binary. Only a few modifications, like [21], were proposed to handle non-binary constraints in the dynamic organization of agents.

3. FUSION OF MAS AND DCSP IN CACS APPROACH

In order to avoid shortcomings of known DSCP methods and propose new principles of combination between MAS and DCSP we developed several software engineering methods and algorithms which comprise a new approach for developing DSS. This approach was called Controller Agent for Constraints Satisfaction (CACS). Based on the ABT Algorithm of M. Yokoo [17] CACS approach introduces two types of agents in MAS: Variables' Agent and Controller Agent.

In one hand, a Variables' Agent holds one variable or more. It chooses its values and proposes these values to Controller Agents. On the other hand, Controller Agent encapsulates inter-agents constraints over these variables. Each Controller Agent holds one constraint or more and validates the propositions received from Variables' Agents.

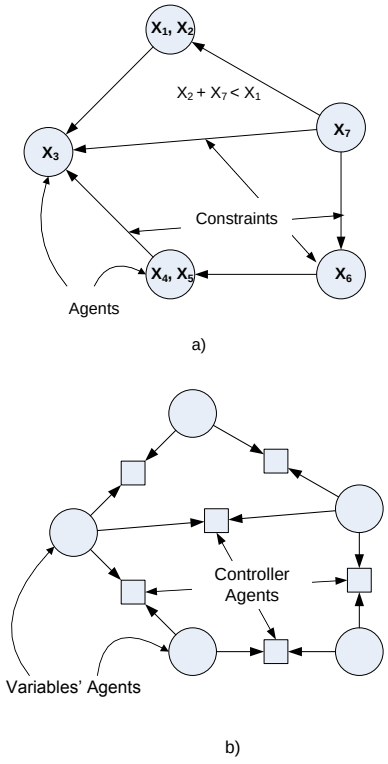

Figure 1. A constraint network example: a) without or b) with Controller Agent

We can see in Figure 1 (a) an example of constraint network where Variables' Agent are inter-connected by arcs which represent constraints. These inter-agent constraints are encapsulated in Figure 1 (b) by Controller Agents. The same network can be modified as in Figure 2 by grouping some inter-agent constraints inside a controller agent. With this ability, we can change the scale of constraints grouping from total distribution to total centralization. The problem can vary from designating a controller agent for each constraint to total centralizing by gathering all constraints inside one central controller agents.

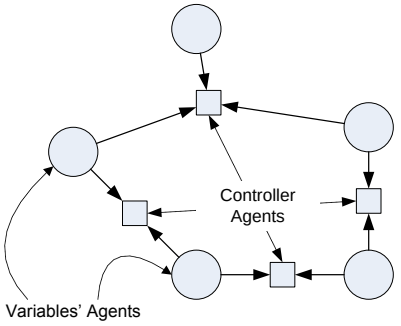

Figure 2. Grouping constraints inside Controller Agents.

For abbreviation purposes we will use the term *VAgent* to refer to Variables' Agents and *CAgent* to refer to Controller Agents. In fact, these terms are used as the name of classes used in the implementation of the prototype. The complete DCSP is formulated in terms of VAgents and CAgents. The solution of the problem is seeking during communication between these types of agents. The proposed algorithm of communication is divided into two stages: (1) domain reducing stage and (2) value proposing and validating stage. These stages are explained as follows:

A. Domain reducing stage

This stage assures constraints consistence by preprocessing variables' domains. The results are reduced domains by eliminating values that would be surly refused by them. This is done as follows:

1. A VAgent sends information concerning the domain of its variable to all linked CAgents. The message takes the form of (variable, domain).
2. After receiving the domains of all variables involved in its constraint, the CAgent uses consistency algorithms [22] in order to reduce these domains to new ones according to its local constraint(s). Then, the controller sends these domains back to their VAgents.
3. Every VAgent receives the new domains sent by CAgents and combines them (by the intersection of received domains) in order to construct a new version of its variable domain.
4. If any new version of a variable domain was empty then we can say that this DCSP is an *over-constrained* problem [23] where no solution can be found. In this case, the system signals that no solution was found (failure). As a prospective, another solution can be investigated by using constraints relaxation [23, 24], in which a VAgent returns

to an older version of the domain and reconstruct a new version after neglecting the domains sent by the CAgent that represents the soft constraints that the system may violate according to certain constraint hierarchy [23]. On the other hand, if all variables end with single-value domains then one solution is found. Otherwise, the domain reducing stage is repeated as long as we obtain a different new version of a variable domain. When domain reducing is no longer possible (no more change in variables' domains), we can proceed to the next stage.

The result of the domain reducing stage may be one of the three following kinds: 1) The domain of a variable is reduced to an empty field. Having at least one empty domain for a variable means the problem is over-constrained. If there is no solution that satisfies all the constraints and which contains a value for this variable. 2) The former is reduced to a new domain. This reduction may be the result of responses to a controller or more. This change must be propagated to other controllers. For this, the final stages must be repeated. 3) No change in the domain for this particular variable. In this case, we are faced with two situations: a) there are no changed domains at all. This means that the stage is over and we can proceed with the next stage. b) a change to succeed because of the spread of change in the domain of other variables. These variables can be linked directly or indirectly to the variable concerned.

A. Value proposing and validating stage

In this stage VAgents make their propositions of values to related CAgents to be tested. Value proposing can be considered as a domain information message in test mode. A test mode means that when a "no-solution" situation occurs because of a proposition the system backtracks to the last state before that proposition. This proceeds as follows:

1. From now on, every VAgent starts instantiating values for its variable according to the new domains. It sends this proposition to the related CAgents.
2. The CAgent chooses the value received from the VAgent with the highest priorities. This value is considered as a domain with a single value. CAgent uses consistency algorithms as in the previous stage to reduce other variables' domains. These new domains are sent to their VAgents to propagate domains change. This step may be viewed as a distributed form of forward checking in an enhanced backtracking algorithm.
3. Like in the previous stage, if all variables end with single-value domains then one solution is found. Unlikely, if the result of this propagation was an empty domain for any variable then the proposed value is rejected and another value is requested. If no more value can be proposed then system signals a no-solution situation to user.
4. If the result of the domain propagation was some new reduced domains with more than one value then steps 1-3 are repeated recursively with the value proposed by the VAgent that have the next priority.

The second stage involves one of three situations: 1) The proposed value is rejected if the spread of this value gives an empty domain for one variable at least. The refusal of a value involves retraction of the former domain and demand for another value. 2) Otherwise, the proposed value is accepted and distributed among the agents. The proposal and validation of values for the other variables continue recursively. 3) If there are more values to be proposed for a variable, the value proposed by the agent who has a higher priority is denied. The algorithm ends in failure when the agent has more priority over proposals valid.

Let's consider an example of MAS where three variables *x*, *y*, *z* with original permitted domain {0, 1, 2} are distributed on three VAgents A_1, A_2 and A_3, and two constraints exist: $x \neq y$ and $x + y < z$. These constraints are placed into two CAgents C_1 and C_2.

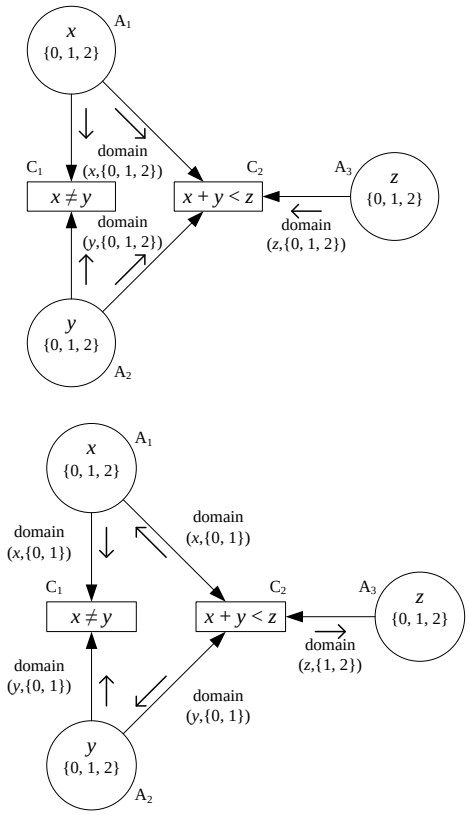

Figure 3. Illustration of domain reducing stage of CACS.

During the first stage of CACS (fig.3) three agents A_1, A_2 and A_3 are sending the domain { 0, 1, 2} for the three variables *x*, *y* and *z* respectively agents C_1 and C_2. C_1 tries to reduce the domains of *x* and *y*. Obviously, no change is possible. On the contrary, the agent C_2 changes the domains of variables *x* and *y* in {0, 1} and the domain of *z* in {1, 2}. This change will be propagated to the agent C_1 which returns the same domains for variables *x* and *y* (i.e. {0, 1}). The domain reducing stage finishes with the domain {0, 1} for the variables *x* and *y* and the {1, 2} for the variable *z*.

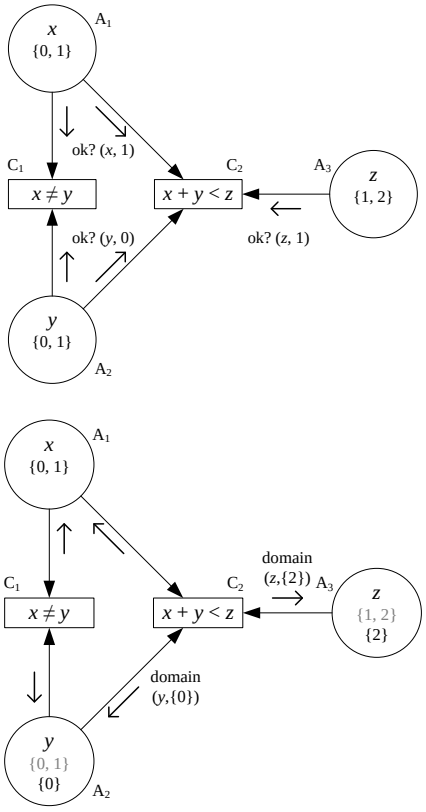

Figure 4. Illustration of value proposing and validating stage of CACS.

During the second stage (fig.4), algorithm will assign priorities to the agents A1, A2 and A3 according to their index. So the agent A1 will have the highest priority, and the agent A3 will have the lowest priority. Suppose that the agent A1 proposes value 0 for the variable x to the agents C1 and C2. C1 treats this value as the domain {0} and reduces the domain of the variable y to {1}. The spread of this new domain reduces the domain of the variable z to {2}. A2 tries to offer as the value 0 for variable y. His proposal will be of lower priority than the agent A1 and will be refused because they are inconsistent. The same result is obtained for any other value.

According to the results of the first and second stages, we can say that the CACS algorithm solves DCSP: 1) When the DCSP is over-constrained, we are faced with two different situations: Either the initial domains of the variables are inconsistent. This means that at the end of the first stage there is at least one empty domain of a variable. This involves termination of the algorithm and the declaration of a state of non-solution. Either the initial domains of the variables are consistent. 2) Where there is a unique solution of DCSP, we face two situations: The domains are consistent as long as there is a solution to the DCSP. If the first stage ends with single-vale domains, it means that the solution is found and the algorithm stops. Otherwise, in the second stage, the value proposed by a variable if it is not inconsistent with a value proposed by another agent with higher priority. The proposals of the agent with the highest priority are a priori accepted by all CAgents (it is necessary that this value is part of the final solution to be finally accepted). 3) When the DCSP is under-constrained, many solutions exist. The order of each proposed agent determines convergence towards any particular solution. In other words, the agents start the proposals by the most suitable for their purposes. For example, if an agent tries to minimize the value of its variable, it must begin proposing values from the minimum to the highest values.

4. SOFTWARE IMPLEMENTATION OF CACS

To prove the proposed methods of constraints satisfaction based on two types of the agents we developed an object-oriented CACS software prototype which can be considered as a generic framework for distributed information syste4ms in logistics. As we can see from Figure 5, the developed CACS prototype uses hierarchical multiple-layer architecture.

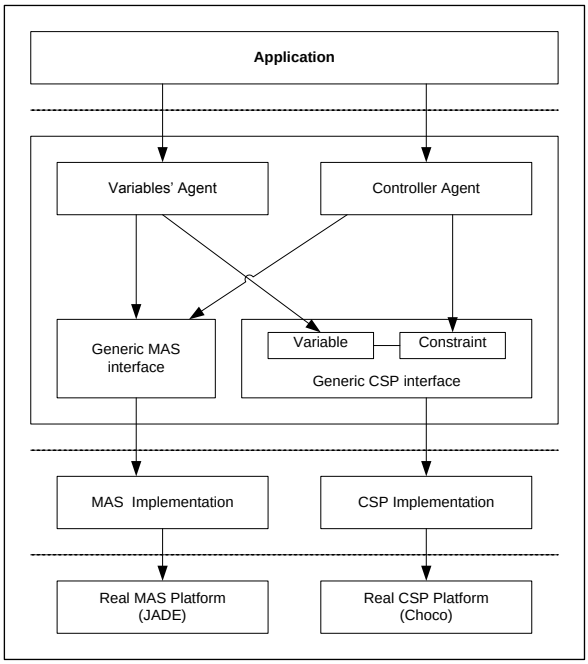

Figure 5. Software architecture of CACS prototype.

This architecture allows developing applications more flexibly by separating it into specialized layers. The very top layer is the application layer which is the implementation of a DCSP problem using the proposed system underneath it. From the application view point, the system is composed directly from the two principal types of agents: the CAgent and the VAgent. Both agents are inherited from CommonAgent class that defines some shared functionalities between both types of agents. The user can create the necessary VAgents according to its problem definition. He also creates the constraints and associates them to CAgents.

The second layer is the intended system (CACS) where our two-stage interaction algorithm is implemented in accordance with previous definition. Figure 7 shows the interaction between agents during the domain reducing stage. The interaction protocol is a loop of repeated domain informing from the VAgents side to CAgents side and new domain proposing as response. This loop is repeated until no further domain reduction

is possible (or an empty reduced domain is found which signify that there is no solution).

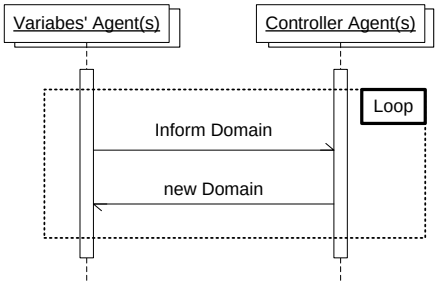

Figure 6. Implementation of interaction during domain reducing stage

The interaction between agents during value proposing stage is shown in Figure 6. as nest loops: the internal loop is similar to the domain reducing loop in Figure 6. Variables' domains are reduced according to the proposed value in the external loop. In the external loop, values are proposed and evaluated after the domain reduction to be either accepted or rejected. The external loop continues until we obtain single value domains for all variables.

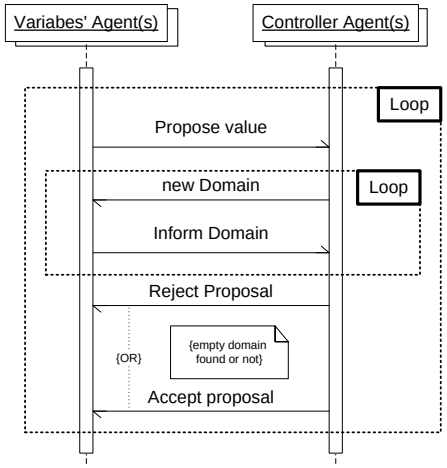

Figure 7. Implementation of interaction during value proposing stage

The system layer uses generic interfaces for both MAS and CSP platforms. This allows the system to use any existing MAS and CSP platforms by implementing these interfaces. At the same time this isolates the internal structure from the changes of choice of platforms. An intermediate layer between the system and the real MAS or CSP platform is necessary in order to separate the structure of the system from that of the real MAS and CSP platforms. This layer works as an adapter; it implements the generic platforms in the system layer using the real platforms. This implementation difficulty varies according to the MAS and CSP platforms used for the realization of the final system.

The whole CACS prototype was developed in Java language. Due to the object oriented nature of Java language agents and the messages are represented by objects (Figure 8, 9).

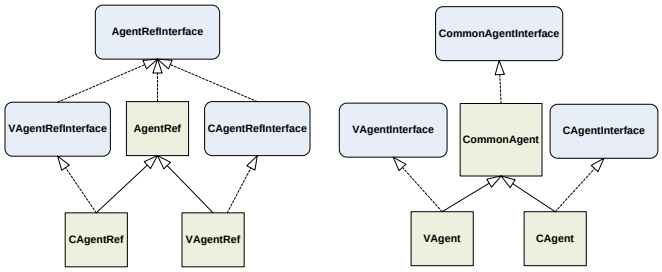

Figure 8. The hierarchy of the main components of agents (agents and reference to the agents). Rectangles with rounded corners represent interfaces; rectangles with sharp corners represent classes

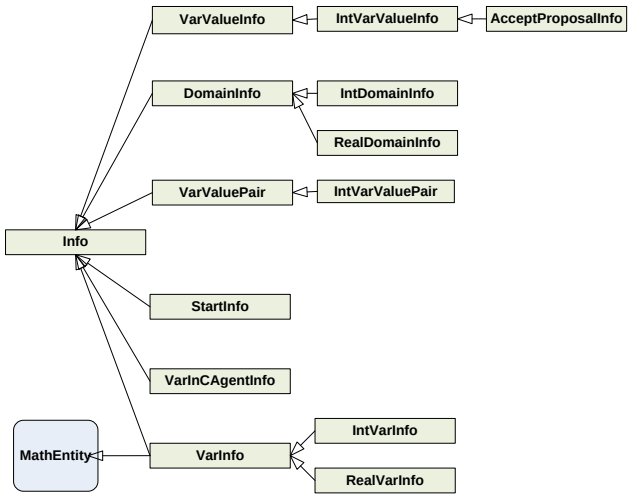

Figure 9. The hierarchy of agent messages. Rectangles with rounded corners represent interfaces; rectangles with sharp corners represent classes

However, from the point of view of Multi-Agent System design, agents should not be referenced by a simple public reference that is accessible by any other object in the system. The reason for that is to prevent any direct access to the agent internal functionality. Normally, references to agents should be kept hidden by the MAS platform and communicating with an agent is made by messages that would be delivered by the system using the agent address. Mapping from agent address to its real reference is an internal functionality of the MAS platform.

In order to be more generic, we distinguish in the prototype implementation between the agent and its reference. For this purpose, VAgentRef and CAgentRef classes have been designed. Both classes are inherited from the abstract AgentRef class. They are used as references to either variables' agents or controller agents. When an instance of the class DCSP is used to create an instance of VAgent or a CAgent, it returns an instance of either VAgentRef or CAagentRef classes respectively according to created agent. In the same manner, a variable inside

variables' agents cannot be referred directly. In fact, a controller agent keeps a copy of that variable inside it and propagates any change on that variable to the owner agent. Instead of dealing with variables directly between agents, they deal with variables identifiers. A variables identifier is an instance of VID class. It is simply the name of the variables and the identifier of its owner agent. An instance of VAgentRef is used to create variables inside the corresponding VAgent. A variable creation process returns an instance of VID class identifying the created variable.

Among additional features we added to our prototype a possibility to declaratively define a simple DCSP via the use of XML notation. The XML file that describes a DCSP problem should be built according to the following model (fig.10):

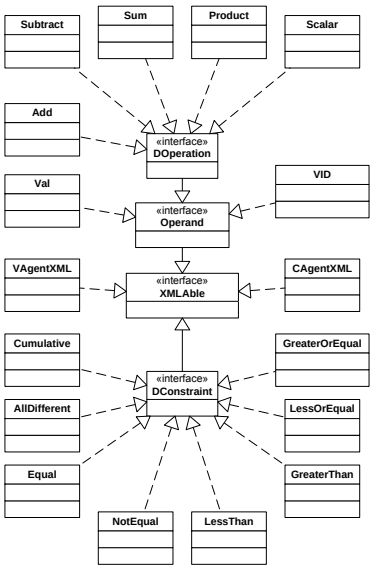

Figure 10. The hierarchy of the main components of agents (agents and reference to the agents)

The choice of multi-agent platforms and multi-solver constraints required a study and testing of several platforms. We reviewed our work over multiple platforms including JADE and Madkit and several constraints solvers as CHOCO, Cream and JCK. Finally we chose for the role of MAS JADE (Java Agent DEvelopment Framework) multi-agent framework [25], and for CSP platform, we have chosen Choco [26, 28].

JADE is a multi-agent framework compliant with the FIPA specifications [27] and is fully implemented in Java language. JADE was established by the laboratory TILAB Telecom Italia. JADE has three main modules (fig.11): DF (Directory Facilitator): provides a service of "yellow pages" to the platform; ACC (Agent Communication Channel) handles communication between agents; AMS (Agent Management System) oversees the registration of agents, authentication, access and use of the system. Each JADE agent is composed of a single thread of execution (thread). Each task agent is represented by an instance of class Behavior. Jade offers the possibility of agents' multi-threaded, although the user leaves the responsibility for managing competition (except the timing of the messages file ACLs).

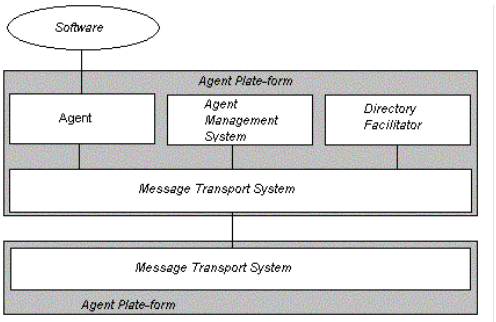

Figure 11. Architecture II software platform JADE

In order to implement a behavior, the developer must define one or more objects of class Behavior, the instantiate and add them to the thread of execution of the agent. Every object type has a Behavior method *action ()* (which is the treatment to be performed by it) and a method d*one ()* (which checks if the treatment is completed). In detail, the scheduler executes the method *action ()* of each object in the queue of the tasks of the agent. Once this is completed, the method *done ()* is invoked. If the task has been completed then the Behavior object is removed from the queue. The scheduler is non-preemptive and does only one behavior at a time, one can consider the method *action ()* as atomic. It is then necessary to take certain precautions during the implementation of the latter, to avoid endless loops or operations too long. The most classic program behavior is to describe it as a finite state machine. The current status of the agent is stored in local variables.

Also JADE simplifies the implementation of multi-agent systems through a set of graphical tools that supports the debugging and deployment phases.

Choco is a library for constraint satisfaction problems (CSP), constraint programming (CP) and explanation-based constraint solving (e-CP) [28]. It is built on an event-based propagation mechanism with backtrackable structures. Choco is implemented in Java and takes advantage of the principle of inheritance to allow the programmer to define its own types and constraints. This is achieved by using abstract classes (fig. 12):

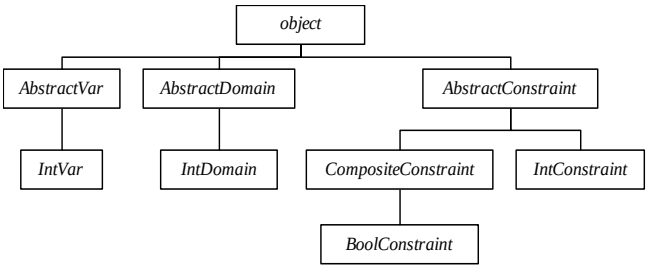

Figure 12. Hierarchy of constraints in Choco

It permits the use of multiple solvers for different problems separately. This allows each CAgent to have its own solver. A distributed constraint problem is created as an instance of the class DCSP. This instance represents the problem to be solved and is used to create the different needed agents.

Our prototype in its current state is composed of three main packages containing more than 80 classes and Java interfaces and approximately 4300 lines of code.

5. DESIGN METHODOLOGY IN CACS

A specific methodology was designed to allow the user to develop distributed multi-agenty systems using Swarm Intelligence paradigm and CACS approach. In general this methodology consists of the following steps:

1. Identify the key actors of the problem (VAgents). These actors are the entities of the system modeled.
2. Determine the properties (variables) of these actors that are restricted by constraints with properties of other actors.
3. Determine all the constraints of the problem.
4. Classify constraints logically in separate groups.
5. Specify a set of Controller Agents to monitor each group of constraints.

To provide a developer with flexible practical methods of the design we offer two refinements of the general methodology: simple and complex.

To prove the proposed methods of constraints satisfaction based on two types of the agents we developed an object-oriented CACS software prototype which can be considered as a generic framework for distributed model-driven DSSs. As we can see from Figure 5, the developed CACS prototype uses hierarchical multiple-layer architecture. The following steps correspond to a given DCSP:

6. Creation of the problem P. This is done by creating an instance of the class DCSP from the package dcsp.
7. Creation of agents to control variables (specifically, their references) via the prolem P. using the method *makeVAgent* () to create a variable and method *makeCAgent* () to create a controller. 3) Creating variables distributed via agents which own variables. This is done through the method *makeBoundedIntVar* () which creates a variable with two upper and lower limits.
8. Creation of constraints on variables.
9. Addition of constraints to CAgents.
10. Start the algorithm of resolution through the DCSP P.

The use of the prototype can be demonstrated via the following simple example:

$V = \{x, y, z\}$ is the set of variables from the domain $\{1, \ldots, 100\}$ for all of them, $C = \{c_1, c_2, c_3\}$ is the set of constraints:

$c_1 : x \neq y, y \neq z, x \neq z$ (or alldifferent (x, y, z))

$c_2 : x \geq y$

$c_3 : z \geq y$

In order to model this problem using the proposed prototype the user should proceed as follows. We start by assigning variables to VAgents. In this example, agents v1, v2, and v3 own variables x, y, and z respectively. Note that the distribution of variables may be a problem dependant issue which means that the user chooses the owner agent of each variable according to the problem specifications. In the same manner, constraints also should be assigned to CAgents. In this example, we assign each constraint to a CAgent.

1. Create a distributed problem p (an instance of DCSP class). This class will be used in order to create VAgents and CAgents and to start our CACS algorithm.

```
DCSP p = new DCSP("example");
```

This creates a distributed problem with which agents, variables and constraints will be created.

2. Use this instance to create both types of agents. This is done by calling makeVAgent() and makeCAgent() methods from the DCSP instance created in step 1 as follows:

```
VAgentRef v1 = p.makeVAgent ("v1");
VAgentRef v2 = p.makeVAgent ("v2");
VAgentRef v3 = p.makeVAgent ("v3");
CAgentRef c1 = p.makeCAgent ("c1");
CAgentRef c2 = p.makeCAgent ("c2");
CAgentRef c3 = p.makeCAgent ("c3");
```

3. Create variables inside VAgents. In other word, assign variables to variables agents. The method makeBoundedIntVar() is used to achieve this as follows:

```
VID x = v1.makeBoundedIntVar ("x", 1, 100);
VID y = v2.makeBoundedIntVar ("y", 1, 100);
VID z = v3.makeBoundedIntVar ("z", 1, 100);
```

4. Create the constraints and post them to CAgents. The constraints are created separately and posted to their owner agents using the method post():

```
c1.post(new AllDifferent(new VID[]{x,y,z}));
c2.post(new GreaterOrEqual(x, y));
c3.post(new GreaterOrEqual (y, z));
```

5. Start the CACS algorithm by calling solve() method from the DCSP instance:

```
p.solve();
```

This last instruction initiates communication between the different agents in the system in accordance the algorithm described previously in Section 3. If an agent finds a value for its variable that corresponds to a solution then it will notify to this value. The solution will be the combination of all values from all agents. Otherwise, no-solution state is declared.

Also the developer can express the structure of DCSP in declarative manner using XML. For instance, the problem described in previous sub-section can be written in XML as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE dcsp SYSTEM "dcsp.dtd">
<dcsp>
<name>example</name>
<vagent><name>v1</name><var><name>x</name>
  <inf>1</inf><sup>100</sup></var></vagent>
v2 and v3 by the same manner
<cagent>
  <name>c1</name>
  <constraint><alldiff>

<vid><name>x</name><owner>v1</owner></vid>

<vid><name>y</name><owner>v2</owner></vid>

<vid><name>z</name><owner>v3</owner></vid>
  </alldiff></constraint></cagent>
c2 and c3 by the same manner
</dcsp>
```

6. CACS APPROACH IN TRANSPORT LOGISTICS

We consider modern transportation problems as a natural candidate domain for evaluation of the proposed CACS approach. Although there is a lot of different centralized algorithms in this area we believe that multi-agent techniques can radically improve efficiency and fairness of negotiation between participants in the course of problem solving as well as improve reactivity of the logistics systems. Among different benefits of logistics management within the CACS framework we can point out such positive features as: better consideration of individual preferences and ability of their dynamical changes in the course of solving, early availability of partial solutions and inherently distributed structure of the system.

In order to create solid foundations for application of Swarm Intelligence and CACS approach in transportation logistics we developed a distributed multi-agent application which mimics major features of modern ship loading problems, and evaluated its feasibility and performance. Our CACS application is based on a simplified ship loading scenario which was originally presented in studied in Chips constraint solver by Kay Chips (Kay 1997) and later was expressed in terms of Java-based Choco constraint solver by prof. A. Aggoun.

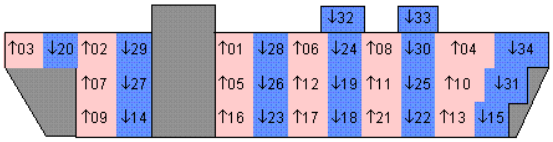

Figure 13. Graphical representation of the original Kay's ship loading problem (Kay 1997)

In the discussed problem a specific precedence function *pred* in defined over the loading items. For each of the items the number of workers needed for loading is specified.

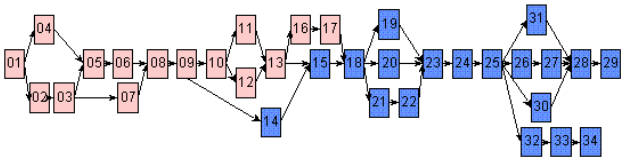

Figure 14. The feasible order of loading tasks (the loading plan) in accordance with the constraints given (Kay 1997)

According to CACS methodology each loading task is realized as a separate Variable Agent in our CACS application. Variable Agent holds three specific variables. These variables determine start time of loading (t^i_{start}), finish time of loading (t^i_{end}) and predetermined loading duration (d^i) accordingly.

All constraints of the considered problem are grouped inside Controller Agents. We recognize three different groups of Controller Agents according to the semantics of the constraints. The first group contains Controller Agents which hold duration constraints. The agent of that group is responsible for verifying that the loading tasks are scheduled within the time frame. It means that for each task i the following constraint should be satisfied: $t^i_{start} + d^i \le t^i_{end}$.

The second group contains Controller Agents which are responsible for verifying that the loading plan satisfies precedence constraints given (like one on the fig. 13). Finally the third group contains Controller Agents which are responsible for verifying availability of the resources for the loading plan. Controller Agent of that kind holds cumulative constraint over the number of workers available for finishing the ship loading within the total time.

That cumulative constraint may be expressed using current values of t^i_{start}, and t^i_{end} variables, as well predetermined workforce effort needed for each task w^i. Given these values we can define the scheduling matrix *SC*.

$$SC = \begin{bmatrix} 0 & 0 & w^1 & w^1 & w^1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & w^2 & w^2 & 0 & 0 & \dots & 0 \\ & & & & & & & & \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & w^N \quad w^N \end{bmatrix}$$

The element SC_{ij} is equal to w_i iff at the time moment j the loading task i is performed, and it is equal to 0 in the opposite case.

Using that matrix we may define the maximum number of workers needed at each moment of the time and the needed cumulative constraint: $\max_j \left(\sum_{i=1}^{N} SC_{ij} \right) \le \max Persons$.

With such problem interpretation we may completely describe it in terms of our CACS approach. The original structure of the agents is presented on Figure 15.

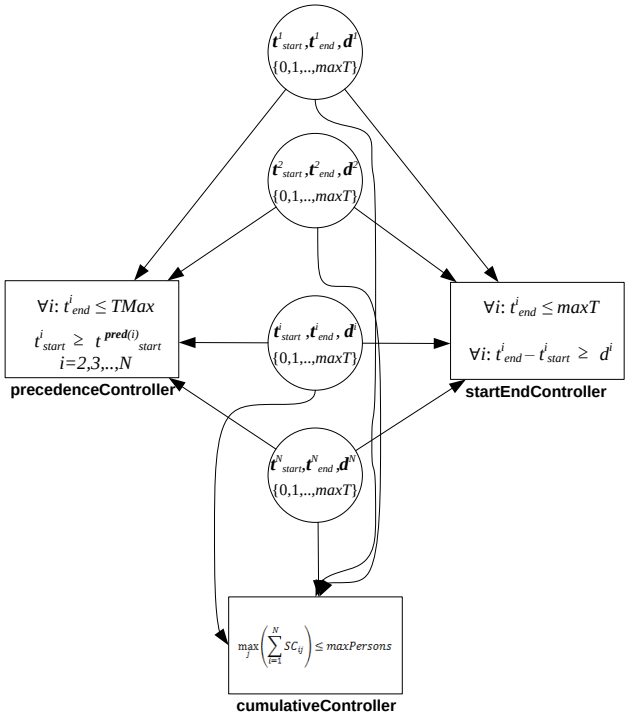

Figure 15. Connected structure of Variable Agents (circle) and Constraint Agents (rectangle) for the ship loading problem

Using the proposed methodology we designed the Java-based application that solves the ship loading problem. In that program at the first moment the ControllerAgents are created:

```
CAgentRef startEndController =
  dpb.makeCAgent("startEndController");
CAgentRef precendenceController =
  dpb.makeCAgent("precendenceController");
CAgentRef cumulativeController =
  dpb.makeCAgent("cumulativeController");
```

Then auxiliary VariableAgent is created which stores the total time of loading operations:

```
VAgentRef general = dpb.makeVAgent("General");
VID generalEnd = general.createVar("General_End",
  0, timeHorizon);
```

After that in the cycle thirty-four VariableAgents are created which correspond to the loading tasks and store needed variables t^i_{start}, t^i_{end} and duration d^i. In the same cycle the duration constraints are created and attached to the corresponding ControllerAgent.

```
for (int j = 0; j < nbTasks; j++) {
taskAgents[j] = dpb.makeVAgent("task_agent_" +
  (j + 1));
taskStarts[j] = taskAgents[j].
  createVar("Start", 0, timeHorizon);
taskEnds[j] = taskAgents[j].createVar("End",
  0, timeHorizon);
taskDurations[j] = taskAgents[j].
  createVar("Duration", durations[j],
  durations[j]);

DOperation startEndOperation = new
  Subtract(taskStarts[j], taskEnds[j]);
DConstraint startEndConstraint = new
  Equal(startEndOperation, taskDurations[j]);
startEndController.post(startEndConstraint);

DConstraint endConstraint =
  new LessOrEqual(taskEnds[j], generalEnd);
startEndController.post(endConstraint);
}
```

Finally the precedence constraint and cumulative constraint are determed and the process of solution search is started.

With the given conditions in the result of application run the values of variables t^i_{start}, t^i_{end} will constitute a feasible solution ot the ship loading problem.

7. CONCLUSIONS & FUTURE WORK

In this article we proposed a new approach for combination of MAS and DCSP in multi-agent swarm systems. This approach called CACS (Controller-Agent for Solving Constraints) based on the use of a specific type of agents called Agent Controller and Variables' Agent. We believe that proposed process of constraint satisfaction in the multi-agent system fits well the general principles of Swarm Intelligence. In particular, the stage of domain reduction in our algorithm may be seen exchange of "rules, tips and believes about how to process the information [4]".

Also in our approach we implemented a principal feature of Swarm systems, which is principal ability to modify multi-agent structure in response of various influencing factors. First of all, declarative manner of constraints based formalization of the problem allows for changing inter- and intra-agent behavior. Secondly, the composition of inter-agent constraints inside ControllerAgent may be changed during evolution of the system (as it shown on fig.1 and fig .2).

In the proposed CACS architecture we see good opportunities for further moving towards to implementation of advanced swarm intelligence capabilities. Modern MAS platforms like JADE implement different peer-to-peer communication mechanisms for which direct correspondence may be found in computational biology. Given such mechanisms as foundation for reliable distributed inter-agent communication we will extend discussed algorithms of interaction between Controllers Agents and Variables Agents by adaptation framework. In such framework agents will be able to discover critical changes in MAS configuration (faults of agents, misbehavior, etc), negotiate responsibilities and change the roles accordingly in order to continue proper collective operations.

The model of distributed constraints satisfaction proposed in SACS also offers two main contributions in DCSP research. First, it is the possibility of a direct and easier dealing with non-binary constraints without having to use methods of transformation of non-binary constraints to binary constraints. Second, CACS offers us the possibility to organize the constraints logically related groups. This grouping of constraints allows us to form sub-problems, each group is monitored and processed by a single controller. This also helps reduce the total number of Controller Agents needed.

Non-binary constraints are more common in real problems than binary ones. Some methods are used in order to allow using binary constraint solving techniques on non-binary ones. Methods like hidden and dual transformation [14, 15] convert non-binary constraints into equivalent binary ones. Other methods are proposed in the DCSP domain in order to deal with non-binary constraints. I. Brito [21, 33,34] has proposed organizing agents involved in a non-binary dynamically in order to form a proper propose-validate sequence. Agents then follow that sequence to find a solution for that constraint.

Our algorithm proposes another direct alternative. Any constraint is encapsulated inside a controller agent regardless this constraint is binary or non-binary. Agents involved in any constraint are not forced to follow any order in proposing values for their variables.

The increase in number of agents is an inconvenience of our model. We can investigate the possibility of using a hybrid system of both, our model and a standard ABT model, in order to model a DCSP. In such hybrid system, binary constraints relate variables' agents directly while non-binary constraints are encapsulated inside controller agents. The possibility of gathering constraints gives also the possibility of decreasing the number of agents. The user can group some constraints according to the modeled problem logic.

To prove the feasibility of the proposed theoretical principles we implemented software prototype of CACS. It uses generic interfaces for integration with different third-party MAS-

platforms and CSP-solvers. In the final implementation we used the MAS platform JADE and the Choco CSP solver. Apart from direct Java programming of DCSP problems our prototype also provides an opportunity to describe the problem using XML facilitating the modeling of simple problems without the need to write and compile a Java program.

Demonstrated applicability of CACS for solution of logistics problems opens opportunity for further progress in developing Swarm Intelligence applications. Following that direction we plan to continue in design of meta-communication protocol between ControllerAgents, which will permit define formal methods of re-composition of constraints inside different ControllerAgents during evolution of the system.

Another interesting problem for CACS application comes from the domain of modern transportation systems. Here we wish to apply CACS approach for the "transport on demand" challenge and solution of complex logistics problems in real conditions of modern warehouses. Also we are going to investigate ways to add optimization mechanism to the system similar to DPOP algorithm [34]. This will allow the user to adjust the Variables' Agent value choosing according to a given optimizing mechanism.

This work was partially supported by HSE grant # T3-61.1.

REFERENCES

[1] Feng, G., Yu, G., and Jiang, W. Logistics Management in China. Building Supply Chain Excellence in Emerging Economies. (Hau L. Lee, M. Eric Johnson eds.). pp.177-199. Springer US. (2007).

[2] Simonova. L. Growth Prospects of Russian Transportation and Logistics Market. Das Beste der Logistik:Innovationen, Strategien, Umsetzungen. pp.369-379. Springer Berlin Heidelberg. (2008).

[3] Bonabeau, E., Dorigo , M., and Theraulaz G. Swarm Intelligence: From Natural to Artifical Systems. New York, Oxford University press. (1999).

[4] Eberhart, R. C., Shi, Y., and Kennedy, J. Swarm Intelligence. Morgan Kaufmann. (2001).

[5] Hülsmann, M., Windt, K. Approaches to Methods of Autonomous Cooperation and Control for the Management-, Information- and Communication-Layer of Logistics. Understanding Autonomous Cooperation and Control in Logistics (Katja Windt and Michael Hülsmann eds.). pp. 163-167. Springer Berlin Heidelberg. (2007).

[6] Davidsson, P. Henesey, L., Ramstedt, L., Törnquist, J., and Wernstedt, F. Agent-Based Approaches to Transport Logistics. Applications of Agent Technology in Traffic and Transportation. pp.1-15. Birkhäuser Basel. (2005).

[7] Izquierdo-Torres, E. Collective Intelligence in Multi-Agent Robotics: Stigmergy, Self-Organization and Evolution. (2004).

[8] Atlee, T., Benkler, Y., Homer-Dixon, T., Levy, P., Malone, T., Martin, R. H. P., Masum, H., Steele, R. and Tovey, M. COLLECTIVE INTELLIGENCE: Creating a Prosperous World at Peace, Earth Intelligence Network. (2008).

[9] Bousquet, F. Modélisation d'accompagnement Simulations multi-agents et gestion des ressources naturelles et renouvelables. (2001).

[10] Ferber, J. Multi-agent Systems-An Introduction to Distributed Artificial Intelligence. Addison-Wesley, Boston, (1999).

[11] Charrier, R.; Bourjot, C.; Charpillet, F. A Nonlinear Multi-agent System designed for Swarm Intelligence: the Logistic MAS. SASO '07. First International Conference on Self-Adaptive and Self-Organizing Systems, 9-11 July. pp.32 – 44. (2007).

[12] Barták, R. Constraint Programming: In Pursuit of the Holy Grail, in Proceedings of WDS99 (invited lecture), pp. 555-564. (1999).

[13] Eisenberg, C. Distributed Constraint Satisfaction for Coordinating and Integrating a Large-Scale, Heterogeneous Enterprise, University of London. (2003).

[14] Bacchus, F., Chen, X., Beek, P. v., and Walsh, T. Binary vs. Non-Binary Constraints, pp. 1-37. (2002).

[15] Bacchus, F., Beek, P. v. On the Conversion between Non-Binary and Binary Constraint Satisfaction Problems. in Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98) and of the 10th Conference on Innovative Applications of Artificial Intelligence (IAAI-98), pp. 311-318. (1998).

[16] Havens, W. S. NoGood Caching for MultiAgent Backtrack Search. in Proceedings of American Association for Artificial Intelligence 1997 Constraints and Agents Workshop, (1997).

[17] Yokoo, M., Hirayama, K. Algorithms for Distributed Constraint Satisfaction: A Review. in Autonomous Agents and Multi-Agent Systems, pp. 198-212. (2000).

[18]Bessiere, C., Brito I., Maestre, A., and Meseguer, P. The Asynchronous Backtracking Family, LIRMM-CNRS, , Montpellier, France March 2003, (2003).

[19] Bessiere, C., Brito, I. Asynchronous Backtracking without Adding Links: A New Member in the ABT Family. pp. 7-24. (2005).

[20] Muscalagiu, I., Horia-Emil, P. and Panoiu, M. Asynchronous Backtracking with temporary and fixed links: A New Hybrid Member in the ABT Family. INFOCOMP - Journal of Computer Science vol.5: 29-37.

[21] Brito, I. and Meseguer, P. Distributed Stable Matching Problems with Ties and Incomplete Lists. In Proceedings of the Twelfth International Conference on Principles and Practice of Constraint Programming (CP-2006), Nantes, pp. 675-680, (2006).

[22] Zhang Y., Wu, H. Bound Consistency on Linear Constraints in Finite Domain Constraint Programming. In 13th European Conference on Artificial Intelligence Young Researcher Paper, (1998).

[23] Rudova, H. Constraint Satisfaction with Preferences. In Faculty of Informatics Brno - Czech Republic: Masaryk University, (2001).

[24] Mailler, R. and V. Lesser, V., A Cooperative Mediation-Based Protocol for Dynamic, Distributed Resource Allocation. In IEEE Transaction on Systems, Man, and Cybernetics, Part C, Special Issue on Game-theoretic Analysis and Stochastic Simulation of Negotiation Agents, New York, pp. 438-445, (2004).

[25] Bellifemine, F. L., Caire G. and Greenwood, D. Developing Multi-Agent Systems with JADE, Wiley. (2007).

[26] "CHOCO" : http://choco.sourceforge.net/. (2010).

[27] FIPA: http://www.fipa.org/specifications. (2010).

[28] Stamatopoulou, I., Kefalas, P., and Gheorghe., M. OPERAS: A Framework for the Formal Modelling of Multi-Agent Systems and Its Application to Swarm-Based Systems. LNCS 4995/2008. Engineering Societies in the Agents World VIII.pp.158-174, 2008. Springer Berlin / Heidelberg. (2008).

[30] Charrier, R., Bourjot, C., and Charpillet, F. Study of Self-adaptation Mechanisms in a Swarm of Logistic Agents. SASO '09. Third IEEE International Conference on Self-Adaptive and Self-Organizing Systems, 14-18 Sept. 2009.pp. 82 - 91. (2009).

[31] Brito, I. and Meseguer, P. Asynchronous Backtracking Algorithms for Non-binary DisCSP. In Workshop on Distributed Constraint Satisfaction Problems at the 17th European Conference on Artificial Intelligence (ECAI-2006), Riva del Garda, (2006).

[32] Petcu, A. Dynamic distributed optimization for planning and scheduling. AAAI-05 Workshop; Pittsburgh, PA. pp.52-53. (2005).

[33] Al-Maqtari, S., Abdulrab, H., and Nosary, A. Constraint Programming and Multi-Agent System mixing approach for agricultural Decision Support System. Emergent Properties in Natural and Artificial Dynamical Systems in ECCS'O5 International Conference, pp. 113-119.(2005).

[34] Al-Maqtari, S., Abdulrab, H., and Babkin, E. Towards a robust software framework for DCSP solving in multi-agent systems. The Second AIS SIGSAND - European Symposium on Systems Analysis and Design, 2007. (2007).

Swarm Intelligence to Distribute Simulations in Computational Ecosystems

Antoine Dutot* and **Damien Olivier*** and **Guilhelm Savin*†**

Abstract. This paper deals with distribution of complex system simulation. A first reflexion is conducted about the distributed environment and we show that it can be considered as a complex system *i.e* an artificial ecosystem, in other words a computational ecosystem. In a second time we propose to manage the complexity of the simulation and its environment by an algorithm based on swarm intelligence. Once again, this proposition is based on interactions and the mechanisms of cooperation and competition that help us to detect the self-organizations which evolve during the simulation in the distributed environment. In a last time, we outline a middleware allowing to distribute dynamically the simulation. This middleware allows mobile code and offers advices to the simulation to reduce the cost of communications. To do that we detect the self-organizations during the simulation and we facilitate their placement on a same node of the distributed system.

1 Introduction

In this paper, we are concerned by complex systems simulation distribution using swarm intelligence. We consider that the simulation evolves in an open distributed environment. We will demonstrate in the following that this environment is a "computational ecosystem". According to this characteristic, we can try to control the trajectory of the system using/encouraging the self-organizations.

This paper is organized as follows. Section 2 defines what is a complex system and our perception of an environment in which complex system simulations can be distributed. Section 3 describes the structure we used to model complex systems. Sections 4 and 5 describes the swarm intelligence algorithm and the platform used to distribute simulations. Finally, section 6 concludes this paper.

2 Computational Ecosystem used to distribute

2.1 What is a complex system ?

A complex system is composed of a massive set of entities with interactions between these entities. A global behavior of the system, which is non-trivial according to entities behaviors, emerges from these interactions. In such system, set of entities is not static: some entities can appear and other disappear: these systems are opened.

Interactions between entities feed an informations flow which structure our system. Furthermore as the system is open, flows cross it bringing the necessary energy to avoid the natural evolution to disorder. We are in front of a kind of dissipative system.

*L.I.T.I.S., University of Le Havre, France
*Authors are sorted alphabetically
†Corresponding author: guilhelm.savin@litislab.fr

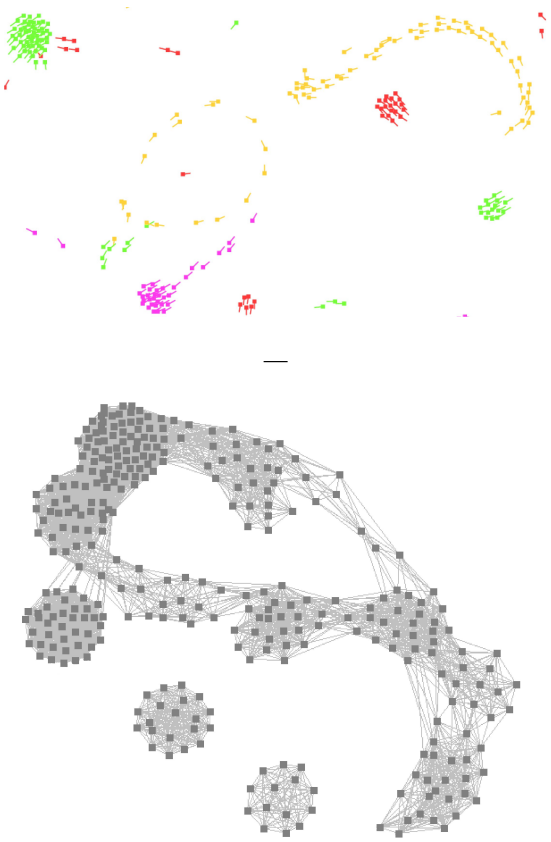

Figure 1. This picture represents one zoom in part of a boids-based simulation (on the top) and its representation with a dynamic graph (on the bottom). On the top part, a snapshot of the simulation is proposed. The boids (interacting entities) can only perceive and interact with fellow creatures located in a limited area. During the process, some groups appear corresponding to global structures obtained from local interactions. On the bottom part of the figure, the graph represents at a given moment the interaction graph corresponding to the state of the system made of boids.

Entity could have preferential interactions between a restricted set of other entities. Entities having such interactions composed a group called organization. There will have some organizations in the system, with a large amount of interactions between organization members, and a few interactions between different organizations.

As organizations are the result of the system dynamic and not a pre-established entry, the mechanism of organizations appearing/disappearing is a mechanism of self-organization. So if it is il-

lusory to try to control each entities to conduct the system, we can plan to affect the system by means of self-organizations.

2.2 Simulation in a computational ecosystem

The environment where the simulations evolve is composed by a set of machines which can connect or disconnect themselves at anytime. On theses machines eventually a lot of executing units (processes, threads, objects, ... process will be used in the following) are running.

There are interactions between these processes which can be direct or not. Direct interactions can be, for example, a communication between two processes or can be done through a shared file or memory zone. Indirect interactions can be a trace drop in the environment like a file. The processes population need resources to *live*: some computing power and some memory at least. Information flow crosses the system as an input data which is consumed by the processes and dissipated as output data. This stream structures the environment.

In the previous description we recognize the definition of a complex system. Furthermore the processes belong to classes as demon, scheduler .... and they are in competition to access to memory for example and collaborate to compute a result for example. There are strong analogies between natural ecosystems and the system that we consider, thus classes can be describe as species. Due to this parallel we define the environment where simulations take place as a computational ecosystem.

The kind of simulation we are interested in is complex system simulations. As we seen above, such simulations are composed of a massive set of entities, needing a large amount of computing resources which can not be provided by a single machine. A way to solve this resources limitation is to distribute the simulation, expanding our computational ecosystem by using a set of machines rather than a single one. The question is now, how to distribute such simulations ?

The elements of answer are in the nature of the simulations and the environment. Finally, environment becomes a complex system that we used to distribute complex systems.

A major raised problem is how to distribute entities in the environment. Load of machines has to be balanced but interactions between remote entities (entities which are located on different machines, these interactions are called *remote interactions*) have to be minimized to reduce network-load. This can be done by trying to control self-organizations. There are a lot of interactions between members of a same organization, if members are located on different machines, amount of remote interactions increases. So, detecting organizations allows to put all members of a same organization on a same machine, reducing remote interactions.

As organizations may change through time, entities location may change too. Problem is that entities are being executed , so they have to stop their execution, store their state and migrate to a new destination. A tool allowing this migration is needed. An interesting concept find in literature is the *active object pattern*[7] which sees the object as an actor receiving requests and executing them one by one. This provides two advantages. First, calls to object methods are seen as requests which allows to model call as an object exportable through the network. A second advantage is that migration becomes trivial: when active object has to migrate, it stops executing its requests, sends them to its next location and starts to execute then on this new location.

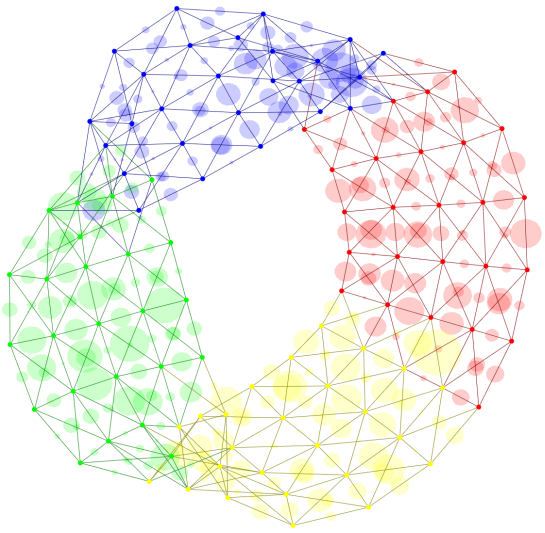

Figure 2. Coloring of a graph representation of the Moebius strip with AntCo2

3 Model

Complex system is composed of a set of entities and there are interactions between these entities. Difficulty of modeling a complex system is that system components may change : entities may appear or disappear and interactions are not static.

The emergence of global properties or behaviors come from this dynamic. Therefore, the study of complex systems leads to three important things to model: entities, interactions and dynamic of the system.

Since interactions can be seen as a couple of entities, a graph can model the system frozen at a time t. A graph G is a peer (V, E) where V is a set of elements called nodes and E is a set of node peer called edges.

To model the dynamic of complex systems, we have to used an other research field which is the one of dynamic graphs. A dynamic graph allows to introduce dynamic in the set of nodes and the set of edges of a graph. A definition of such graphs can be found in [4]. Therefore, a dynamic graph $G(t)$ is a peer $(V(t), E(t))$ where t is a discrete (in this paper) representation of the time. $V(t)$ and $E(t)$ may change with t: $V(t+1)$ (respectively $E(t+1)$) is $V(t)$ (resp. $E(t)$) with eventually some elements added or removed. Each node v and each edge e have a set of properties $P_v(t)$ (resp. $P_e(t)$) which may change with t too.

Figure 1 shows a boids simulation with its corresponding dynamic graph modeling interactions between boids. Boids are a kind of particule introduced by Craig REYNOLDS in [10] to simulate collective animal behavior like birds flocks. If distance between a boid a and a boid b is under a given threshold, then direction of a is influenced by b and inversely: there is an interaction between these two boids modeled in the graph by an edge between nodes representing a and b.

4 Swarm Intelligence - AntCo2

AntCo2 is a distributed algorithm dedicated to load balancing and communication minimisation. It considers only the dynamic graph of the application to compute the distribution. As communications and entities appear and disappear, as the importance of communication

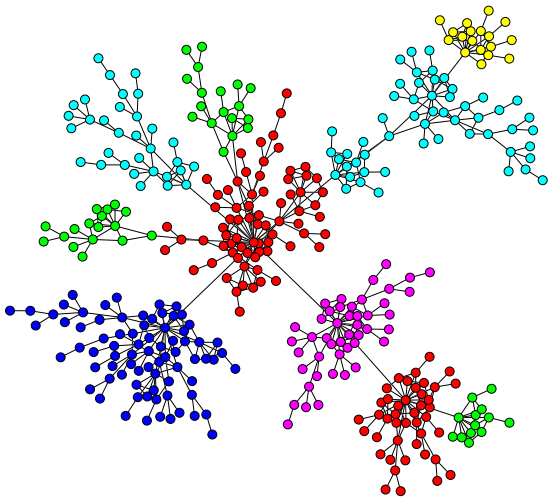

Figure 3. AntCo2 applied to a 400-nodes graph generated using preferential attachment rules.

evolve, the graph changes. Therefore the load balancer should also handle this dynamic process and be able to provide a distribution as the graph evolve.

Each computing resource is associated with a color, then by assigning a color to a node, the algorithm specifies the distribution.

One can see the distribution as a weighted partitioning of the graph. In this partitioning we try to distribute evenly the load (number of entities weighted by their computing demand) and to minimize communications between computing resources to avoid saturating the network. These two criteria are conflicting, therefore a trade-off must be found.

4.1 Detection of organizations

We see the partitioning as a dynamic community detection algorithm. We call such dynamic communities "organizations". Communities are often seen as group of vertices that are more densely connected one with another than with the rest of the graph. An algorithm able to detect organizations is able to follow communities as they evolve when nodes and edges appear, evolve and disappear in the communities.

There exists several graph partitioning algorithms ([8, 6, 5]) and community detection algorithms ([9]), but few handle evolving graphs. It is always possible to restart such algorithms each time the graph changes, but this would be computationally intensive. AntCo2 is an incremental algorithm that starts from the previous partitioning to compute a new partitioning when the graph changes.

Having a load balancer running on a single machine, to distribute applications that are often very large could be inefficient. Another goal of AntCo2 is to be able to be distributed with the application.

4.2 Swarm Intelligence

AntCo2 uses an approach based on swarm intelligence, namely colonies of ants. This algorithm provides several advantages: ants can act with only local knowledge of the graph representing the application to distribute. AntCo2 tries to avoid any global computation, therefore allowing it to be distributed with few communications and no global control.

In AntCo2, each colony represents a computing resource and has its own color. Inside colonies, ants collaborate to colonize organizations inside the graph and assign their color to nodes. Inversely, colonies compete to keep and conquer organizations. Figure 3 and 4 show graphs colored by ants.

Ants color nodes using numerical colored pheromones corresponding to their colony color. Such pheromones "evaporate" and therefore must be maintained constantly by ants. This allows to handle graph dynamics by forgetting old partitioning solutions and discovering new solutions by the constant exploration of ants inside the graph. The details of the algorithm are given in ([2]). Figure 2 shows four ants colonies coloring a graph with pheromones rates on each edge.

The change of a color for a node indicates a "migration advice", meaning that the corresponding entity should migrate on the computing resource associated to the new color. An inertia mechanism allows to avoid oscillatory advices.

4.3 Distribution

Ants of the AntCo2 algorithm use only local informations from the dynamic graph. This offers three ways of distribution as described in [3].

A first solution is to have on a single machine a dynamic graph modeling the overall application and to run AntCo2 on this graph. Migration advices are sent to machines. This solution has some disadvantage. If there is a fault on the machine hosting AntCo2, distribution looses its load-balancer: the solution is not fault-tolerant. Moreover, it leads to increase network communications: informations about interactions have to be sent to load-balancer, and migration advices have to be sent to other machines. This is a problem for an algorithm which aims to reduce network-load.

The next solution is similar to the first. Instead of running on a single machine, AntCo2 runs on a set of machines, each machine hosting a part of the graph. This solution becomes more tolerant to fault but still leads to increase network-load.

The last solution is to have one instance of AntCo2 running on each machine. Each instance considers only local entities and interactions: the system becomes decentralized. If there is a fault on a machine, this does not affect the system : this solution is fault-tolerant. As instances using only local informations, there is no need to communicate interactions informations to another machine: network-load not increases. Moreover, computing-load needed by an instance of AntCo2 depends of the amount of entities hosted on the machine, thus distribute entities leads to the distribution AntCo2 itself : AntCo2 is self-distributed.

5 Dagda

Dagda is a middleware dedicated to the distribution of Complex Systems simulations. It uses an existing middleware as a base which is extended with new features. The final aim is to provide a simple way to create distributed complex system simulation.

The main words of Dagda are decentralized, portable, load-balanced. Decentralized means that there is no restricted set of machines on which depend all machines. Dagda aims to be as portable as possible, *ie* any machines (computer,pda,phone,supercalculator...) can participate to the distribution.

Dagda can be divided in three components. The first is a local representation of the part of the application running on the machine. This is model by a dynamic graph. This graph is maintained by the second component of Dagda, called *agency* which is a middleware

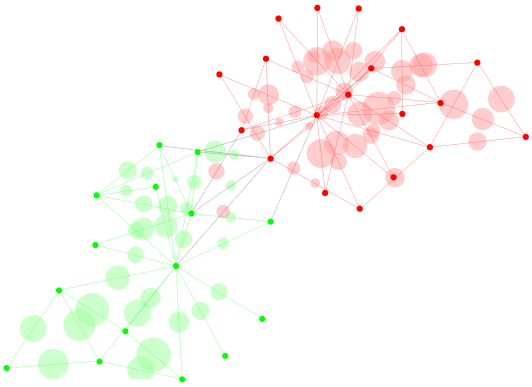

Figure 4. AntCo2 applied to the Zachary Karate Club[11] graph.

with Dagda features. A final aim is to provided a standard connection to middlewares able to manage active object and migration, but actually Dagda allows the use on its internal middleware or the use of ProActive[1], a middleware developed by INRIA. The last component of Dagda is a load-balancing algorithm used to colorize the graph. When node color changes, migration advices are sent to the middleware which proceed to the migration.

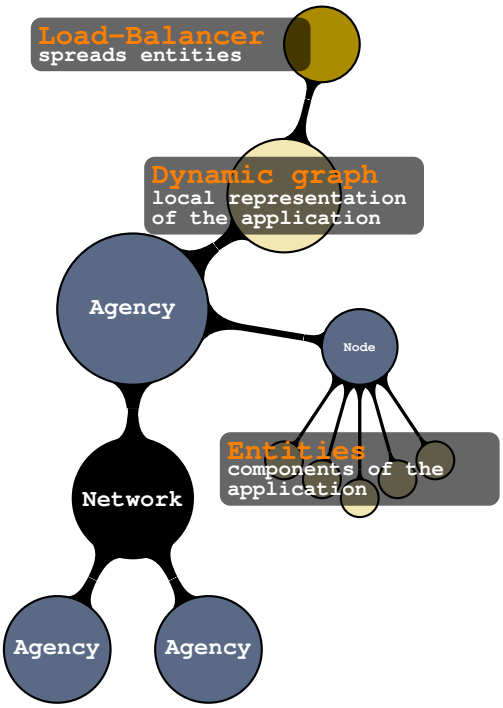

Figure 5. Dagda overview

5.1 Interactions Graph

Dagda is based on the concept that the distributed application is composed of a massive set of objects. These objects are called *entities* and are hosted on a machine by an *agency*. The active object pattern is used to model these entities.

Each machine models its hosted entities in a dynamic graph. Entities are nodes of this graph. When an entity communicates with an other, this is modeled in the graph with an edge between nodes associating to the implicated entities. Greater is the number of interactions between two entities, greater is the weight of the corresponding edge. There is a mechanism which decreases edge's weight through the time.

Entities can migrate from one agency to an other. This raises a problem: how to identify each entity through the network and how to get a remote entity ? The second part of this problem, how to get entity, is treated on section 5.2. Entities are identified by an *id* which is unique through time and network. Uniqueness is assumed by the fact that *id* depends on the agency's address (agency who creates the entity) and on a time-stamp.

Dagda profiles method calls between entities. For example, if an entity `A` calls a method `m()` of an entity `B`, this call will be detected and registered. Then this detection of interactions between entities is used to maintain the dynamic graph which models these interactions through the time.

The GRAPHSTREAM[4][1] API is used to create the graph.

5.2 Agency

The part of Dagda managing remote objects and remote operations is called Agency. This agency is the base of interactions between machines. It allows implementation of the active object pattern by entities and uses a middleware to allow remote method calls and migration of entities. A basic middleware is provided but the aim is to allow the use of any middleware ables to manage active objects.

Dagda aims to have a decentralized architecture so there is no master server to reference informations as for example entities location. Therefore, another aim of agencies is to provide global features without global control.

One is these features is to maintaine a shared context. Overall, environment, composed of all machines, has two kinds of properties: local properties which are specific to a single machine, and global properties which are shared by all machines. Managing local properties is trivial, but managing shared properties without global control raises some problems: if a property is changed on a machine, how spreads this change before other machines look for this property ? Agency aims to solve this problem and provides a valid access to environment properties.

5.3 Load-balancing

The last component of Dagda is a load-balancing algorithm. This algorithm uses the interactions graph and attributes colors to node. Dagda uses the AntCo2 algorithm. This choice allows to :

- balance the work-load of machines;
- reduce the network-load;
- distribute the load-balancer.

Distribution of the load-balancer is an important thing to have a decentralized platform. As describes in 4.3, there are three solutions to run the AntCo2 algorithm. First and second solution use a centralized approach which is not the aim of Dagda. Therefore, the last solution, having one instance of AntCo2 on each machine, has been retained for this middleware.

[1] `http://www.graphstream-project.org`

6 Conclusion

In this paper, an approach of complex system simulation distribution has been presented. This approach considers the execution environment as a computational ecosystem. We have seen that this environment becomes a complex system used to compute complex system simulations.

A major problem raised in this paper is how to distribute entities composing complex system. A swarm intelligence algorithm of organizations detection has been presented and used as load-balancing algorithm. Then a platform dedicated to complex system simulations distribution and using the previous load-balancing algorithm has been presented.

Next steps of this work can be divided in two parts. The first is about the platform, Dagda, which still needs some development and a phase test. Second part is about AntCo2 results and the adaptivity of load-balancing. Some mechanisms need to be added to avoid oscillatory migration advices. Adaptivity means that organizations have to adapt to environment, taking account for example of machine resources, other processes running on the machine...

REFERENCES

[1] Laurent Baduel, Françoise Baude, Denis Caromel, Arnaud Contes, Fabrice Huet, Matthieu Morel, and Romain Quilici, *Grid Computing: Software Environments and Tools*, chapter Programming, Deploying, Composing, for the Grid, Springer-Verlag, January 2006.

[2] Cyrille Bertelle, Antoine Dutot, Frédéric Guinand, and Damien Olivier, 'Organization detection for dynamic load balancing in individual-based simulations', *Multi-Agent and Grid Systems*, **3**(1), 42, (2007).

[3] Antoine Dutot, *Distribution Dynamique Adaptative à l'aide de mécanismes d'intelligence collective*, Ph.D. dissertation, Université du Havre - LIH, 2005.

[4] Antoine Dutot, Frédéric Guinand, Damien Olivier, and Yoann Pigné, 'Graphstream: A tool for bridging the gap between complex systems and dynamic graphs', in *EPNACS: Emergent Properties in Natural and Artificial Complex Systems*, (2007).

[5] C. M. Fiduccia and R. M. Mattheyses, 'A linear time heuristic for improving network partitions', in *ACM IEEE Design Automation Conference*, pp. 175–181, (1982).

[6] B. Hendrickson and R. Leland, 'An improved spectral graph partitioning algorithm for mapping parallel computations', *SIAM J. Scien. Comput.*, **16**(2), 452–469, (1995).

[7] Carl Hewitt, Peter Bishop, and Richard Steiger, 'A universal modular actor formalism for artificial intelligence', in *t*, pp. 235–245, (1973).

[8] B.W. Kernighan and S. Lin, 'An efficient heuristic procedure for partitioning graph', *The Bell System Technical Journal*, **49**(2), 192–307, (1970).

[9] M. E. J. Newman and M. Girvan, 'Finding and evaluating community structure in networks', *Phys. Rev*, **69**, (2004).

[10] Craig W. Reynolds, 'Flocks, herds and schools: A distributed behavioral model', in *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pp. 25–34, New York, NY, USA, (1987). ACM.

[11] W. W. Zachary, 'An information flow model for conflict and fission in small groups', *Journal of Anthropological Research*, **33**, 452–473, (1977).