

# Controller-Agent based approach for Solving Distributed Constraint Problem

Sami AL-MAQTARI<sup>1</sup> and Habib ABDULRAB<sup>1</sup>

**Abstract.** The interesting domain of constraint programming has been studied for years. Using Constraint Satisfaction Problem in association with Multi-Agent System has emerged the research in a new field known as Distributed Constraint Satisfaction Problem (DCSP). Many algorithms are proposed to solve DCSP. Inspired from ABT algorithm, we introduce in this paper our algorithm for solving DCSP where we divide agents in the system into two groups: Variables' and Controller Agents, which allow reformulating of different inter-agent communication algorithm in our framework. This division allows the separation between the constraints verification and other functions of agents. The proposed algorithm is not only capable of treating binary constraints; it can be used easily in order to treat non-binary constraints. This facility gives also the possibility of grouping constraints in order to form a set of quasi-independent sub-problems. These sub-problems may be interconnected by some common variables. The instantiation of these variables can be done by negotiation in order to separate the sub-problems into totally independent ones.

## 1 INTRODUCTION

Constraint Satisfaction Problem or CSP is a very interesting paradigm for modeling in real life. A lot of real world problems can be described and modeled as a set of variables where their values are restricted by a set of constraints. Many methods and techniques are already in use in order to treat this kind of problems. Backtracking and arc-consistency and their alternatives are already discussed over and over in literature [1, 2].

Mixing CSP with the autonomy of agents in a Multi-Agent System emerges the birth of a sub-branch of CSP research known as Distributed CSP or DCSP. Multiple methods are used to solve a DCSP. Among them we find the Asynchronous BackTracking (ABT) proposed by Yokoo and its alternative [3-7] that we inspired our approach from.

This Approach is very adequate for modeling a specific kind of problems like water usage in agricultural domain. In such system constraints are suitable for modeling the ecosystem equations. However, they cannot be used in order to model the autonomous behavior of the different actors in the system (like farmers and resource owner, etc.); where multi-agent systems are more convenient.

We propose a new framework based on two kinds of agents: Variables' agents and Controller Agents. By defining these two kinds of agents we separate between the treatment of constraints

and the other functionalities of the agents in the system. Also, it allows dividing a general constraint problem into multiple sub-problems easier to be treated. This algorithm is firstly introduced in our paper [8] for a proposed model intended to be used for water agricultural usage management to be applied to the region of Sadah in Yemen.

In the next sections we start by giving some basic definitions then, we introduce our proposition for an alternative algorithm for solving DCSPs.

## 2 Definitions

We start by giving some definitions of CSP and DCSP:

### 2.1 Constraint Satisfaction Problem (CSP)

Formally, a Constraint Satisfaction Problem (CSP) is a triple  $(V, D, C)$  where:

- $V = \{v_1, \dots, v_n\}$  is a set of  $n$  variables,
- a corresponding set  $D = \{D(v_1), \dots, D(v_n)\}$  of  $n$  domains from which each variable can take its values from,
- and  $C = \{c_1, \dots, c_m\}$  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  $k: c_i(v_a, \dots, v_k)$  that maps the cartesian product  $D(v_a) \times \dots \times D(v_k)$  to  $\{0, 1\}$ . As usually the value 1 means that the value combination for  $v_a, \dots, v_k$  is allowed, and 0 otherwise.

A solution for a CSP is an assignment of values for each variable in  $V$  such that all the constraints in  $C$  are satisfied.

Constraints involving only two variables are called binary constraints [9]. 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 binarized using techniques such that dual graph method and hidden variable method [10]. Translating non-binary constraints into binary ones allows processing the CSP using efficient techniques adapted only for binary constraints. However, this translation normally implies an increase in number of constraints.

<sup>1</sup> LITIS Lab. – INSA of Rouen – France. Email: {sami.almagtari, abdulrab}@insa-rouen.fr.

## 2.2 Distributed Constraint Satisfaction Problem (DCSP)

A Distributed Constraint Satisfaction Problem (DCSP) is a CSP where the variables are distributed among Agents in a Multi-Agent System (MAS). A DCSP can be formalized as a combination of  $(V, D, C, A, \lambda)$  where:

- $V, D, C$  are the same as explained for a normal CSP,
- $A = \{a_1, \dots, a_p\}$  is a set of  $p$  agents,
- and  $\lambda: V \rightarrow A$  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.  $\forall v_1, \dots, v_k \in V_i \Leftrightarrow \lambda(v_1) = \dots = \lambda(v_k)$  where  $V_i \subset V$

represents the subset of variables that belong to agent  $a_i$ . These subsets are distinct, i.e.  $V_1 \cap \dots \cap V_p = \phi$  and the union of all subsets represents the set of all variables, i.e.  $V_1 \cup \dots \cup V_p = V$ .

The distribution of variables among agents divides the constraints set  $C$  into two subsets according to the variables involved within the constraint. The first set is the intra-agent constraints  $C_{intra}$  that represent the constraints over the variables owned by the same agent  $C_{intra} = \{C(V_i) \mid \lambda(v_1) = \dots = \lambda(v_k), v_1, \dots, v_k \in V_i\}$ .

The second set is the inter-agent constraints  $C_{inter}$  that represent the constraints over the variables owned by two or more agents. Obviously, these two subsets are distinct  $C_{intra} \cap C_{inter} = \phi$  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.  $C_{intra} = \phi$ . Therefore, all variables in  $V$  are interface variables  $V = V_{interface}$ .

Many techniques are used to solve DCSPs. Mainly we mention the Asynchronous Backtracking (ABT) algorithm that was proposed by Yokoo [3, 11-13] and some of its alternatives [4, 5]. In the following section we introduce another alternative based on the concept of Controller Agent, which we propose, in order to validate and test inter-agent constraints.

## 2.3 Over-constrained problem and constraint relaxation

A CSP is called an over-constrained problem if no possible combination exists that satisfies all constraints. In other words, at least one constraint can not be satisfied. Solving such problem implies neglecting some constraints in the system (constraint relaxation [14]). This can be achieved by dividing constraints according to some kind of preference levels [15]. Constraint types may vary from hard to soft constraints according to their importance. The hard ones represent the constraints that should absolutely be satisfied while the soft constraints may be neglected. In this case we search the optimal solution in trying to maximize the number of satisfied soft constraints. We say that no solution exists if we cannot satisfy all the hard constraints.

## 3 Controller Agent approach

Here we will explain how to extend the original DCSP-based model of multi-agent cooperation. We will describe a structure of DCSP based on the idea of dividing agents into two types: Variables' Agent and Controller Agent. Originally introduced in our former works [8, 16], a Variables' Agent is an agent who possesses one or more variables from the set  $V$  and a subset of intra-agent constraints from  $C_{intra}$ , while the role of Controller Agents is to encapsulate the inter-agents constraints. For short, we will use the term VAgent as an abbreviation for Variables' Agent and CAgent for Controller Agent.

CAgents take in charge the responsibility of verifying the constraints satisfaction. In other words, from the beginning all global constraints are distributed among an initial set of CAgents  $CA = \{ca_1, \dots, ca_q\}$  where  $1 \leq q \leq m$  and  $m$  is the number of inter-agent constraints. In the centralized form of DCSP all constraints in  $C_{inter}$  are grouped inside one CAgent ( $q=1$ ), while in its simplest form each CAgent contains only one constraint from  $C_{inter}$  ( $q=m$ ).

Figure 1 shows a constraint network of a set of agent related by binary constraints either directly (a) or via CAgent (b). As in a formal constraint network, Figure 1 (a) shows some constraints that represent relations directed from an agent (called proposing agent) to another (called evaluating agent) according to a predetermined order. The resolution is done by passing value propositions according to the directions of arcs from proposing agent to evaluating agent.

In our approach shown in Figure 1 (b), relations between agents are represented by links directed from a VAgents to a CAgents. These links indicate that the VAgent has at least one variable that is involved in a constraint of the CAgent.

The CAgent supervises one or more constraints. This means that we can group more than one constraint into Controller to form a kind of local problem as shown in Figure 1 (b). The propositions of values for variables are sent by VAgent to be received and tested by CAgents. VAgents are given different priority degrees (by default, their order of creation). Such application of VAgents and CAgents facilitates seamless combination of MAS and CSP.

Using the concept of VAgents and CAgents needs reformulating inter-agent communication algorithm in our

framework. In the next section we will illustrate how the modification of well-known asynchronous backtracking algorithm by Yokoo [3, 11] can be easily achieved. For simplicity reasons, some limitations are made in this paper and they are not an obligation: we will consider only the case where a VAgent has one variable only. In the same manner, we will treat the case where a CAgent contains one binary constraint only. The proposed approach can be easily extended and applied to other cases without these limitations as we will see later in this paper.

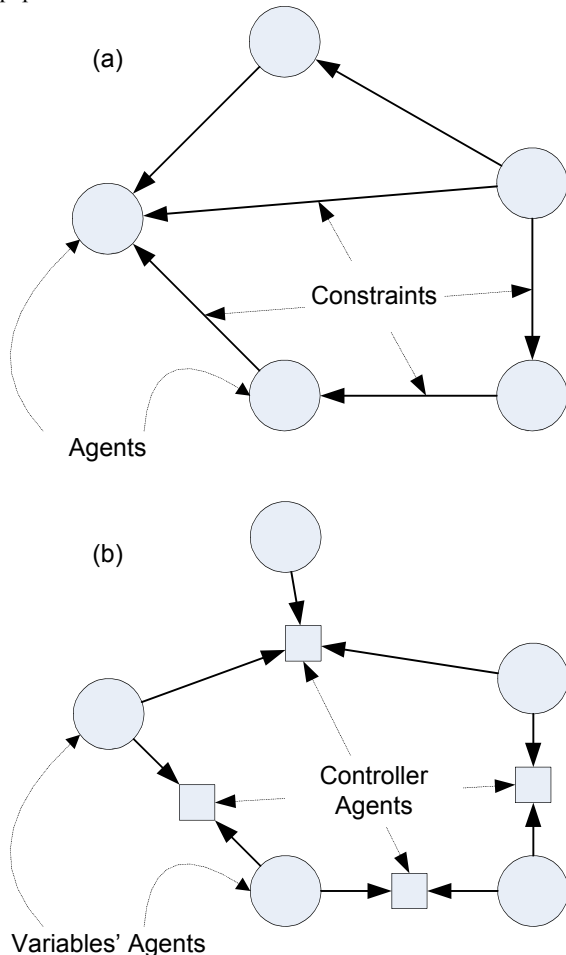


Figure 1 Constraint Network (a) without or (b) with Controller Agent

### 3.1 Algorithms

As mentioned before, the proposed algorithm may be considered as a modified version of the ABT algorithm of Yokoo[3, 4]. The proposed algorithms of both types of agents are divided into two stages:

- **Domain reducing stage**

This stage implies a preprocessing of variables' domains. The result is reduced domains by eliminating values that would be surely refused from 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 use bound consistency algorithm 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. In this case, the system signals a no-solution to user (failure). Another solution can be investigated by using constraints relaxation [14, 15], in which a VAgent It 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 [15]. 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.

- **Value proposition and validation stage**

In this stage VAgents make their propositions of values to related CAgents to be tested. Value proposing can be considered as a domain inform message in test mode. This proceeds as follows:

5. From now on, every VAgent starts instantiating values for its variable according to the new domains. It sends this proposition to the related CAgents.
6. 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 bound consistency algorithm as in the previous stage to reduce other variables' domains. These new domains are sent to their VAgents to propagate domains change.
7. 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.
8. If the result of the domain propagation was some new reduced domains with more than one value then steps 5-7 are repeated Recursively with the value proposed by the VAgent that have the next priority.

### 3.2 Advantages and disadvantages

Main advantage of the system is the possibility of treating non-binary constraints simply without need of modification. The algorithm is still working for non-binary constraints in the same manner. This allows avoiding the need for constraints translation into binary ones. Extending the algorithm to cover non-binary constraints will be explained later in this paper.

The possibility of treating non-binary constraints allows us to gather constraints to create subsets of constraints where each

subset contains a group of constraints that could be related and included inside one CAgent. Searching a solution for this group represents a sub-problem of the general one. In one of its extreme case, this grouping allows centralizing all the resolution (in case where all constraints are included inside one CAgent). In normal case, this allows dividing constraints into groups that are related by some common variables. Giving value for these variables separates these groups of constraints and forms a set of sub-problems that are totally independent. Many methods [17, 18] are proposed in order to partition the problem to allows processing it in parallel manner. These methods can be used with our algorithm in order to assign each partition of the problem to a CAgent.

A main disadvantage of the proposed algorithm is the increase in agent number. In fact, using CAgent for each constraint (mostly binary ones) may overload the system because of the large total number of agents (the increase according to the number of constraints). However, the benefits of grouping constraints into subsets and putting each group inside a controller can compensate the increase in number of agents. We still have some improvement to do in order to increase the performance of the algorithm. This performance can be improved by taking in consideration some enhancements that will be discussed later.

### 3.3 Algorithm Correctness

For any CSP we can encounter one of three cases: if it is an over-constraint problem and then the solution does not exist. Otherwise, one solution or more exist. Here, we will explain how this approach can detect these cases.

As described in step 4 in the domain reduction stage of the algorithm, the detection of an empty domain for any variable means that this DCSP is over-constrained problem and no solution exists. In the other side, if all variables after the domain reduction stage have only one single value domains, then this signifies that only one solution exists which is the combination of all domain (single) values.

Otherwise, one or more solutions may exist. By considering value proposing and validating stage of the algorithm, we consider value proposing as domain informing with a single value in test mode. We call it test mode because unlike domain reduction stage here we can request backtracking.

To understand backtracking in our algorithm we can imagine a situation where a VAgent V1 proposes a value for its variable. This value is considered by the related CAgent C as a single-value domain and would propagate to other related VAgents. If another VAgent V2 ends with an empty domain for its variable because of V1 proposal then it would report a backtrack signal to the CAgent C. in this case, CAgent C will request another value from VAgent V1 and VAgent V2 should retrieve its previous variable domain before domain propagation.

The recursive nature of value proposing stage results always either single-value domains for all variables which means that a solution exists or an empty domain case which means that no solution exists.

### 3.4 Example

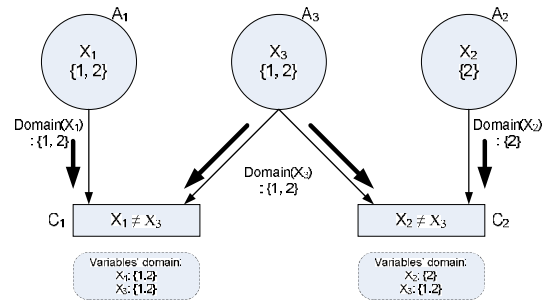


Figure 2 domain sending

The following figures show an example of the proposed algorithm. Three VAgents  $A_1, A_2$  and  $A_3$  having three variables  $X_1, X_2$  and  $X_3$  respectively. Each variable has its respective domain. These agents are related to two CAgents  $C_1$  and  $C_2$  which contain two constraints over the variables.

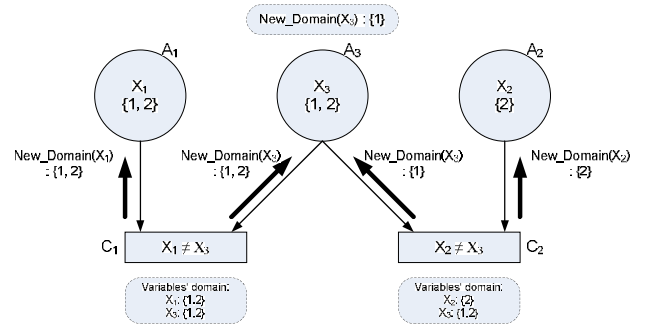


Figure 3 returning reduced domains

In Figure 2 the VAgents start sending information about the domains of their variables to the CAgents. The CAgent calculates the validity of these domains according to its constraints. In Figure 3, the controller  $C_1$  cannot reduce the domains of  $X_1$  and  $X_3$ . However, the controller  $C_2$  can reduce the domain of  $X_3$  and send back the new domain.

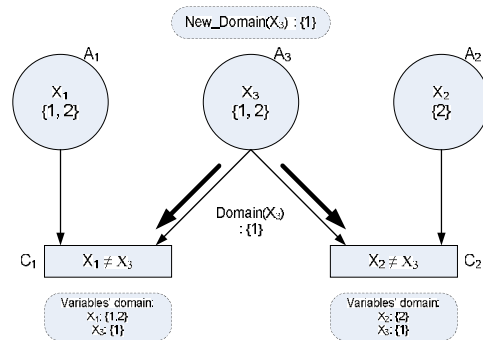


Figure 4 propagation of domain change

In order to propagate the domain change in the agent  $A_3$ , it will resend its domain to the related controllers as shown in Figure 4. This propagation also informs the controllers about the new domain of the variables.

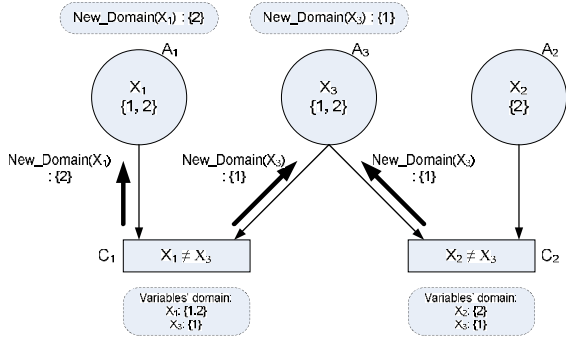


Figure 5 continue domain reduction

The domain change propagation can cause another domain change in another VAgent. In Figure 5, agent  $A_1$  receives another proposition for its variable new domain. While this propagation has no effect on the domain of the variable in agent  $A_2$ .

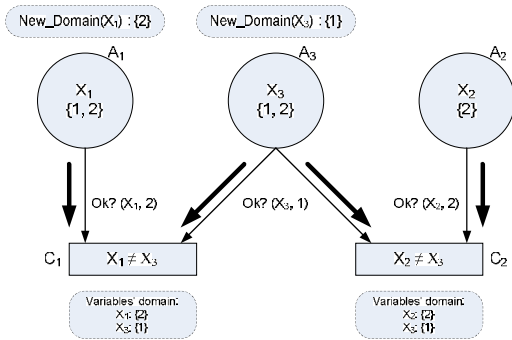


Figure 6 value proposing

Each VAgent receives the reduced domain and creates a new domain by the intersection between the returned domains from each controller. In Figure 3, Agent  $A_3$  has created the new domain  $\{1\} = \{1, 2\} \cap \{1\}$ .

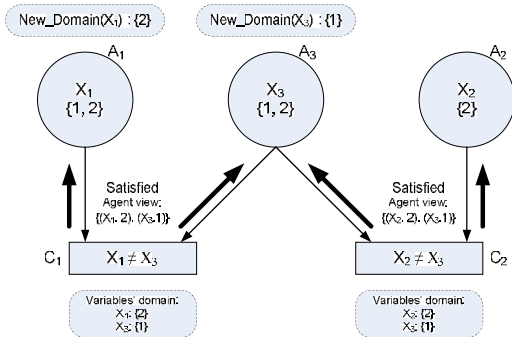


Figure 7 result of proposition validation

The domain reduction continues until we have stability in the variables' domain. The agents can start then proposing values for their variables. In Figure 6, agents  $A_1, A_2$  and  $A_3$  propose 2, 2 and 1 as values for their variables. They send these propositions to the related CAgents  $C_1$  and  $C_2$  for validation.

After validating the propositions, the CAgents send back the result of either to be satisfied or not (Figure 7).

### 3.5 Algorithm extension

As we said earlier in this paper, for simplicity reasons we will consider only the case where a VAgent has one variable only. In the same manner, we will treat the case where a CAgent contains one binary constraint only. The extension of the proposed algorithm to be used with VAgents that have more than one variable is simple: If a VAgent  $VA_i$  has a set of interface variables that contains more than one variable  $\{v_{i1}, \dots, v_{ij}\}$

where each variable is involved in an external constraint, the agent should be related to all CAgents owning these constraints. The agent  $VA_i$  sends the domain of the variable  $v_{ij}$  only to the concerned controller(s) in order to avoid overloading communication. The rest of the manipulation stays the same as for an agent with one variable only.

Treating non-binary constraints is straight forward: the CAgent containing non-binary constraint exchange all necessary information with all VAgent having variables involved in its constraint. The same thing applies when the controller contains more than one constraint.

## 4 Framework implementation general structure

The general schema of the proposed framework is shown in Figure 8. The intended architecture represents a layer between the application and the layer of the real MAS and CSP platforms. From the application view point, the system is composed directly from the two principle types of agents: the CAgent and the VAgent. The user can create the necessary VAgents according to its problem definition. He also creates the constraints and associates them to CAgents.

The system layer uses generic interfaces for both MAS and CSP platforms. This allows the system to use any existed platforms by implementing these interfaces. In 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 and its difficulty changes according to the MAS and CSP platforms used for the realization of the final system.

In our work we have chosen JADE [19] as a MAS platform. JADE (Java Agent DEvelopment Framework) is a software Framework compliant with the FIPA specifications and is fully implemented in Java language. It simplifies the implementation of multi-agent systems through a set of graphical tools that supports the debugging and deployment phases. JADE is free software developed and distributed by Telecom Italia Lab.

In which concern the CSP platform, we have chosen Choco. Choco [20] is a library for constraint satisfaction problems (CSP), constraint programming (CP) and explanation-based constraint solving (e-CP). It is built on an event-based propagation mechanism with backtrackable structures. It is an open-source software implemented in java also. It permits the use of multiple solvers for different problem separately. This allows each CAgent to have its own solver.

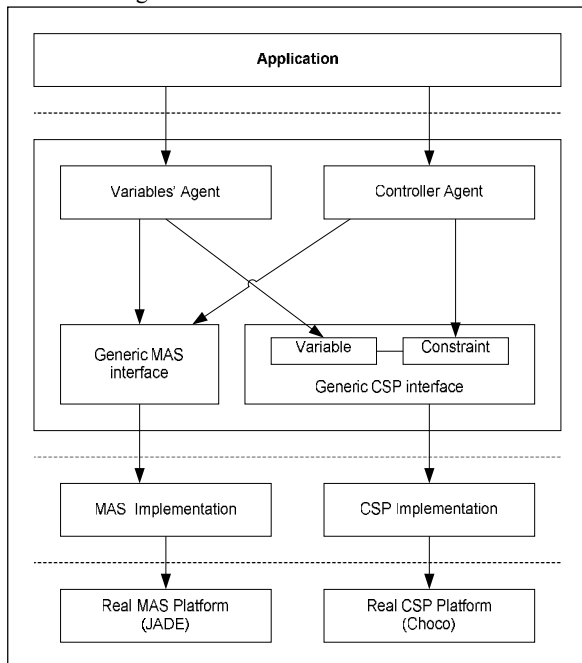


Figure 8 general schema of the proposed system

Both types of agent use the generic MAS platform that provides the base for constructing agents and manage the communication between them. CAgents use the generic CSP interface to have the capability of solving and validating constraints.

## 5 Conclusion and perspective

In this paper, we have introduced our different approach for the solution of Distributed Constraint Satisfaction Problems in a MAS consisting of two architectural types of agents: CAgents and VAgents. This approach allows the separation between the treatment of constraints and the other functionalities of the agents in the system. Also, it allows dividing a general constraint problem into multiple sub-problems easier to be treated.

Two aspects are investigated for enhancing the algorithm. We have the intention of enhance the algorithm role of the CAgent in two aspects: (a) by giving the CAgent the capacity of proposing solutions to the VAgents. These resolutions are established by negotiations with other CAgents, (b) by giving a preference level to every constraint. This allows constraints relaxing and permits CAgents to accept proposals which may not satisfy some weak constraints. This can be useful in case of over-constraint problems.

Another perspective is to guide the strategy of proposals of a VAgent by an optimization function. This function is defined according to the specific objectives to this agent.

## References

- [1] G. Ringwelski, "An Arc-Consistency Algorithm for Dynamic and Distributed Constraint Satisfaction Problems."
- [2] C. Fernandez, R. Bejar, B. Krishnamachari, C. Gomes, and B. Selman, "Communication and Computation in Distributed CSP Algorithms."
- [3] M. Yokoo and K. Hirayama, "Algorithms for Distributed Constraint Satisfaction: A Review," in *Autonomous Agents and Multi-Agent Systems*, 2000, pp. 198-212.
- [4] C. Bessiere, I. Brito, A. Maestre, and P. Meseguer, "The Asynchronous Backtracking Family," LIRMM-CNRS, Montpellier, France March 2003 2003.
- [5] C. Bessiere and I. Brito, "Asynchronous Backtracking without Adding Links: A New Member in the ABT Family," 2005, pp. 7-24.
- [6] M. Yokoo, "Asynchronous Weak-commitment Search for Solving Distributed Constraint Satisfaction Problems," in *International Conference on Principles and Practice of Constraint Programming*, 1995, pp. 88-102.
- [7] C. Bessière, A. Maestre, and P. Meseguer, "Distributed Dynamic Backtracking."
- [8] S. Al-Maqtari, H. Abdulrab, and A. Nosary, "A Hybrid System for Water Resources Management," in *GIS third international conference & exhibition*, 2004.
- [9] F. Bacchus, X. Chen, P. v. Beek, and T. Walsh, "Binary vs. Non-Binary Constraints," 2002, pp. 1-37.
- [10] F. Bacchus and P. v. Beek, "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)*, 1998, pp. 311-318.
- [11] M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara, "The Distributed Constraint Satisfaction Problem: Formalization and Algorithms," in *IEEE Transaction on Knowledge and Data Engineering*, 1998, pp. 673-685.
- [12] M. Yokoo, T. Ishida, E. H. Durfee, and K. Kuwabara, "Distributed Constraint Satisfaction for Formalizing Distributed Problem Solving," in *12th IEEE International Conference on Distributed Computing Systems*, 1992, pp. 614-621.
- [13] M. Yokoo, "Distributed Constraint Satisfaction Algorithm for Complex Local Problems," in *Third International Conference on Multiagent Systems (ICMAS-98)*, 1998, pp. 372-379.
- [14] R. Mailler and V. Lesser, "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, 2004, pp. 438-445.
- [15] H. Rudova, "Constraint Satisfaction with Preferences," in *Faculty of Informatics Brno - Czech Republic: Masaryk University*, 2001.
- [16] S. Al-Maqtari, H. Abdulrab, and A. Nosary, "Constraint Programming and Multi-Agent System mixing approach for agricultural Decision Support System," in *Emergent Properties in Natural and Artificial Dynamical Systems*, 2006, pp. 199-213.
- [17] P. Berlandier and B. Neveu, "Problem Partition and Solvers Coordination in Distributed Constraint Satisfaction," in *Workshop on Parallel Processing in Artificial Intelligence (PPAI-95)*, Montreal, Canada, 1995.
- [18] R. H. Bisseling, J. I. Byrka, and S. Cerav-Erbas, "Partitioning a Call Graph."
- [19] JADE: <http://jade.tilab.com>.
- [20] Choco: <http://choco.sourceforge.net/>.