JChoc DisSolver
Bridging the Gap Between Simulation and Realistic Use

I. Benelallam[1,2], Z. Erraji[1], G. Elkhattabi[1], J. Ait Haddou[1] and E. H. Bouyakhf[1]
[1]*LIMIARF – FSR,University Mohammed V, Rabat, Morocco*
[2]*INSEA, Rabat, Morocco*

Keywords: Constraint Programming (CP), Multi-Agent Systems, Distributed Problem Solving, Agent Models and Architectures, Distributed Constraints Reasoning, Realistic Use, Constraint Satisfaction Problem (CSP), Distributed CSP (DisCSP).

Abstract: The development of innovative and intelligent multiagent applications based on Distributed Constraints Reasoning techniques is obviously a fastidious task, especially to tackle new combinatorial problems (e.i. distributed resource management, distributed air traffic management, Distributed Sensor Network (Béjar et al., 2005)). However, there are very few open-source platforms dedicated to solve such problems within realistic uses. Given the difficulty that researchers are facing, simplifying assumptions and simulations uses are commonly used techniques. Nevertheless, these techniques may not be able to capture all the details about the problem to be solved. Hence, transition from the simulation to the actual development context causes a loss of accuracy and robustness of the applications to be implemented.
In this paper, we present preliminary results of a new distributed constraints programming platform, namely JChoc DisSolver. Thanks to the extensibility of JADE communication model and the robustness of Choco Solver, JChoc brings a new added value to Distributed Constraints Reasoning. The platform is user-friendly and the development of multiagent applications based on Constraints Programming is no longer a mystery to users.
A real distributed problem is used to illustrate how the platform can be appropriated by an unsophisticated user and the experimental results are encouraging for more investigations.

1 INTRODUCTION

Since the onset of real time electronic devices, mobile, ubiquitous, and intelligent computing, new combinatorial problems have emerged in the AI community such as: distributed resource management, distributed air traffic management, Distributed Sensor Network (Béjar et al., 2005), disaster rescue (Kitano et al., 1999) and distributed Meeting Scheduling Problems (SMP), for which it is not suitable to collect all data of problem in one site, to solve it by a centralized algorithm. The reasons are communication time and cost of translation of each subproblem in a common format. In addition, to give a single agent all data of the problem can also be excluded for reasons of security and confidentiality. Therefore, some of the AI communities are motivated to take an interest in Distributed Constraint Reasoning (DCR), giving birth to other distributed formalism (Yokoo, 2001), whose work focused on developing techniques for modeling and solving distributed combinatorial problems with or without optimization criterion. Distributed Constraint Satisfaction Problems (DisCSP) and Distributed Constraint Optimization Problems (DCOP) provide a useful framework of multiagent systems for distributed resolution of combinatorial problems (Yokoo and Hirayama, 1995; Yokoo et al., 1992; Yokoo, 2001; Yokoo, 2000; Yokoo et al., 1998).

In this context, an agent must have a communication platform that allows the exchange of information or dialogue to coordinate their decision-making. This reliable communication tool allows agents to send and receive messages according to a given distributed protocol. However, various sophisticated solvers have been developed: DisChoco (Wahbi et al., 2011), Disolver (Hamadi, 2003), MELY (Galley, 2000), Frodo (Petcu, 2006). These solvers rely on several algorithms for solving DisCSP problems such as Asynchronous Backtracking (ABT (Yokoo

Benelallam I., Erraji Z., Elkhattabi G., Ait Haddou J. and Bouyakhf E..
JChoc DisSolver - Bridging the Gap Between Simulation and Realistic Use.
DOI: 10.5220/0005208600660074
In *Proceedings of the International Conference on Agents and Artificial Intelligence* (ICAART-2015), pages 66-74
ISBN: 978-989-758-073-4

et al., 1992), ABT Family (Bessiere et al., 2005)), Asynchronous Forward Checking (AFC) (Meisels and Zivan, 2007) and Nogood-based Asynchronous Forward-Checking (AFC-ng) (Ezzahir et al., 2009). Asynchronous Distributed constraints OPTimization (Adopt) (Modi et al., 2005), Asynchronous Forward Bounding (AFB) (Gershman et al., 2009), Asynchronous Branch-and-Bound (Adopt-BnB) (Yeoh et al., 2008) and Dynamic Backtracking for distributed constraint optimization (DyBop) (Ezzahir et al., 2008) were developed to solve DCOP problems. As well as the authors recognise that most of these tools are specially developed for simulation context. This fact can be clearly observed from its experimental setups. Given the difficulty that researchers are facing, they often make many simplifying assumptions (i.e. simple agent (one variable per agent), agents as multi-thread, single physical platform, communication via simulated perfect FIFO channels, etc...) about the underlying distributed problem, which might affect the predictions obtained from the simulation in non-trivial ways. Switching from the simulation to the actual development practice often leads to loss of accuracy. Hence, bridging the gap between simulation and actual development and deployment within distributed constraints solvers is the primary motivation for presenting the different ideas discussed in the present paper.

In this paper we focus on the development of a multiagent platform for Distributed Constraint Reasoning, namely JChoc DisSolver. This proposed platform allows non-expert user to address and solve easily real Distributed Constraint Satisfaction Problems.

This document is organized as follows. Section 2 presents a brief definition of Distributed Constraint Satisfaction Problem (DisCSP) and gives an example. In section 3, we present related work . Section 4 presents the global architecture of JChoc platform. In section 5, we show how use this platform in a distributed environment. And finally, in section 6 we conclude the paper by experiment this platform within a real Distributed Constraints Satisfaction Problem.

2 PRELIMINARIES

2.1 Distributed Constraint Satisfaction Problems

Constraint Programming distinguishes between the description of the constraints involved in a problem on the one hand, and the algorithms and heuristics used to solve the problem on the other hand. Modeling and solving problems is through a very elegant mathematical formalism, called the Constraint Satisfaction Problems CSPs.

The Distributed Constraint Satisfaction Problem (DisCSP) is represented by a constraint network where variables and constraints are distributed among multiple automated agents.

Definition. A finite DisCSP is defined by a 5-tuple(A,X,D,C,ψ), where:

- $A = \{A_1,...,A_p\}$ is a set of p agents.
- $X = \{x_1,...,x_n\}$ is a set of n variables such that each variable x_i is controlled by one agent in A.
- $D = \{D(x_1),...,D(x_2)\}$ is a set of current domains, where $D(x_i)$ is a finite set of possible values for variable x_i.
- $C = \{c_1,...,c_m\}$ is a set of m constraints that specify the combinations of values allowed for the variables they involve. We note that the constraints are distributed among the automated agents. Hence, constraints divide into two broad classes: inter-agent and intra-agent.
- $\psi : X \longmapsto A$ is a function that maps each variable to its agent.

A solution to a DisCSP is an assignment of a value from its domain to every variable of the distributed constraint network, in such a way that every constraint is satisfied. Solutions to DisCSPs can be found by searching through the possible assignments of values to variables such as ABT algorithm (Yokoo et al., 1992).

2.2 Meeting Scheduling Problem as a DisCSP

The Distributed Meeting Scheduling Problem (MSP) is a real distributed problem where agents may not desire to deliver their personal information to a centralized agent to solve the whole problem (Meisels and Lavee, 2004; Wallace and Freuder, 2002).

The MSP involves a set of n agents having a personal private calendar and a set of m meetings each taking place in a specified location. Each agent, $A_i \in A$, knows the set of the k_i among m meetings he/she must attend. It is assumed that each agent knows the traveling time between the locations where his/her meetings will be held. The traveling time between locations where two meetings m_i and m_j will

be hold is denoted by $TravellingTime(m_i, m_j)$. Solving the problem consists in satisfying the following constraints: (i) all agents attending a meeting must agree on when it will occur, (ii) an agent cannot attend two meetings at same time, (iii) an agent must have enough time to travel from the location where he/she is to the location where the next meeting will be held.

We illustrate in Figure 1 the encoding of the instance of the meeting scheduling problem in the distributed constraint network formalism. This figure shows 4 agents where each agent has a personal private calendar and a set of meetings each taking place in a specified location. Thus, we get the following DisCSP:

- $A = \{A_1, A_2, A_3, A_4\}$ each agent A_i corresponds to a real agent,
- For each agent $A_i \in A$ there is a variable m_{ik}, for every meeting m_k that A_i attends,
- $X = \{m_{11}, m_{13}, m_{14}, m_{21}, m_{22}, m_{32}, m_{33}, m_{34}, m_{44}\}$.
- $D = \{D(m_{ik}) | m_{ik} \in X\}$ where,
  - $D(m_{11}) = D(m_{13}) = D(m_{14}) = \{s \mid s \text{ is a slot in } calendar(A_1)\}$.
  - $D(m_{21}) = D(m_{22}) = \{s \mid s \text{ is a slot in } calendar(A_2)\}$.
  - $D(m_{32}) = D(m_{33}) = D(m_{34}) = \{s \mid s \text{ is a slot in } calendar(A_3)\}$.
  - $D(m_{44}) = \{s \mid s \text{ is a slot in } calendar(A_4)\}$.
- For each agent A_i, there is a private arrival-time constraint (c^i_{kl} intra-agent constraint) between every pair of its local variables (m_{ik}, m_{il}) (e.g. Omar must attend tree meetings m_1, m_2 and m_3). For each two agents A_i, A_j that attend the same meeting m_k there is an equality inter-agent constraint (c^{ij}_k) between the variables m_{ik} and m_{jk}, corresponding to the meeting m_k on agent A_i and A_j (e.g. Omar and Jean participate in the same meeting m_1). Then, $C = \{c^i_{kl}, c^{ij}_k\}$

Given this example, a Distributed Constraint Reasoning (DCR) platform must allow agents to have a reliable communication tool that allows sending and receiving messages, in order to find the feasible solutions.

3 RELATED WORK

Recently, B. Lutati and al. (Lutati et al., 2014) have proposed a MAS platform, called AgentZero. This

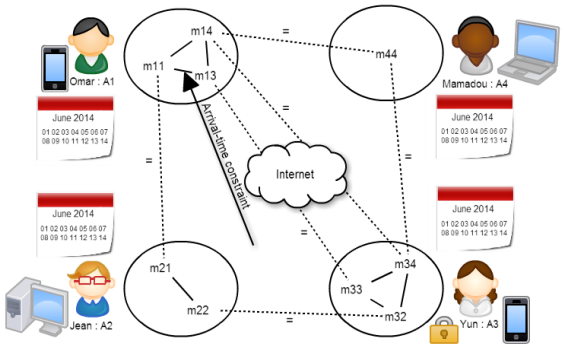

Figure 1: Meeting Scheduling Problem modeled as DisCSP.

tool can be considered as a new addition to the available MAS tools in general and to the DCR research field in particular. The authors claim that AgentZero is generic and applicable to many domains, specifically introducing benefits for the DCR simulation domain. However, the platform has been designed only for simulation use and used only by researchers in Distributed Constraint Reasoning. So developing and setting computer software for real problems based on DCR is not simple and remains a difficult task for users in general.

In (Petcu, 2006) A. Petcu. Proposes a Framework for Open Distributed Optimization (FRODO). The framework is implemented in Java, and simulates a multiagent environment in a single Java virtual machine. Each agent in the environment is executed asynchronously in a separate execution thread, and communicates with its peers through message exchange. FRODO comes with several built in algorithms and a suite of problem generators for benchmarking.

The authors of (Sultanik et al., 2007) proposed a open-source tool for solving DCR, called DCOPolis. DCOPolis is an open-source framework designed to abstract algorithm implementation from the underlying platform (i.e. hardware, network, operating system). This allows a single implementation of an algorithm to be run in simulation (i.e. on top of the NS2 network simulator with AgentJ).DCOPolis differs from existing DCR frameworks and simulators, however, it supports a novel type of simulation in which the runtime of any distributed algorithm can be accurately estimated on a single physical computer.

Researchers in DCR are concerned with developing new algorithms, and comparing their performance with existing algorithms. Therefore, in (Wahbi et al., 2011) the authors present an open source Java library , called DisChoco which aims at implementing DCR algorithms from an abstract model of agent . DisChoco allows to represent both DisCSPs and DCOPs, as opposed to other platforms. A single im-

plementation of a DCR algorithm can run as simulation on a single machine.DisChoco is a elegant platform, but all the different issues of realistic uses and actual deployment have not been addressed.

Developing intelligent software applications based on DCR algorithms is a difficult task, because the programmer must explicitly juggle between many very different concerns, including centralized programming, distributed programming, asynchronous and concurrent management of distributed structures, communication concerns and others. In addition, there are very few open-source tools for solving DCR problems in a physically distributed environment. In this paper we have been looking for a singular platform that would possess not only simulation qualities, but especially designed for realistic and actual deployment. JChoc platform is a new added value which allows bridging the gap between simulation and realistic use. To our knowledge, this is the first DCR platform respecting FIPA standards and specifications.

4 JCHOC PLATFORM

4.1 JChoc Description

The best way to prove the effectiveness of a proposed distributed constraint reasoning algorithm, is to use it in a realistic multi-platform agent. This is how we can reduce the gap between theory and practice. JChoc is a distributed constraints multiagent platform proposed for solving combinatorial problems within a specific distributed environment. It can also be used to analyze and test the algorithms proposed by constraints programming community. This platform is presented in the form of programming environment (API) and applications to help different types of users. Hence, JChoc can be easily appropriated by two main actors:

- Developers to design and develop applications (e.i. client application, web application, mobile application, etc...) within distributed constraints programming based on JChoc API;
- Non-expert user to interact directly with applications based on distributed constraints programming.

This proposed platform has several advantages:

- A distributed constraints problem can be easily addressed and solved in a realistic environment by unsophisticated users;
- The performances of the proposed protocols (i.e. ABT, AFC, Adopt, etc...) can be actually tested and proved in a realistic communication channel (i.e. WLAN WPAN WMAN WWAN);
- It offers a modular software architecture which accepts extensions easily (i.e. security, confidentiality, cryptography, etc...);
- Thanks to the extensibility of JADE communication model (JADE, 2013), JChoc allows the development of multiagent systems and applications consistent with Foundation for Intelligent Physical Agents (FIPA)[1] standards and specifications;
- Thanks to the the robustness of Choco platform (Jussien et al., 2008), complex agent (i.e. multiple variables per agent) can easily address and solve its local sub-problem and use solutions as a compiled domain.

This platform consists of several modules presented as services. The main constraint programming services offered are based Distributed Constraint Reasoning Protocols (DCRP) and Choco Solver (CS). Choco is a platform for research in centralized constraint programming and combinatorial optimization. This choice of Choco enabled us to benefit from the modules already implemented in it. In the next section, we will study the different elements of JChoc platform.

4.2 JChoc Architecture

JChoc architecture is motivated by FIPA specifications, it allows the development of multiagent systems and applications conforming to MAS standards. It is implemented in JAVA and provides classes that implement and inherit from JADE and Choco platforms to define the behavior of specific agents. Figure 2 represents the main JChoc architectural elements. This platform has five main modules.

- DCRP ≪Distributed Constraint Reasoning Protocols≫ provides distributed constraints protocols as service. This element defines new types of messages and implements the behavior of the agent when receiving and sending a specific type of information (e.i. ABT, AFC, Adopt, etc...);
- CS ≪Choco Solver≫ provides the ability to address and resolve local CSP sub-problem;
- DF ≪Director Facilitator≫ provides a service of ”yellow pages” to the platform;
- ACC ≪Agent Communication Channel≫ manages the communication between agents;

[1] http://www.fipa.org/

- AMS ≪Agent Management System≫ oversees the registration of agents, their authentication, their access and the use of the system.

These five modules are activated at each time the platform is started.

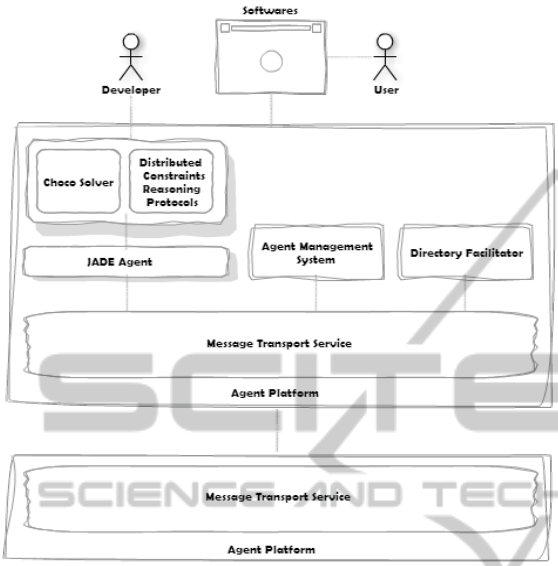

Figure 2: The JChoc Architecture.

The JADE agent is also a key player in our platform. Thanks to JADE an Agent Identifier (AID) identifies an agent uniquely.

JChoc uses extensively a sniffing tool for debugging, or simply documenting conversations between agents. The sniffer subscribes to AMS agent to be notified of all platform events and of all message exchanges between a set of specified agents. When the user decides to monitor an agent or a group of agents, every message directed to, or coming from, that agent/group is tracked and displayed in the sniffer GUI. The user can select and view the details of every individual message, save the message or serialize an entire conversation as a binary file.

5 USING JCHOC IN DISTRIBUTED ENVIRONMENT

In this section we present how to use the JChoc platform in real distributed environment. The MSP problem depicted in figure 1 is used to illustrate this proposed platform. Initially we generate a sub-problem for each agent involved in the global DisCSP problem, modeled by an expert as an XML file, which allows standardizing the syntactic structure of the sub-problems. A sub-problem containing only the information necessary for a single agent, so he can participate in solving the global problem in a real distributed environment.

Figure 3 shows an example of representation of the MSP sub-problem defined above in the XDisCSP format. Each variable has a unique ID, which is the concatenation of the ID of its owner agent and index of the variable in the agent. This is necessary when defining constraints (scope of constraints). For constraints, we used two types of constraints: TKC for Totally Known Constraint and PKC for Partially Known Constraint. Constraints can be defined in extension or as a Boolean function. Different types of constraints are predefined: equal to $eq(M_i,M_j)$, different from $ne(M_i,M_j)$, greater than or equal $ge(M_i,M_j)$, greater than $gt(M_i;M_j)$,etc. In this sub-problem there is 1 complex agent A_3 which controls exactly 3 variables. The domain of A_3 con-

```
<?xml version="1.0" encoding="UTF-8"?>
<instance>
        <presentation name="MSP" type="DisCSP"
        model="Complex" constraintModel="TKC" format="XDisCSP 1.0" />
    <domains nbDomains="2">
        <domain name="D1" nbValues="14">1..14</domain>
    </domains>

    <variables nbVariables="3">
        <variable name="M3.2" id="1" domain="D1" description="M_2" />
        <variable name="M3.3" id="2" domain="D1" description="M_3" />
        <variable name="M3.4" id="1" domain="D1" description="M_4" />
    </variables>

    <constraints nbConstraints="3">
        <constraint model="TKC" name="C0" reference="ArrivalTime"
            scope="M3.2 M3.3 2" arity="2">
            <parameters>M3.2 M3.3 2</parameters>
        </constraint>
        <constraint model="TKC" name="C1" reference="ArrivalTime"
            scope="M3.3 M3.4 2" arity="2">
            <parameters>M3.3 M3.4 2</parameters>
        </constraint>
        <constraint model="TKC" name="C2" reference="ArrivalTime"
            scope="M3.2 M3.4 2" arity="2">
            <parameters>M3.2 M3.4 2</parameters>
        </constraint>
    </constraints>
    <predicates nbPredicates="2">
        <predicate name="ArrivalTime">
            <parameters>int Mi,int Mj,int cte</parameters>
            <expression>
                <functional>ge(abs(sub(Mi,Mj)), cte)</functional>
            </expression>
        </predicate>
        <predicate name="eq">
            <parameters>int Mi,int Mj</parameters>
            <expression>
                <functional>eq(Mi,Mj)</functional>
            </expression>
        </predicate>
    </predicates>
    <agents_neighbours>
        <agents_parent>
            <agent name="A1">
                <constraints nbConstraints="2">
                    <constraint model="TKC" name="C0" reference="eq"
                        scope="M1.4 M3.4" arity="2">
                        <parameters>M1.4 M3.4</parameters>
                    </constraint>
                    <constraint model="TKC" name="C1" reference="eq"
                        scope="M1.3 M3.3" arity="2">
                        <parameters>M1.3 M3.3</parameters>
                    </constraint>
                </constraints>

            </agent>
            <agent name="A2">
                <constraints nbConstraints="1">
                    <constraint model="TKC" name="C0" reference="eq"
                        scope="M2.2 M3.2" arity="2">
                        <parameters>M2.2 M3.2</parameters>
                    </constraint>
                </constraints>

            </agent>
        </agents_parent>

        <agents_children>
            <agent name="A4" id="5" variable="M3.4" />
        </agents_children>
    </agents_neighbours>
</instance>
```

Figure 3: Definition of DMS sub-problem in XDisCSP format.

tain 14 values $D_3 = \{1...14\}$. There are three constraints of Arrival time $ge(abs(sub(M_i,M_j))$: the first constraint is between $M_{3.2}$ and $M_{3.3}$ the second one is between $M_{3.3}$ and $M_{3.4}$ and the third is between $M_{3.2}$ and $M_{3.4}$, three constraints of equality $eq(M_i,M_j)$: between $M_{1.4}$ and $M_{3.4}$, between $M_{1.3}$ and $M_{3.3}$, between $M_{2.2}$ and $M_{3.2}$ after defining our sub-problem we can configure our solver.

Once the sub-problem is generated, we can test the functioning of the platform in a physically distributed environment. So we chose machines that simulate the different agents of the problem, and filed each sub-problem in a machine, before launching it.

Figure 4 shows how the master launches its communication interface listening on the network. We start with instantiate the dissolver object (line 6), This class models the distributed problem when JChoc is used to solve a problem in a real distributed environment. All information on distributed problem is encapsulated in this object (identities of agents, inter-agent constraints, protocol, etc.). Then, we define the type of master (line 7) (ABT in this case). Finally, we trigger the container and we launch the master (lines 8-9).

```
1  import JChoc.DisSolver;
2
3  public class Master
4  {
5      public static void main(String[] args)
6      {
7          DisSolver js = new DisSolver();
8          js.setType("MasterABT");
9          js.setGui(true);
10         js.run();
11     }
12 }
```

Figure 4: How the master launches its communication interface.

Figures 5-8 show how to launch JChoc agents. We start with instantiate the DisSolver object (line 6), followed by the agent and distributed sub-problem declaration which specifies the resolution algorithm to be used (line 7-8). Next, the declaration of the container containing the master with its IP address (line 9). Eventually, we launch the agent (line10).

```
1  import JChoc.DisSolver;
2
3  public class Omar
4  {
5      public static void main(String[] args) {
6          DisSolver js1 = new DisSolver();
7          js1.setType("AgentABT");
8          js1.addAgent("A1", "Problem1.xml",true,true);
9          js1.setContainer("192.168.1.21");
10         js1.run();
11     }
12 }
```

Figure 5: How to implement and launch JChoc DisSolver in Omar agent (A1).

```
1  import JChoc.DisSolver;
2
3  public class Jean
4  {
5      public static void main(String[] args) {
6          DisSolver js1 = new DisSolver();
7          js1.setType("AgentABT");
8          js1.addAgent("A2", "Problem2.xml",true,true);
9          js1.setContainer("192.168.1.21");
10         js1.run();
11     }
12 }
```

Figure 6: How to implement and launch JChoc DisSolver in Jean agent (A2).

```
1  import JChoc.DisSolver;
2
3  public class Yun
4  {
5      public static void main(String[] args) {
6          DisSolver js1 = new DisSolver();
7          js1.setType("AgentABT");
8          js1.addAgent("A3", "Problem3.xml",true,true);
9          js1.setContainer("192.168.1.21");
10         js1.run();
11     }
12 }
```

Figure 7: How to implement and launch JChoc DisSolver in Yun agent (A3).

```
1  import JChoc.DisSolver;
2
3  public class Mamadou
4  {
5      public static void main(String[] args) {
6          DisSolver js1 = new DisSolver();
7          js1.setType("AgentABT");
8          js1.addAgent("A4", "Problem4.xml",true,true);
9          js1.setContainer("192.168.1.21");
10         js1.run();
11     }
12 }
```

Figure 8: How to implement and launch JChoc DisSolver in Mamadou agent (A4).

The master waits for the confirmation of creation of all agents before ordering the start of the search. Thus, the problem can be solved using a specified implemented protocol (ABT for example).

6 EXPERIMENTAL RESULTS

6.1 Configuration Example

To experiment the JChoc platform in a physically distributed environment, we chose five machines with features **2.93 GHz**, **CORE(TM) 2 duo** with **2 GB RAM** that simulate agents. These machines are connected via the **WLAN** of our laboratory. We also chose *ABT* algorithm to solve Meeting Scheduling problems (MSP). In figure 1 above, we depict an example of problem solved by this platform in a live distributed environment network. This figure illus-

trates an instance of MSP viewed as DisCSP where each agent has a personal private calendar and a set of meetings each taking place in a specified location. In that example, there are four agents, A_1, A_2, A_3 and A_4, and four meetings, m_1, m_2, m_3 and m_4. Each agent has its own calendar divided into **14** slots. The time required for traveling among places where meetings can be scheduled is **2 slots**.

We have intentionally limited the number of agents to 4 for this problem needs, but the number of the agents can be easily extended to $\mathbf{N} \gg \mathbf{4}$ for the neediest problems.

Figures 9 and 10 show the GUI of the sniffer agent at the start and the end of ABT resolution. The canvas provides a graphical representation of the messages exchanged between sniffed **ABTagents**, where each arrow represents a message and each color identifies a type of conversation. For example agent A_1 sends an **OK?** message to informs A_2 that he has done a new assignment $m_{1.1}$**:1** (line 5).

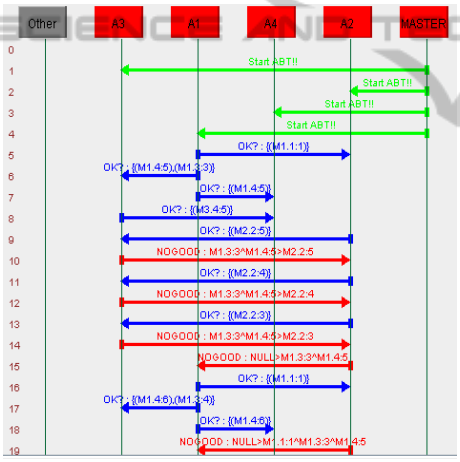

Figure 9: The start on sniffer agent GUI.

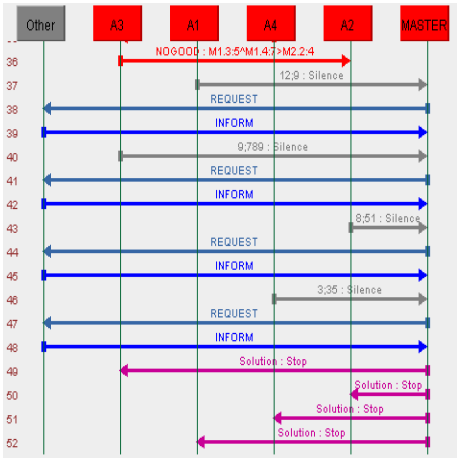

Figure 10: The finish on sniffer agent GUI.

If no new consistent value is found (line 10), A_3 generates a new **nogood** $m_{1.3}$**:3** $\wedge$ $m_{1.4}$**:5** $\Rightarrow m_{2.2} \neq 5$ by the resolution of existing nogoods. Eventually, the system can stabilize in a state where each agent has a value and no constraint is violated. This state is a global solution and the network has reached quiescence, meaning that no message is traveling through it (lines 37, 40, 43, 46). Once the solution is found, the master should be advised to spread the stop order to all agents (lines 49-52).
A solution to this example is :
$A_1 \longrightarrow (m_{1.1} : 3;\ m_{1.3} : 7;\ m_{1.4} : 1)$, $A_2 \longrightarrow (m_{2.1} : 3;\ m_{2.2} : 5)$, $A_3 \longrightarrow (m_{3.2} : 5;\ m_{3.3} : 7;\ m_{3.4} : 1)$, $A_4 \longrightarrow (m_{4.4} : 1)$.

6.2 Platform Scalability

The scalability of JChoc is the ability of the system, network, and process to handle a growing amount of work in a capable manner and its ability to be enlarged to accommodate that growth. In order to experiment our platform, we consider a large number of MSP instances. These Meeting Scheduling Problem are characterized by $< m, p, n, d, h, t, a >$, where m is the number of meetings, p is the number of participants, n is the number of inter-agent constraints d determines the number of days. Different time slots are available for each meeting, and h is the number of hours per day, t is a duration of the meeting and a is the percentage of availability for each participant. We present our results for the class $< m, p, n, 5, 10, 1, 90\% >$ and we vary three parameters : m, p, n (each agent has 2 meetings):

#p	#m	#n	#messages	Time (ms)
4	8	3	11	17070
5	10	5	11	17204
6	12	6	14	16144
7	14	7	14	17073
8	16	8	19	19180
9	18	9	24	20210
10	20	10	22	18294
11	22	11	32	20197
12	24	12	27	18516
13	26	15	30	20370
14	28	33	51	26073
15	30	35	105	31103
16	32	29	69	28914
17	34	33	175	38324
18	36	35	139	43172
19	38	38	141	37121
20	40	43	94	33457

Figure 11: Performance of JChoc platform using ABT protocol on the Meeting Scheduling Problem (MSP).

As shown in experimental results, in figure 7, the performance of our platform is measured in terms of network load (number of messages) and run-time execution. From these preliminary results we see that JChoc platform performs rapidly in small instances (#p $\in$ [4, 14]). The number of messages increases for #p $\in$ [15, 18] and reduces for #p $>$ 18. This scalability behavior is due to complexity of MSP problems. When the instance is hard the problem can be solved rapidly.

7 CONCLUSION

In this paper, we have proposed a modular, reliable, deployable and distributed software architecture, called JChoc DisSolver, which can be used easily for several real combinatorial problems. The main purpose of our platform is to break down the barriers to building new and innovative applications. The possibility of combining the expressiveness of Choco, the extensibility of JADE and our powerful Distributed Constraint Reasoning Add-On bring a strong added value in the development of innovative applications based on Constraints Programming paradigm. The JChoc platform presented in this paper has been designed to support extensions: security, cryptography.

In this work, we have implemented ABT protocol and solved the Meeting Scheduling problem (MSP) in a real distributed environment. We found that by using this platform we can adopt easily any proposed protocol for solving distributed constraint problem in such environment.

Future activities are focusing on enhancing the platform by the implementation of other DCR algorithms and to enrich the graphical user interface to make it easier to use for researchers. Another direction of improvement is to allow JChoc platform to be suitable to mobile devices. We plan also to implement new approaches of confidentiality.

REFERENCES

Béjar, R., Domshlak, C., Fernández, C., Gomes, C., Krishnamachari, B., Selman, B., and Valls, M. (2005). Sensor networks and distributed csp: communication, computation and complexity. In *Artificial Intelligence 161:1-2, 117-148*.

Bessiere, C., Brito, I., Maestre, A., and Meseguer, P. (2005). Asynchronous backtracking without adding links: A new member in the abt family. In *Artificial Intelligence,161:724*.

Ezzahir, R., Bessiere, C., Benelallam, I., Bouyakhf, E., and Belaissaoui, M. (2008). Dynamic backtracking for distributed constraint optimization. In *Proceeding of the 2008 conference on ECAI 2008, 901-902*. Amsterdam, The Netherlands, The Netherlands, IOS Press.

Ezzahir, R., Bessiere, C., Wahbi, M., Benelallam, I., and Bouyakhf, E. (2009). Asynchronous interlevel forward-checking for discsps. In *Principles and Practice of Constraint Programming (CP-09)*.

Galley, M. (2000). Distributed constraint programming platform using sjavap. In *http://cs.fit.edu/Projects/asl/#MELY*.

Gershman, A., Meisels, A., and Zivan, R. (2009). Asynchronous forward bounding for distributed cops. In *Journal of Artificial Intelligence Research, 34, 61-88*.

Hamadi, Y. (2003). Disolver : A distributed constraint solver. In *Technical Report MSR-TR-2003-91, Microsoft Research*.

JADE (2013). Java agent developpement framework. In *URL http://jade.tilab.com/*.

Jussien, N., Rochart, G., and Lorcal, X. (2008). Choco: an open source java constraint programming library. In *CPAIOR'08 Workshop on Open-Source Software for Integer and Contraint Programming(OSSICP'08)*. France, Paris.

Kitano, H., Tadokoro, S., Noda, I., Matsubara, H., Takahashi, T., Shinjou, A., and Shimada, S. (1999). Robocup rescue: Search and rescue in large-scale disaster as a domain for autonomous agents research. In *IEEE International Conference on System, Man, and Cybernetics*.

Lutati, B., Levit, V., , and Meisels, A. (2014). Agentzero: A framework for simulating and evaluating multiagent algorithms. In *Agent-Oriented Software Engineering*. Springer.

Meisels, A. and Lavee, O. (2004). Using additional information in discsp search. In *5th workshop on distributed constraints reasoning, DCR'04*.

Meisels, A. and Zivan, R. (2007). Asynchronous forward-checking for discsps. In *Constraints 12, 131-150*.

Modi, P., Shen, W., Tambe, M., and Yokoo, M. (2005). Adopt: asynchronous distributed constraints optimization with quality guarantees. In *Artificial Intelligence 161:1-2, 149 180*.

Petcu, A. (2006). Frodo: a framework for open/distributed optimization. In *Technical Report Technical Report EPFL:2006/001, LIA, EPFL, CH-1015 Lausanne,http://liawww.epfl.ch/frodo/*.

Sultanik, E., Lass, R., and Regli, W. (2007). Dcopolis: A framework for simulating and deploying distributed constraint optimization algorithms. In *CP-DCR*.

Wahbi, M., Ezzahir, R., Bessiere, C., and Bouyakhf, E. (2011). Dischoco 2: A platform for distributed constraint reasoning. In *DCR'11, pages 112-121*. URL http://www.lirmm.fr/coconut/dischoco/.

Wallace, J. R. and Freuder, C. E. (2002). Constraint-based multi-agent meeting scheduling: effects of agent heterogeneity on performance and privacy loss. In *3rd workshop on distributed constrait reasoning, DCR'02, pages 176-182*.

Yeoh, W., Felner, A., and Koenig, S. (2008). Bnb-adopt: an asynchronous branch and bound dcop algorithm. In

AAMAS08: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems, 591-598.

Yokoo, M. (2000). Algorithms for distributed constraint satisfaction problems: A review. In *Autonomous Agents and multiagent Systems 3, 185-207*.

Yokoo, M. (2001). Distributed constraint satisfaction. In *:Foundation of Cooperation in multiagent Systems*. Springer.

Yokoo, M., Durfee, E., Ishida, T., and Kuwabara, K. (1992). Distributed constraint satisfaction for formalizing distributed problem solving. In *International Conference on Distributed Computing Systems, 614-621*.

Yokoo, M., Durfee, E., Ishida, T., and Kuwabara, K. (1998). The distributed constraint satisfaction problem: formalization and algorithms. In *IEEE Transactions on Knowledge and Data Engineering, 10(5):673-685*.

Yokoo, M. and Hirayama, K. (1995). Distributed breakout algorithm for solving distributed constraint satisfaction problems. In *Proceedings of the First International Conference on MultiAgent Systems*. MIT Press.