Mobile Robotic JChoc DisSolver
A Distributed Constraints Reasoning Platform for Mobile Multi-robot Problems

Zakarya Erraji[1], Mounia Janah[1], Imade Benelallam[1,2] and El Houssine Bouyakhf[1]
[1]*LIMIARF − FSR, University Mohammed V, Rabat, Morocco*
[2]*INSEA, Rabat, Morocco*

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

Abstract: Due to the computational complexity (NP-Complete) of Constraint Programming (CP), several researchers have abandoned its use in robotic research field. In the last decade, as many approaches of real-time constraint handling have been proposed, constraint programming has proved to be a stand-alone technology that can be used in several research fields.
Even if mobile robotics is a complex research area, in this paper, we prove that distributed constraint reasoning techniques can be utilized as a very elegant formalism for multi-robot decision making. First, we describe dynamic distributed constraint satisfaction formalism, the new platform architecture "RoboChoc" and specify how decision making can be controlled in multi-robots environment using dynamic communication protocols. Then we provide an example application that illustrates how our platform can be used to solve multi-robot problems using constraint programming techniques.

1 INTRODUCTION

With increasing deployment of applications involving multiple robots and unmanned air vehicles, there is growing need for programming platform with realistic use context. Coordination between robots to solve a combinatorial problem, is one of the most complex research area in mobile robotics. In the last decade, as many approaches of real-time constraint handling have been proposed, constraint programming has proved to be a stand-alone technology that can be used in several research fields. This paper presents a new multi-robot platform named Mobile Robotic JChoc DisSolver (RoboChoc), based on constraint programming to solve multi-robot combinatorial problems in dynamically changing environments. The resolution of such problem is impacted by the tools that are used. The RoboChoc platform is based on a powerful constraint solver, named Choco (Jussien et al., 2008), a multi-agent platform JADE (JADE, 2013) that insures best management and communication between agents, and a Robot Operation System (ROS) (Quigley et al., 2009) that provides libraries and tools to help software developers create robot applications. RoboChoc uses a ROS Layer that benefits from the ROS community contributions (e.g. navigation (Zaman et al., 2011), localization (Thomas and Ros, 2005), etc). Since a problem is modeled using constraint programming techniques, users can implement the application and solve it easily.

The paper is organized as follows. Section 2 presents related works. Section 3 shows a brief definition of Distributed and Dynamic Constraint Satisfaction Problems (DisCSPs and DDisCSPs). Section 4 presents the RoboChoc platform architecture. Section 5 describes an example of application. Finally, Section 6 presents our conclusions and perspectives.

2 RELATED WORKS

Many research focus upon the application of constraint programming in the robotic field. ThingLab (Borning, 1981) proposed a software package that allows users to graphically design and simulate a system which can use constraints to define the properties of the system and guided simulation of circuits, mechanical linkages, and other geometries.

Erraji, Z., Janah, M., Benelallam, I. and Bouyakhf, E.
Mobile Robotic JChoc DisSolver - A Distributed Constraints Reasoning Platform for Mobile Multi-robot Problems.
DOI: 10.5220/0005833003040310
In *Proceedings of the 8th International Conference on Agents and Artificial Intelligence (ICAART 2016) - Volume 1*, pages 304-310
ISBN: 978-989-758-172-4

The research of D. K. Pai (Pai, 1989; Pai, 1991) describes a new framework for robot programming using constraint satisfaction techniques. Pai defined two special classes of CSP variable. Input variables were used to capture current state and output variables were defined such that their values can be taken from the solution and translated to vehicle actuator commands. For each iteration, the previous solutions were fed into the solver, and then the input variables were updated. A repair algorithm then fixed the solution given the changes and the constraints.

Zhang and Mackworth (Mackworth, 1997; Zhang and Mackworth, 1994; Zhang and Mackworth, 2002) proposed a constraint-based system for the design of intelligent and hybrid agents. The hybrid agents are those which can interact continuously and discretely with their environments. A controller based on constraint satisfaction drives the system toward satisfying all constraints. During the task, system detects the dissatisfaction of any constraint, the controller can initiate a backtrack.

Other contributions were recently proposed. the researches of Richard S. Stansbury and Arvin Agah (Stansbury and Agah, 2012) presents a decision making framework that help robots to perform the best decision at any given moment. The selected task must satisfy all constraints. The framework models this decision as a Constraint Satisfaction Problems (CSPs). Where the contribution of Doru Panescu and Carlos Pascal (Panescu and Pascal, 2014) presents a multi-robot system planning based on a combination between a multiagent system and a constraint satisfaction problem approach. They make some simulation tests with a multi-robot system presented by four robots to solve an assembly goals. They use in these simulations the coloured Petri net models, specifically developed for a distributed constraint satisfaction algorithm.

To date, no researchers have proposed a constraint programming platform to practically implement multi-robot application. DisCSP has been used as a formalism for diagnosing coordination failures in multi-robot systems (Meir et al., 2006). This work shows that, in general, a trade-off exists between the computational requirements of the algorithms, and their diagnosis results. In contrast, we use sophisticated APIs (JADE, ROS and Choco) to model the ideal coordination relationships between robots. We present results from experiments conducted in realistic usage conditions. In addition, previous contribution doesn't use standard and distributed platform, in contrast to our work.

3 BACKGROUND

3.1 Distributed Constraint Satisfaction Problems

Constraint Programming distinguishes between the description of the constraints involved in a problem on the one side, and the algorithms and heuristics used to solve the problem on the other side. 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 DisCSP (or a distributed constraint network) has been formalized as a 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.
- $D = \{D(x_1), ..., D(x_n)\}$ 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).

3.2 Dynamic and Distributed Constraint Satisfaction Problems

Many hard practical multi-robot problems can be seen as DisCSPs. Most Distributed Reasoning platform assume that problems are static. This is a limitation for dynamic problems that change over time and especially in dynamically changed environments. For example, any robot belonging to a set of robots that explore the moon in a cooperative mission can be lost if hit by a meteor. Thus the problem will change. Hence in a dynamic environment a DisCSP may change over time, these changes could be due to addition/deletion of variables, constraints, or agents. The Distributed and Dynamic Constraint Satisfaction Problems (DDisCSPs) can be described as a tuple (A, X, D, C, δ) where:

- A, X, D and C remain as described in DisCSPs.
- δ is the function that introduce changes.

Many DDisCSPs approaches (e.i : DynABT (Omomowo et al., 2008)) are proposed to solve such problem, and can be easily implemented in RoboChoc platform.

4 PLATFORM ARCHITECTURE

4.1 RoboChoc Description

RoboChoc is a JADE-based platform. This platform is a distributed constraint multi-robot system, proposed for mobile robotic applications. It can also be used to analyze and test algorithms proposed by constraints programming community. This platform is presented in the form of programming environment (API) and applications to help different types of robotic users. Hence, RoboChoc can be easily adopted by two main actors:

- Developers to design and develop multi-robot applications.
- Non-expert user can interact directly with applications based on distributed constraint programming.

This proposed platform has several advantages:

- A multi-robot problem modeled as distributed constraint problem, can be easily addressed and solved in a realistic environment by non-expert users;
- The performance of the proposed protocols (e.g. ABT, AFC, Adopt, etc) can be actually tested and proved in a realistic robot cordination, using communication channel (i.e. WLAN WPAN WMAN WWAN);
- It offers a modular software architecture which accepts extensions easily (e.g. security, confidentiality, cryptography, etc);
- Thanks to the extensibility of JADE communication model (JADE, 2013), RoboChoc 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.
- Thanks to the the Robot Operation System ROS (Quigley et al., 2009), The distributed nature of ROS also fosters a large community of user-contributed packages that add a lot of value on top of the core ROS system.

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 RoboChoc platform.

4.2 RoboChoc Architecture

RoboChoc architecture allows FIPA specifications. 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 1 represents the main RoboChoc architectural elements. This platform has five main modules.

- Distributed Constraint Reasoning Protocols unit: 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.g. ABT, AFC, Adopt, etc...);
- Constraint Solver Unit: provides the ability to address and resolve local CSP sub-problem by using the Choco Solver;
- Services Management Unit: manages services in the platform. It's based on the Director Facilitator (DF) agent of JADE;
- Communication Management Unit: manages the communication in the platform. It's based on the Agent Communication Channel (ACC) of JADE;

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

- Authority Unit: oversees the registration of robots, their authentication, their access and the use of the system.
- ROS Layer Unit : provides a set of tools to apply a decision or to read the robot situation (e.g. scan sensor state, moving the robot, etc.).

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

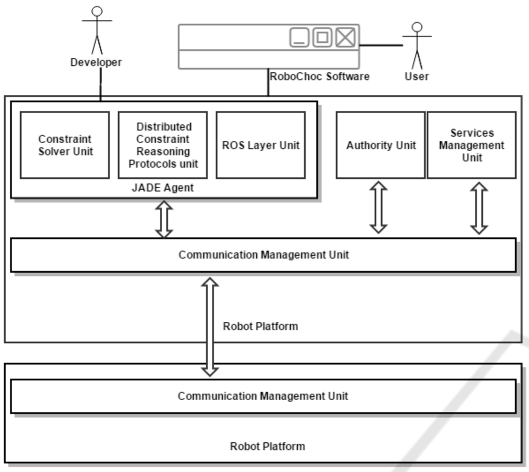


Figure 1: RoboChoc Architecture.

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

RoboChoc uses extensively a sniffing tool for debugging, or simply documenting conversations between robots. The sniffer subscribes to Authority Unit to be notified of all platform events and of all message exchanges between a set of specified robots. When the user decides to monitor a robot or a group of robots, every message directed to, or coming from, that robot/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.

The platform can be used with all robots that use ROS as an operating system, and no additional configuration is needed. The user has to choose the appropriate robots for its application.

5 EXAMPLE APPLICATION

5.1 Description of 3D N^2 Queens Problem

The n-queens problem is a classical combinatorial problem that can be formalized and solved by constraint satisfaction problem. In the n-queens problem, the goal is to put n queens on an n×n chessboard so that none of them is able to attack (capture) any other. Two queens attack each other if they are located on the same row, column, or diagonal on the chessboard. This problem is called a constraint satisfaction problem because the purpose is to find a configuration that satisfies the given conditions (constraints).

Three Dimensional Queens Problems is an extension of the well known n-queens problem in 3D dimensional space. The goal in this problem is to place N^2 queens in an $N \times N \times N$ cube such that none of them is able to attack any other. A queen can attack another if they exist in the same row, column, or diagonal formed by the division of the tree dimensional space. We mention that the space is divided into n^3 position which the n^2 queens will take place.

This problem was treated, modeled and solved in the following paper, using linear programing: Three Dimensional Queens Problems (Allison et al., 1989). In this research they found that there is no solution for the n less than 11, n=12, n=14. They found exactly for n=11, 528 solutions using an other formalism far from constraint programming.

In our work, we propose for the first time, a DisCSPs modeling for the same problem. The DisCSP that model the problem is formalized as a tuple (X, D, C, A, ψ) where:

- X = { $X_{1,1}$, ..., $X_{n,n}$} is a set of $n \times n$ variables.
- D = {$D_{1,1}$={1..n}, ..., $D_{n,n}$={1..n}} is a set of domains, where $D_{i,j}$ is a finite set of possible values for the variable $X_{i,j}$.
- C = {
  C1=$\{(X_{i,j1} \neq X_{i,j2}) and (\left|(X_{i,j1} - X_{i,j2}\right| \neq |j1-j2|) \; / \; (i,j1,j2) \in [1,4]\}$ where $X_{i,j1}$ and $X_{i,j2}$ designate queens existing in the same row.

  C2=$\{(X_{i1,j} \neq X_{i2,j}) and (\left|(X_{i1,j} - X_{i2,j}\right| \neq |i1-i2|) \; / \; (i1,i2,j) \in [1,4]\}$ where $X_{i1,j}$ and $X_{i2,j}$ design queens existing in the same column.

  C3={if $i1 \neq i2$ and $j1 \neq j2$ and $|i1-i2| \neq |j1-j2|$ then $X_{i1,j1} \neq X_{i2,j2}$ / $(i1,i2,j1,j2) \in [1,4]$}
  } is the set of constraints.
- A = { $A_{1,1}$, ..., $A_{n,n}$} is a set of $n \times n$ agents.
- ψ is the function that assign the variable $X_{i,j}$ to the agent $A_{i,j}$.

In this modeling, each queen $Q_{i,j}$ is represented by a variable $X_{i,j}$

5.2 Using RoboChoc to Solve the 3D N^2 Queens Problem

The problem can be solved by using RoboChoc. Hence, each agent is presented as a robot. We use XML files to describe the sub-problems for each robot. The figure 2 shows an example of the XML file used to describe the sub-problem for the robot named Robot3.1.

Each variable has a unique ID, which is the ID of its owner robot. This is necessary when defining constraints (scope of constraints). For constraints, we used two types: 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 $eq(V_i,V_j)$, different from $ne(V_i,V_j)$, greater than or equal $ge(V_i,V_j)$, greater than $gt(V_i;V_j)$, etc. In this sub-problem the constraints presented in the DisCSPs modeling are defined by the predicate p1. In the tag topics we indicate the topics to be created and used by the ROS Layer Unit. these topics will be used to control the robot motion in this application. After defining our sub-problem we can configure our solver.

The problem can be solved using the ABT algorithm assuming that no perturbation will occur during the resolution, because we will use the Modular Open Robots Simulation Engine(Morse) ((Echeverria et al., 2011)) with ROS as a middle ware. The adequate type of robot that can represent queens for this application is the drone Quadrotor_Dynamic.

We solved the modeled problem with the Constraint Solver Unit in the platform, and we have found exactly the same results presented in the research of (Allison et al., 1989). To simplify the problem in this application, we will use just n=4 as the figure 5 shows. For the reason that there is not any solution for n=4, the robots who can't belonging to the solution will be taken from the operational scene. Thus only robots that will satisfy all constraints will fly to take their positions. We can test the functioning of the platform in a physically distributed environment. So we chose machines that simulate the different robot of the problem, and filed each sub-problem in a machine, before launching it.

Figure 3 shows how the master launches its communication interface listening on the network. We begin by instantiating the AgentsContainer object (line 7), This class models the distributed problem when RoboChoc 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 8) (ABT in this case). Finally, we trigger the container and we launch the master (lines 10).

```
<?xml version="1.0" encoding="UTF-8"?>
<instance>
      <presentation type="DisCSP"/>
   <domains nbDomains="1">
       <domain name="D1" nbValues="4">1..4</domain>
   </domains>
   <variables nbVariables="1">
      <variable name="V3.1" domain="D1" />
   </variables>
   <topics nbTopics="1">
      <topic name="/drone3.1/waypoint3.1" />
   </topics>
   <predicates nbPredicates="1">
      <predicate name="p1">
      <parameters>int X int Y int i int j</parameters>
      <expression>
       <functional>and(neq(X, Y), neq(abs(minus(X, Y)),abs(minus(i-j))))</functional>
      </expression>
     </predicate>
   </predicates>
   <agents_neighbours>
      <agents_parent>
         <agent name="Robot1.1">
            <constraints nbConstraints="1">
               <constraint model="TKC" name="C5" reference="p1" scope="V1.1 V3.1 1 3">
                  <parameters>V1.1 V3.1 1 3</parameters>
               </constraint>
            </constraints>
         </agent>
         <agent name="Robot2.1">
            <constraints nbConstraints="1">
               <constraint model="TKC" name="C5" reference="p1" scope="V2.1 V3.1 2 3">
                  <parameters>V2.1 V3.1 2 3</parameters>
               </constraint>
            </constraints>
         </agent>
         <agent name="Robot1.3">
            <constraints nbConstraints="1">
               <constraint model="TKC" name="C5" reference="neq" scope="V1.3 V3.1">
                  <parameters>V1.3 V3.1</parameters>
               </constraint>
            </constraints>
         </agent>
         <agent name="Robot2.2">
            <constraints nbConstraints="1">
               <constraint model="TKC" name="C5" reference="neq" scope="V2.2 V3.1">
                  <parameters>V2.2 V3.1</parameters>
               </constraint>
            </constraints>
         </agent>
      </agents_parent>
      <agents_children>
         <agent name="Robot4.1" variable="V3.1" />
         <agent name="Robot3.2" variable="V3.1" />
         <agent name="Robot3.3" variable="V3.1" />
         <agent name="Robot3.4" variable="V3.1" />
         <agent name="Robot4.2" variable="V3.1" />
      </agents_children>
   </agents_neighbours>
</instance>
```

Figure 2: Example of a sub-problem for the robot3.1.

```
import JChoc.DisSolver;

public class Master
{
    public static void main(String[] args)
    {
        AgentsContainer mainConstainer = new
              AgentsContainer();
        mainConstainer.setAgentType("MasterABT");
        mainConstainer.addAgent("MASTER", false);
        mainConstainer.run();
}
```

Figure 3: How the master launches its communication interface.

```
 1  package jchoc.run;
 2  import jchoc.tools.AgentsContainer;
 3
 4  public class RunDrone
 5  {
 6      public static void main(String[] args) {
 7
 8          AgentsContainer agentContainer = new
                  AgentsContainer();
 9          agentContainer.setAgentType("AgentABT");
10          agentContainer.addAgent("Robot3.1","Robot3
                  .1.xml",true);
11          agentContainer.setMainContainer("
                  192.168.1.14");
12          agentContainer.run();
13              }
14  }
```

Figure 4: Run drone.

Figures 4 shows how to launch robots as agents. We start with instantiate the AgentsContainer object (line 8), followed by the agent and distributed subproblem declaration which specifies the resolution algorithm to be used (line 9-10). Next, the declaration of the container containing the master with its IP address (line 11). Eventually, we launch the agent (line12).

The master waits for the confirmation of creation all agents before ordering the start of the search. Thus, the problem can be solved using a specified implemented protocol (ABT for this example). Figures 5 and 6 show the initial and final state of resolution of the problem. Between these two states, the solving phase was performed to satisfy all constraints.

Figures 7 shows the exchanged messages in the resolution phase sniffed by the Sniffer Agent in the platform. ABT algorithm is executed autonomously by each robot. Robots do not have to wait for decisions of others but they are subject to a total(priority) order. Each robot tries to find an assignment satisfying the constraints with what is currently known from higher priority neighbors. When an robot assigns a solution to its local problem, the selected value is sent to lower priority neighbors. When no solution is possible for a local problem, the inconsistency is reported to higher robots in the form of a *nogood*. ABT computes a solution (or detects that no solution exists) in a finite time. The total ordering on robots is static.

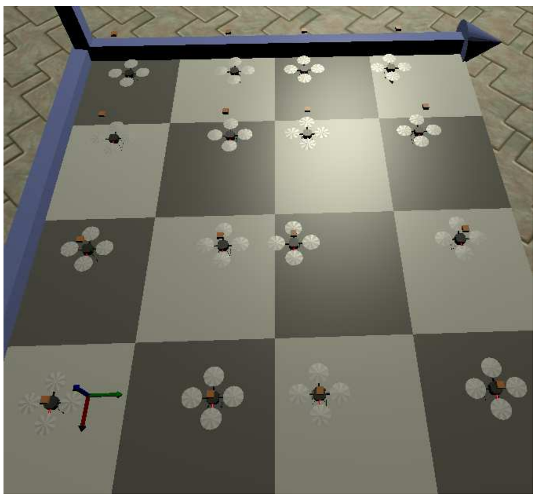


Figure 5: Initial state.

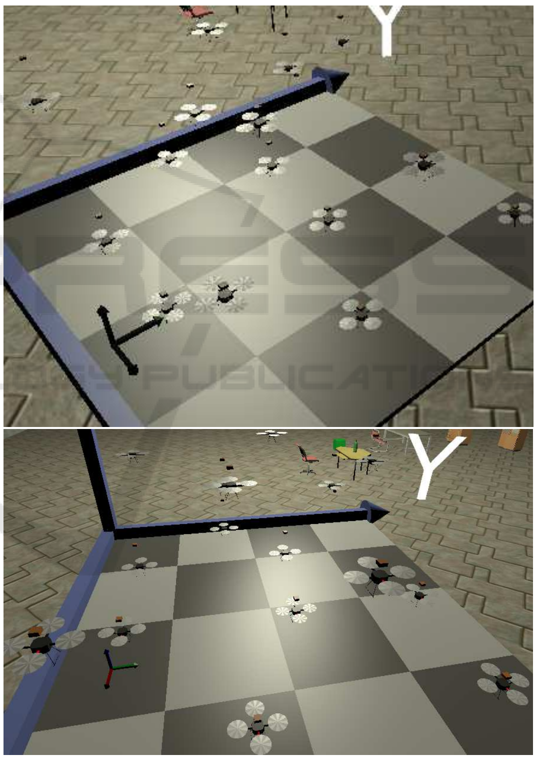
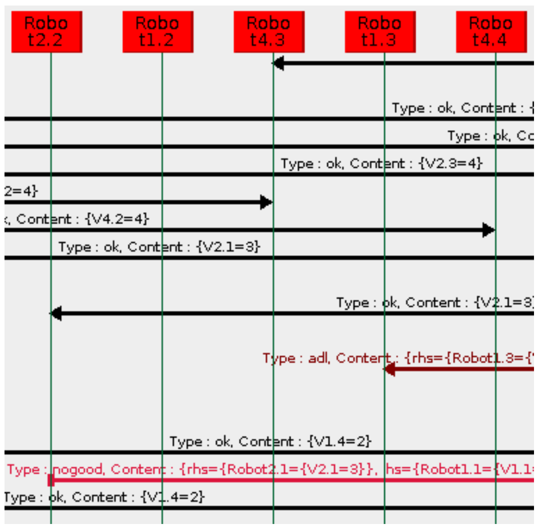


Figure 6: Final state.

6 CONCLUSIONS

The goal of this work was to propose a Distributed Constraints Reasoning (DCR) platform, which can be easily used to solve robotic application in a realistic use.RoboChoc platform presented in this paper has been designed to solve any multi-robot problems de-



```
Type : ok, Content : {
Type : ok, Co
Type : ok, Content : {V2.3=4}
2=4}
, Content : {V4.2=4}
Type : ok, Content : {V2.1=3}
Type : ok, Content : {V2.1=3
Type : adl, Content : {rhs={Robot1.3={
Type : ok, Content : {V1.4=2}
Type : nogood, Content : {rhs={Robot2.1={V2.1=3}}, hs={Robot1.1={V1.1
Type : ok, Content : {V1.4=2}
```

Figure 7: Sniffed messages in the solving phase.

signed as a DisCSP and to support security and cryptography extensions (thanks to JADE API).

In this work we have modeled the 3D N^2 queens problems as a DisCSP problem. We have used the MORSE Simulator to run drone-robots, where each queen is presented by a drone. Each drone performs as an ABT agent that applies the decision made by the algorithm or reads his actual situation using the ROS Layer Unit, which is independent of the solving protocol. Users can contribute in the ROS Layer Unit by adding new ROS applications to simplify the use of new destined robotic task.

Future works are focusing on enhancing the platform by the implementation of other necessary API in ROS layer, and by diagnosing and proposing new techniques for coordination failures between mobile robots.

REFERENCES

Allison, L., Yee, C., and McGaughey, M. (1989). Three-dimensional queens problems. Monash University, Department of Computer Science.

Borning, A. (1981). The programming language aspects of thinglab, a constraint-oriented simulation laboratory. volume 3, pages 353–387. ACM.

Echeverria, G., Lassabe, N., Degroote, A., and Lemaignan, S. (2011). Modular openrobots simulation engine: Morse. In *Proceedings of the IEEE ICRA*.

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.

Mackworth, A. K. (1997). Constraint-based design of embedded intelligent systems. volume 2, pages 83–86. Springer.

Meir, K., Gal, A. K., Amnon, M., and Yehuda, E. (2006). Diagnosis of multi-robot coordination failures using distributed csp algorithms. In *American Association for Artificial Intelligence, 970-975*.

Omomowo, B., Arana, I., and Ahriz, H. (2008). Dynabt: Dynamic asynchronous backtracking for dynamic discsps. In *Artificial Intelligence: Methodology, Systems, and Applications*, pages 285–296. Springer.

Pai, D. K. (1989). Programming parallel distributed control for complex systems. In *Intelligent Control, 1989. Proceedings., IEEE International Symposium on*, pages 426–432.

Pai, D. K. (1991). Least constraint: A framework for the control of complex mechanical systems. In *American Control Conference, 1991*, pages 1615–1621.

Panescu, D. and Pascal, C. (2014). A constraint satisfaction approach for planning of multi-robot systems. In *System Theory, Control and Computing (ICSTCC), 2014 18th International Conference*, pages 157–162.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y. (2009). Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5.

Stansbury, R. S. and Agah, A. (2012). A robot decision making framework using constraint programming. volume 38, pages 67–83. Springer.

Thomas, F. and Ros, L. (2005). Revisiting trilateration for robot localization. volume 21, pages 93–101. IEEE.

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*.

Zaman, S., Slany, W., and Steinbauer, G. (2011). Ros-based mapping, localization and autonomous navigation using a pioneer 3-dx robot and their relevant issues. In *Electronics, Communications and Photonics Conference (SIECPC), 2011 Saudi International*, pages 1–5.

Zhang, Y. and Mackworth, A. K. (1994). Specification and verification of constraint-based dynamic systems. In *Principles and Practice of Constraint Programming*, pages 229–242.

Zhang, Y. and Mackworth, A. K. (2002). A constraint-based robotic soccer team. volume 7, pages 7–28. Springer.