TP 7 des architectures logicielles
Séance 7 : Architecture Multi-Agents

JChoc is a multi-agent platform for solving Distributed Constraints Reasoning (DCR) problems. It was written in Java and it's available as a Java (.jar) library.

The Distributed Meeting Scheduling Problem (DisMSP) is a truly distributed problem where agents may not desire to deliver their personal information to a centralized agent to solve the whole problem.

Let's consider an example of DisMSP and solve it with JChoc.

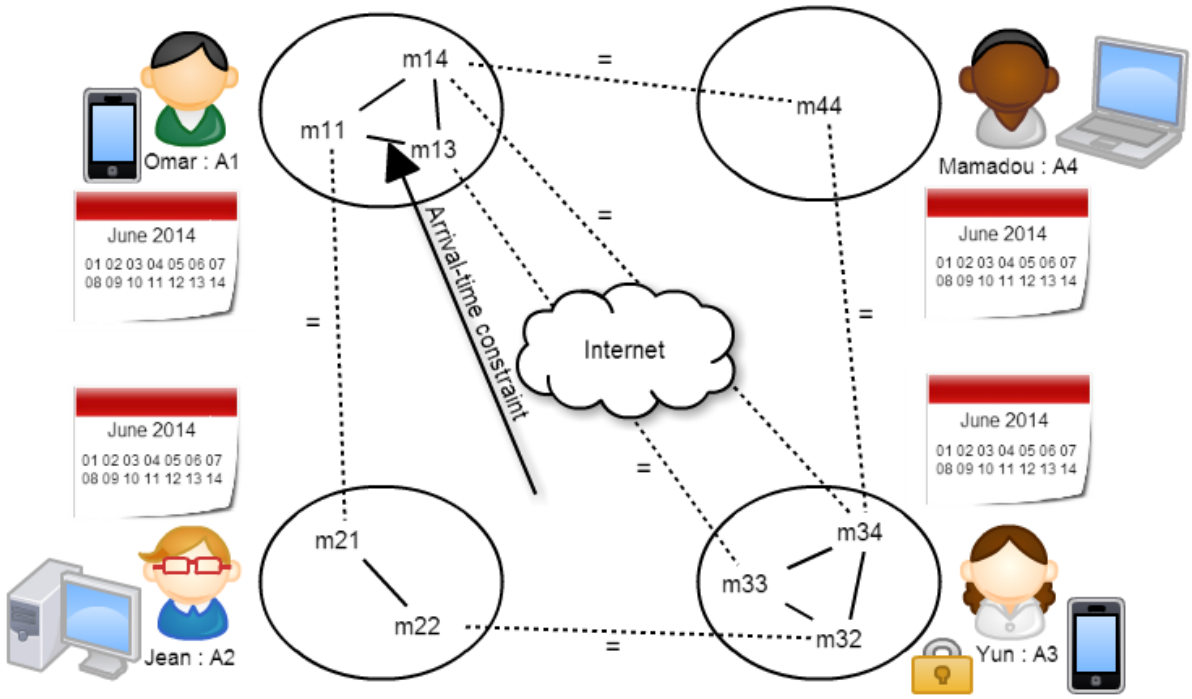


Figure 1 : The distributed meeting-scheduling problem modeled as DisCSP

Figure 1 shows four 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,

$X = \{m_{11}, m_{13}, m_{14}, m_{21}, m_{22}, m_{32}, m_{33}, m_{34}, m_{44}\}$.

- $D = \{D(m_{ik}) \mid m_{ik} \in X\}$ where,
  - $D(m_{11}) = D(m_{13}) = D(m_{14}) = \{s \mid s \text{ is a slot in calendar (A1)}\}$.
  - $D(m_{21}) = D(m_{22}) = \{s \mid s \text{ is a slot in calendar (A2)}\}$.

- $D(m_{32}) = D(m_{33}) = D(m_{34}) = \{s \mid s \text{ is a slot in calendar (A3)}\}$.
- $D(m_{44}) = \{s \mid s \text{ is a slot in calendar (A4)}\}$.

- For each agent A_i, there is a private arrival-time constraint (c^i_{kl}) between every pair of its local variables (m_{ik}, m_{il}). 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. Then, $C = \{ c^i_{kl}, c_{ij}^k \}$.

Step 1: Start a Master that can detect the silence in the network and inform the other agents of the end of solving.

- Create a new java project
- Import JChoc.jar file in your project library
- Create a Class "Master.Java", then follow the following steps:

Master.java

```
1
2  import JChoc.DisSolver;1
3
4  public class Master {
5
6      public static void main(String[] args) 2
7
8          DisSolver js= new DisSolver(); 3
9          js.setType("MasterABT");4
10         js.setNumberOfAgents(4);5
11         js.setGui(true)6
12         js.run();7
13
14     }
15
16 }
```

Figure 2: Class Master.java

  - Step 1: Import DisSolver class from the JChoc package
  - Step 2: Add a main method in the Master class
  - Step 3: Instantiate a DisSolver object
  - Step 4: Choose the type of the Master that you went to use: in the example, we chose to use an ABT master, in other words, ABT (Asynchronous BackTracking) as solving protocol.
  - Step 5: Add the number of agents to use.
  - Step 6: Choose to display the GUI or not.
  - Step 7: Call the run method to start the multi-agent Platform JChoc.
- Create and start Agents:
  - Each agent can code his part of the whole problem and put it in a xml file.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<instance>
    <domains nbDomains="1">
        <domain name="D1" nbValues="7">1..7</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: Example of the XML file in the JChoc Platform for the MSP problem

Figure 3 describes a part of the whole problem affected to Agent 3.

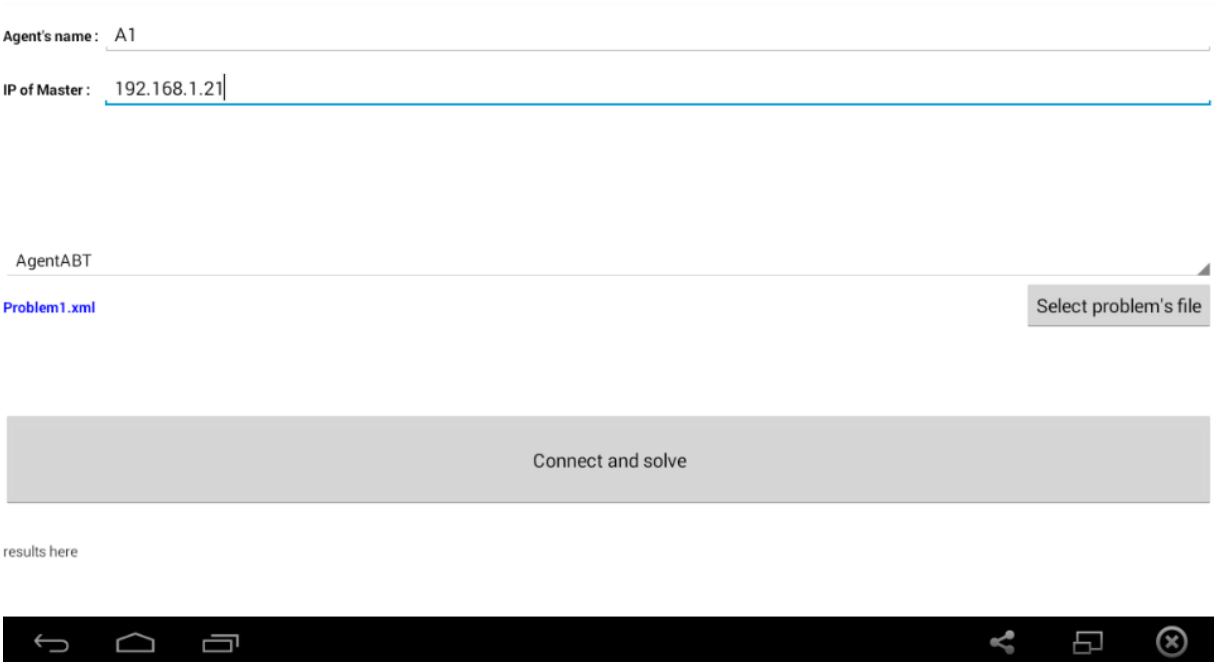
b) If we use machines, we can create a new class for each agent and add the following lines by modifying the arguments depending on the problem to solve.

```
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("192.168.1.37");
10         js1.run();
11     }
12 }
```

Figure 4: Class Agent

- Step 1: Import DisSolver class from JChoc package
- Step 2: Add the main method in the agent class
- Step 3: Instantiate a DisSolver object
- Step 4: Choose the type of the agent that you went to use: in the example we chose to use an ABT agent and automatically we will use ABT protocol to communicate and solve the whole problem.
- Step 5: Set agent arguments:
  - Name of agent.
  - His XML file
  - Show the GUI or not (true/false).
- Step 6: IP of Machine where we started the Master.
- Step 7: Call the run method to start the agent.

Or we can use Android devices to run agents and solve the problem; this is the mobile interface using the same steps to run the A1 agent:


Agent's name : A1
IP of Master : 192.168.1.21
AgentABT
Problem1.xml
Select problem's file
Connect and solve
results here