NTNU – Trondheim
Norwegian University of
Science and Technology

TDT4280
Multi Agent Systems and Game Theory

Project Part 1a

Working Together

Author:
Silje Irene Hansen
Marius Fagerland

February 15, 2015

1 Introduction

For this task we have used JADE to create our mulitagent system. The purpose of our system is to solve an arithmetic problem. The system consists of a TaskAdministrator agent, AdditionSolver agents, SubtractionSolver agents, DivisionSolver agents, MultiplicationSolver agents and DirectoryFacilitator (DF) agent. The last one, DirectoryFAcilitator is where the other solver agents can register their services and the TaskAdministrator can query for certain agents. The TaskAdministrator takes the arithmetic problem as argument. The arithmetic problem is written in postfix notation, the reason for this is explained in the Postfix Notation section. Further the TaskAdministrator decomposes the problem into sub-problems and delegates the sub-problems to the other solver agents which return an answer. A more detailed overview of this is to follow.

2 How to start JADE

To use JADE we used the IDE Eclipse to implement new agents and run the JADE environment. To do this we added the jade folder and jade.Boot to the build path of the project. The arguments in the build path were set to

-gui -agents add:Oving2.AdditionSolver;sub4:Oving2.SubtractionSolver;
mul:Oving2.MultiplicationSolver;div:Oving2.DivisionSolver;
TA:Oving2.TaskAdmin("3 4 1 - * 8 2 - * 6 + 2 /").

3 Postfix Notation

For our math problem we have chosen postfix notation. The reason for this is because the first operator will be a sub-problem. In contrast prefix where we would have to iterate through many operators before finding the first sub-problem. In infix notation we would have to consider paranthesis which would result in more complexity.

A sub-problem in postfix notation is found by first searching for an operator and then check if the two previous strings are numbers. If they are, we have a sub-problem that can be solved immediately. If one of the strings are another operator we have to wait until the sub-problem belongig to the operator is solved.

An example of postfix problem is shown in equation 3.1. The left side of the equal mark is postfix and the right side is the infix notation of the problem.

$$a\ b\ + = a + b \tag{3.1}$$

4 Auction Type

For the auction type we have chosen First-Price Sealed-Bid. This way each agent evaluates how well they are going to solve the problem and does not take into account other agents. Hence the system will perform more efficient.

5 Implementation

This section is going through the implementation of the TaskManager and the Solver agents. The DirectoryFacilitator is not included since it is a part of the framework.

5.1 TaskAdministrator

This agent takes a arithmetic problem as argument. Then it decomposes the problem into subproblems, and for each subproblem it asks the DF for which agents that can solve this problem. Further an auction is being held between the agents to solve the subproblem. The agent with the best bid is chosen to solve the problem. When an answer is returned the problem and subproblems are updated, and a new iteration of asking the DF and holding an auction is done. This is repeated until all of the subproblems are solved and we have one answer to our math problem.

Since this agent is only asking the DF for other agents, this agent does not need to register its services at the DF. The agent is using three different types of behaviours. A **CyclicBehaviour** is used to continously try to make new subproblems. This updates a list of sub-problems. When this list is not empty, the CyclicBehaviour will call an **OneShotBehaviour** to search for possible agents to solve the task. This behaviour is the one asking the DirectoryFacilitator for agents. The code in this behaviour will be executed just once for each time the behaviour is called. The list of agents are then sent to a **Behaviour** called RequestPreformer. This behavoiur consists of four steps. The first step is to send requests to the all the agents in the list. In step two it compare the tasktime receieved in responses from all the agents. In the third step it sends the problem to the agent that will solve the task fastest. And in the fourth step it will receive the solution and update the problem. This behaviour is a general Behaviour and will be executed over and over again. This is important becuse the agents will not respond immediatly and the TaskAgent therefor have to wait before proceeding to the next step. A switch is used to go from one step to another.

5.2 Solver agents

Theses are agents which solves math problems. First they need to register their services at the DF agent. This is done in the setup function. When this is done, two behaviours are added. The first behaviour handles the auction part, and the second behaviour handles the problem solving and informing the TaskAdministrator about the solution. The behaviours we used were CyclicBehaviour. This behaviour were chosen because the solver agents continuously need to listen for requests.

Since many of our solver agents do the same things except for the problem solving part, we made an abstract solver class which the other solver agents implements. This general class implements the setup function and the behaviours, and we've made an abstract problem solving function which each agent have to implement.

5.3 Communication protocol

This section is going in detail in how the agents are communicating. As mentioned before the TaskAdministrator queries the DF agent for spesific agents. So for the auction the TaskAdministrator sends a CFP message to the solver agents. Further the solver agents sends a proposal back to the TaskAdministrator. The TaskAdministrator then picks the best proposal or bid and sends a accept_proposal with the problem to the the agent with the best bid. Then the agent solves the problem and informs the TaskAdministrator about the solution. This protocol can be viewed in figure 2

6 Optimal Strategy

In our strategy the TaskAdministrator finds all the subproblems that can be solved directly and begin an auction for each one of those. This is an efficient solution because it follows that all the agents receiving a sub-problem will be working in parallel. But our solution is not optimal. When an agent is finished with a sub-problem, a new sub-problem might be revealed. In our solution however no sub-problem will be analysed before all the agents of the first round are finished. If you think of the arithmetic problem as a tree, our solution will solve sub-problems level by level. In an optimal solution sub-problems should be solved as they are reveald and the solver agents should be working in parallel.

7 Summary of our thoughts

It took a long time before we found a good way to synchronize the agents. Also we found it difficult to make the TaskAgent ask for bids and send the problem just once for each sub-problem. After some time we found a way to make sure that all tasks that were started never would be started again.

8 Example

An example problem that were solved with our solution is displayed in the following equation

$$\frac{(4-1)*3*(8-2)+6)}{2}.$$

For this example we have chosen one agent of type addition, one of multiplication, two of subtraction and two of division. How the agents operate are shown in figure **??**. We also made a solution where the agents are forced to work in serial and not in parallel. How these agents operate are shown in figure 2.

From the figures we see that for each sub problem the TaskAdministrator asks the DF agent for which agents that can solve the problem, and then the TaskAdministrator holds auctions and determines which agents to solve the tasks. For the first sub-problem a subtraction agent is used, then a multiplication agent, subtraction again, multiplication, addition and a division agent. For the parallel implementation we see that both of the subtraction problems are being done in parallel. The final solution is 30.

9 Further Extensions

For further extension one might consider other auction types, for instance English or Dutch auction type. The English auction type is the standard auction we all know where one starts small and have several bidding rounds. While the Dutch auction where one starts with a high asking price and lowers the price until one bids. For these auctions one can have strategy that also takes into account other agents. However for this problem the agents should focus more on their own status. Hence the First-Price Sealed-Bid auction is a more efficient option as mentioned before.

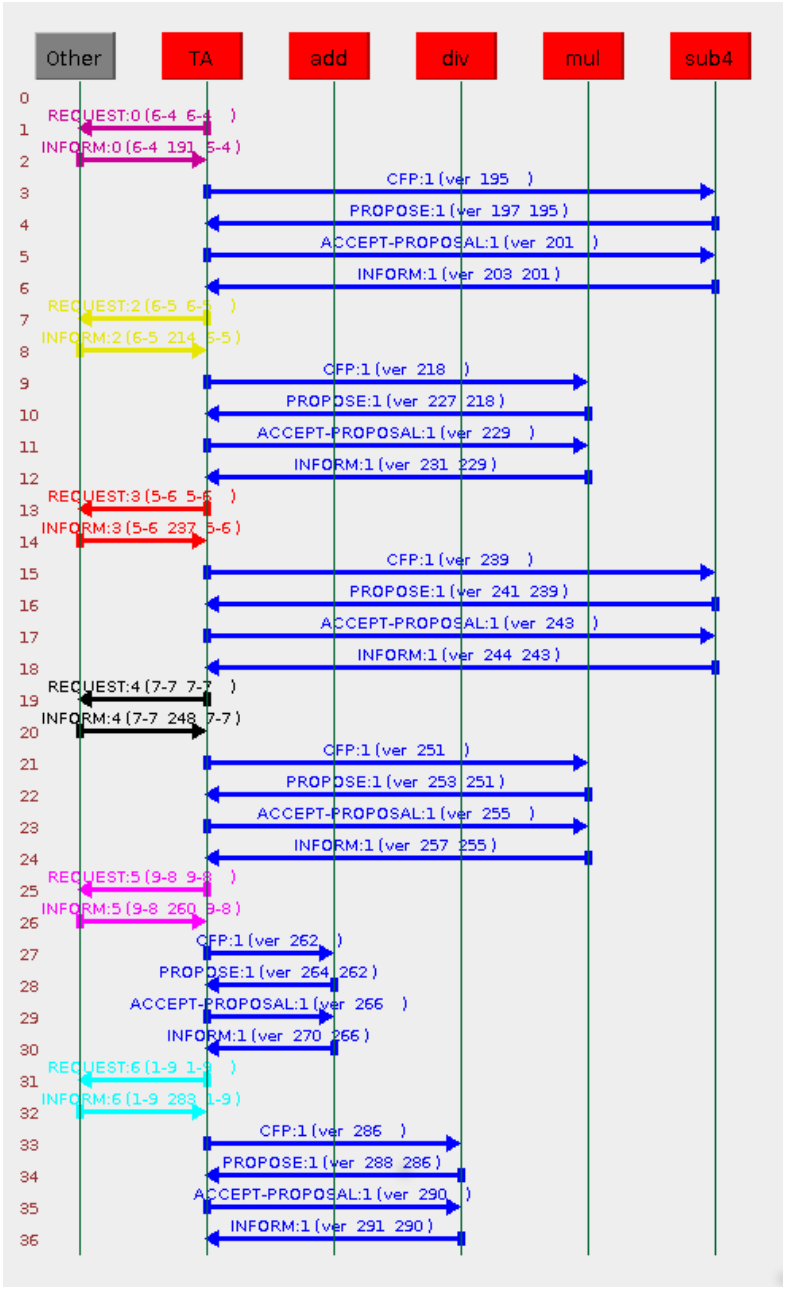

Figure 1: This is the diagram of the sniffer agent where the system is not in parallel. The diagram show how the different agents interact with each other.

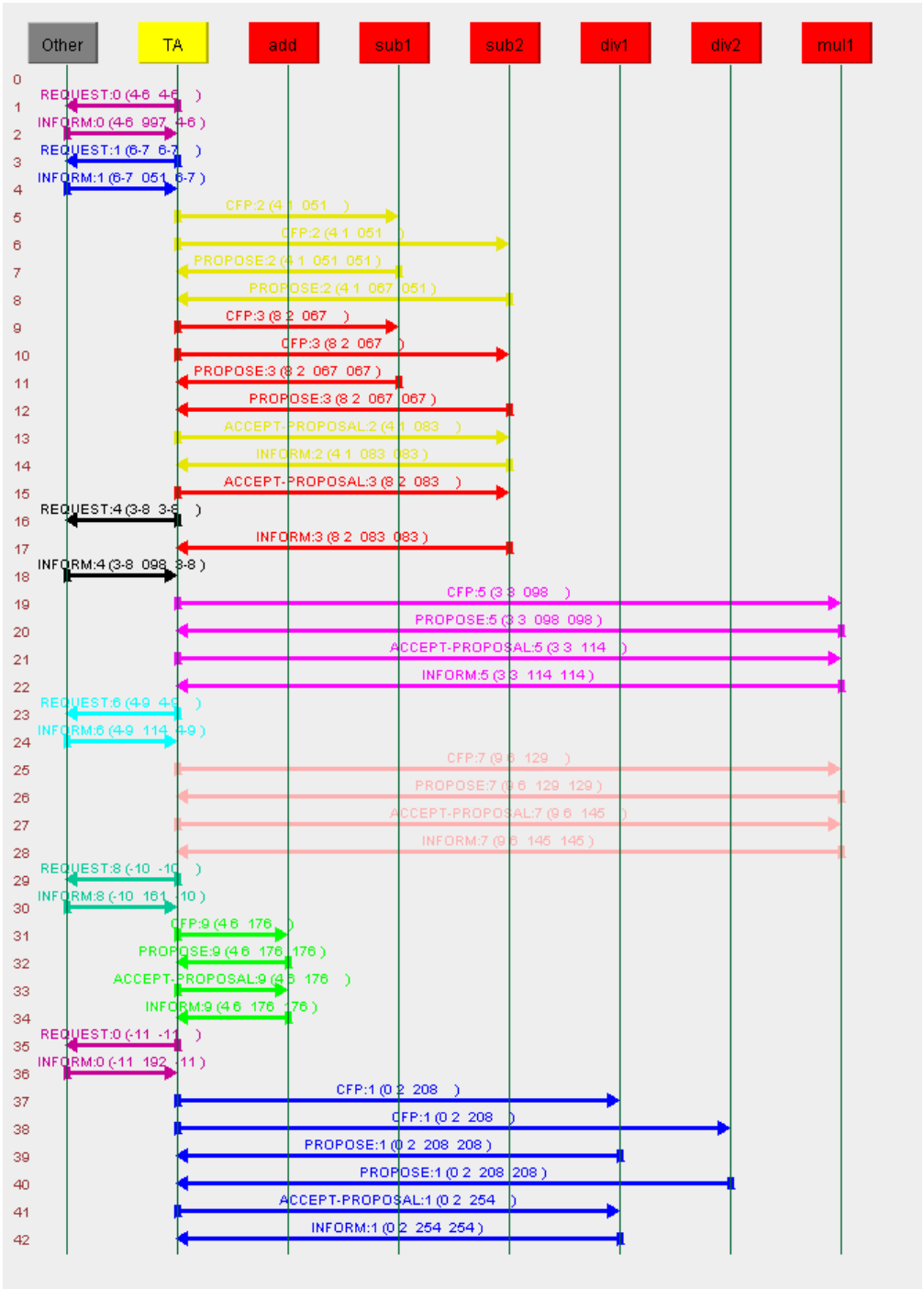

Figure 2: This is the diagram of the sniffer agent where the system is in parallel. The diagram show how the different agents interact with each other.