Exercise 3 - Programming in JADE

TDT4280
Distributed Artificial Intelligence and Intelligent Agents

February 19, 2012

Topics: task decomposition, distributed problem solving, auctions, the FIPA ACL

1 Introduction

The goal of this exercise is to implement a multi-agent system capable of distributed problem solving. The problems we will be solving are simple arithmetic expressions. You will implement several types of agents, and these needs to cooperate in order to solve the problem.

Through this exercise you will become familiar with agent communication, and in particular the FIPA ACL. You will also implement an auction, and get practical experience with problem decomposition and problem sharing.

You must document your work in a report.

2 JADE

You will base your implementation on JADE (Java Agent DEvelopement Environment). JADE is a framework for implementation of multi-agent systems, and can be downloaded from `http://jade.tilab.com`.

JADE Resources

See `http://jade.tilab.com/doc/` for some excellent resources for getting familiar with JADE, including the JADE API. It is recommended that you read through the *JADE Programming Tutorial* before starting to code in order to get an understanding of the framework.

The *JADE Programmer's Guide* and the *JADE Administration Tutorial* also provide a lot of useful information. Finally, JADE comes with a set of helpful and illustrating examples.

Getting Started With JADE

The most important classes that are provided by JADE are `jade.core.Agent` and `jade.core.behaviours.Behaviour`, and you will need to extend these with your code. To get started using JADE, follow these steps:

- Download JADE from `http://jade.tilab.com`.
- Include the `jade.jar` library in your classpath.
- JADE is run from the main class `jade.Boot`.
- Run JADE with the the `-gui` option to use the graphical tools.
- Implement your agents by extending `jade.core.Agent`.

3 Agents

You will need to implement different types of agents, specifically an task administrator and a set of math solvers.

The **TaskAdministrator (TA)** agent will be responsible for solving the problem. This agent need to decompose the problem and assign experts to solve each subproblem.

The TA must be able to receive tasks, and return their results.

Solver agents must be implemented for each arithmetic operator. The following four are required, but you are free to implement additional experts if you like.

AdditionSolver (AS), **SubtractionSolver (SS)**, **DivisionSolver (DS)**, **MultiplicationSolver (MS)**.

You may choose to implement one agent type per operator, or let agents provide several services. No agent may, however, be able to solve more than 2 types of operations.

You will also need the **DirectoryFacilitator (DF)** agent, which is provided by the framework. This agent provides a yellow pages service where the agents register their services. It can be accessed through the `jade.domain.DFService` class.

The solver agents must register their services with the DF, and the TA must ask the DF for agents providing the appropriate service when holding auctions.

Once the TA have divided the expression into subproblems, it need to assign these to the solvers. To decide the assignments, the TA should hold auctions where the solvers bid on each task.

A few suggestions

JADE comes with several extensive behaviours that might help you in your implementation. You may use these if you feel they are helpful, but they may also steepen the the learning curve for JADE. As an alternative you may base your solution on simpler behaviours such as `CyclicBehaviour`, `OneShotBehaviour` and `WakerBehaviour`. These will require you to write a little more code, but are very simple and intuitive to use. The `WakerBehaviour` is especially useful for waiting for the replay to auctions and to simulate solving of the subtasks.

You are also encouraged to use many small behaviours rather than to implement everything in one or a few large ones. Note that it is possible to add new behaviours from within executed behaviours.

4 Tasks and Task Decomposition

The tasks are arithmetic expressions with addition, subtraction, division and multiplication.

You are free to choose whether to use the standard infix notation (e.g. a+b), prefix notation (e.g. +ab) postfix notation (ab+) for the expressions. Your choice should be documented in the report.

You will need to split the expressions into the individual tasks that can to be solved. Then each part is solved by different agents, and the result combined to solve the problem.

The TA must be able to receive `query-ref` messages with the tasks as content. Once such a message is received, it should start working on it, and report back the answer as soon as possible.

A Simple Example

The expression [`+ * 5 2 - 7 2`] (in prefix notation) could be decomposed as shown in figure 1. We see that both the multiplication and the subtraction tasks are ready to be solved. These can be assigned to different solvers. Once the result of these have been returned, the TA can have someone solve the addition task.

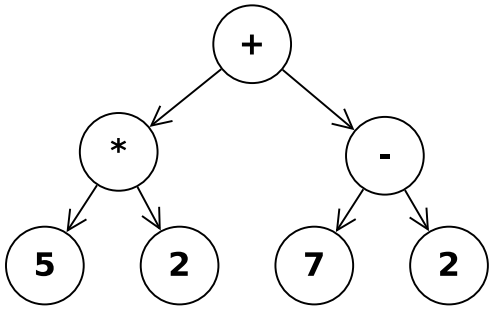

Figure 1: A task decomposition of the expression [`+ * 5 2 - 7 2`].

Implementation Hints

Although you may implement the task decomposition any way you see fit, it may prove easier to parse expressions in prefix or postfix notation rather than the more common infix. Parsing the experssions into simple *task trees* like in the example above is also advisable, as it will enable your TA to easily identify any subtasks that are ready for delegation. Using trees will also make it trivial to integrate the partial answers into the problem solution.

5 Auction

You will use auctions to determine the best solver for each subproblem.

A call for proposals (CFP) must be multicast to all agent capable of solving the task. They must then be allowed to bid, and a winner selected to perform the task.

Which auction type to use is up to you, and you may select any you deem appropriate as long as you can explain why in the report. The simplest option is, perhaps, the First-Price Sealed-Bid (FPSB) auction, and it is described below.

Read more about single item auctions in Wooldrige chapter 14.2.

First-Price Sealed-Bid Auction

Figure 2 show the different interactions performed by agents during a FPSB auction with two solvers bidding on an addition problem.

The TA sends out a description of the task to be done. The task-description is a call for proposals, sent to all problem solving agents capable of performing the task.

This description is used by the problem solvers to assess how much time they will need to perform the task. If the agent has capacity to solve the task, he sends his bid to the TA.

After the TA have received all the bids, or a deadline is met, the TA selects the best bidder and assign him the task. The rest of the bids are rejected.

When the problem solver finishes a task, it informs the TA about the result.

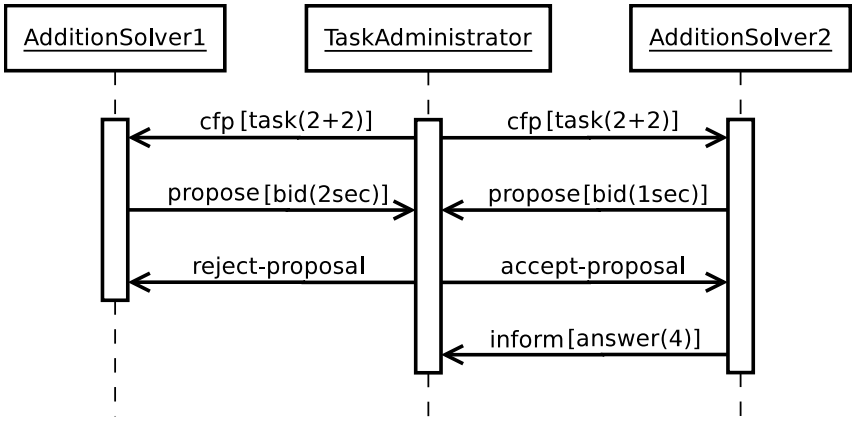

Figure 2: The interactions involved in the First-Price Sealed-Bid auction. AdditionSolver2 provides the best bid, is awarded the task, and solves the problem before sending the result back to the TA.

Bidding

You must implement bidding strategies for the solver agents. The strategies should take into account how long the agent estimate this particular task will take to solve, and how much other work the agent needs to finish before he can start working on the task.

This means that the agents will need some way to estimate the magnitude of the task. You could hardwire this into the agent, or create a heuristic to base the estimate on.

The agents also need a memory of the tasks it has been assigned. If the agent have previously assigned tasks it needs to finish first, this should be included in the bid.

6 Result Sharing and Integration

When assigned a task, the solver agents should work on this until solved, and inform TA about the result once finished. Since the tasks in this domain are rather trivial, the solvers should simulate working on the tasks. If a solver won an auction with a bid of one second, the agents should wait one second before returning the answer to the TA.

When the solver agents informes TA about the subproblem solutions, these must be integrated to the final answer to the problem. How to do this will depend on how you

break down the problem to begin with.

7 Implementation requirements

When implementing the system, you will need to make a lot of decisions. You are free to make these decisions as you like, as long as as you document your decisions properly. Your system is not required to be optimal, but must meet the requirements outlined below.

- TA must decompose the problem into subproblems.
- The subproblems must be solved by the appropriate solver agents, i.e. the TA cannot do any arithmetic operations.
- TA must auction the subproblems, and assign them to the winners.
- The TA must be able to receive `query-ref` messages with tasks, and `inform` the sender of the result.

You are, of course, encouraged to do more than the bare minimum. See Section 10 below for some ideas.

8 The Optimal Strategy

As stated above, your implementation is not required to be optimal. You should therefore consider in the report how your solution could be improved, and how an optimal strategy would look. Of course, if you feel confident that your implementation is optimal, you must explain why.

On a conceptual level, explain the optimal strategy for the TA in your report. Describe how the task should be decomposed, and how the subproblems should be assigned to solvers in order to make the system optimal in regard to time needed to solve the task. Consider such factors as how to solve as many subproblems as possible in parallel, and how the TA should choose which solvers to involve in auctions. Remember to explain your assumptions.

The implemented task is very simple. It may help in the discussion if you imagine that more complex problems are being solved by the same approach.

9 Deliveries

The delivery of this exercise consist of your code and a report. Deliver the report as .pdf (no word-files) together with the code to it's learning.

Code

Deliver a zip-file named YourLastName(s).zip with all the .java and .class for this exercise. The package name you use for your agents must be the same as the name of this zip file!

Remember to make your code easy to read, by formatting it nicely and commenting where appropriate. If your code cannot be understood, you will not be given the benefit of the doubt.

Report

Write a short report (about 4 to 6 pages). The report will need to

- Describe the implementation of your system.
- Explain how you solved the task decomposition problem.
- Explain which auction type you chose, and why. Advantages, disadvantages?
- Describe the optimal strategy for the Task Administrator (see Section 8 above).
- Summarize your final thoughts, problems encountered, lessons learned, etc.
- Illustrate how your system decompose and solve tasks with an example.
  - Describe what is done when, and which messages are sent to whom.
  - Include a sequence diagram. (This can be automatically generated by JADE's Sniffer agent.) The diagram must include all relevant agents: at the very least the task administrator and all the solver agents.
- List the parameters needed to start JADE with the GUI and enough instances of your agents.

Illustrate your report with figures where appropriate.

10 Further Extensions

If you like, you are welcome to try to extend the system beyond what is described above. This is not required, but could be both instructive and fun.

Some suggestions for extensions that could be made:

- Add more complex operators to the task expressions and agents to solve them.
- Enable the TA to auction complex subproblems to other TAs for an even more distributed problem solving.
- Experiment with different types of auctions, such as Vickrey, English or Dutch.
- Add a time cost for holding auctions with many participants, and experiment with the trade-off between getting the best possible solver and having smaller auctions.