Proceedings of the International Multiconference on Computer Science and Information Technology pp. 235–241
ISSN 1896-7094

JABAT—An Implementation of the A-Team Concept

Dariusz Barbucha, Ireneusz Czarnowski, Piotr Jędrzejowicz, Ewa Ratajczak-Ropel and Izabela Wierzbowska

Department of Information Systems
Gdynia Maritime University
Morska 83, 81-225 Gdynia, Poland
{barbucha, irek, pj, ewra, iza}@am.gdynia.pl

Abstract. The paper proposes a JADE-based A-Team environment (in short: JABAT) as a middleware supporting the construction of dedicated A-Team architectures that can be used for solving a variety of computationally hard optimisation problems. The paper includes a general overview of the functionality and structure of the proposed environment and a description of how this functionality can be extended to solving new problems.

1 Introduction

Formerly, research into systems composed of multiple agents was carried out under the banner of Distributed Artificial Intelligence, which has historically been divided into two main fields: Distributed Problem Solving and Multi-Agent Systems. Recently, the term "multi-agent systems" has a more general meaning, and is rather used to refer to all types of systems composed of multiple autonomous components [1].

The field of autonomous agents and multi-agent systems is a rapidly expanding area of research and development. It is based on many ideas originating from such areas as artificial intelligence, distributed computing, object-oriented systems and software engineering.

Recently, a number of significant advances have been made in both the design and implementation of autonomous agents. A number of applications of agent technology is growing systematically. Nowadays agent technology is used to solve real-world problems in a range of industrial and commercial applications. Also a number of agent-based approaches have been proposed to solve different types of optimisation problems [2, 4, 5].

One of the successful approaches to agent-based optimization is the concept of an asynchronous team (A-Team), originally introduced by Talukdar [9]. An A-Team is a collection of software agents that cooperate to solve a problem by dynamically evolving a population of solutions. It is especially dedicated for solving computationally difficult problems. An A-Team usually uses combination of features taken from a variety of natural and synthetic systems, including for example insect societies [6], genetic algorithms [7] and tabu search [8].

An A-Team is a cyclic network of autonomous agents and shared, common memories. Each agent contains some problem solving skills and each memory contains a population of temporary solutions to the problem to be solved. All the agents can work asynchronously and in parallel. During their works agents cooperate by selecting and modifying these solutions.

This paper proposes a JADE-based A-Team environment (in short: JABAT) as a middleware supporting the construction of the dedicated A-Team architectures used for solving a variety of computationally hard optimization problems. JADE is an enabling technology for the development and run-time execution of peer-to-peer applications which are based on the agents paradigm and which can seamlessly work and interoperate both in wired and wireless environment [10]. From the functional point of view, JADE provides the basic services necessary to distributed peer-to peer applications in the fixed and mobile environment. JADE allows each agent to dynamically discover other agents and to communicate with them according to the peer-to-peer paradigm.

The paper contains overview of JABAT functionality and a description of how this functionality can be extended to solving new problems.

2 Overview of JABAT

JABAT is a middleware allowing to design and implement an A-Team architecture for solving combinatorial optimization problems. The problem-solving paradigm on which the proposed system is based can be best defined as the population based approach.

The main features of JABAT are the following (Fig. 1):

- The system can in parallel solve instances of several different problems.
- A user, having a list of all algorithms implemented for given problem may choose how many and which of them should be used.
- The optimization process can be performed on many computers. The user can easily adjoin or delete a computer from the system. In both cases JABAT will adapt to the changes, commanding the agents working within the system to migrate.
- The system is fed in the batch mode—consecutive problems may be stored and solved later, when the system assesses that there is enough resources to undertake a new search.

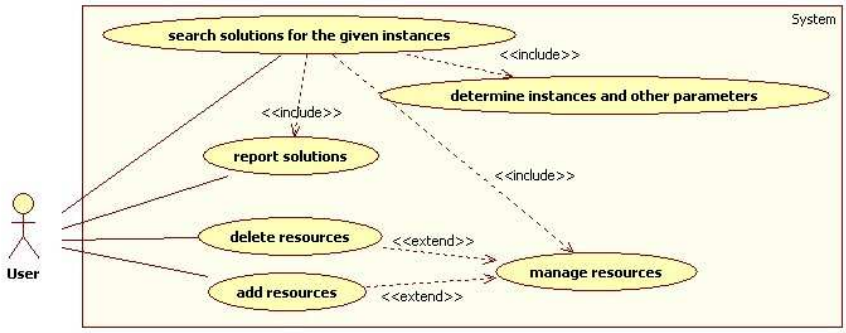

Fig. 1. Working with JABAT

The JABAT produces solutions to combinatorial optimization problems using a set of optimising agents, each representing an improvement algorithm. To escape getting trapped into a local optimum an initial population of solutions called individuals is generated or constructed. Individuals forming an initial population are, at the following computation stages, improved by independently acting agents, thus increasing chances for reaching a global optimum.

2.1 The process of searching for the best solution

Main functionality of the proposed environment includes organizing and conducting the process of search. It involves a sequence of the following steps:

- Generating an initial population of solutions.
- Applying solution improvement algorithms to individuals drawn from the common memory and storing them back after attempted improvement, with the use of a user chosen replacement strategy.
- Continuing reading-improving-replacing cycle until a stopping criterion is met.

This functionality is realized by mainly two types of agents: *OptiAgents* and *SolutionManagers*. Each *OptiAgent* working within JABAT represents a single optimising algorithm. Each *SolutionManager* is responsible for finding the best solution for a single instance of a problem and maintains one population of solutions of this problem. The agents of both types act in parallel and communicate with each other exchanging solutions that are either to be improved (when solutions are sent to *OptiAgent*) or stored back (when solutions are sent to *SolutionManager*).

In the process of solving tasks *OptiAgents* and *SolutionsManagers* exchange a number of messages. An *OptiAgent* can communicate with all *SolutionManagers* that work on instances of the same problem, so the messages containg the details describing the task and solutions (an agent may need more than one solution to conduct the optimization process).

Apart from *OptiAgents* and *SolutionManagers* there are also other agents working within the system—responsible for initialising, organising the process of migrations, storing the results etc. (Fig. 2)

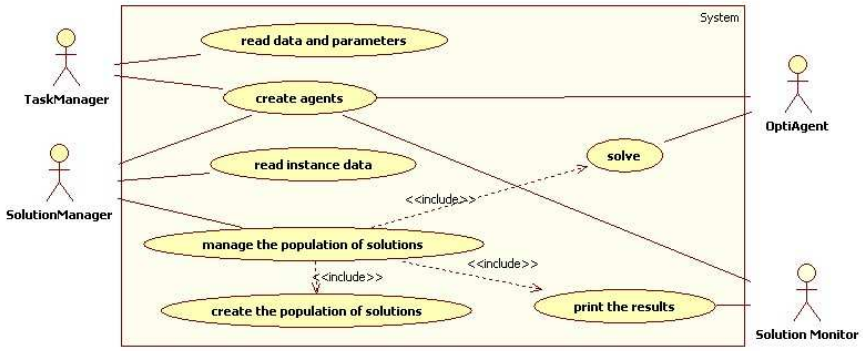

Fig. 2. Overwiew of JABAT architecture

2.2 Managing parallel searches

JABAT can deal with several searches conducted in parallel, and the process is administered mainly by *TaskManager*. *TaskManager* is run as the first. It is responsible for creating other agents and reading all needed parameters from so called initial directory and from configuration file where global parameters are stored.

At first *TaskManager* creates a single *PlatformManager*. The *PlatformManager* manages optimization agents and system platforms. It can move the optimization agents among platforms and create their copies to improve the efficiency. The *PlatformManager* reports its successful running sending message.

The parameters for solving instances of different problems are stored in the XML files in the initial directory. These files are read by *TaskManager* in order of loading. An example of data stored in XML files with the set of instance descriptions for resource contrained project scheduling problem (RCPSP) is presented in the Example 1. Every file contains information on the problem name, instance data and managing and optimization agents which should be run. One input file may contain instances (in several file containers), also there may be more *OptiAgents* of each type.

Example 1. Example of XML data for solving an instance of RCPSP

```
<?xml version="1.0" standalone="no" ?>
<!DOCTYPE boundle SYSTEM "boundle.dtd">
<boundle>
<problemname>RCPSP</problemname>
<solutionmanager>
<dataclass>rcpsp.Data</dataclass>
<strategy>rcpsp.Strategy1</strategy>
</solutionmanager>
<datapath>
<urldir>\RCPSP\data\mm_j10\</urldir>
   <file>j10*.mm</file>
</datapath>
<optiagent>rcpsp.LocalSearch</optiagent>
<optiagent>rcpsp.TabuSearch</optiagent>
<optiagent>rcpsp.CrossingHeuristic</optiagent>
</boundle>
```

The parameters from each XML file are used by *TaskManager* to create a number of agents: *SolutionMonitors*, *SolutionManagers* and optimization agents. For each problem instance *TaskManager* creates one *SolutionMonitor*, which monitors solutions of this instance. The *SolutionMonitor* reports its successful running by sending its identifier. This identifier is used to create other agents: one *SolutionManager* and all optimization agents required by the parameters file. All these agents report their successful running by sending messages.

The Example 2 presents initial messages sent and received by *TaskManager* while solving the istance of RCPSP problem with three optimization agents, as described in the XML file presented in the Example 1.

Example 2. Messages sent and received by *TaskManager* while solving RCPSP problem instance with three optimization agents.

```
Open rcpsp_j1002_06mm.xml
TM: Running SMo_rcpsp_j1002_06mm
TM get msg: Start SMo_rcpsp_j1002_06mm
Data found in \RCPSP\dane\mm_j10\j1002_06.mm
TM: Running SMa__SMo_rcpsp_j1002_06mm__RCPSP_j10j1002_06mm
SMo get msg: Start SMa
TM get msg: Start SMa__SMo_rcpsp_j1002_06mm__RCPSP_j10j1002_06mm
TM: Running OA__SMo_rcpsp_j1002_06mm__rcpsp.LocalSearch__1
TM: Running OA__SMo_rcpsp_j1002_06mm__rcpsp.CrossingHeuristic__2
[init c=19,c=19,c=19,c=20,c=20,c=20,c=20,c=20,c=20,c=21,c=21,
c=21,c=21,c=21,c=21,c=21,c=29,c=29,c=29,c=29,c=29,c=30,c=30,c=32,
c=33,c=33,c=34,c=34,c=34,c=35,c=35,c=36,c=36,c=36,c=37,c=37,c=38,
c=38,c=38,c=39,c=39, c=40, c=40,c=43,c=43,c=43,c=44,c=44,c=48,
c=48,c=58]
TM: Running OA__SMo_rcpsp_j1002_06mm__rcpsp.TabuSearch__3
TM get msg: Start OA__SMo_rcpsp_j1002_06mm__rcpsp.LocalSearch__1
TM get msg:
Start OA__SMo_rcpsp_j1002_06mm__rcpsp.CrossingHeuristic__2
TM get msg: Start OA__SMo_rcpsp_j1002_06mm__rcpsp.TabuSearch__3
SMo get msg: Stop SMa 1 1
TM get msg: Stop SMa__SMo_rcpsp_j1002_06mm__RCPSP_j10j1002_06mm
No data in initial directory \projava\jabat_input.
TM get msg: Stop SMo_rcpsp_j1002_06mm
TM: Stopping OA__SMo_rcpsp_j1002_06mm__rcpsp.LocalSearch__1
TM: Stopping OA__SMo_rcpsp_j1002_06mm__rcpsp.CrossingHeuristic__2
TM: Stopping OA__SMo_rcpsp_j1002_06mm__rcpsp.TabuSearch__3
```

The further behaviour of *TaskManager* depends on whether there are other instances waiting and whether the maximum number of opimization agents has been achived. The number of agent running at the same time depends on the number of accessible resources. If it is possible to create new agents, *TaskManager* initialises solving the next instance from the same file if there is one, or checks the initial directory for next files. The checking is done periodically. If starting new agents is not possible, *TaskManager* waits for a message from any of running *SolutionManagers* indicating that the manager has finished with his task and may be destroyed together with other agents that has been created for solving this particular task.

TaskManager may be created and destroyed only by a user.

To create a *SolutionManager* two pieces of information are needed:

- the name of the class that represents task,
- the name of the chosen replacement strategy.

Replacement strategies in JABAT define:

- how the initial population is created (for example how many solutions it consists of),
- how solutions are chosen to be sent to optimising agents,
- how solutions that has been received from optimising agents are merged with the population,
- when the process of searching stops.

There are several different predefined strategies in the system to choose from. A simple strategy can, for example, draw a random solution and, when an improved solution is received, replace the worst solution in the population with it.

During the whole process of solving a problem, an agent called *SolutionMonitor* gathers data to create a report of obtained solutions. The data are sent periodically by *SolutionManager* and stored in a text file. The information include: name of the *SolutionManager* (which contains the name of the problem), the best solution obtained so far (that of the maximum or minimum value of fitness), average value of fitness among solutions from current population, the actual time of running and the time in which the best solution was last reached. The report may later be analysed, for example it may be read into a spreadsheet (i.e. MS Excel) and converted into a summary report with the use of a pivot table tool.

2.3 Managing more platforms

JABAT, using a functionality of JADE, makes it possible for optimization agents to migrate to containers on other computers that have joined the main platform. In this way the system can work with better efficiency.

Managing the process of migration is the responsibility of *PlatformManager*, created by *TaskManager* when the system starts working. *PlatformManager* does the following:

- manages the number of containers available,
- manages the number of working *OptiAgents*, also destroys agents indicated by *TaskManager*,
- registers in which containers *OptiAgents* are working, sends requests to change location to the optimising agents.

The Fig. 3 presents a view on the main container with all the main agents working and a number of containers where some optimization agents migrated.

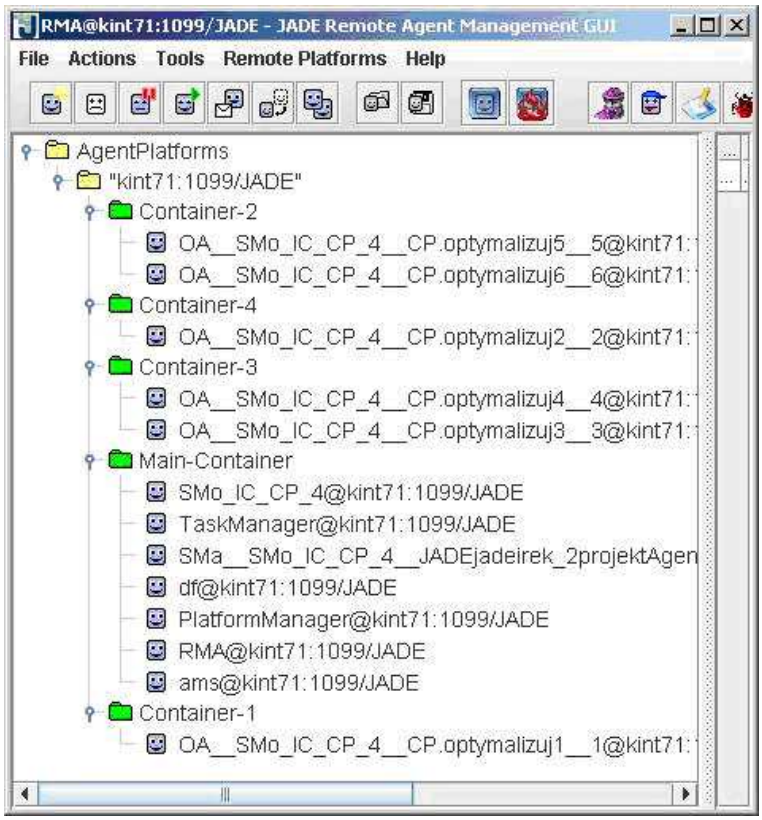

Fig. 3. Agents working in the main other containers

3 Implementing new problems in JABAT

The system was designed so that the functionality described in Section 2 could be easily extended to solving new problems or solving them differently. The main idea is to reduce the amount of work of a programmer who wants to use the system with new problems, new ways of representing tasks or solutions, new optimising algorithms or finally new replacement strategies. JABAT makes it possible to focus only on defining these new elements (Fig. 4) and assure that the process of communication and improvement procedures will still work.

All elements of JABAT work on classes like *OptiAgent*, *Strategy*, *Task* and *Solution*, predefined at a rather general level and responsible for optimising, handling the population of solutions and defining individual task and solution respectively.

In order to solve a problem not yet implemented in JABAT classes inheriting from *Task* and *Solution* should be defined. Apart from defining constructors and properties specific to the problem, several inherited methods should be overridden. For a task these are for example function *createSolution*() creating an initial solution (drawn at random or empty), function *readDataFile*() reading the instance data from a file and function *ontology*() returning the JADE ontology of the class. An ontology defines how a class may be transformed into a text message and how the text message is used to construct the class. Solution should also override some methods, for example a method that counts the fitness of the solution, a method that compares the solution with other solutions and again a method returning the ontology of the class.

In addition to *Task* and *Solution* at least one optimising algorithm should be implemented. It is done by simply creating a class inheriting from *OptiAgent* with overriden method *Optimise*().

It is also possible (but not neccesary) to easily create own replacement strategies, which requires defining a class inheriting from one of strategies that are already available in the system and to adjust it by overriding chosen methods. Strategies work on population of individuals of the type *Solution* and thus are independent from the problem definition or implementation details of particular solution and can be used with solutions of all types.

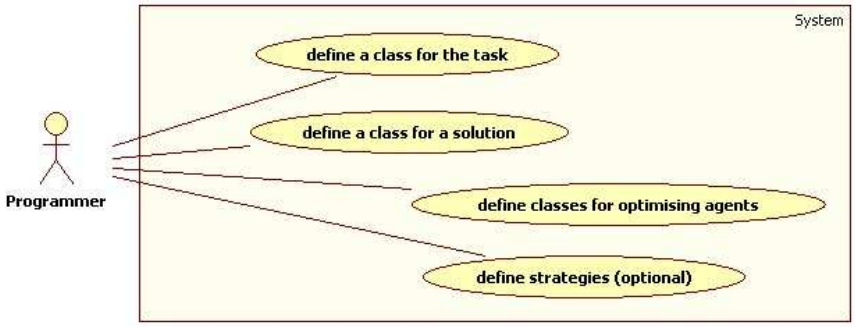

Fig. 4. Preparing JABAT for solving new problems.

So far JABAT has been succesfully tested in solving four different problems: resource constrained project scheduling problem (RCPSP), clustering problem (CP), euclidean planar traveling salesman problem (TSP) and vehicle routing problem (VRP) [3]. For each of the problems several optimising algorithms have been implemented: local search algoritm, algorithm based on the simple evolutionary crossover operator and tabu search algorithm for RCPSP, random local search, hill-climbing local search and tabu search for CP, implementation of the Lin-Kerninghan algorithm and a crossing algorithm for TSP and 2-optimum algorithm operating on a single route or different routes, λ interchange local optimisation method and two more algoritms interchanging sequences of visits and random elements on routes for VRP. For these experiments also several different strategies have been implemented.

4 Conclusions

The proposed JADE-based A-Team environment is a "middleware plus", supporting development of A-Team systems. Its main advantages that have been inherited from JADE, include:

- Ability to simplify the development of the distributed A-Teams composed of autonomous entities that need to communicate and collaborate in order to achieve the working of the entire system.
- Interoperability JADE-A-Team agents can cooperate with other agents provided that they comply with the FIPA standard.
- Uniformity and portability JADE-A-Team provides a homogeneous set of APIs that are independent from the underlying network and Java version.

Moreover, a software framework that hides all complexity of the distributed architecture plus a set of predefined objects are made available to users, who can focus just on the logic of the A-Team application and effectiveness of optimization algorithms rather than on middleware issues. It is expected

that the approach will result in achieving scalable, flexible, efficient, robust, adaptive and stable A-Team architectures at a low development costs. Parallel actions of independent agents working in the distributed environment is the main factor behind efficiency of the A-Team based architectures.

During the test and verification stages JADE-A-Team has been used to implement several A-Team architectures dealing with well known combinatorial optimization problems including traveling salesman, project scheduling under resource constraints, vehicle routing problem and clustering problem. Functionality, ease of use and scalability of the approach have been confirmed. Further research will concentrate on providing a friendly human computer interface and developing a set of classes that can even further support the construction of advanced A-Teams architectures featuring population based algorithms.

Acknowledgement: The research was supported by the KBN, grant no. 3T11C05928

References

1. Jennings N. R., Sycara K., Wooldridge M.: *A Roadmap of Agent Research and Development,* Autonomous Agents and Multi-Agent Systems **1**, (1998) 7–38.
2. Aydin M. E., Fogarty T. C.: *Teams of autonomous agents for job-shop scheduling problems: An Experimental Study,* Journal of Intelligent Manufacturing, **15**(4), (2004) 455–462.
3. Barbucha D., Czarnowski I., Jdrzejowicz P., Ratajczak E., Wierzbowska I.: *JADE-Based A-Team as a Tool for Implementing Population-Based Algorithms,* Proceedings of ISDA 2006—6th IEEE International Conference on Intelligent System Design and Applications (in print).
4. Marinescu D. C., Boloni L.: *A component-based architecture for problem solving environments,* Mathematics and Computers in Simulation, **54**, (2000) 279–293.
5. Parunak H. V. D.: *Agents in Overalls: Experiences and Issues in the Development and Deployment of Industrial Agent-Based Systems,* International Journal of Cooperative Information Systems, **9**(3), (2000) 209–228
6. Oster G. F., Wilson E. O.: *Caste and Ecology in the Social Insects,* Princeton University Press, Princeton, NJ8 (1978)
7. Davis L. (ed): *Handbook of Genetic Algorithms,* Van Nostrand Reinhold (1991).
8. Glover F.: *Tabu SearchParts I and II,* ORSA Journal of Computing, Vol. **1** No. 3, Summer 1989 and Vol. **2**, No. 1, Winter 1990.
9. Talukdar S., Baerentzen L., Gove A., de Souza P.: *Asynchronous Teams: Co-operation Schemes for Autonomous,* Computer-Based Agents, Technical Report EDRC 18-59-96, Carnegie Mellon University, Pittsburgh (1996).
10. Bellifemine F., Caire G., Poggi A., Rimassa G.: *JADE. A White Paper,* Exp, **3**(3) (2003) 6–20.