BathTUB team description – multi-agent programming contest 2016

Christian Bahrdt, Oguz Serbetci and Axel Heßler*

Technische Universität Berlin/DAI-Labor,
Fakultät IV für Elektrotechnik und Informatik,
Sekretariat TEL 14, Ernst-Reuter-Platz 7,
D-10587 Berlin, Germany
Email: axel.hessler@dai-labor.de
*Corresponding author

Abstract: Team BathTUB reports on their approach to the complex scenario of multi-modal logistics in the agent programming contest 2016. The solution has been created by beginners in multi-agent programming and using the JIAC V agent framework.

Keywords: multi-agent programming; multimodal logistics; JIAC.

Reference to this paper should be made as follows: Bahrdt, C., Serbetci, O. and Heßler, A. (2018) 'BathTUB team description – multi-agent programming contest 2016', *Int. J. Agent-Oriented Software Engineering*, Vol. 6, No. 1, pp.118–127.

Biographical notes: Christian Bahrdt is a member of the Bachelor degree program in Computer Science at TU Berlin.

Oguz Serbetci is a member of the Bachelor degree program in Computer Science at TU Berlin.

Axel Heßler studied Computer Science at Technische Universität Berlin. He works as a researcher at DAI-Labor of TU Berlin with focus on agent technologies. He has worked on a number of projects within the laboratory, including the development of the JIAC V agent frameworks. He received his doctor degree for work on agent-oriented methodologies in the context of real-world projects and has been an active participant in the multi-agent programming contest since 2007.

1 Introduction

Team *BathTUB* is the contribution of six students of Technische Universität Berlin in the Computer Science Bachelor's program. They were attending the course 'Multi-Agent Contest' that is provided in the form of a software project. Besides learning the principles of multi-agent programming, the use of the JIAC V agent framework (Hirsch et al., 2009) and understanding and developing a solution for the Multi-Agent Programming Contest [MAPC (https://multiagentcontest.org/)], most of the students make their first steps in managing and cooperating in a larger software engineering project. The course is supervised by Dr. Axel Heßler, who is researching and teaching agent-oriented principles for over a decade and is also co-developer of the agent framework JIAC V and successful participant in former MAPCs. Around 1,000 man hours have been invested in the project during the semester and another week has been spent just before the contest.

Participating in the contest has long tradition at Technische Universität Berlin (see e.g., in Heßler et al., 2007, 2010, 2013]), starting in 2007, when the former team has won the gold digging challenge using predecessor framework of JIAC V, the JIAC IV agent framework (Fricke et al., 2001).

2 System analysis and design

The methodology, which is used in a trimmed-down version, borrows from the JIAC methodology (Heßler, 2013), and can be described as bottom-up and agile methodology: The whole MAPC scenario is very complex. In order to understand the challenge and develop the solution we start with domain analysis, which is to build a first ontology or world model reflecting the four different agent roles (car, drone, motorcycle, truck) with their different capabilities and properties (maxSpeed, batteryLoad, carryingCapacity)..

The next step was to define the notion of items and facilities with their dependencies. Items must be bought or crafted to get them and can be stored, delivered or dumped. Dependencies of items span an item graph (see Figure 1). We create the graph during initialisation phase of the simulation and it is used during job planning.

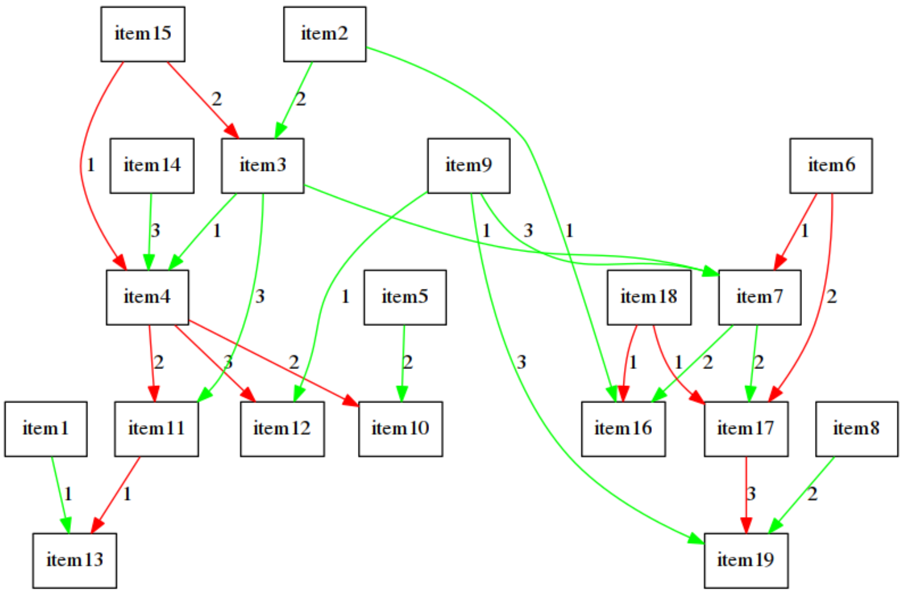
Facilities are points-of-interests on the map that sell, store or support crafting of items. We model the types of facilities, their price and current inventory. All facilities and their actual state are gathered in the yellow pages.

Finally in ontology modelling, we model jobs (those activities that make money) and other activities so that we can rank and sequence activities for scheduling and execution.

The main strategy of the team is going for fast money. We focus on doing simple jobs as fast as possible by means of decent job planning, complexity reduction and enemy surveillance. What we mean by this is the following: the JobPlanner ranks jobs by complexity (see item graph above) and than disassembles the job according to the number of free agents and assigns tasks to them. We always try to guess what the opponent is doing by observing the inventories of the facilities.

Our job planning is a centralised approach to team planning. In principle, it would also be possible to distribute the agents on several machines. However, except for the initialisation phase, the system performs fast enough meaning that the agents algorithms (message parsing, communication between team agents, decision cycle, team coordination, etc.) work faster than the request loop so there is no need to distribute at the moment. The current implementation of the contest team also does not allow for this kind of distributed agents because the JobPlaner has been implemented as Java Singleton (see Section 4 for more details).

Figure 1 The item dependency graph that is created during simulation initialisation phase (see online version for colours)



Notes: Green arrows mean that items are used as tools. Red lines mean that items are consumed as ingredients during crafting.

Altogether we have a hybrid approach, where each agent maintains its own world model and decides on this basis: this is completely decentralised, whereas job coordination is centralised.

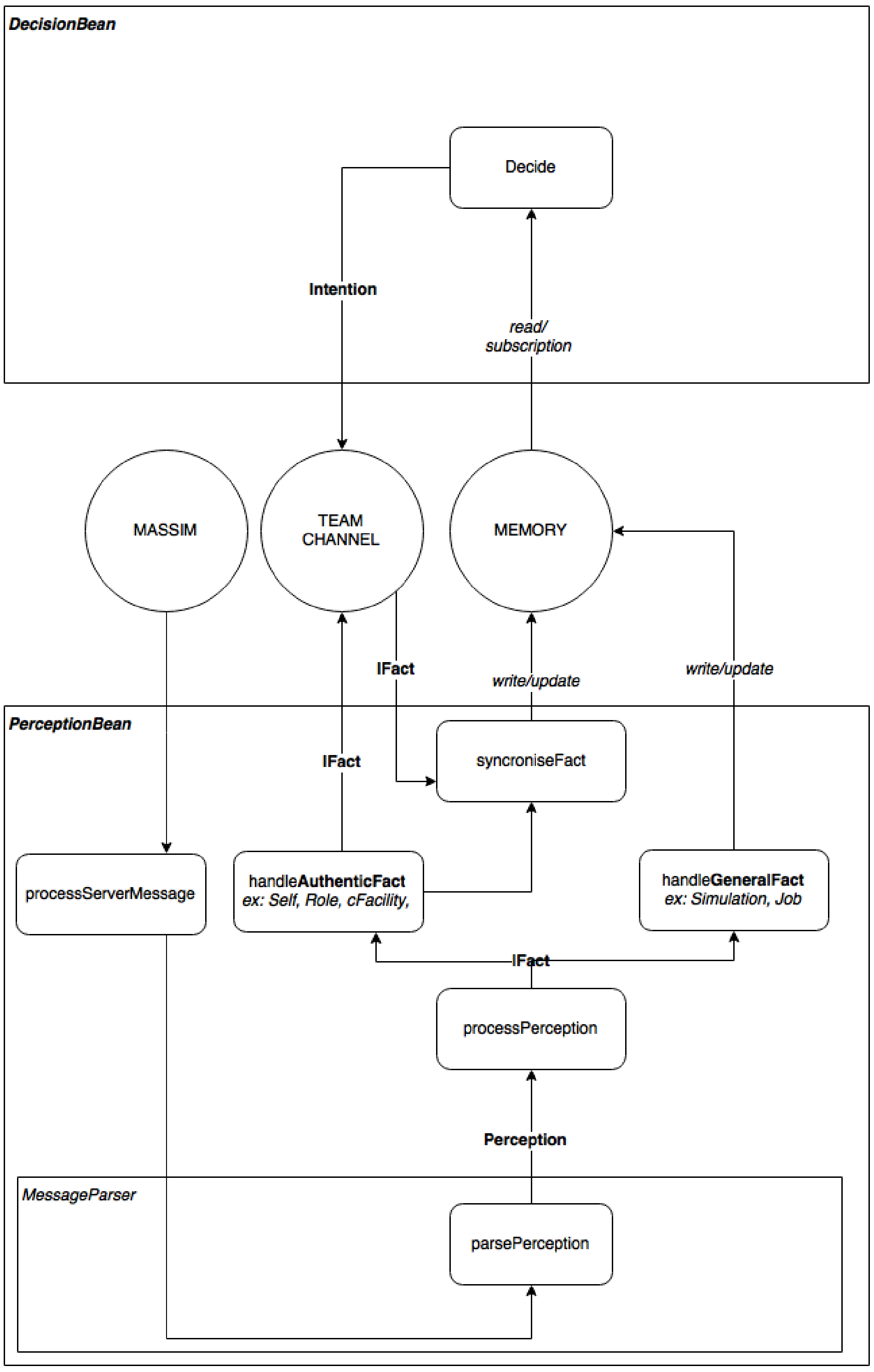
When coming to the communication strategy in the agent team, we estimate the message traffic load cause it is essential for overall system performance. All agents share their own perception with all other agents in every step using messages. Every message must be processed by each agent, it updates the world model of the agent. The number of resulting message load is ($\mathcal{O}(n^2)$, n being the number of agents in the system). Also there is some coordination going on which involves $\mathcal{O}(n)$. Finally, each agent tells all other agents what it intends to do (assemble, assistAssemble or recharge) in the next step where we also estimate ($\mathcal{O}(n^2)$.

The team coordination strategy is controlled by a central instance that maintains and coordinates the agents in the team. However every agent can be this controlling instance (for more details see Section 4).

The principle agent structure and decision process has been visualised in Figure 2. It is close to the BDI agent model. The agent lifecycle is controlled by the JIAC runtime. Each agent has its own thread of control, thus acting autonomously, and does not require user interaction. Map discovery, enemy surveillance and job planning is proactively done by every agent. They also proactively share their perception and intention with other team members.

However, agents react to server messages and their perception input and can act according to that. In fact, the decision process is triggered once every simulation step. Agents update their world state from perception input and from what they are told by other agents.

Figure 2 The single agent structure based on the JIAC agent model



3 Software architecture

Team BathTUB uses JIAC as the basis for multi-agent programming. JIAC is not a specific programming language but rather an agent middleware for distributed computing. JIAC is based on the Java programming language and does not prohibit nor enforce special concepts. JIAC has been successfully used by former TUB teams in former contests. JIAC is also the runtime environment for the agents, which runs reliably and with good performance.

Apart from JIAC we use Git along with Gitlab for development team synchronisation and Slack as communication platform to discuss design decisions and problems.

Team BathTUB uses one well-known algorithm. For routing we rely on Dijkstra (1959) [offered by GraphHopper (https://www.graphhopper.com/)], mainly because of its ease of use. For planning we invented our own algorithm/protocol as described in the next section, because we needed an algorithm that is also capable of easily coordinating the agents without much overhead.

The waypoint routing algorithm calculates the fastest route to destination considering the battery state of the agent. It does not consider the price for charging. The algorithm checks whether it is possible to reach the destination and charge later. Routes with a lower number of charging stations are preferred, choosing the shortest route beyond all possible routes. If no route can be found, which happens when no charging stations can be reached within the current charge, an exception will be thrown resulting in an emergency service call.

4 Agent team strategy

In this section, we describe the team coordination strategy of our team. Any time an agent has nothing to do it starts to plan a job, which means it will take control over a job-planning singleton, sending a call for proposal to all agents. Using a Java singleton pattern at this point is a break in agent-oriented design, considered as a development and performance shortcut, which is possible and not prohibited when using the JIAC framework. The resource the singleton is controlling is the job-planner. In theory, it should have been a job-brokering or job-planning agent or an agent component, but in fact, it is now a class and instance in the Java virtual machine. It can be seen as an infrastructure functionality and component, where all agents have access to it. By this, the singleton instance makes sure that only one job-planning instance exists and can be used.

The job planning process has three steps: Proposals, Assignments and ExecutionStart. When an agent starts the JobPlanner it plans all active jobs, one at a time. For every job a proposal request is generated for distributing the items of a job to the agents, one item type at a time. The agents now calculate their subjectively best way to retrieve some of these items and send the result (the cost) back to the planner as their personal *JobProposal*. The planner then evaluates these proposals and chooses the fastest partly job-fulfilling proposals. This is repeated until every item is retrievable. After all that, the agents are again informed about who is doing what in a central manner (*JobAssignment*). If all items of a job are assigned, the JobPlanner starts the execution of the job by sending a *JobExecutionStart* message.

Information exchange in the agent team has already been discussed earlier. Following the earlier description, the complexity of perception exchange results simply from every agents sending its perception and intention to every other agent by means of JIAC-Messages. Only job bidding and task assignment is a bit more complex, but still does not break the server cycle time limit.

Between subsequent simulation steps every agent calculates its proposal (see description above), either it has something to do in the next step or not. Idle agents directly execute one of the following activities: map discovery, job planning or enemy surveillance.

Failing actions do not cause severe problems to the agents. Agents can also recover from crashes by controlling the agent's lifecycle, or they are simply restarted on the agent node (the JIAC runtime environment running in a JVM) of the current JIAC application.

5 Evaluation

Team BathTUB ranked fifth in the 2016 edition of the Multi-Agent Programming Contest. This means four other teams performed better than us and our overall performance was not satisfying.

Thinking about reasons why, at first raises some questions in general software engineering. Main principles of software engineering are valid also in agent-oriented programming and development. Questions here are source code version control, build process and testing. And they tell a lot about communication, cooperation and coordination between members of the development team.

We used the free and open source code management system Git (https://git-scm.com/). It is fast, it is distributed, every participant has the full history of everything in the repository. The main challenge when using Git is integration. While we have 23 branches for separate functions, we missed the point when it became essential to integrate dependent functionalities and test them together. Furthermore, we neglected the old software developer rule to always have a running system. In the end, we merged a lot of useful functionalities just before the contest began and noticed too late that this version is not working. We could not fix it before or during the contest, either. As a consequence, we had to find an earlier version, that was reliably participating in the contest, but was missing major functionalities that would gain more score.

Another important point when evaluating our team is the contest scenario, which simulates a complex problem in logistics. The JIAC V Agent Framework is not a special application in logistics and our developers had to learn, understand and implement building blocks for detailed organisation of complex operations in this field, too, in addition to learning and understanding the agent-oriented paradigm, principles of agent-oriented programming and the use of Java and JIAC V.

6 Conclusions

In this report we describe our approach to solve the complex multi-agent contest scenario. The performance of our agent team was considerably lower than its potential. In our opinion we could have done a better job when observing some few best practice rules in general software engineering. Nevertheless, we will learn from this experience and show excellent results. We are looking forward to the next edition of the Multi-Agent Programming Contest.

7 Team overview: short answers

7.1 Participants and their background

- *What was your motivation to participate in the contest?*

  Course in Computer Science Bachelor's program 'Multi-Agent Contest'.

- *What is the history of your group?*

  (course project, thesis, …) The course was a software engineering project lasting the summer semester 2016.

- *What is your field of research? Which work therein is related?*

  Agent-Oriented Software Engineering (AOSE), Multi-Agent Framework JIAC, applications of the agent-oriented paradigm in real world.

7.2 The cold hard facts

- *How much time did you invest in the contest (for programming, organising your group, other)?*

  About 1,000 man hours + one week preparation just before the contest.

- *How many lines of code did you produce for your final agent team?*

  4,546 lines of code.

- *How many people were involved?*

  6.

- *When did you start working on your agents?*

  18 April 2016.

7.3 Strategies and details

- *What is the main strategy of your agent team?*

  To make easy jobs fast, we maintain an item dependency graph and observe shops and opponent behaviour, if we can be faster we do the job with all available agents, we avoid crafting.

- *How does the team work together? (coordination, information sharing, ...)*

  Every agent sends its perception to every other agent in the team, also the intention (the action that will be executed in the next step) is sent each to all. The jobplanner ranks active jobs and distributes it over free agents.

- *What are critical components of your team?*

  Crafting was working fine in the preparation phase, but during the contest we had massive problems.

- *Can your agents change their behaviour during runtime?*

  If so, what triggers the changes? Agents observe job completion, if a job is completed by the other team, the current activity (and job) is aborted.

- *Did you make changes to the team during the contest?*

  Yes, we tried hard to fix the crafting bug, but did not succeed. We also fixed minor bugs resulting in slightly better performance during the contest.

- *How do you organise your agents? Do you use e.g., hierarchies? Is your organisation implicit or explicit?*

  Every agent is an individual software entity, running in an own thread, responsible for their own world state update, map exploration. The jobplanning is done in a centralised way, but could be done by every agent.

- *Is most of your agents' behaviour emergent on an individual or team level?*

  We observed many emergent behaviour during preparation and the contest on agent level and on team level, but non has been positive effect though.

- *If your agents perform some planning, how many steps do they plan ahead?*

  Our agents do not have an explicit notion of steps during planning, the number of steps results from the job that has been selected and the actual position of the agents and the availability of items.

- *If you have a perceive-think-act cycle, how is it synchronised with the server?*

  The perceive-think-act cycle is completely triggered by the server, every time a server message arrives the perception updates the world model of every agent, after this the decision process is started and the desired action is sent back to the server. The duration of a server cycle is usually long enough to do all that except during initialisation where many things must be calculated, the map routing graph, the item dependency graph, after 4 to 5 cycles the agents are synchronous with the server.

7.4 Scenario specifics

- *How do your agents decide which jobs to fulfil?*

  Simple jobs first and fast, with the start of each simulation we build an item dependency graph, we also explore the shopping facilities to learn how many items are available, the jobplanner ranks jobs after their complexity, less edges and are better, crafting is worse. We also try to observe the opponent behaviour in order to guess which job the enemy is working on. Agents near shop observe the item count and compare missing item to jobs.

- *Do your agents make use of less used scenario aspects (e.g., dumping items, putting items in a storage)?*

  Dumping, storage and job creation is not used at the moment.

- *Do you have different strategies for the different roles?*

  Yes, speed matters, and also the battery charging, routes with less charging stops are preferred.

- *Do your agents form ad-hoc teams for each job?*

  Yes, when the jobplanner selects a job for execution the number and roles differ every time.

- *What do your agents do when they do not pursue any job?*

  This is not the case, when an agent is not involved in a job execution, it takes control over the jobplanner and starts the next job.

7.5 And the moral of it is …

- *What did you learn from participating in the contest?*

  It is nice to hear the theory, but when it comes to implementation it is often hard and time-consuming to implement, especially debugging takes many more time, also distributed programming can often only be synchronised on compiler level.

- W*hat are the strong and weak points of your team?*

  The team runs robust and fast is positive, the weak point is crafting.

- *How viable were your chosen programming language, methodology, tools, and algorithms?*

  The JIAC runtime and agent skeleton was very reliable and showed good performance, tool support could be better, GraphHopper was also very good and has an elaborate API.

- *Did you encounter new problems during the contest?*

  Multi-modal logistics (or transport) is an open research area that was also new to all members of our team.

- *Did playing against other agent teams bring about new insights on your own agents?*

  Yes, it was nice to monitor other agents and to guess what is the goal, the strategy and tactics, in order to forecast the behaviour and then upset the plans.

- *What would you improve if you wanted to participate in the same contest a week from now (or next year)?*

  Crafting is definitely our short term goal, we do not really examine whether a job is worth the execution (in terms of brings a good return).

- *Which aspect of your team cost you the most time?*

  The jobplanner.

- *What can be improved regarding the contest/scenario for next year?*

  Some of the problems came only up when we played against another team, that has not been tested before, maybe play against other during preparation phase.

- *Why did your team perform as it did? Why did the other teams perform better/worse than you did?*

  The main reason is that our changes in crafting right before the contest could not be fixed during the contest

References

Dijkstra, E.W. (1959) 'A note on two problems in connexion with graphs', *Numerische Mathematik*, Vol. 1, No. 1, pp.269–271.

Fricke, S., Bsufka, K., Keiser, J., Schmidt, T., Sesseler, R. and Albayrak, S. (2001) 'Agent-based telematic services and telecom applications', *Commun. ACM*, Vol. 44, No. 4, pp.43–48.

Git-Distributed Source Code Management System [online] https://git-scm.com/ (accessed 17 March 2017).

Heßler, A. (2013) *MIAC: Methodology for Intelligent Agents Componentware*, PhD thesis, Technische Universität Berlin.

Heßler, A., Hirsch, B. and Keiser, J. (2007) 'Collecting gold. jiac iv agents in multi-agent programming contest', *Proceedings of the Fifth International Workshop on Programming Multi-Agent Systems. At AAMAS 2007*, 2007 Honolulu, HI, USA.

Heßler, A., Hirsch, B. and Küster, T. (2010) 'Herding cows with JIAC v. annals of mathematics and artificial intelligence', in *Annals of Mathematics and Artificial Intelligence*, August, Vol. 59, Nos. 3–4, in Dix, J., Behrens, T., Dastani, M., Koester, M. and Novak, P. (Eds.): *Special Issue: The Multi-Agent Programming Contest: History and Contestants in 2009*, pp.335–349, Springer, Netherlands, , doi: 10.1007/s10472-010-9178-x.

Heßler, A., Konnerth, T., Napierala, P. and Wiemann, B. (2013) 'Multi-agent programming contest 2012: tub team description', in Köster, M., Schlesinger, F. and Dix, J. (Eds): *The Multi-Agent Programming Contest 2012 Edition: Evaluation and Team Descriptions*, number IfI-13-01 in IfI Technical Report Series, pages 86–97. Institut für Informatik, Technische Universität Clausthal.

Hirsch, B., Konnerth, T. and Heßler, A. (2009) 'Merging agents and services – the JIAC agent platform', in Bordini, R.H., Dastani, M., Dix, J. and Seghrouchni, A.E.F. (Eds.): *Multi-Agent Programming: Languages, Tools and Applications*, pp.159–185, Springer Dordrecht Heidelberg, London, New York.

The Graphhopper Directions Api [online] https://www.graphhopper.com/ (accessed 13 March 2017).

The Multi-Agent Programming Contest [online] https://multiagentcontest.org/ (accessed 13 March 2017).