Ann Math Artif Intell (2018) 84:57–74
https://doi.org/10.1007/s10472-018-9588-8

Engineering a multi-agent system in Jason and CArtAgO

Multi-agent programming contest 2017

Jørgen Villadsen[1] · **Oliver Fleckenstein**[1] · **Helge Hatteland**[1] · **John Bruntse Larsen**[1]

Published online: 1 June 2018

Abstract This paper presents the overall strategy utilized by Jason-DTU to achieve a shared second place in the annual Multi-Agent Programming Contest. It provides an overview of the implementation details considering perception, task handling, agent logic and more. The paper analyzes the team's results in each match, and is summarized by evaluating strengths and weaknesses of the proposed multi-agent system.

Keywords Multi-agent programming contest 2017 · Multi-agent systems · Agents in the city · Jason · CArtAgO

Mathematics Subject Classification (2010) 68T42

1 Introduction

The name of our team is Jason-DTU. We participated in the *Multi-Agent Programming Contest* (MAPC) in 2009 and 2010 as the Jason-DTU team [2, 6], in 2011 and 2012 as the Python-DTU team [3, 9], in 2013 and 2014 as the GOAL-DTU team [8] and in 2015/2016 as the Python-DTU team [7].[1]

The MAPC 2017 scenario consists of two teams of agents, each moving through the streets of a realistic city as vehicles. The goal for each team is to earn as much money

[1] https://multiagentcontest.org/

✉ Jørgen Villadsen
jovi@dtu.dk

[1] Algorithms, Logic and Graphs Section, Department of Applied Mathematics and Computer Science, Technical University of Denmark, Richard Petersens Plads, Building 324, DK-2800 Kongens Lyngby, Denmark

as possible, which is rewarded for completing certain jobs. Jobs comprise the acquisition, assembly and delivery of items, utilizing certain facilities placed randomly on the map.[2]

To earn more money than the opposing team, our multi-agent system uses centralized planning, deciding which jobs to solve; problem decomposition, dividing jobs into several tasks, which in turn are delegated to individual agents; and hierarchical planning, defining high-order plans on how to complete their assigned tasks.

The scenario features four distinct roles for the agents: drones, motorcycles, cars and trucks, sorted by increasing capacity and energy, and decreasing speed. More interestingly, each role is associated with a set of usable tools, such that no agent is single-handedly able to assemble all items. As a result, the scenario encourages agent coordination, having to use several agents to assemble certain items.

The competition is an attempt to stimulate research in the area of multi-agent system development and programming, which has also been the goal of Jason-DTU. The team's effort has been focused on devising fitting solutions for the key problems introduced; on how to implement it in practice, emphasizing a high level of abstraction and robustness; and on collecting and analyzing results, constantly looking for improvement.

The overall strategy of Jason-DTU's multi-agent system is described in Section 2. The section also covers prior attempts at solutions, having developed a system targeting last years version of the scenario as a part of a bachelor thesis. Section 3 goes into detail about how the system is implemented, from server communication to agent coordination, focusing on the crucial elements of the system. Section 4 gives an overview of Jason-DTU's matches in terms of results as well as statistics, concluding with a summary of the team's performance. Section 5 concludes. Finally, Appendix A includes a questionnaire, intended to give insight into the participants' background, the developed system and their take on the competition.

2 Strategy

During the development of the Jason-DTU multi-agent system for this year's contest, multiple approaches and strategies have been considered and explored for the best possible way to solve the problems presented in the scenario. The overall goal was to build a solution capable of solving problems spanning a wide range of parameters. It should be able to scale to simulations of different lengths, and handle simple and more complex items, all with a varying number of agents. The early work for this project was based on last year's contest, but extended to this year's version as soon as possible. The first working version of the system was also developed during the bachelor thesis [4].

2.1 Early strategies

A decision was made to make a centralized planner, subject to delegating tasks to different agents, while each individual agent would be responsible for planning how its assigned task should be solved. However, there was a question of how much the work should be done on a centralized level versus a decentralized level.

The early strategies for the system were heavily leaned toward doing much of the work in the central planner. For example, agents would be given concrete shops, items, and

[2] https://github.com/agentcontest/massim/blob/master/docs/scenario.md#background-story

delivery locations, and then just execute the necessary actions to achieve their objectives. This was fine for the early strategies, but problems began to rise when assembling more complex items. This part of the system was however reused and expanded for later use. In order to retrieve all the items necessary for a job, a shopping list was created, being a map of the shops and which items and the amount that should be bought. Then the planning to retrieve these items could be done easily by the agents by visiting each shop in turn.

The first solution developed was able to solve simple jobs, meaning jobs agents could handle entirely by themselves. However, as many jobs required items that were too complex for one agent to assemble alone, this strategy had to be revised. Because all synchronization and coordination was handled through the central planner, unnecessary complexity arose, without the planner having any need to influence these situations.

To solve this problem, trucks were assigned to each shop and as many workshops as possible, to build a hierarchical structure of the agents. These trucks would act as a middle layer between the central planner and the rest of the agents, and be responsible for assembling all items. All other agents would then have much simpler tasks, only going between different trucks and storage locations to retrieve, give, and deliver items. However, this solution introduced bottlenecks when too many jobs should be completed in parallel, as the trucks were unable to handle the workload.

2.2 Centralized vs decentralized planning

The initial strategy relied on decentralized planning, using the Contract Net Protocol to delegate tasks between agents. As a result, the agent best suited for each task was chosen to complete it, realizing optimal task delegation on an individual level. This strategy was based on last year's scenario, where assembling did not necessarily require tools, hence neither did it require agent coordination. For this year's competition however, tools became an important part of solving jobs, thus requiring the use of multiple agents to assemble certain items. Using the decentralized approach, agent coordination became increasingly difficult with the number of agents involved, and the development eventually took a turn toward centralized planning.

With a centralized planner, the system is able to decide which jobs to complete based on the available agents, in contrast to a decentralized planner, where free agents would immediately start solving jobs, prior to knowing whether the required resources are at their disposal. This is checked prematurely by the centralized planner, not assigning any tasks to the agents before the whole job has been successfully delegated, based on the parameters of the available agents' vehicles. While the planner delegates tasks, the agents are responsible for figuring out how to solve the tasks themselves. In a sense, the centralized planner commands agents to achieve high-level goals, in which each individual agent continues planning in a decentralized manner on how to proceed.

Using a combination of centralized and decentralized planning, the system is only responsible for maintaining a global state of the simulation, and telling the agents what to do. Doing so allows the agents to decide for themselves how their goals are best achieved, taking their current charge, carrying capacities and so on into consideration.

2.3 Final strategy

The final approach used for the contest moved most of the heavy planning away from the central part of the system, simplifying the overall structure. This was done by only having

the central planner handle high-level tasks. These were assigned to a group of agents, where the number and type of agents would be estimated based on the difficulty of the job. As specific tools were needed to solve certain jobs, the roles of the agents also had to match the requirements. Doing so, the system was able to easily handle different complexities of jobs, using different groups of agents based on the requirements and agents' availability.

All of the planning and coordination for actually solving jobs is done between the agents in the group. The central planner delegates to each agent a shopping list, including tools, and after retrieving the items, every agent meets at the same workshop and waits. Next the agents work together to assemble and deliver the items, using the assembling protocol described in Section 3.4.2.

This approach was still limited with too complex jobs. However, the problem was in the end solved by dividing jobs into smaller jobs, when not enough agents were available. For example, if a job required multiple items to be delivered, each item could be handled as an independent partial job. Using this strategy, the system was able to handle both simple and more complex jobs with a varying amount of agents available.

Auctions often provided more profit per step compared to normal jobs, and was therefore an option to gain larger profits, without risking other teams completing the job before our team could. To handle auctions as efficient as possible, the agents only bid on auctions in the last possible step, in order to avoid bidding wars with other teams. Bidding on jobs did not cost anything in terms of money, but did cost the bidding agent an action, hence removed the option for the agent to do anything else in the given step. As there often were more than enough auctions available, it was not prioritized to win every auction. Agents only bid on auctions if they were sure the team would be able to solve the job. After winning an auction, the agents would prioritize the job the same way as missions, as assigned auctions have a penalty if they are not completed on time.

2.4 Free agents

The simulation server did not always post enough jobs to keep every agent busy, thus the agents proactively prepare for new jobs. They would first of all ensure that their charge were above a certain threshold, which is done by visiting the nearest charging station. Secondly, each agent would try to gather resources at one of the resource nodes. Lastly, the agents never made use of the SKIP action, but instead used RECHARGE to get as much charge as possible, when not doing anything else.

3 Implementation

The multi-agent system has been developed using two frameworks, namely Jason, a Java-based interpreter for an extended version of AgentSpeak; and CArtAgO, a **c**ommon **art**ifact infrastructure for **ag**ents **o**pen environments.

Jason implements the operational semantics of AgentSpeak, and provides a platform for the development of multi-agent systems, including several customizable features. The extended version of AgentSpeak is a logic based agent-oriented programming language with Prolog-like syntax, allowing for succinct agent logic [1].

CArtAgO is a general purpose framework/infrastructure that facilitates the programming of virtual environments for multi-agent systems. The framework is based on the Agents & Artifacts meta-model, introducing high-level metaphors taken from human cooperative

working environments such as *agents*, *artifacts* and *workspaces*. Artifacts are resources and tools, which can be dynamically constructed, used and manipulated by agents to realize their individual or collective goals [5].

3.1 Server communication

In the contest the multi-agent system communicates with the Multi-Agent Systems Simulation (MASSim) server through a piece of client-side software called EISMASSim.[3] EISMASSim is based on the Environment Interface Standard (EIS), a proposed standard for agent-environment interaction. It maps the communication between agents and the MASSim server, (i.e. sending and receiving XML-messages), to Java method calls. Also, it automatically establishes and maintains connections to a specified MASSim server. In other words, EISMASSim is a proxy environment on the client side which handles communication with the MASSim server completely by itself.

EISMASSim is fully configurable, allowing percepts to be delivered as notifications; scheduling of actions, such that subsequent calls to PERFORMACTION will block until a valid action-id is available; and queued percepts, such that GETALLPERCEPTS only yields one collection of percepts per invocation. By receiving percepts as notifications, percept handlers can be attached to entities, enabling the system to respond or trigger execution when certain percepts are perceived. Furthermore, instead of having multiple PERFORMACTION invocations block the agent thread, the system defines its own scheduling mechanism, where subsequent calls to PERFORMACTION are ignored, except for the first action. The scheduling mechanism allows agents to queue a single action, but only if the server's response deadline has passed. Doing so makes the system robust toward a slow server connection, allowing the agents to continue planning, regardless of the data being outdated.

3.2 Perception

Percepts are sent by the server as XML files, and contain information about the current simulation. Initial percepts (sent via SIM-START messages) contain static information while other percepts (sent via REQUEST-ACTION messages) contain information about the current simulation state.[4]

Once a new step is perceived, a series of events occur, starting by collecting all current information available to the agents. While doing so, agent specific percepts are passed on to the agent's personal artifact, responsible for parsing and updating both the agent model and the agent's observable state. After having collected all percepts, they are passed on to one or two different artifacts depending on the message type. Static information remains unchanged, thus only needs to be perceived the very first time while other information must be perceived in every step, reflecting changes to facilities and jobs.

When all the information has been perceived, new jobs are evaluated and a task delegation algorithm is run asynchronously not to interfere with the perception cycle. Finally, the agents are notified that the simulation has advanced to the next step, continuing their reasoning cycle. As a result, the system comprises a perceive-act cycle doing the thinking in parallel, opposed to the more general perceive-think-act approach.

[3] https://github.com/agentcontest/massim/blob/master/docs/eismassim.md

[4] https://github.com/agentcontest/massim/blob/master/docs/scenario.md#percepts

3.3 Task handling

In short, a job comprises the acquisition of base items, assembling items into the job's required items, and delivering these items at a given storage facility. The central planner evaluates a job in terms of profit, minimum number of agents required and an average amount of steps needed to complete it. While the profit is easily calculated, being the job's reward without the cost of buying base items, finding the number of agents and amount of steps calls to use for extensive approximations. First of all, a shopping list is created, considering all base items needed and in which shops they can be bought. Note that shops have different item availability, both in terms of assortment and quantity. Then the number of assembling agents is based on the volume of the items, while the number of agents to retrieve items is based on both the volume and the number of shops to visit. The sum of assembling and retrieving agents comprise the minimum number of agents required, being a rough approximation. Finally, to solve the job agents must retrieve the base items from shops, assemble items in workshops and deliver items in storage facilities. Based on an average agent speed and location, an average amount of steps needed to complete the job is calculated from the steps required to get between all facilities, in addition to the steps required to purchase and assemble all items.

Having evaluated the profit and a step estimate for each job, jobs can be compared in terms of profit per step. This is done by adding all job evaluations to a prioritized queue, where the job with the highest profit per step has first priority. The task delegation algorithm attempts to delegate tasks according to this prioritization, considering whether there are enough agents available and whether the job can be completed in time. Note that all jobs have a time frame in which they are active, given by a start step and an end step. A job is delegated by dividing it into several tasks. Firstly, a job is split into partial jobs, where the job's required items are distributed across multiple assemblers. Each assembler is responsible for assembling and delivering a subset of the job's required items, mobilizing a team of agents for efficiency. A shopping list is created for the subset of items, and agents are selected to retrieve items and assist assemble based on the contents of their inventory and how much they can carry. The remaining step is to define the teams and assign the tasks. While each agent knows what to assemble or what to retrieve, how to do so is up to the individual agents. The task delegation algorithm simply uses CArtAgO elements to signal agents of new tasks, providing all necessary information to allow for autonomous problem solving and agent coordination.

3.4 Agent logic

Each agent tries to complete one task at the time, and assigning an agent a new task will cause the agent to drop all ongoing plans. By doing so, agents are able to do several tasks such as charging or gathering resources while they are awaiting assignments.

Jobs are solved using hierarchical planning, having defined a limited number of tasks agents can be delegated with and several higher-order plans to do so. All plans eventually boil down to compositions of the same primitive action, namely PERFORMACTION, sending a given action to the MASSim server. Since only one action can be performed each step, this primitive action is the natural synchronization point of the agents' reasoning cycle. While evaluating several steps into the future is reasonable, random failures and unpredictable events can occur, thus the agents only consider one step at the time. This is achieved by synchronizing the agents after performing an action, having them wait to receive a signal before advancing. As a result, the agents are proactive, working toward completing tasks;

reactive, considering changes to the environment in every step; and autonomous, doing both on their own.

Agents perceive their surroundings in the form of beliefs, and each agent has a personal belief base, containing all information the agent considers to be true or not. As mentioned earlier, each agent has its personal artifact as well, and this is directly related to the agent's beliefs. An artifact defines a subenvironment, which agents can selectively observe to obtain beliefs about the environment. More specifically, the agent's personal artifact contains information about its current state, for instance in which facility it is located, how much charge is remaining and how much it carries. Even more crucial is the information about what kind of vehicle the agent is, and the vehicle's attributes. By incorporating this data into a subenvironment, each agent only considers what is relevant to them.

3.4.1 Planning

Even though agents only reason about the current step of the simulation, they are capable of autonomous planning, choosing which goals to fulfill and which plans to select to do so. Plans are usually associated with one or more rules, deciding whether a plan is applicable. Rules are the second-most primitive part of an agent's repertoire, comprising literals and arithmetic, and relying on unification to evaluate to some boolean value. The simplest of rules are used to retrieve information from the agent's belief base, such as the speed of the vehicle, while more advanced rules execute internal actions.

Internal actions allow AgentSpeak to interact with Java, and retrieve information unavailable to the agent. As previously mentioned, a shop's assortment is subject to change every step, leaving it unfeasible to model in every agent. As a result, internal actions are defined to, among other, get the amount available of a specific item in a given shop; get the location of a facility of a given type; and get the base items needed for a list of items. By doing so, only agent-specific data is available to each individual agent, while global data must be retrieved using internal actions. This requires some extra work on the agent's part, however the system only has to manage a single model.

Jason plans are defined using compositions of primitive actions, rules to select plans, and internal actions to retrieve information from CArtAgO artifacts. Goals are fulfilled by executing plans, and an example of such a goal is to charge a vehicle, which is fulfilled by the plan: go to a charging station and perform the CHARGE action. Using a logical approach, different cases are considered depending on the state of the vehicle, corresponding to one plan each. Continuing with the charge example, there are three cases to consider: (1) the vehicle is fully charged, and the goal is therefore fulfilled; (2) the vehicle is not fully charged and in a charging facility, fulfilling the goal by performing a charge action and executing the plan once more; and (3) the vehicle is neither fully charged nor in a charging facility, thus fulfilling the goal by first executing the plan of getting to a charging facility followed by another attempt at executing the plan. All plans except the one where the vehicle is fully charged, will recursively execute itself, making sure that when the entirety of the plan succeeds, the vehicle is in fact fully charged.

This example shows how plans can be used in other plans to achieve a form of hierarchical planning. Returning to the agents' most abstract and high-level plans, namely the tasks delegated by the system, they can be depicted as hierarchical plan trees as illustrated by the assemble task in Fig. 1.

Note that all leaves of the tree are primitive actions, and that plans can be used for multiple purposes taking one or more arguments, for instance to specify in which facility to go. As a result, each plan is uttermost specific to fulfilling a goal, facilitating the use

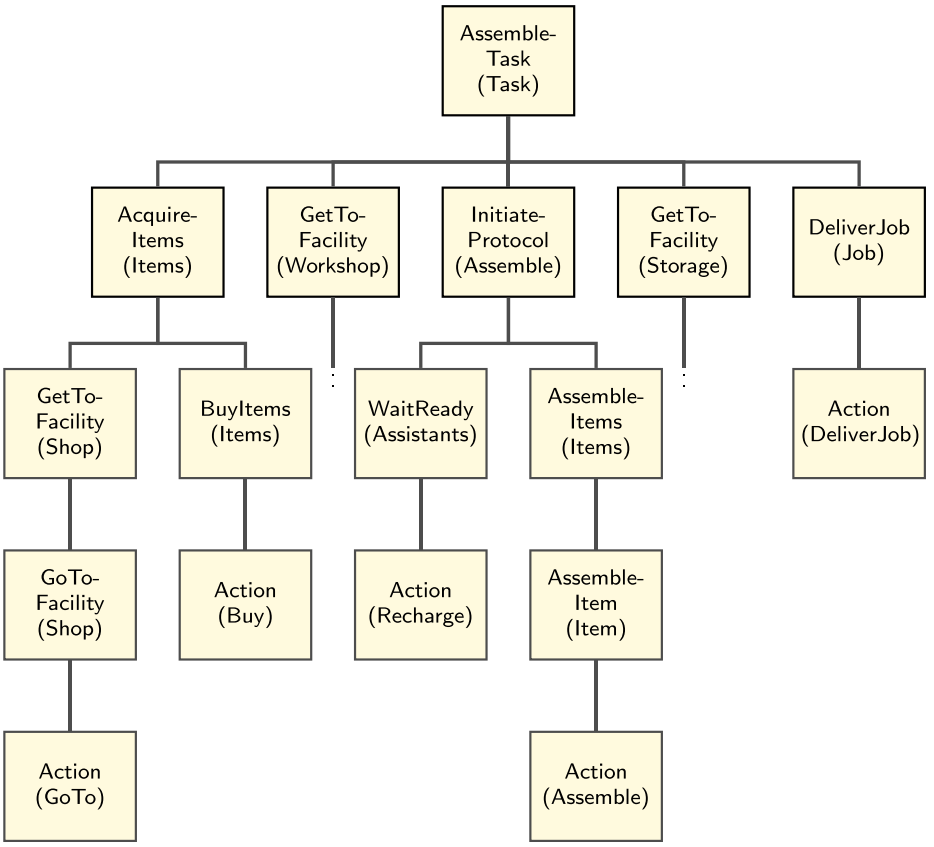

Fig. 1 Hierarchical plan tree for AssembleTask

of problem decomposition to devise elegant hierarchical plans. Furthermore, by using the recursive approach the agents are robust toward random failures and unpredictable events, as they check in every step whether their goal is fulfilled. Some goals do however require the help of other agents, in which plans such as the INITIATEPROTOCOL has been defined, describing an agent coordination procedure.

3.4.2 Cooperation

Even the simplest assemble task may require help from multiple agents, given that vehicles can only use a specific subset of tools. This requirement is accounted for by the delegation algorithm, assigning each task to a team of agents. A team consists of an assembler, responsible for performing assemble actions and delivering the assembled items to a storage facility, and an arbitrary number of retrievers, responsible for retrieving items from shops and performing assist assemble actions. The number of retrievers selected for a task depends on several factors, namely the volume of the required base items, the tools required to assemble all items and whether agents carry any needed items in their inventories. Using items intended for previous failed jobs is prioritized to save both money and time on retrieving them.

Once a team has been assigned, each individual agent completes their task of retrieving a set of items. After doing so, each retriever sends a message to the assembler, notifying that they are ready to assist. The assembler waits until all assistants are ready, before setting a flag to indicate the start of the assembling process. Once all the required items have been assembled, the assembler removes the flag, concluding the assemble protocol.

While assisting with an assembly, the retrievers have no knowledge of which items are being assembled, nor how many steps it will require. This may seem disadvantageous, but has the benefit of not having to take random failures into account. With several agents assisting, the chance for any of them to randomly fail increases, and by simply using a flag, only the assembler has to keep track of which items are successfully assembled or not. To ensure flexibility, the assistants are released once they have used up all items they carried. A more dynamic approach would be to time the arrivals of the required assistants according to which base items are needed when; however doing so is unfeasible due to random failures and unpredictable events.

The system is designed to utilize all free agent capacity, not having idle agents when there are jobs to complete. As a result, the system attempts to divide jobs into tasks which can be solved by the agents individually, keeping agent coordination to a minimum. While coordination is necessary to assemble items, it generally involves having one or more agents wait, making the problem solving inefficient. By doing as much as possible on an individual level, the system is capable of solving a vast amount of tasks in parallel.

4 Results

The multi-agent system developed by this year's DTU team was able to claim a shared second prize in the 2017 competition, with many interesting matches. The contest was especially interesting for the developers, as the system had never been tested against other multi-agent systems. We explore each of Jason-DTU's matches, and try to compare the weaknesses and strengths between the different systems and their respective strategies. We first separately analyze the six matches.

4.1 Analysis of Jason-DTU's matches

4.1.1 BusyBeaver vs. Jason-DTU

Jason-DTU's first match was against BusyBeaver, whom turned out to be the winning team, and therefore the most challenging match. While the system performed badly in the second and third simulations of the match, the first simulation was quite exciting, with DTU being in the lead for most of the time but ending up loosing. By analyzing the graphs from all three simulations, it is easy to see the difference in strategy. BusyBeaver invests a lot in items early on, trying to have the necessary resources ready when a new job is posted. Jason-DTU's strategy uses a just-in-time approach, by only buying and assembling the required items when needed. Looking at the average time it takes each team to complete missions, which can be seen in Table 1, the superiority of BusyBeaver's strategy is seen; it is able to complete jobs around three times faster. One positive note for the Jason-DTU team was its ability to complete all of the auction jobs it bid on, while BusyBeaver did not complete any. Looking at the actions performed by the agents on the two teams, BusyBeaver is in general doing more actions, and is utilizing the GIVE and RECEIVE actions which Jason-DTU is not.

Table 1 The number of jobs completed, the average time to complete a job, and the average reward per job for each team in the match between BusyBeaver and Jason-DTU

	Sim 1	Sim 2	Sim 3	Sim 1	Sim 2	Sim 3	Sim 1	Sim 2	Sim 3
Team	Completed jobs			Average time			Average reward		
BusyBeaver	31/48	53/59	33/49	7	15	23	2697	3202	5223
Jason-DTU	17/48	6/59	16/49	33	42	77	3367	4429	4111
	Completed auctions			Average time			Average reward		
BusyBeaver	0/4	0/3	0/1	–	–	–	–	–	–
Jason-DTU	4/4	3/3	1/1	36	33	112	2354	2705	8962

Final scores, BusyBeaver vs. Jason-DTU, Sim 1 88646:77716, Sim 2 138592:53370, Sim 3 128250:73308

4.1.2 Jason-DTU vs. TUBDAI

While the second match, against TUBDAI, was a big victory for Jason-DTU, the statistics still show some interesting data. The first two simulations were quite similar in the kind and amount of actions the agents were doing. Both of the teams were very active with ASSEMBLE and ASSISTASSEMBLE. However, when analyzing the number of jobs that each of the teams completed, there is a clear difference. The Jason-DTU system completes far more jobs than TUBDAI, while also delivering the jobs faster on average, both of which can be seen in Table 2. One very interesting point from the data is that TUBDAI is able to complete many auctions, showing that their system is optimal when completing jobs without competition.

4.1.3 Chameleon vs. Jason-DTU

The match between Chameleon and Jason-DTU was less interesting, as Chameleon was quite passive, however a very good display of how well the Jason-DTU system works. While the earning of money stalled and even lowered a bit at the end of the first simulation, the second and third simulation displayed a stable, linear growth of money throughout the simulation, which illustrates how good the final strategy is. This data is displayed in Fig. 2. The good results were because of the many jobs completed, both normal jobs and auctions, which were done rather quickly on average. It can also be seen in the statistics from the

Table 2 The number of jobs completed, the average time to complete a job, and the average reward per job for each team in the match between Jason-DTU and TUBDAI

	Sim 1	Sim 2	Sim 3	Sim 1	Sim 2	Sim 3	Sim 1	Sim 2	Sim 3
Team	Completed jobs			Average time			Average reward		
Jason-DTU	48/52	53/58	31/32	24	56	63	2076	2902	2925
TUBDAI	4/52	5/58	1/32	30	93	81	1958	3461	3806
	Completed auctions			Average time			Average reward		
Jason-DTU	9/15	16/19	0/0	30	51	–	2570	3182	–
TUBDAI	6/15	3/19	0/0	35	67	–	3291	4231	–

Final scores, Jason-DTU vs. TUBDAI, Sim 1 132481:3675, Sim 2 176406:10630, Sim 3 115192:37275

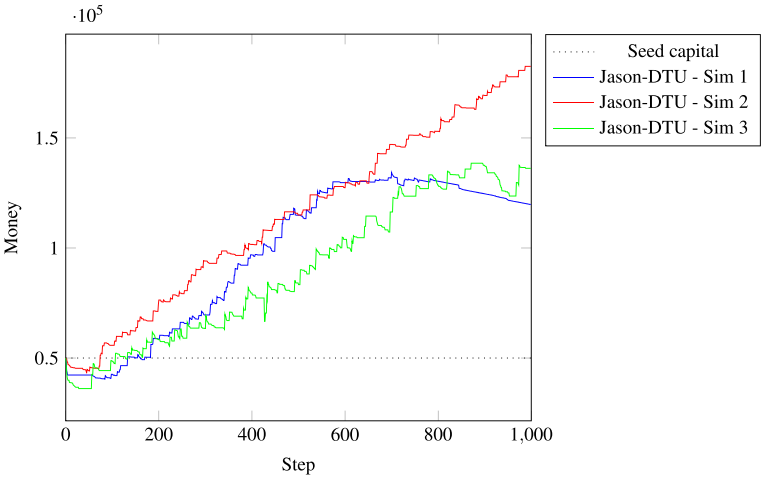

Fig. 2 Results for Jason-DTU in the match with Chameleon. Final scores, Jason-DTU vs. Chameleon, Sim 1 119821:26018, Sim 2 182542:29233, Sim 3 136206:11819

contests that the agents are very active compared to many of the other simulations, where almost all of the actions are based on assembling and assisting each other.

4.1.4 Flisvos vs. Jason-DTU

The match with Flisvos was the most interesting match in the entire competition for Jason-DTU, from which some problems with the Jason-DTU system were exposed. This forced an error in the first of the simulations, making the Jason-DTU system stall, and thereby leading to the only match where the agents were unable to make a profit. However, because of this match, bugs in the system were identified and fixed quickly. It also showed some non-fatal problems with the multi-agent system, where items from failed jobs were not being collected, and existing items were not reused effectively. Hence, a lot of items and the agents' time were wasted.

Table 3 shows that Flisvos was able to complete more jobs than Jason-DTU in general. From the data, the two teams seems to use almost exactly the same time on average

Table 3 The number of jobs completed, the average time to complete a job, and the average reward per job for each team in the match between Flisvos and Jason-DTU

	Sim 1	Sim 2	Sim 3	Sim 1	Sim 2	Sim 3	Sim 1	Sim 2	Sim 3
Team	Completed jobs			Average time			Average reward		
Flisvos	50/60	51/65	47/58	20	31	63	2963	2960	4847
Jason-DTU	10/60	14/65	11/58	20	34	56	2376	3032	5340
	Completed auctions			Average time			Average reward		
Flisvos	0/10	0/23	0/9	–	–	–	–	–	–
Jason-DTU	10/10	23/23	9/9	36	52	57	3950	4624	5583

Final scores, Flisvos vs. Jason-DTU, Sim 1 121141:35837, Sim 2 125639:135220, Sim 3 157291:48352

to complete jobs. However, since only completed and not attempted jobs are included, the data does not necessarily show the whole picture. Only the faster of the two teams completing a job will be counted, hence Flisvos must be faster at completing jobs on average. Jason-DTU was able to complete auction jobs, which the Flisvos system could not. Especially in the second match, with far more auctions available, the two teams were fairly close.

4.1.5 Jason-DTU vs. SMART-JaCaMo

The match against SMART-JaCaMo was very intense, especially after the first simulation had to be replayed because of a server side crash with only 200 steps remaining, and less than 10k in difference between the two teams' scores! The replay of the match turned out to be just as close, unfortunately with a loss for Jason-DTU. The money for all of the matches can be found in Fig. 3.

Overall, the two systems seemed to apply a quite similar strategy, both being able to solve all kinds of jobs. The average time to complete these jobs and the average reward for them also seemed very similar for the two teams, however, Jason-DTU was capable of completing more standard jobs than SMART-JaCaMo, which determined the match. SMART-JaCaMo was able to complete more auctions in all of the simulations, even though Jason-DTU performed far more BIDFORJOB actions. From this match, it was also clear that Jason-DTU's agents were able to retrieve items from failed jobs, which had caused issues in earlier matches. Hence, items were no longer wasted if the other team was able to complete a job faster, which resulted in both faster completion times for future jobs, as well

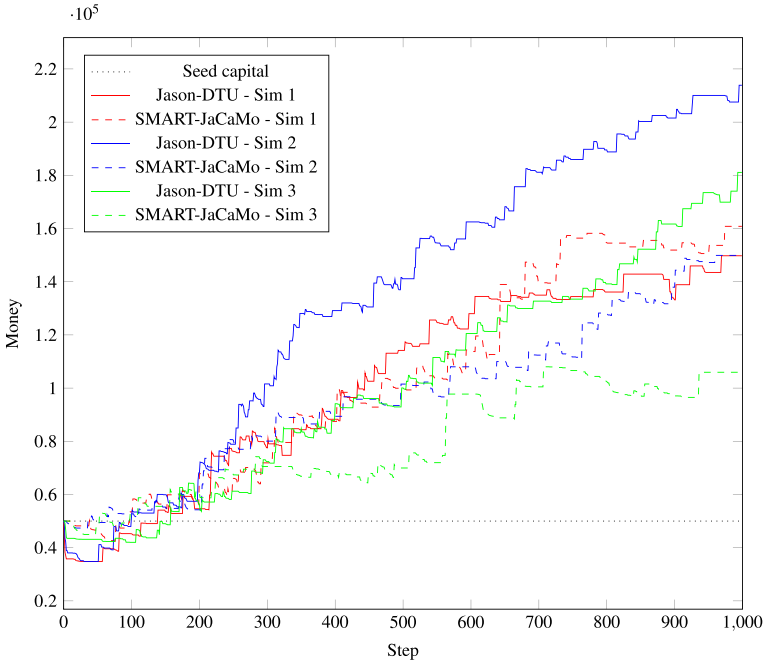

Fig. 3 Overview of Jason-DTU and SMART-JaCaMo's money in the match between them. Final scores, Jason-DTU vs. SMART-JaCaMo, Sim 1 149751:160820, Sim 2 213863:149896, Sim 3 181097:105952

as saved money that would have otherwise been spent on buying new items. An interesting difference to note, is that SMART-JaCaMo was utilizing the storage facilities, as the only team in the competition. This seemed like a very good strategy, reusing items and making them available to every agent on the team, instead of repeatedly buying the same items.

4.1.6 Jason-DTU vs. lampe

The final match for Jason-DTU was against lampe, which had produced varying results during its other matches. The Jason-DTU system preformed very well in this match, having all of its agents active most of the time, thereby completing many jobs. This led to a clear victory in all of the three simulations, along with setting one of the highest scores of the contestants on the last map (final scores, lampe vs. Jason-DTU, Sim 1 53628:157063, Sim 2 85659:139869, Sim 3 100880:225755). Note that lampe earned money in all three simulations, but simply at a slower rate. The system was able to complete both standard jobs and auctions, but did not win nearly as many auctions as Jason-DTU.

4.2 Summary of analysis

As the results from the matches show, Jason-DTU performs well in general. The contest showed some of the strengths and weaknesses in Jason-DTU that did not show in testing. In this section, we summarize the analysis of the matches by providing an overview of overall strengths and weaknesses.

4.2.1 System robustness

Outside the match with Flisvos, Jason-DTU was very stable and was able to make money. Even in the match against BusyBeaver, where Jason-DTU could not complete jobs fast enough, Jason-DTU completed jobs and was able to make money. Part of this was due to Jason-DTU's ability to retrieve items from failed jobs and thus saving money. This shows that the system is robust.

4.2.2 System performance

Jason-DTU earned a shared second place in the competition and as Table 4 shows, it generally earned more money than the opponents. In most matches, Jason-DTU completed

Table 4 Average differences in system performance between Jason-DTU and the other teams, and the overall average performance difference

Opponent	Money difference	Job %	Auction %	Job time difference	Auction time difference
BusyBeaver	−50364.67	26.08	100.00	−35.67	–
TUBDAI	133703.00	93.52	72.11	20.33	10.50
Chameleon	123833.00	100.00	100.00	–	–
Flisvos	−61554.00	18.95	100.00	1.33	–
SMART-JaCaMo	42681.00	73.05	37.75	−4.67	2.33
lampe	94173.33	61.81	91.53	−19.00	−1.50
Overall	47078.61	62.3	83.56	−7.53	3.78

more jobs than the opponent, and it consistently completed auctions in all matches, which provided an additional income. This shows that the system generally performs well.

4.2.3 Customization

The overhead of implementing new strategies is low, which means that it was easy to implement new strategies into the system. During development, Jason-DTU was frequently tested by playing against itself with different strategies to identify strengths and weaknesses in strategies. The use of CArtAgO also made it easy to introduce new artifacts and experiment with different testing setups. This shows that the system is customizable.

4.2.4 Weaknesses in strategy

The just-in-time strategy of only buying and assembling items as needed was slower than investing in many items early on to be ready for new jobs.

4.2.5 Static strategy

As the competition shows, exploiting the weaknesses in the just-in-time strategy can be critical to winning. In complex games such as MAPC, where there are rarely dominant strategies, it is necessary to make the system able to recognize the exploitation early and dynamically switch strategy. However in Jason-DTU, the strategy is static with no self-adjustments.

5 Conclusion

We have presented the overall strategy of the multi-agent system of Jason-DTU which achieved a shared second place in MAPC 2017. A central planner prioritizes and delegates jobs to groups of agents. One agent in the group is then assigned as "leader" whom is responsible for assembling and delivering the required items. All other agents in the group assists the leader with assembling.

We have provided an overview of implementation details including perception, task handling and agent logic. We have implemented the system in Jason and have used the CArtAgO framework to make the agents able to perceive their surroundings in the form of beliefs. It is also used to implement agent communication in task handling. The agents use hierarchical plans to compute their assigned jobs quickly and protocols to synchronize actions with each other.

CArtAgO was initially used to decentralize the task delegation, using the Contract Net Protocol. As the complexity of the scenario increased by introducing tools and more constrained roles, our system leaned toward a more centralized approach. As a result, the role of CArtAgO in the system became less crucial and was mostly used for handling percepts and dividing the environment into several optionally observable sub-environments, separating agent-specific percepts.

CArtAgO allowed for easily extending agent functionality by defining various operations, however CArtAgO operations do not rely on unification, and could thus not be used in AgentSpeak rules. Furthermore, internal operations executed by CArtAgO artifacts run asynchronously, which introduces race conditions when agents execute operations to access

data the artifact is updating. To avoid these problems, all operations were implemented as Jason internal actions instead.

We have analyzed the results of the competition and evaluated strengths and weaknesses of the multi-agent system. Overall the system performed very well in the competition and the goal of developing a highly sophisticated multi-agent system has been achieved. Fitting solutions to key problems have been devised and implemented in practice, emphasizing a high level of abstraction utilizing problem decomposition and agent coordination.

Acknowledgements We are affiliated with DTU Compute, short for Department of Applied Mathematics and Computer Science, Technical University of Denmark (DTU), and located in the greater Copenhagen area. We are grateful to Innovation Fund Denmark for partially funding John Bruntse Larsen's Industrial PhD project *Hospital Planning with Multi-Agent Goals* between PDC A/S and DTU Compute.

Appendix A: Team overview: short answers

A.1 Participants and their background

What was your motivation to participate in the contest?
We have a general interest in artificial intelligence and logic, and in particular in multi-agent systems.

What is the history of your group? (course project, thesis, . . .)
Our group consist of two master students, Helge and Oliver, who together did their bachelor thesis on multi-agent systems, using the contest as the platform for the implementation, and with Jørgen and John as supervisors.

What is your field of research? Which work therein is related? Helge and Oliver have taken courses in multi-agent systems theory and development as part of their bachelor degree. In his PhD, John is doing research on applying multi-agent systems theory for hospital decision support systems. Jørgen's field of research include formal logic and multi-agent systems. He has supervised DTU's team for the contest since 2009.

A.2 The cold hard facts

How much time did you invest in the contest (for programming, organizing your group, other)?
We invested about 300 man hours, including 180 hours of programming for two people.

How many lines of code did you produce for your final agent team?
We produced 5361 total lines of code. Of these, the project included 282 lines of Jason code, while the remaining lines were Java code.

How many people were involved?
Two people have been responsible for implementing the project, with guidance from their supervisors.

When did you start working on your agents?
Helge and Oliver started implementing the agents doing their bachelor thesis in the spring of 2017. These agents were based on the contest from 2016. Most of the work done for the agents used in the actual competition was done in August and September 2017.

A.3 Strategies and details

What is the main strategy of your agent team?

The main strategy is to delegate each job to a group of agents. Each agent will be responsible for collecting some of the items, go to a workshop and assemble them if necessary, and have the leader of the job deliver them.

If the job is too big to be completed by one single group, the job is split into multiple smaller jobs, which can be completed in parallel.

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

Information about the facilities in the world are shared between all agents. This also include information about what items there are located at the different facilities, along with what items the agents are planning to buy.

When new jobs are received, the system estimates which resources (amount and type of agents, along with items) that are necessary to complete the job.

Afterwards, the job is delegated to the free agents of the required types. One agent is assigned "leader", whom is responsible for assembling and delivering the job. All other agents on the job will assist the leader with assembling.

What are critical components of your team?

Some of the critical components of our team are the job delegation and synchronizing actions.

Can your agents change their behavior during runtime? If so, what triggers the changes?

The agents follow the same behavior throughout a simulation.

Did you have to make changes to the team during the contest?

We fixed a bug that caused the system to crash between simulations. During the contest we also found a bug, causing our agents to not always reuse items from failed jobs, which was fixed as well.

How do you organize your agents? Do you use e.g. hierarchies? Is your organization implicit or explicit?

All agents are the same at the start. We do not use hierarchies. Groups of agents are assigned to jobs, where one agent in each group is the leader and responsible for completing the job.

Is most of your agents' behavior emergent on an individual or team level?

Most of the agents' behavior is on their individual level, and only assembling items require team actions.

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

Our agents use hierarchical planning, using as abstract plans as possible. As such, no concrete sequence of actions is chosen for future steps, however, the agents would be committed to their abstract plans for many steps into the future. This could wary a lot, based on the difficulty of the task, but we would assume planning for 100 or more steps would be very unlikely.

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

The agents perceive the latest information from the server, and deliberate this asynchronously. Whenever the agents have finished and found the actions they want to take, this is sent to the server. As the system had trouble computing this within the time limit for each step, any further synchronization protocol was not implemented.

A.4 Scenario specifics

How do your agents decide which jobs to complete?
Jobs are evaluated, estimating steps and agents required to complete the job. The jobs are then sorted according to profit per step, and a delegation algorithm attempts to divide the job into tasks, assigning the tasks to the available agents.

Do you have different strategies for the different roles?
Jobs are delegated depending on the agents' capacity and usable tools, hence the strategy remains the same across all roles.

Do your agents form ad-hoc teams for each job?
Agents solve tasks individually, although tasks involving assembly may require multiple agents, thus in a sense working as a team.

What do your agents do when they do not pursue any job?
Free agents charge, gather resources, and after doing so, they go to a random location within the center of the city.

How did you go about debugging your system?
The Java part of our system has been debugged using the standard Eclipse debugging tools, and the AgentSpeak part has been analyzed using the debugging tools provided by the Jason MAS Console.

What were prominent questions you would have asked your system during development? (i.e. "why did you just do X?")
Prominent questions include: Why did agents sometimes try to complete unsolvable jobs? In what parts of the system were bottlenecks likely to occur? How much time do the agents spend on solving jobs vs waiting for other agents?

A.5 And the moral of it is ...

What did you learn from participating in the contest?
Among other, we gained experience with working with CArtAgO and identified missing features that we found useful; gained experience with optimizing Jason programs; and learned that there is a huge difference between running simulations and competing in the actual contest.

What are the strong and weak points of your team?
We are able to solve multiple jobs simultaneously, utilizing all available agents and resources. Our system did not perform as well when there was a lack of jobs and not being able to utilize our full potential.

How viable were your chosen programming language, methodology, tools, and algorithms?
Jason was an excellent platform for developing a large-scale multi-agent system, allowing us to implement succinct agent logic in AgentSpeak. We also integrated key features of CArtAgO, although keeping the use to a minimum since the framework would quickly overcomplicate simple tasks.

Did you encounter new problems during the contest?
We encountered some problems with delayed perception, thus forcing us to revise the perceive-think-act cycle. We also had to shift our priorities in terms of gameplay.

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

Playing against as well as watching other teams compete gave a lot of new insight, especially in terms of overall strategies.

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

Our goal has been to develop a highly sophisticated multi-agent system, thus focusing on problem decomposition and agent coordination instead of focusing on gameplay and applying game theory.

Which aspect of your team cost you the most time?

The most challenging and time consuming aspect of the system has been to implement an algorithm to efficiently decompose tasks and assign them optimally.

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

We would like more emphasis on flexibility and scalability, possibly by running simulations with a varying amount of steps; utilizing the different roles for different purposes, possibly by restricting the number of partial deliveries allowed for a job; and agent coordination, possibly by limiting the amount of agents staying at the same location.

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

We were able to perform very well due to efficient problem decomposition, using abstract plans and being able to quickly adapt to new information. This allowed our agents to dynamically work together and divide tasks between them.

We had a bug the first two days, which in turn affected our score, however our results improved significantly after resolving it.

References

1. Bordini, R.H., Hübner, J.F., Wooldridge, M.: Programming Multi-Agent Systems in AgentSpeak Using Jason. Wiley, New York (2007)
2. Boss, N.S., Jensen, A.S., Villadsen, J.: Building multi-agent systems using Jason. Ann. Math. Artif. Intell. **59**, 373–388 (2010)
3. Ettienne, M.B., Vester, S., Villadsen, J.: Implementing a multi-agent system in Python with an auction-based agreement approach. Lect. Notes Comput. Sci. **7217**, 185–196 (2012)
4. Hatteland, H., Fleckenstein, O.: Multi-Agent Systems. Bachelor thesis, Technical University of Denmark (2017)
5. Ricci, A., Piunti, M., Viroli, M.: Environment programming in multi-agent systems: an artifact-based perspective. Auton. Agent. Multi-Agent Syst. **23**(2), 158–192 (2011)
6. Vester, S., Boss, N.S., Jensen, A.S., Villadsen, J.: Improving multi-agent systems using Jason. Ann. Math. Artif. Intell. **61**, 297–307 (2011)
7. Villadsen, J., From, A.H., Jacobi, S., Larsen, N.N.: Multi-agent programming contest 2016 — the python-DTU team. Int. J. Agent-Oriented Softw. Eng. **6**(1), 86–100 (2018)
8. Villadsen, J., Jensen, A.S., Christensen, N.C., Hess, A.V., Johnsen, J.B., Woller, Ø.G., Ørum, P.B.: Engineering a multi-agent system in GOAL. Lect. Notes Comput. Sci. **8245**, 329–338 (2013)
9. Villadsen, J., Jensen, A.S., Ettienne, M.B., Vester, S., Andersen, K.B., Frøsig, A.: Reimplementing a multi-agent system in Python. Lect. Notes Comput. Sci. **7837**, 205–216 (2013)