ROS Hybrid Behaviour Planner: Behaviour Hierarchies and Self-organisation in the Multi-Agent Programming Contest

TUBDAI Team Description Multi-Agent Programming Contest 2018

Christopher-Eyk Hrabia(✉), Michael Franz Ettlinger, and Axel Hessler

Faculty of Electrical Engineering and Computer Science,
DAI-Labor, Technische Universität Berlin,
Ernst-Reuter-Platz 7, 10587 Berlin, Germany
christopher-eyk.hrabia@dai-labor.de

Abstract. While the decision-making and planning framework ROS Hybrid Behaviour Planner (RHBP) has been used in a wide variety of projects, newer features have not yet been tested in complex scenarios. One of those features allows creating multiple independent levels of decision-making by encapsulating a separate behaviour network into behaviours. Another one is an extension for implicit coordination through self-organisation. This paper discusses our system that was developed for the multi-agent contest 2018 using RHBP, while especially making use of newer features wherever possible. Our team TUBDAI achieved the shared top spot in the contest, showing that RHBP and in particular the new features can be used successfully in a complex scenario and measures up to the multi-agent frameworks, other teams have used. Especially, when a last-minute change to the contest environment required us to integrate substantial strategy changes in last-minute, it turned out that RHBP fostered adaptiveness during our development.

Keywords: Artificial Intelligence · Autonomous systems · Multi-agent programming · Decision-making · Planning · Self-organisation

1 Introduction

The multi-agent programming contest (MAPC) provides a testbed for evaluating multi-agent research results in an applied and competitive setting since many years. Participating in the contest has long tradition at Technische Universität Berlin (TUB) (e.g. [4–6]). The motivation for participation in the contest was always twofold. First, to use it as a platform to evaluate our multi-agent frameworks in complex multi-agent problems of the contest. Secondly, to use the competition setting as a platform for introducing our research to new users like students, which apply our frameworks either in their thesis projects or project courses. In 2017, we have introduced the framework ROS Hybrid Behaviour Planner (RHBP) [11] for the realisation of our contest team [10], which has its roots in the robotics domain.

In particular, RHBP is applied for the implementation of the individual task-level decision-making and planning of the agents as well as the coordination amongst the agents. RHBP is targeting the multi-robot domain and is based on the Robot Operating System (ROS) [17] framework that provides means for deployment, decentralised execution, and communication. The RHBP combines the advantages of reactive opportunistic decision-making and goal-oriented proactive planning in a modular hybrid architecture. Decision-making is based on behaviour networks [14] that allow for dynamic state transitions and the definition of goals, while the deliberative part is realised on the foundation of the Planning Domain Description Language (PDDL) [15] and the particular planner Metric-FF [7].

The reason for evaluating RHBP in a simulated comparable abstract multi-agent scenario instead to real robot applications is that real robot applications require to address a huge overhead of other domain-specific challenges, such as hardware failures, very uncertain environments, and difficulties in basic robotic capabilities like object detection and localisation [8]. Due to the reason that RHBP is a generic framework for decision-making, planning and coordination of multi-robot systems the evaluation in the MAPC, which focuses on task-level agent control, allows us to concentrate on these research aspects.

Since 2016 the contest scenario has been using the discrete and distributed last-mile delivery simulation (MASSim) [2] on top of geographic map data from different real cities (*OpenStreetMap* data). The simulation allows competition of several teams consisting of independent agents. Delivery jobs are randomly generated and split into three categories: *Mission jobs* are compulsorily assigned, *auction jobs* are assigned by prior auction and *regular jobs* are open to everyone. Jobs are monetarily rewarded on fulfilment and can only be accomplished once. Moreover, jobs consist of several items which can be purchased at shops (2016–2017) or gathered in resource nodes (2017–2018), as well as stored in warehouses. Furthermore, the 2018 edition of the contest scenario was extended with the obligation of building wells to generate score points required to win matches. Building wells required money that is earned by completing delivery jobs or selling resource items in a shop. The well-building extension in 2018 fosters more interaction and direct competition between the teams aside from increasing the overall search space for finding the most optimal solution. Moreover, the number of used agents per team was increased from 28 to 34 agents.

In the 2018 participation, the goal of our team was to evaluate more recent features of RHBP that have not been tested before in a complex application. Team TUBDAI focuses on two features in particular. First, so-called *Network-Behaviours* that allow creating multiple independent levels of decision-making by encapsulating a separate behaviour network into behaviours of the behaviour network model of RHBP. Secondly, the extension *so_data* for implicit coordination through self-organisation [9]. The extension *so_data* incorporates the concept of a virtual gradient space as a common data structure and communication mean for various self-organisation patterns. Additionally, the *so_data* package already contains several abstract implementations of self-organisation patterns, while the particular integration with RHBP is enabled through the package *rhbp_selforga*, which contains special sensor and behaviour components.

The participants of the 2018 edition of the contest consisted of five international teams, with two independent participations from TUB, namely *TUBDAI* and *Dumping to Gather*. Both teams from TUB are applying a RHBP-based implementation as a follow-up of the introduction of RHBP in the 2017 contest [10]. However, both teams did only share the starting point with general components developed in the year before, like the protocol proxy mac_ros_bridge and the integration of the routing library GraphHopper. Despite these general components, both teams developed their solution completely from scratch, which resulted in two very different general strategies. Both teams have been supported by the technical supervision of the first author of this article. Nevertheless, strategic decisions or implementations were never communicated or shared between both teams. The resulting implementations are both highly decentralised with all agents taking operational decisions autonomously using RHBP behaviour models. Only the evaluation of published delivery jobs is done in centralised components in both cases. Moreover, both teams apply an own contract-net based implementation for the coordination of the assembly and delivery for fulfilling the jobs.

The remainder of the paper is structured as follows. In the Sect. 2, we analyse and summarise the particular challenges of the MAPC 2018 for our team. Next, Sect. 3 outlines and describes our general team strategy, whereas Sects. 4 and 5 provide details about our implemented architecture and coordination approach. Subsequently, Sect. 6 describes the behaviour model we have implemented for the autonomous decision-making of our agents making use of our RHBP framework. In Sect. 7 we describe and discuss our contest results based on statistics and observations we made in the individual matches. Finally, Sect. 8 concludes this work with a summary of the contributions as well as emphasising future steps we plan to address.

2 MAPC 2018 Challenges

The complex contest scenario offers a comprehensive environment to design, implement and evaluate a multi-agent system. This results in some unique challenges which are covered in this section.

In the last two years, cooperation between agents became a bigger focus of the contest. While in preceding scenarios, it was possible to develop viable solutions without cooperation between agents by letting each agent independently work on parts of the problem, now there are key actions that require cooperation between multiple agents [1]. In this year, jobs only use items which need to be assembled first, requiring the implementation of complex cooperation and communication between agents.

Some environmental parameters are generated randomly, which results in much variation between simulations, thus making it harder to develop a solution that can work well with different configurations. The number of possible finished products to build can vary quite substantially between simulations. Strategies like proactively assembling and hoarding items work best with a few finished products, while a higher number of finished products makes just-in-time gathering, assembly and delivery more efficient. The maps differ in size and street layout, therefore changing the necessary effort for discovery as well as the effectiveness of different agent roles. These and other differences in the unknown contest configuration make it harder for teams to develop solutions that work well in all cases.

Following [12] the characteristics of multi-agent systems are having incomplete information or capabilities, no global system control, decentralised data, and asynchronous computation. In that spirit, a primary goal of participating in a multi-agent contest should be to develop a decentralised solution. Nevertheless, some previous submissions had shared data structures between agents and used a central planner, which decided on actions for all agents. The individual agents were then only responsible for making sure that enough battery is available to perform those predefined actions. The benefit of such systems is that they are easier to implement, as not much effort has to be put into agent communication and coordination. Furthermore, the effectiveness of such systems can be observed for example with last year's winning submission by BusyBeaver [16]. However, our goal was to decentralise as many decisions as possible, letting each agent autonomously decide what to do next, except on those few cases where this is impossible (e.g. job execution). Moreover, this decentralised approach allows that each agent could potentially be run on a different machine.

Another challenge is the large number of possible options for strategy decisions. The diversity in facilities and actions allow for many different strategies. It would take a massive development effort to make use of all available facilities and actions, so it is necessary to evaluate options and implement only the most promising ones.

During each simulation, a team has to compete against another team on the same map, which results in unique challenges. Regular job rewards are only awarded to the team, who can perform the job faster, therefore making job delivery speed a major design goal. The competitive setting also requires weighting actions of increasing one's own score with actions to decrease the opponents' score. Also, because of the broad spectrum of available strategies, it is harder to design a system, that performs well against varying opposing strategies.

One limitation imposed by the contest is that actions have to be submitted at latest four seconds after a step percept is published. Due to the limited timeframe, it is hard to find the optimal solution and a compromise has to be found between an ideal decision-making process and one, that can be finished quickly.

3 Strategy

The main strategy of the TUBDAI implementation is a stockpile with feedback strategy. Here, it is the goal to gather resources, assemble items in advance, and fulfil jobs that can be performed with the available item stock. First, the agents explore the environment until a resource node for each base item is discovered. Then all agents start to gather resources to achieve a stock of base items depending on the assigned priority for further assembly. Agent groups are formed dynamically to assemble items once the agent capacity is exceeded. The jobs are then prioritised according to a calculated reward. The reward is given by a ratio of required work and revenue. If a job reaches the defined reward threshold and all items of the job are currently in stock, the job is executed. Furthermore, urgent item demands such as induced by mission jobs, which would result in fines if not fulfilled, are also considered on demand based on feedback from other higher-level components like the job planner by dynamically adjusting the priorities of the respective items. The feedback allows reacting and changing the priorities of finished products and base items, which results in better adaptability to changes on demand, while still maintaining the efficiency of a general stockpile strategy. The advantage of this stockpile with feedback strategy is a fast job performance, enabling decentralised decision-making by individual agents to avoid a single point of failure, while also enabling on-demand execution based on the priority feedback. A disadvantage is that assembled finished products may not match any job and cannot be used, hence potentially wasting time resources.

Even though we aimed for a decentralised solution, the rating of jobs is centralised in one agent for simplification because there is no difference in between the job perception amongst the agents and the cost-benefit analysis is as well agent independent. The distribution of task from the decomposed jobs is then realised using a contract net protocol [19] involving all agents in a decentralised and distributed fashion. All other components are completely decentralised, too. Each agent has its own RHBP-based decision-making and planning component.

For the execution of jobs, an algorithm involving a chain of decisions has been developed. At the end of the chain, the scenario-specific job planner inspects which items currently provide high money returns. This information is converted into a finished product prioritisation. This prioritisation is then used by the next link in the decision chain, the assembly decision. The agents always decide autonomously which items should be assembled next and share their decision with the others. This decision is mainly based on what is needed for job execution as well as on the available items, and the finished products that are already assembled. In turn, this decision creates a prioritisation of base items that are most needed for assembly. These prioritisations are then used by the gathering algorithm to decide which base item to gather. An important point is that the taken decisions are exchanged amongst the agents to avoid conflicts and unwanted parallel work.

One advantage of the stockpiling strategy is that it creates job idle time, respectively agents available for dismantling, building, and exploration because not all agents are always striving for fulfilling the currently available jobs. Particularly, this additional freedom for individual context-specific decisions of the agents is fostering the adaptation and reaction to varying opponent strategies.

The exploration of the environment is implemented with the so_data library of RHBP in a decentralised and self-organised manner. The framework extension of RHBP containing so_data has been introduced in [9] and enables implicit coordination through various self-organisation patterns. Self-organisation patterns are reusable design patterns abstracted from specific self-organisation algorithms, respectively self-organisation mechanisms [3]. The patterns are classified in movement patterns, decision patterns, and basic functionality patterns.

Basic Functionality Patterns are represented by gradients. Gradients are subject to spreading, aggregation and evaporation. The exchanged gradient messages (*SoMessages*) contain the position where they were emitted, as well as other information and metadata, necessary for advanced patterns to base their calculations on. Movement Patterns are used to control movement of the agents to allow for implementation of behaviours like foraging, chemotaxis, and exploration. Decision patterns allow agents to make collective decisions. The framework provides samples of these behaviours including Quorum Sensing, Morphogenesis and Gossip.

In our MAPC implementation, each agent emits its own location with *SoMessages*, so others do not explore these points as well, while selecting a target location close-by that has not been visited for the longest time. The exchanged SoMessages are filtered in a decentralised manner in each agent by the SoBuffer module instance of the so_data library, which provides the base for the calculation of a discrete heat map. Likewise, the heat map is used to select the appropriate exploration target locations, while the initial exploration phase is stopped after resource nodes for all available base items have been discovered. Later during the match, one drone agent is also exclusively patrolling the map border to discover opponent wells, which would be difficult to discover accidentally during normal job operation.

The general idea behind the well-building is to build wells at locations that are difficult to discover or difficult to reach by the agents of the other team to neglect the requirement of an explicit well defence strategy. The introduction of well building in MAPC 2018 further emphasised the role differences between the agents. Particularly, some locations are only reachable by drones (e.g. some parks, rivers) because there are no close-by roads to allow other agent roles approaching these locations. These locations make great spots for wells as only drones can reach them, so the opponent would need to use drones for dismantling. This strategy has the advantage that not all opponent agents are able to attack our wells as well as that drones in general are very inefficient for dismantling purposes, although this is at the same time a trade-off for the well building because our drones are also inefficient in building up wells. Nevertheless, the strategy has the potential to confuse the opponent because it either needs special consideration or advanced adaptation capabilities.

Dismantling opponent wells keeps the opponent from gaining score points and it allows to gain additional money for further construction of one's own wells.

After an opposing well is discovered, the dismantling behaviour is executed by all agents that are not busy building wells themselves. In particular the agents are prioritising the closest near-by wells for dismantling. The dismantling itself is not further coordinated amongst the agents thus combined or split attacks by groups of agents emerge from the situation.

4 Architecture

The agent architecture of TUBDAI is based on the 2017 approach but replaces big parts of the ROS-topic-based communication in the perception with custom information provider modules, which are feeding the information more directly into RHBP sensors. The information providers avoid some communication overhead coming from ROS communication to increase the efficiency of the implementation. Furthermore, the TUBDAI implementation shares information distributively with the self-organisation library so_data of RHBP.

Figure 1 shows the general architecture as well as how the individual components communicate. The MASSim contest server is running the actual contest simulation and sending the percepts to all agents as well as receiving chosen actions using an XML-based protocol. Each *mac_ros_bridge* receives percepts for one agent, converts them into ROS messages and publishes them into the ROS runtime environment, respectively communication space. Additionally, the bridge takes action messages from the agent and converts them back into an XML format to be passed to the contest server. Each agent in the simulation requires a dedicated *mac_ros_bridge* and one *RHBP Agent*. *RHBP Agent* is a ROS node and is responsible for receiving environment information from the *mac_ros_bridge*, deciding for the best action and communicating the decision to the bridge. Hence, a *RHBP Agent* follows the sense-think-act paradigm within the ROS runtime environment. Each *RHBP Agent* consists of several components, including RHBP components like Behaviours, Sensors, Effects, Conditions and Goals. Additionally, we make use of the *DecisionPattern* from the self-organisation extensions. Particularly, we also use the pattern not only in the way the framework intended it but extensively for all kinds of decisions that do not necessarily involve self-organisation. Basically, this pattern allows to easily share further calculated sensor information between behaviour objects and decision objects. Thus, a certain calculation can be used twice, first, for the task-level decision with RHBP, secondly, for the particular implementation of a behaviour. The design pattern that was used to implement *DecisionPattern* was found to be very useful in many situations and therefore was reused for other components. The *DecisionPattern* is not visualised in the architecture diagram, it is used within the particular behaviour implementations or condition objects.

Providers are responsible for most interaction between *RHBP Agent* and other components like *mac_ros_bridge* and the graphhopper node. They subscribe to messages from the *mac_ros_bridge*, pre-process them and keep them ready for various components to use. This reduces the number of subscribers, and also improves performance by eliminating duplicate code execution that would be necessary if each component would subscribe individually. Providers can also send actions to the *mac_ros_bridge* and interact with the graphhopper node.

The GraphHopper node is responsible for calculating distances between given locations. This allows other components to estimate the steps required to achieve their goals.

The manager component of RHBP retrieves the status of all RHBP components and is responsible for the task-level decision-making that leads to the execution of a behaviour at the end. The chosen behaviour then emits an action through the Action Provider to the *mac_ros_bridge*, which sends it to the MASSim server.

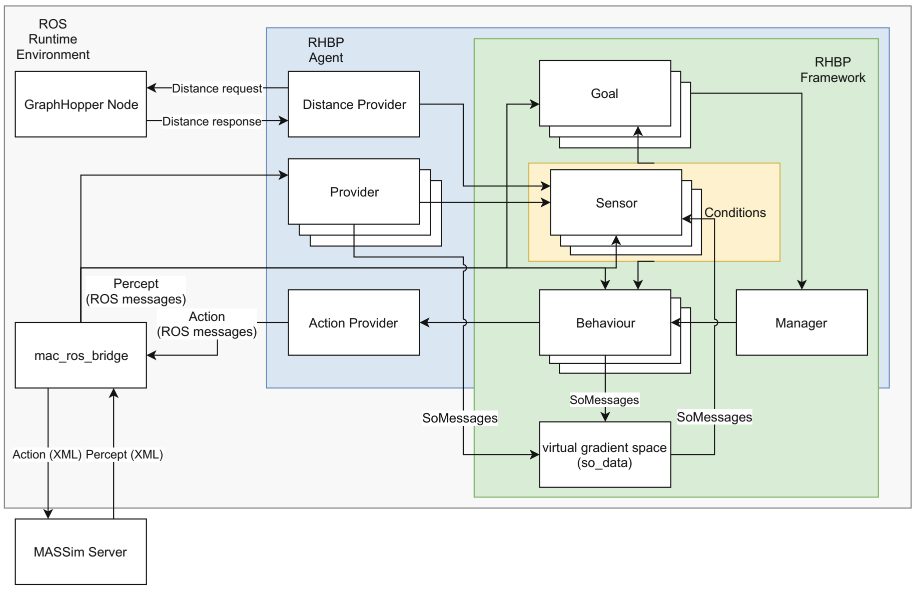

Fig. 1. Component communication diagram showing how information flows between components. The architecture is simplified to only show components that interact with others.

5 Coordination

In this section we are providing more details about our different approaches for coordination within our implementation. First, we explain the explicit coordination with a contract net that is used to coordinate the job fulfilment. Secondly, we discuss details of the implicit coordination for the self-organised exploration and information gathering about opponent well locations.

5.1 Contract Net

A Contract Net with Confirmation Protocol (CNCP) [18] is used to coordinate assembly and job coordination explicitly. This light-weight flexible and fast protocol provides scalability, robustness against errors, adaptivity and few communication bottlenecks [13]. For both of these tasks, one agent initiates the coordination by starting an auction, not to be confused by the auction jobs of the simulation, and requests help from other agents. Next, other agents that are able to help send bids. Our implementation is limited to one coordination taking place at a time. If another agent has already initiated coordination, the agent has to wait until the current coordination cycle is complete before starting its coordination.

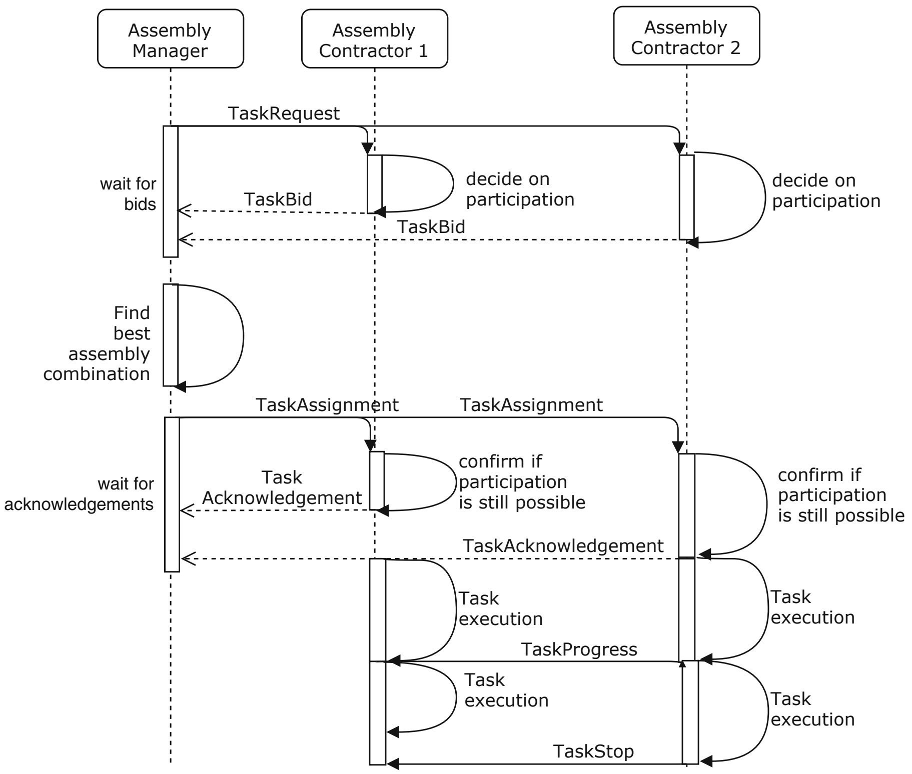

Fig. 2. Interaction diagram of assembly with one initiating manager and two participating contractors.

When an agent has filled up its stock, it initiates the assembly process by starting the assembly manager (see Fig. 2). The manager requests all other agents to send bids to help with assembly. If another agent wants to coordinate, it first has to wait until the current coordination cycle is finished. Other agents then respond with their bid which includes their location, role, and the items they can offer. Once the manager decides for a combination, it sends assignments to all chosen agents. The agents then return an acknowledgement and the assembly can start. Participating agents coordinate assembly by sending a *TaskProgress* message. At any time, agents can interrupt assembly by sending a *StopTask* message, which forces all participating agents to stop their task. This could happen, when one agent receives a job task, which has a higher priority than assembly tasks.

The job coordination strategy works similar to assembly coordination strategy. It is initiated when a job component decides for a job to execute. The JobManager then sends out an initial request. Each agent that doesn't have an active task checks if they can help with the task and respond with the items they can offer. The manager then looks for a combination of agents that can perform the job and sends out an assignment to all of them. The agents confirm by sending an acknowledgement back and the task is started. Similar to the assembly CNP, the task can be stopped at any time by emitting the *StopTask* message.

5.2 Self-organisation

The self-organisation extension for RHBP allows agents to share information about their environment in a virtual gradient space, which can then be used for implicit coordination [9].

In our implementation agents publish self-organisation messages whenever they move around to let all other agents know which parts of the map they have visited. The receiving agents then aggregate these messages from all agents in a distributed fashion. This allows them to decide which locations require further exploration, which is especially relevant in the initial exploration phase to detect all necessary resource nodes in the beginning of each simulation. This self-organisation exploration algorithm is enhanced further by using two other types of messages. Agents publish the location they plan to go to, so other agents are able to avoid it. This allows for the prevention of exploring certain locations twice at the same time. Moreover, agents publish a message when a location is not reachable, so other agents do not try to go there.

In detail, we are creating a heat map from self-organisation messages of specific frames, indicating either how often each spot has been subject to said frames or the last *SoMessage* that has been recorded at each position. The heat map is realised using a grid of numbers instead of a vector-based system. This reduces the accuracy but greatly simplifies later calculations. Initially, the decision pattern creates a two-dimensional array filled with zeros, which represents the map. Whenever a *SoMessage* arrives, that matches one of the desired frames, the map is updated. The location of the *SoMessage* is converted to a mask. The mask is then applied to the map. *MapDecision* has two modes, *oldest_visited* and *seen_count* Depending on the mode, the mask is applied to the existing map to result in a heat map indicating how often a location has been visited, or when each location has been visited last.

The other way self-organisation is used for coordination is for agents to check if an opponent well has successfully been dismantled. Whenever an agent locates an opponent's well, it informs all other agents about it through SoMessages. Agents combine this information with location information sent by other agents. If at some point, the well is not seen anymore, even when an agent is in range of the well, the agent can be sure, that the well has been successfully dismantled.

6 RHBP Behaviour Model

A RHBP behaviour model allows to describe the relationships between behaviours, goals, and sensors through conditions and effects. These models provide the foundation for the autonomous decision-making executed in the manager component of RHBP.

In contrast to the last-years participation of TUB using RHBP with a single behaviour model layer for decision-making, our implementation partitions the model into various nested behaviour models. This became possible through the recently introduced NetworkBehaviour feature. NetworkBehaviours are frequently used for structuring and controlling the major responsibilities of the agents on the highest decision-making level. Such partitioning fosters a separation of concerns, reuse of code, and a reduction of the decision-space through grouping of certain behaviour options. NetworkBehaviours are a special type of behaviours that directly inherit from the behaviour base class of RHBP. In contrast to normal behaviour implementations in RHBP, the NetworkBehaviours are not directly executing any actions that have an influence on the environment. Instead, NetworkBehaviours are triggering a nested decision-making and planning process to select suitable behaviours from their encapsulated behaviour model to achieve the targeted effects.

In particular, NetworkBehaviours are modelled for controlling resource exploration, discovering opponent wells, dismantling opponent wells, building wells, gathering base items, assembling finished item products, and delivering jobs. All NetworkBehaviour implementations are inheriting from an abstract scenario-specific NetworkBehaviour implementation GoAndDoNetworkBehaviour that incorporates battery management and travelling on the simulated map, which is a basic capability of all higher-level tasks in the MAPC scenario. The high-level decision-making behaviour model is visualised in Fig. 3. It shows the high-level first class entities and their relationships, this in turn describes the agent behaviour declaratively. Here, it has to mentioned that all further implicit dependencies are automatically determined by the system. Hence, there is no direct relationship between behaviours and goals in the shown model.

Considering the fact that this model covers only the highest-level of decision-making with additional nested models within each of the shown NetworkBehaviours a comparison with the model of the previous participation in 2017, please see [10], indicates that the complexity of the 2018 TUBDAI implementation is considerably larger. Overall, we see that the structure of TUBDAI is more fine-grained, e.g. we have distinct NetworkBehaviours for assembly, delivery, and gathering. Likewise, the entire number of behaviour and goal instances is considerably larger but this is partly because of redundant charging goals and behaviours in each NetworkBehaviour. In detail, the old TUBDAI model contained only 5 behaviours and 2 goals, whereas TUBDAI 2018 has 3 goals and 7 NetworkBehaviours each again containing 2 goals and 4–5 behaviours.

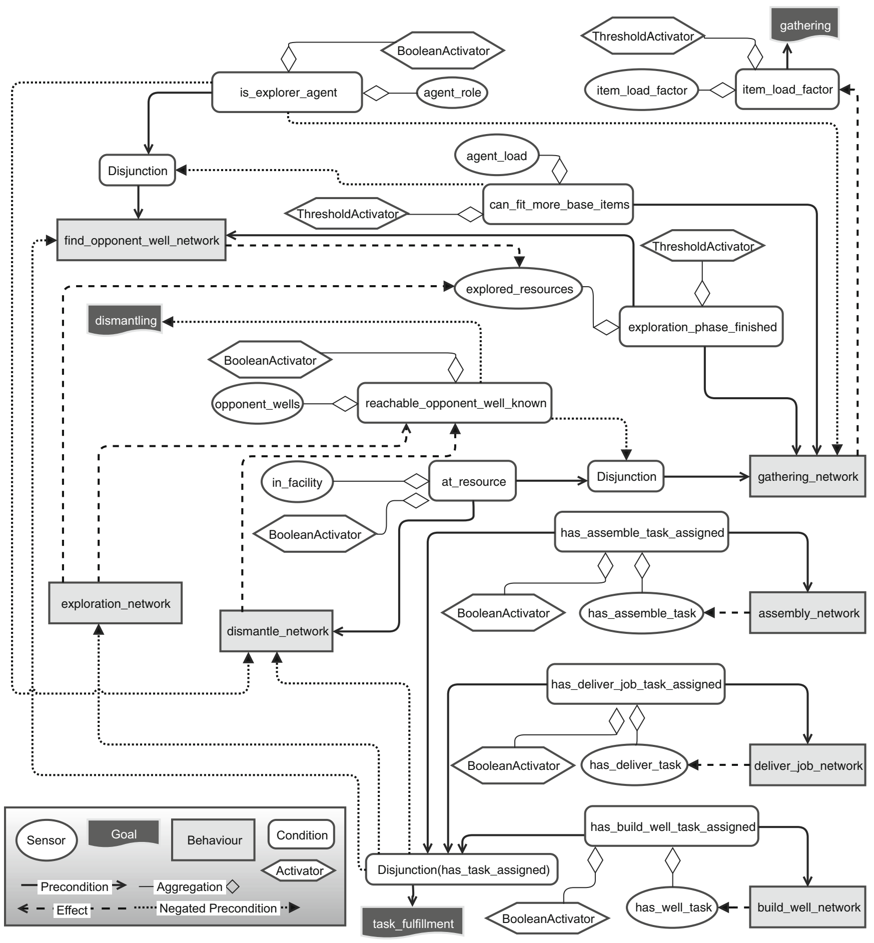

Fig. 3. High-level decision-making behaviour model for agent task execution of TUBDAI 2018. All listed behaviours are NetworkBehaviours containing nested behaviour models. Each NetworkBehaviour contains 2 goals and 4–5 behaviours.

During the realisation of the TUBDAI implementation we discovered a new general implementation pattern for lower-level decisions in sensor, condition, behaviour, and goal implementations. The new implementation pattern is taken from the self-organisation extension of RHBP, which offers a component called DecisionPattern that can be used by certain behaviours and sensors to share low-level decisions between the decision-making layer of RHBP and the actual implementation of the behaviour. This implementation pattern is further generalised and not anymore applied only for self-organisation related low-level decisions. Instead, the DecisionPattern works as an aggregate that takes a low-level decision like selecting the closest charging station, which is then shared amongst the sensors and conditions of behaviours and goals as well as the actual behaviour implementation. This pattern was found to be useful for sharing information between behaviours and sensors of one agent.

7 Evaluation

Four other teams participated in the 2018 contest: SMART_JaCaMo (Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)), Dumping to Gather (TUB), Jason-DTU (Technical University of Denmark) and Akuanduba-UDESC (Santa Catarina State University (UDESC)). Three of these teams were defeated, only one team was able to win against our submission. Table 1 shows main stats that were achieved by each team against TUBDAI. The last column shows an average of these stats for all matches of TUBDAI.

Table 1. Comparison of the performance of all teams in the matches with TUBDAI vs overall average performance of TUBDAI.

Team	Jason-DTU	Akuanduba-UDESC	Dumping to Gather	SMART_-JaCaMo	TUBDAI (avg)
Ranking	3	5	4	1	2
Match points	0	0	0	9	6.75
Tournament points	21	0	9	33	27
Successful jobs	83	0	35	163	79
Opponent jobs	71	127	94	24	70.25
Score after 3 matches	2236	0	2206	2923	45628.75
Opponent score	57619	71928	42935	10033	1841
No actions	15.92%	97.95%	10.40%	8.33%	8.88%
Opponent no action	8.07%	2.51%	6.40%	18.53%	33.15%
Goto actions	58695	993	70730	73039	75648.25
Goto failure rate	30.92%	41.29%	43.27%	0.00%	0.70%
Dismantle actions	0	0	3354	1780	1471
Retrieve & delivered	1078	0	94	2000	0
Build actions	402	0	1336	1371	3604.75
Successful assembly	712	688	293	1350	378
Agent upgrades	0	0	0	14	0

In the following subsections we will briefly summarise the most important observations we made during the matches against each opponent, before we finally discuss our performance and results in more detail. If the reader wants to replicate our observations the source code of all teams, the simulation server, and replays from all matches are available on the official contest homepage[1].

[1] https://multiagentcontest.org/2018/.

7.1 Jason-DTU

Jason-DTU won the second place in the previous edition of the contest and also exhibited a good performing and successful solution in this year's contest.

To finish jobs, Jason-DTU makes excessive use of storages and specialised task groups. Particularly, they use certain road agents for item gathering as well as specialised agent teams for taking care of assembly and delivery. When there is enough massium available, trucks move to the edge of the map and build wells there.

While the team had an aggressive dismantling strategy against other teams, during the match against TUBDAI, the team did not perform any dismantle action. As shown in Table 1, the team had a 30.92% failure rate of the goto action. This likely results from road agents trying to dismantle wells, but not being able to reach them as they were built at off-road locations.

Their job strategy was very effective, due to the centralised workshop and storage being used as intermediary item holder reduced the coordination efforts. As items are stockpiled, the jobs can be performed quickly while also keeping the agent's efficiency on a high level. This led to a slightly better job performance of 83 jobs executed, compared to 79 jobs on average in three simulations for TUBDAI.

7.2 Akuanduba-UDESC

The second opponent in the contest was Akuanduba-UDESC. Due to an error in their system, the team could not send actions in time. This resulted in almost no actions by their agents.

Due to the inaction of this team's agents, their strategy cannot be analysed. However, it allows to evaluate our submission in a very special case. Due to the fact, that effectively no opponent was present, no dismantling was needed. This allowed the agents to shift more resources to job execution.

While the average job execution rate of our team against all other teams was 63 jobs, during the matches against Akuanduba-UDESC, the job execution rate was 127 job (see Table 1). This shows that our agents adapted well to a situation without opponent wells by shifting priorities accordingly.

7.3 Dumping to Gather

Dumping to Gather, the other team from TUB, started with the same basis for their project including the *mac_ros_bridge* and RHBP as framework. However, they followed the different on-demand strategy for job execution. When a job was announced, they coordinated agent teams who were then responsible for the whole chain of actions including gathering, assembly and delivery.

Their approach to build wells was to use multiple agents of all roles at the same time. This allowed them to build up wells almost instantly but had the drawback of multiple agents having to use many steps for moving to the destination location. Wells were built next to each other in a line. We assume this was done to reduce the number of movement steps that had to be taken between building two wells. However, this has the disadvantage that dismantling agents usually immediately after dismantling one well find the next one. This allowed our team to dismantle their wells relatively quickly.

Dismantling was done using both air and road agents. This allowed Dumping to Gather to attack our wells that were placed in off-road locations using drones. However, other agents were not prepared to handle off-road locations and got stuck in an error loop, trying to reach the wells. This led to a goto action failure rate of 43.27% (see Table 1).

7.4 SMART_JaCaMo

The team SMART_JaCaMo was a very strong opponent in multiple regards. First, the team's agents seem to have been divided into different responsibilities, six trucks were used to only build wells, two drones were only used for exploration and dismantling and the rest performing jobs.

Secondly, the team was able to perform 163 jobs, which was substantially more than any other team (see Table 1). The agents who were tasked with job execution, gathered items, assembled them together and delivered them according to the available jobs. They also made use of storing items in storages in order to improve efficiency.

Thirdly, when enough massium was available, a number of truck-agents were responsible for building wells. The locations seemed to have been chosen randomly somewhere close to the edge of the map.

Finally, dismantling was done by two drones. Their skill was upgraded at simulation start, so they can perform dismantling actions efficiently. Afterwards they were only responsible for finding wells and dismantling them. As drones are able to go to off-road locations, this defeated our strategy pretty well. Due to the road-agents not dismantling at all, they also did not suffer from failed goto actions like other teams.

7.5 Discussion

All in all, the last-minute changes of the well-building strategy paid off because this has been a unique strategy, which was not expected by the opponents. Here, this particular strategy becomes especially attractive, as the original well-building strategy has been changed three days before the contest and the RHBP-based architecture supported a quick integration of the new strategy, which did not require comprehensive code changes. Originally, it was the plan to build wells with trucks at the edges of the map area. Unfortunately, this turned out to be less efficient with the final contest maps published three days before the contest. Instead, we shifted to the strategy that making use of so-called off-road locations on the map in order to neglect an explicit well defence strategy. Off-road locations are locations that are not connected to the street network and thus only reachable by drones.

The conducted last-minute changes comprise shifted role responsibilities like only drones building wells; a changed exploration that is not focusing anymore on the map borders; less priority on the job fulfilment because the mandatory and rare drones are often busy with well-building; and a higher priority on dismantling to efficiently use the increased job idle time for non-drone agents.

The encapsulation of code within behaviours and aggregation in NetworkBehaviours made these changes very intuitive and robust. By duplicating certain NetworkBehaviours and switching around preconditions and effects, most of the strategy was adapted, requiring only small code changes within the behaviours. In our opinion this would have been potentially much more difficult to achieve with a traditional sequential programming approach.

The runtime adaptiveness of RHBP could be observed at the match against Akuanduba-UDESC. While many resources were usually used for dismantling opponent wells, there was no dismantling required against Akuanduba-UDESC because of their timeout issues, which resulted in almost 100% inactivity of their agents. This freed up resources for other tasks for our agents. The agents were able to adapt to this unexpected situation and increased their job performance from an average of 63 jobs to 127 jobs. This shows that TUBDAI agents adapted well to a situation without opponent wells by shifting priorities accordingly.

Moreover, the simulation configuration used in the contest was very different from the sample configurations that have been published together with the server source code for the contest preparation. The biggest difference was that it was very easy to gain money for building wells within the contest. While in the sample configurations (which we assumed to be similar to the contest configuration) most jobs offered rewards of less than 500, the jobs in the contest had much higher rewards, , i.e. jobs exceeding 10,000 in reward, whereas building wells stayed on the same price level. In consequence, building wells became easier, and the strategy of building and defending more critical. Nevertheless, the TUBDAI implementation has shown that it was able to adapt and handle this unexpected setup successfully.

In the end, *TUBDAI* only lost the final match against SMART_JaCaMo. The reason was that they efficiently dismantled our non-defended off-road constructed wells exclusively with two of their drones, which have been only responsible for discovering and dismantling of our wells. Moreover, their skill was upgraded directly at simulation start, so they dismantled more efficiently. Furthermore, due to the road-agents not dismantling at all, team SMART_JaCaMo did not suffer from failed goto actions like other teams. A question that might come up at this point is why our RHBP-based approach was not able to adapt automatically to this situation. The reason is that RHBP is only having the opportunity of adaptation if alternative behaviour implementations are available, which was not the case for the TUBDAI implementation.

Nevertheless, a detailed analysis of the replays and the published code of SMART_JaCaMo showed that their unique strategy was also not a result of their adaptive strategy or implementation, but rather a result of their last-minute changes in implementation the human team has made after analysing the matches before the very last match of the competition between TUBDAI and SMART_JaCaMo. We could prove this by playing about 30 simulations with the a priori published simulation sample configurations and the not modified SMART_JaCaMo code in which SMART_JaCaMo was not able to win any simulation against our team. In detail, the SMART_JaCaMo team added a second drone for exploration, implemented immediate skill upgrade after simulation start, disabled dismantling in trucks, enabled dismantling for exploration drones, and created a second drone exploration algorithm, that targets locations that are typically used by our agents to build wells. All these changes were made in short time-frame while we were competing against the other three teams. The changes seemed to be very robust and side-effect free, which is impressive for such a substantial last-minute change. Nevertheless, it has to be stated that the SMART_JaCaMo approach did not follow the rules of the competition because teams are encouraged to refrain from code changes during the contest that are not pure bug fixes of their own strategy. This fact also leads to an official correction of the final placement by the steering committee of the competition resulting in a shared top spot between TUBDAI and team SMART_JaCaMo[2].

8 Conclusion

In the presented article we described our successful solution for the MAPC 2018 that allowed us to win the shared top spot of the competition. Our solution enabled us to address the described challenges of the contest. The required coordination is achieved by a combination of explicit coordination based on a contract net protocol, and implicit coordination on the foundation of the RHBP self-organisation extension. Here, both coordination approaches are also supporting a decentralised solution. Adapting to varying environments and situations was possible through the application of our framework RHBP that fosters a separation of concerns of agent capabilities, which are then used for autonomous decision-making. Moreover, this autonomous decision-making enabled our system to quickly react on different opponent behaviours. The given computational constraints and requirement to handle an increased number of agents in comparison to the previous year have also been addressed successfully.

All in all, we could show that a multi-agent system developed on the foundation of our RHBP framework is able to compete with other multi-agent approaches even though it is actually targeting the different application domain of multi-robot system. Particularly, using RHBP showed to be advantageous especially in terms of adaptation capabilities during development as well as in runtime of the system. Furthermore, our focus in 2018 on testing in practise the recently introduced RHBP features for creating behaviour model hierarchies by nesting and encapsulating behaviours and goals within other behaviours as well as realising implicit coordination through sharing and filtering information with the support of our self-organisation extension turned out to be beneficial.

[2] https://multiagentcontest.org/2019/01/23/results.html.

For the future, we would like to further explore the challenge of selecting the most appropriate high-level strategy like on-demand job completion or stockpiling in such a complex scenario. So far the high-level strategy, even though successful in our case, is the result of human considerations and engineering. Future work could explore if we are able to select autonomously the most appropriate high-level strategy, especially by applying RHBP, from several implemented strategies depending on the opponent's behaviour.

1 Team Overview: Short Answers

1.1 Participants and Their Background

What was your motivation to participate in the contest?
The motivation was to further evaluate the decision-making and planning framework ROS Hybrid Behaviour Planner (RHBP). While the framework has been used in a wide variety of projects (also in MAPC 2017), newer features have not been tested in complex scenarios. One of those features allows to create multiple independent levels of decision making by encapsulating a separate behaviour network into a behaviour. Another one is an extension for implicit coordination on the foundation of self-organisation.

What is the history of your group? (course project, thesis, . . .)
Researchers of the DAI-Labor started to participate in the contest in 2007. Since then they have contributed to every edition of the contest and have won four of them using successive generations of the JIAC multi-agent framework. The TUBDAI 2018 team originates from a Master's Thesis student and its supervising PhD student. The applied framework RHBP is developed in one Ph.D. thesis and several independent Bachelor and Master's theses.

What is your field of research? Which work therein is related?
Our field of research is multi-agent systems applied in the robotics domain.

1.2 Development

How much time did you invest in the contest for programming vs. other tasks (for example organization)? creating vs. optimizing your agents?
We invested approximately 600 h without time for framework development and the communication proxy (mac_ros_bridge). The mac_ros_bridge maps the xml-based socket communication of the MASSim simulation server to ROS communication means (which was also partly reused from MAPC 2017). Furthermore, 400 h of the invested time budget are spend on programming tasks while 200 h are used for optimising our approach.

How many lines of code did you produce for your final agent team?
The scenario specific code contains approximately 7000 LOC.

How many people were involved and to which degree?
Christopher-Eyk Hrabia (Ph.D. Student at Technische Universität Berlin) provided the general supervision, was especially responsible for the consultation about scientific approaches as well as giving technical support for the RHBP framework and its application.
Michael Franz Ettlinger (M.Sc. Student at Technische Universität Berlin) was responsible for the scenario specific implementation and execution of the contest.
Axel Hessler (Post-Doc at Technische Universität Berlin) was responsible for the infrastructure and overall administration.

When did you start working on your agents?
The major work started mid May 2018, communication infrastructure (e.g. mac_ros_bridge) was already done mid of April 2018.

1.3 System Details

How do your agents work together? (coordination, information sharing, ...)
Information sharing for implicit coordination (exploration, opponent well states) as well as explicit coordination (jobs) through a contract-net protocol implementation.

What are critical components of your team?
The most critical component is the job planning component which coordinates the job tasks amongst the agents.

Can your agents change their behaviour during runtime? If so, what triggers the changes?
Yes. The agents select the most appropriate behaviour based on the current perception and the results of the hybrid planning decision-making component of RHBP.

Did you have to make changes to the team (e.g. fix critical bugs) during the contest?
No.

How do you organize your agents? Do you use e.g. hierarchies? Is your organization implicit or explicit?
No hierarchy. We partly use implicit (self-organised) and partly explicit (contract-net protocol) coordination, see above.

Is most of your agents' behaviour emergent on an individual or team level?
All behaviour if it was possible emerge from individual level, which results from the autonomously taken decision by each individual agent.

If your agents perform some planning, how many steps do they plan ahead?
They plan one task ahead. A task can technically have unlimited amount of steps but practically has no more than 40 simulation steps.

If you have a perceive-think-act cycle, how is it synchronized with the server?
Our perceive-think-act cycle is performed as quick as possible as soon as the server delivers the percept of the current simulation step. Through enough calculation power it was made sure that the actions are always delivered in time.

How did you go about debugging your system?
We applied three different debugging techniques. First, RHBP offers extensions visualisation and monitoring of behaviours and their internal states. Secondly, agents can be started in development environment and analysed with a normal Python debugger. Thirdly, we used custom log messages to analyse the runtime behaviour without interference.

Which operating system did you use, and is your team portable to other operating systems?
We used Ubuntu 16.04, our solution is portable to all other Linux distributions that have ROS support. Execution on Windows is also possible through the Windows Subsystem for Linux (WSL) using a Ubuntu-binding.

What hardware did you use to run your agent team? (RAM, CPU, disk space, multiple computers, any other notable requirements)
Intel(R) Core(TM) i7-4930K @ 3.40 GHz CPU (6 cores with hyper-threading), 32 GB RAM and a Samsung SSD 840. It is the same machine that was used already by our team in MAPC 2017.

1.4 Scenario and Strategy

What is the main strategy of your agent team?
Our strategy has three main tiers. Stockpiling items as well as assembled items. Building wells at positions that are only accessible for special agents (drones). Attacking opponent wells aggressively.

How do your agents decide which jobs to complete?
If all required items are on stock the jobs are completed.

Do you have different strategies for the different roles?
Yes, only drones build wells and one drone is responsible for well exploration at the map border.

Do your agents form ad-hoc teams to complete a task?
Yes, the sub-tasks of a job are coordinated ad-hoc with a contract-net protocol implementation. Here, all agents participate in the auctions that are used for the task assignment.

How do your agents decide when and where to build wells?
Random places that can't be reached by other agents (off-road). We use the Graphhopper back-end to determine which map locations are not accessible by road agents.

If your agents accumulated a lot of currency, why did they not spend it immediately?
Our strategy requires drones to execute the well building, due to the fact that drones are comparable inefficient in building it is possible that we accumulate currency if the drones are not able to build fast enough.

1.5 And the Moral of It Is ...

What did you learn from participating in the contest?
What are the strong and weak points of your team?
The strategy decisions proved to be a viable solution for the contest. The stockpile with feedback strategy worked good enough to produce the required money (massium) for wells, while providing enough idle agent time for other tasks. If the contest would have been about massium alone (like last year) the strategy in its current implementation would likely preform worse. In such a scenario the use of storages could massively improve the massium output but would reduce the availability of agents for well building tasks. The off-road well locations strategy was not expected by any opponent and therefore performed well. The main drawback of the strategy was that if it is expected, it is easy to counter. This was observable in the match against SMART JaCaMo, who were able to adapt their strategy in a few hours to beat our team. The only assumption that turned out wrong was that we expected opponents to try to defend their wells once they were built. RHBP proved to be a great framework to use for the project. After an initial learning period, RHBP bore out to be robust and allowed agents to adapt well to changes at run time. The implementation was robust and performed well during the contest. The assembly coordination strategy worked well but resulted in many empty coordination cycles. If this would have been implemented using a client initiating contract net protocol, its performance as well as simplicity would likely have increased.

How viable were your chosen programming language, methodology, tools, and algorithms?
One goal of this project was to use RHBP and its new features and extensions, evaluate them and offer improvement suggestions. RHBP was used quite successfully and allowed to create a fast, adaptive and flexible solution for the contest. It also allowed quick and robust changes to the strategy as discussed in the evaluation. The run-time adaptiveness of RHBP could be observed at the match against Akuanduba-UDESC. While many resources were usually used for dismantling opponent wells, there was no dismantling required against Akuanduba-UDESC, which freed up resources for other tasks.

Did you encounter new problems during the contest?
We have been able to find several bugs and performance bottle necks in our SoBuffer library, which is used for communicating and handling messages for self-organisation.

Did playing against other agent teams bring about new insights on your own agents?
We did not gain major insights, we could only prove the runtime adaptation capabilities in situations we have not especially considered before.

What would you improve if you wanted to participate in the same contest a week from now (or next year)?
We would less emphasis on job completion and would add a well defence strategy.

Which aspect of your team cost you the most time?
Implementing the job coordination and execution.

Why did your team perform as it did? Why did the other teams perform better/worse than you did?
Because other teams didn't expect off-road well locations and our solution adapted robustly to different situations in the games.

1.6 The Future of the MAPC

What can be improved regarding the contest for next year?
Due to incidents in this years contest, we propose to make handing in code before contest obligatory. Furthermore, we think code changes should not be allowed, which also solves the problem of fair schedules. Maybe it could also be a good approach to run everything on the same virtual machines or docker containers, which are running in the organisers department to avoid problems with connection performance or too much deviating hardware requirements. Furthermore, we encourage to focus more on decentralisation and autonomous agent development. while avoiding the focus on optimisation problems.

What kind of scenario would you like to play next? What kind of features should the new scenario have? We would suggest to have a scenario that requires less optimisation of a scenario specific problem, highlighting more features of intelligent agents such as being adaptive, able to learn, robust, ...

Should the teams be allowed to make changes to their agents during the contest (even based on opponent agent behaviour observed in earlier matches)? If yes, should only some changes be legal and which ones (e.g. bugfixes), and how to decide a change's classification? If no, should we ensure that no changes are made and how?
Changes should not be allowed because having modifications during the contest defeats the purpose of finding a great strategy as well as autonomous decision making when developers make decisions based on their observations. Even more, we propose to enforce a code submission before the contest starts. Bugfixes could potentially be allowed but would have to go through a peer-reviewed pull request. For organisational reasons the review of the pull request could also be done after the contest.

Do you have ideas to reduce the impact of unforeseen strategies (e.g., playing a second leg after some time)?
As long as the strategies are done on the foundation of the provided API not exploiting bugs everything should be allowed. Even more, unforeseen strategies should be encouraged.
If the organisers want to prevent this (which we don't think they should), they could request a detailed strategy description to make sure they agree that the strategy is "expected" and not "unforeseen".

References

1. Ahlbrecht, T., Dix, J., Fiekas, N.: Multi-agent programming contest 2017. Ann. Math. Artif. Intell. **84**(1), 1–16 (2018)
2. Ahlbrecht, T., Dix, J., Schlesinger, F.: From testing agent systems to a scalable simulation platform. In: Eiter, T., Strass, H., Truszczyński, M., Woltran, S. (eds.) Advances in Knowledge Representation, Logic Programming, and Abstract Argumentation. LNCS (LNAI), vol. 9060, pp. 47–62. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-14726-0_4
3. Fernandez-Marquez, J.L., Serugendo, G.D.M., Montagna, S., Viroli, M., Arcos, J.L.: Description and composition of bio-inspired design patterns: a complete overview. Nat. Comput. **12**(1), 43–67 (2012)
4. Heßler, A., Hirsch, B., Keiser, J.: Collecting gold. JIAC IV agents in multi-agent programming contest. In: Proceedings of the Fifth International Workshop on Programming Multi-Agent Systems, At AAMAS 2007, Honolulu, HI, USA (2007)
5. Heßler, A., Hirsch, B., Küster, T.: Herding cows with JIAC V. Ann. Math. Artif. Intell. 1–15 (2010). https://doi.org/10.1007/s10472-010-9178-x
6. Heßler, A., Konnerth, T., Napierala, P., Wiemann, B.: Multi-agent programming contest 2012: TUB team description. In: Köster, M., Schlesinger, F., Dix, J. (eds.) The Multi-Agent Programming Contest 2012 Edition: Evaluation and Team Descriptions, number IfI-13-01 in IfI Technical Report Series, pp. 86–97. Institut für Informatik, Technische Universität Clausthal (2013)
7. Hoffmann, J.: The metric-FF planning system: Translating "ignoring delete lists" to numeric state variables. J. Artif. Intell. Res. (JAIR) **20**, 291–341 (2003)
8. Hrabia, C.-E., et al.: An autonomous companion UAV for the SpaceBot cup competition 2015. In: Koubaa, A. (ed.) Robot Operating System (ROS). SCI, vol. 707, pp. 345–385. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-54927-9_11
9. Hrabia, C.-E., Kaiser, T.K., Albayrak, S.: Combining self-organisation with decision-making and planning. In: Belardinelli, F., Argente, E. (eds.) EUMAS/AT -2017. LNCS (LNAI), vol. 10767, pp. 385–399. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-01713-2_27
10. Hrabia, C.-E., Lehmann, P.M., Battjbuer, N., Hessler, A., Albayrak, S.: Applying robotic frameworks in a simulated multi-agent contest. Ann. Math. Artif. Intell. **84**, 117–138 (2018)
11. Hrabia, C.-E., Wypler, S., Albayrak, S.: Towards goal-driven behaviour control of multi-robot systems. In: 2017 3rd International Conference on Control, Automation and Robotics (ICCAR), pp. 166–173, April 2017
12. Jennings, N.R., Sycara, K., Wooldridge, M.: A roadmap of agent research and development. Auton. Agents Multi-Agent Syst. **1**(1), 7–38 (1998)
13. Krakowczyk, D., Wolff, J., Ciobanu, A., Meyer, D.J., Hrabia, C.-E.: Developing a distributed drone delivery system with a hybrid behavior planning system. In: Trollmann, F., Turhan, A.-Y. (eds.) KI 2018. LNCS (LNAI), vol. 11117, pp. 107–114. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-00111-7_10
14. Maes, P.: How to do the right thing. Connect. Sci. **1**(3), 291–323 (1989)
15. Mcdermott, D., et al.: PDDL - The Planning Domain Definition Language (1998)
16. Pieper, J.: Multi-agent programming contest 2017: BusyBeaver team description. Ann. Math. Artif. Intell. **84**, 1–17 (2018)
17. Quigley, M., et al.: ROS: an open-source robot operating system. In: ICRA Workshop on Open Source Software, vol. 3, no. 3.2, p. 5 (2009)

18. Schillo, M., Kray, C., Fischer, K.: The eager bidder problem: a fundamental problem of DAI and selected solutions. In: Proceedings of the first International Joint Conference on Autonomous Agents and Multiagent Systems: Part 2, pp. 599–606. ACM (2002)
19. Smith, R.G.: The contract net protocol: high-level communication and control in a distributed problem solver. IEEE Trans. Comput. **12**, 1104–1113 (1980)