Modeling and Execution of Coordinated Missions in Reconfigurable Robot Ensembles

Martin Schörner[1], Constantin Wanninger[2], Alwin Hoffmann[3], Oliver Kosak[4], Hella Ponsar[5], Wolfgang Reif[6]
Institute for Software and Systems Engineering, University of Augsburg, Germany
Email: {schoerner[1], wanninger[2], hoffmann[3], kosak[4], ponsar[5], reif[6]}@isse.de

***Abstract*—Unmanned aerial vehicles (UAV) can support various scenarios, e.g., serve as measuring instruments for climate research, help rescue forces in disaster scenarios or autonomously inspect critical infrastructure. To accomplish their respective tasks, UAVs are often equipped with different sensors and in some scenarios used in ensembles to work together cooperatively. In this paper, we present the Block Definition Language (BDL) for modeling and executing UAV missions. The BDL supports both the use of exchangeable sensors and appropriate coordination mechanisms for robot ensembles. We demonstrate our plugin mechanism in a proof of concept using the BDL in conjunction with the robotic middleware ROS for hardware control and robot synchronization.**

I. Introduction

The range of applications for mobile robot ensembles, especially those consisting of unmanned aerial vehicles (UAVs), has been steadily increasing during the last years and continues to do so. Users deploy ensembles, e.g., for autonomous search and rescue in major catastrophe scenarios [1], space exploration [2], entertainment [3] and meteorological research [4], [5]. Unfortunately, the amount of different approaches for control and instruction of such ensembles grows as quickly as new use-cases emerge. This entails some serious drawbacks amplified by the diversification in hardware used for robots, sensors, and actuators the ensembles are built with: *1)* Ensemble designers redundantly design mechanisms that they need for controlling ensembles. *2)* Tools for flying robots (UAV) are often tailored to fit one specific use case, which makes them unflexible and hard to reuse in other applications [6]. Applications based on the robot framework ROS [7] requires interfacing with concretely implemented hardware drivers making the approach inflexible [8]. We want to overcome the aforementioned drawbacks with a flexible and parameterizable approach: the *Block Definition Language* (BDL) which we designed for reconfigurable, modular, and mobile robot ensembles. Block definition language provides the possibility to define relevant coordination patterns for those systems, which we call multipotent ensembles [9]. The deduction of abstract tasks using self organizing algorithms for agent missions is covered in Kosak et al [9] and combined capabilities in Eymueller et al [10]. While our previous work [11] focuses on the coordination of whole ensembles, the BDL covers the definition of robot programs, using the robots' sets of capabilities in a generic way. By using generic specifications that can be instantiated at runtime, we need no details on specific hardware implementations at design time. Thereby, our domain specific language BDL enables the modular reuse of program blocks for multipotent robots in different applications. For the concrete definition, execution and the deployment of missions for robots in an ensemble, we propose the Dynamic Mission Definition and Execution framework (DMDE) in the following. It implements the execution engine for the Block Definition Language (BDL) which is based on elements of UML activity diagrams [12] and petri nets [13]. Each mission in BDL consists of blocks that are interconnected with each other and can be executed by a *mission player* at runtime. BDL is agnostic to the used hardware and the targeted middleware. By using an extension mechanism, it can be tailored to the required middleware and the robots. Summarized, the contributions of this paper are the modeling language (BDL) for dynamic, sensor-based missions for robot ensembles, a framework for an automated distribution of modeled missions with plugin mechanisms and a concrete implementation of our approach with a ROS plugin on a mobile sensor platform.

This work is partly funded by the German Research Foundation (DFG) under the COMBO grant.

II. Related Work

Examples for traditional ground control software to define UAV missions are the APM Mission Planner [14] and QGroundControl [15] for the Pixhawk flight controllers, the DJI GCS Pro [16] for UAV of the manufacturer, or the ground control station of the Paparazzi flight controller [17]. Each one of these allow the user to perform useful interactions with the respective flight controller, e.g., monitoring telemetry data during flight, and defining simple missions. Missions in current software usually only consist of waypoints the UAV needs to approach containing simple additional commands, e.g., for taking pictures or deploying payloads. Each of the current software programs is usually bound to a certain flight controller. One exception is UgCS [18], which supports the flight controllers of several manufacturers. Another problem is that the control of more than a few UAVs is impractical or not possible with these tools [19]. Dousse et al. [19] try to solve this problem with a plugin for QGroundControl, which allows the control of more than 15 UAVs by creating a way to perform actions on multiple UAVs. In Intel's UAV light shows [3] several hundred UAVs equipped with RGB LEDs are controlled simultaneously to create a light show in the sky using a proprietary software.

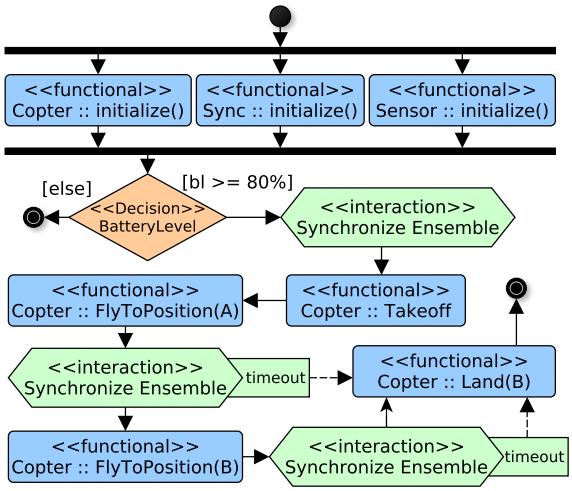

Fig. 1: Graphical representation of a BDL sample mission

The work of Kiener et al [20] focuses on mission modeling of heterogeneous robot ensembles based on different capabilities of the robots used. The missions are modeled with a graph similar to a statemachine without transition semantics. Exception handling is mentioned, but not modeled and also a distributed deployment is not focused in this paper. MacKenzie et al. [21] present a Mission Definition Language that summarizes robots in associations and addresses the capabilities in a strictly sequential manner. Brunner et al. created RAFCon, a graphical tool for the creation and execution of robotic tasks [22]. It uses hierarchical state machines to model these tasks and provides a GUI with various built-in debugging mechanisms. However, pseudo-states (e.g. fork) and orthogonal states are only handled implicitly and communication between different robots is not the focus of this project.

Projects in the field of robotics today are often based on the Robot Operating System (ROS) [7], a middleware for robot applications. ROS also offers various ways to model tasks or missions. Launch files in ROS2 are python scripts that can react to events and changes in the lifecycle of respective nodes [23]. The ROS-based framework smach [24] enables the creation and coordination of applications for robots. It enables the interconnection of the robots' capabilities to hierarchical state machines through python scripts. After creating and executing a program the user can monitor its state graphically at runtime. The missing multi-robot support as well as distributed deployment mechanisms are part of this paper.

III. Modeling Robot Missions with DMDE

In order to enable the definition of missions easily and in a reusable way, we introduce the BDL that enables the interconnection of blocks to form complete robot missions. Blocks fulfill different tasks by each containing a simple functionality, e.g., flying to a waypoint or closing a gripper. Blocks are either control flow blocks or functional blocks. Functional blocks contain the actual functionality of a mission and actively contribute to the execution of the mission. Control flow blocks instead manipulate the control flow of the mission, e.g., express alternatives or parallelization.

To demonstrate the functionality of BDL, we illustrate an example mission that can be deployed on multiple UAV carrying sensors for measuring sensor data in flight (cf. Fig. 1). Missions are designed for a specific robot, i.e., a system that is composed of a physical network of reconfigurable hardware, such as an UAV equipped with different sensors. In this mission the UAV initializes itself before flying to multiple waypoints and landing again. The UAV synchronizes itself with other UAV in its ensemble at each waypoint. The mission starts at a *Start* block (black circle). After that, we use a parallelization block (upper black bar) to split the mission flow into three parallel processes. These blocks initialize the UAV and the ensemble synchronization. A third block waits for a temperature sensor to be plugged in. In order to allow for arguments to be passed to functional blocks, parameters like the type of a required sensor can be defined. By allowing the use of wildcards, we can specify a requirement for an arbitrary sensor that supports the required variable to be measured. It is also possible to specify a concrete type of sensor, e.g., a temperature sensor. By using this approach we allow for dynamic reconfiguration of the robot in case of a hardware failure and don't require knowledge about the used sensors at mission design time. After all three processes reach the join block (lower black bar), the mission continues and reaches a decision block.

This block works like a decision in UML Activity Diagrams and picks the successor that matches the defined condition on the edges. The particular decision BatteryLevel block in Fig. 1 checks the UAV battery status and exits the mission if the battery level is below 80% in order to avoid battery problems mid flight. If the battery is sufficiently charged, the mission continues to a *Barrier* block that stops further mission execution until all other members of the ensemble also reach the Barrier with the same id. After that, we use a series of functional blocks to activate the motors of the UAV before flying along a series of waypoints and finally land again. Note that after flying to a waypoint, we use a barrier to pause the mission flow to ensure that all other participants of the ensemble have reached it before continuing. To enable the definition of alternative behavior when an exception occurs, we allow multiple successors for functional blocks. If a robot does not reach a barrier due a problem, a timeout error is triggered in the barriers of all other robots, resulting in an alternative path causing the UAV to land directly. These alternative paths can be defined for each block in each mission individually. It is also possible to define different successors for different types of errors that can occur in one block.

The mission definition in BDL is generic with regard to the usable blocks in order to be agnostic to specific frameworks, tools, and hardware. However, we allow the addition of new functions with a plugin system, e.g., to bind a robot middleware or other frameworks into the block definition language. A ROS plugin for BDL can, e.g., implement blocks for starting

nodes or calling services.

To allow for synchronization of several robots in the ensemble, we provide a plugin which enables the use of barriers in the missions. In the initialization phase of the mission, a master detects all robots in the ensemble by listening for their heartbeat messages. After initialization the participants of the ensemble are known by every participant and the barriers can be used for synchronization of the individual missions. Each barrier has a unique ID that makes it distinguishable from others. In its individual missions the robot stays in the block for the respective barrier until all other robots have reached it as well. Once all robots reach the barrier, they leave their barrier blocks at the same time. This procedure synchronizes the robot's missions after leaving the barrier block.

IV. Reference Implementation in Python

We implemented the concepts from Section III with Python scripts and JSON files to realize the Dynamic Mission Definition and Execution (DMDE) framework. The work flow therefore is as follows: The user defines a mission in JSON which the DMDE then parses for the mission player which represents the core of the framework. A mission consists of a sequence of blocks which can be parameterized by arguments. A mission can be executed with the mission player. A mission player only contains some basic functionality like start and end blocks or Forks and Joins. By loading plugins with the mission player, we can extend the missions with additional blocks (i.e., functional and logic blocks) to increase the default functionality and logic.

In order to not limit the functional range of the mission player to a single use case, we include the possibility of extending the mission player via plugins. To make use of the extensive functionality of the ROS Framework, a plugin for interfacing with it is needed. The extension enables our approach to use ROS1 as well as ROS2 as underlying middleware. To provide basic functionality for interfacing with ROS, blocks are available to call services, wait for activity in a topic and to start nodes and the logging of sensor data. In addition to interfacing with ROS, a method for synchronizing the ensemble is needed. For this purpose we implemented a simple synchronization plugin that uses the ROS2 messaging system to implement barriers as way of synchronizing multiple missions like described in III.

One problem when dealing with ensembles containing lots of robots is the deployment of the missions to the individual robots. In our framework, we distribute individual missions to the robots via a git repository [25] in which we are able to store missions for all devices centrally. After the start, the onboard computer of each UAV downloads the latest missions. The association between mission files and devices is achieved trough IDs stored on the UAV and the missions. Providing the required internet connection can be challenging especially on outdoor experiments. However, due to their small size, mission files can be downloaded via a mobile internet connection or, if no mobile internet is available, from a locally operating git server.

V. Proof of Concept

To demonstrate the functionality of our system, we tested it with a measurement flight scenario. Three UAVs synchronously ascended to specified altitudes in different places of a field while carrying temperature sensors.

Each of these UAVs carried a *mission controller* in the form of a Raspberry Pi 3B equipped with DS18B20 and DHT22 sensors to measure the temperature and humidity. We used a laptop in combination with a Spektrum DX9 radio remote control to realize the ground control station (GCS). This allowed us to monitor the mission and record the telemetry and sensor data sent by the robots. The laptop was also used for creating, modifying and distributing the mission files for the robots in the ensemble. It was running ROS and was responsible for handling the deployment of missions, starting missions and monitoring the flight of the UAVs. The remote was used to disarm the UAVs in case of an emergency. Up to two sensors could be connected to each of the UAVs which were connected to the GCS via a WiFi network. The sensors automatically published their data to the ROS2 messaging system. For acquisition of robot positions and sensor values, the laptop used the logging system *rosbag2*. The centralized collection of logged data made the subsequent collection of the data of the individual UAVs superfluous. This approach also enabled us to store the collected data in a format that allows the joint plotting and evaluation of collected data from several UAVs without requiring any manual merging efforts.

Each UAV received the mission file depicted in 1. We defined relevant information, e.g., the coordinates including altitudes of each waypoint and the type of sensor required for the flight, in the parameters of the respective blocks. To validate our synchronization concept, in their missions the UAVs performed a climb with several intermediate stops realized by barriers at different heights in the UAVs' missions. The GCS also executed a mission to guarantee safety measures for takeoff, start the logging of all telemetry data and coordinate the synchronization. After the start of the mission, the management node for the synchronization of the ensemble was started immediately. The *UserBreakpoint* block after the ensemble initialization halted the mission and displayed a message box on the screen. After all UAVs were registered in the ensemble, the user closed the message box to continue the program. In addition, the GCS started the logging of relevant parameters with a *StartLogging* block in the GCS's mission, wich recorded all ROS2 messages i.e., sensor data and UAV positions in the entire ensemble. The GCS behaves like a participant in the ensemble. It must therefore also reach all the barriers that the UAVs must reach. After the mission flow arrived at the last barrier in the GCS mission all running blocks with logging functionality were stopped and the mission ended.

The proof of concept was executed on the described hardware. However, the actual flight movements of each UAV were simulated. Therefore, we set up the three UAVs in a row and installed the batteries at the beginning of the

mission. For illustration we provide a screen capture[1] of the execution of the mission. After switching on the copters, they automatically established a connection to the Wi-Fi network and obtained their mission from the git repository set up for this purpose. The operator checked the UAV's connection to the emergency remote control and connected the sensors to the UAV. The mission controllers automatically recognized the sensors and started the software to integrate the sensors into ROS2. The current mission was also downloaded and started on the telemetry laptop. A console output on the laptop made it possible to track which UAVs had already completed their initialization and were ready for the mission. After the launch release by confirming the message box on the screen of the telemetry laptop, the actual mission was executed on the UAVs and the telemetry laptop. It was possible to follow the progress of the mission on the telemetry laptop via the command line output of the mission player and the UAV positions could be monitored graphically via RViz [26]. Parameters such as sensor values and the position of the UAV were available by using the ROS command line interface. These values were also collected and stored centrally on the telemetry laptop with the *rosbag2* framework. After landing, the mission ended on the UAV's mission controllers and on the telemetry laptop.

VI. Conclusion

In this work, problems of systems for distributed and mobile measuring devices in ensembles were first identified. In addition, we illustrated the great need for an appropriate mission definition framework for such applications by highlighting the lack of such in current literature on mission control in robot ensembles and modular hardware. To fill this gap, we then introduced our approach for a mission definition framework for robot ensembles in which we also provided the possibility to include other types of sensors. We further demonstrated our exemplary implementation and realization with actual hardware which we tested in the form of a proof of concept during a partially simulated measurement flight with an ensemble consisting of three UAVs and a ground control station. For this purpose we developed a modular hardware platform as an adapter for easy integration of sensors into the system. In addition, a mission definition framework was developed, which allows the definition and execution of missions and can be extended via plugins. In the future, work needs to be invested in the semantic annotation of the language and blocks. This way a coherence check of missions at ensemble level and a validation of the JSON file could also be accomplished.

References

[1] Swarmix: Synergistic interactions in swarms of heterogeneous agents. http://www.swarmix.org/. [Online]. Available: http://www.swarmix.org/

[2] R. D. Lorenz, E. P. Turtle *et al.*, "Dragonfly: a rotorcraft lander concept for scientific exploration at titan," *Johns Hopkins APL Technical Digest*, vol. 34, pp. 374–387, 2018.

[3] Intel shooting star drone project page. Accessed on: 2020-01-12. [Online]. Available: https://www.intel.co.uk/content/www/uk/en/technology-innovation/aerial-technology-light-show.html

[4] B. Wolf, C. Chwala *et al.*, "The scalex campaign: Scale-crossing land surface and boundary layer processes in the tereno-prealpine observatory," *Bulletin of the American Meteorological Society*, vol. 98, no. 6, pp. 1217–1234, 2017. [Online]. Available: https://doi.org/10.1175/BAMS-D-15-00277.1

[5] O. Kosak, C. Wanninger *et al.*, "Decentralized coordination of heterogeneous ensembles using jadex," in *IEEE 1st Int. Workshops on Found. and Appl. of Self* Systems (FAS*W)*, 2016, pp. 271–272.

[6] P. Ulbrich, R. Kapitza *et al.*, "I4copter: An adaptable and modular quadrotor platform," in *Proceedings of the 2011 ACM Symposium on Applied Computing*. ACM, 2011, pp. 380–386.

[7] M. Quigley, K. Conley *et al.*, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3. Kobe, 2009, p. 5.

[8] M. A. Ma'Sum, G. Jati *et al.*, "Autonomous quadcopter swarm robots for object localization and tracking," in *MHS2013*. IEEE, 2013, pp. 1–6.

[9] O. Kosak, C. Wanninger *et al.*, "Multipotent systems: Combining planning, self-organization, and reconfiguration in modular robot ensembles," *Sensors*, vol. 19, no. 1, 2018. [Online]. Available: http://www.mdpi.com/1424-8220/19/1/17

[10] C. Eymüller, C. Wanninger *et al.*, "Semantic Plug and Play & Self-Descriptive Modular Hardware for Robotic Applications," in *International Journal of Semantic Computing (IJSC)*, 2018.

[11] O. Kosak, F. Bohn *et al.*, "Ensemble programming for multipotent systems," in *2019 IEEE 4th International Workshops on Foundations and Applications of Self* Systems (FAS*W)*, June 2019, pp. 104–109.

[12] Version 2.5.1 of the UML specification. Accessed on: 2020-01-12. [Online]. Available: https://www.omg.org/spec/UML/2.5.1/PDF

[13] J. L. Peterson, "Petri nets," *ACM Comput. Surv.*, vol. 9, no. 3, pp. 223–252, Sep. 1977. [Online]. Available: http://doi.acm.org/10.1145/356698.356702

[14] APM Planner 2 official website. Accessed on: 2020-01-12. [Online]. Available: http://ardupilot.org/planner2/

[15] QGroundControl documentation: PlanView. Accessed on: 2020-01-12. [Online]. Available: https://docs.qgroundcontrol.com/en/PlanView/PlanView.html

[16] DJI GCS official website. Accessed on: 2020-01-12. [Online]. Available: https://www.dji.com/de/ground-station-pro

[17] P. Brisset, A. Drouin *et al.*, "The Paparazzi Solution," in *MAV 2006, 2nd US-European Competition and Workshop on Micro Air Vehicles*, Sandestin, United States, 2006.

[18] UgCS official website. Accessed on: 2020-01-12. [Online]. Available: https://www.ugcs.com/

[19] N. Dousse, G. Heitz, and D. Floreano, "Extension of a ground control interface for swarms of small drones," *Artificial Life and Robotics*, vol. 21, no. 3, pp. 308–316, 2016. [Online]. Available: http://dx.doi.org/10.1007/s10015-016-0302-9

[20] J. Kiener and O. Von Stryk, "Cooperation of heterogeneous, autonomous robots: A case study of humanoid and wheeled robots," in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2007, pp. 959–964.

[21] D. C. MacKenzie, J. M. Cameron, and R. C. Arkin, "Specification and execution of multiagent missions," in *Proceedings 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human Robot Interaction and Cooperative Robots*, vol. 3. IEEE, 1995, pp. 51–58.

[22] S. G. Brunner, F. Steinmetz *et al.*, "Rafcon: A graphical tool for engineering complex, robotic tasks," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 3283–3290.

[23] ROS2 node lifecycle documentation. Accessed on: 2020-01-12. [Online]. Available: https://design.ros2.org/articles/node_lifecycle.html

[24] smach framework for ros documentation. Accessed on: 2020-01-12. [Online]. Available: http://wiki.ros.org/smach

[25] L. Torvalds and J. Hamano, "Git: Fast version control system," *URL http://git-scm. com*, 2010.

[26] Rviz documentation for ROS2. Accessed on: 2020-01-12. [Online]. Available: https://github.com/ros2/rviz/blob/ros2/README.md

[1] https://video.isse.de/bdl