Synthesizing Capabilities for Collective Adaptive Systems from Self-descriptive Hardware Devices Bridging the Reality Gap

Constantin Wanninger, Christian Eymüller(✉), Alwin Hoffmann(✉), Oliver Kosak(✉), and Wolfgang Reif(✉)

Institute for Software and Systems Engineering,
University of Augsburg, Augsburg, Germany
{wanninger,eymueller,hoffmann,kosak,reif}@isse.de

Abstract. In the field of collective adaptive systems (CASs) robotic applications are mostly executed in a simulated environment with simulated hardware and abstract capabilities due to their complexity. These simulated systems usually cannot be applied in reality without major modifications. We propose an approach to bridge the gap between abstract capabilities and the execution of concrete capabilities on real hardware through a semantic description of the hardware itself, its drivers, interfaces and capabilities, enabling the realization of CAS in the real world. With a plug and play mechanism for hardware modules and the semantic description it is now possible to develop a CAS without committing to a concrete set of hardware and, moreover, the set of hardware to the requirements of the system.

1 Introduction

Collective systems exist in almost all areas of nature [6] (e.g., flocks of birds, herds of animals) and technology [20] (e.g., computer networks, robot teams). All of these systems have the common characteristic that groups of individual agents provide more functionality than each individual. Besides collective approaches, adaptive approaches can be added to manage even more complex problems. An example for a collective system is a group of ants carrying one big leaf on a narrow surface. If there are gaps on the forest floor, adaptive systems are needed to react to the environmental influences, like building a bridge out of other ants [28]. Such compound systems are called collective adaptive system (CAS) [21]. Mobile robots are typically used for the illustration of CASs on real hardware (e.g., [12,14,18]). These systems mostly use robots that are heavily customized for accomplishing specific tasks. For example, Unmanned Aerial Systems (UAS) equipped with gas sensors are used to find chemical clouds [23] or for the detection of forest-fires infrared cameras are mounted to UAS [10]. One problem of such systems is that they are very inflexible and must be redesigned

in case of changing requirements or new use cases. In this case hardware specific code, mostly written in C or C++, must be altered or rewritten.

This paper proposes an approach for the reconfiguration of robot hardware at runtime without the need to alter or rewrite hardware specific code. Therefore a plug and play mechanism with semantic self-descriptions of the hardware modules is used. These self-descriptions include static information about the hardware (e.g., weight of a hardware module) in form of *properties*. In addition, executable *capabilities* (e.g., "measure temperature") of a hardware module are deposited in the self-description of each hardware module. Capabilities can be either provided by a single hardware element (e.g., a quadcopter has the capability "fly") or by a combination of multiple hardware elements through combining their self-descriptions (e.g., a quadcopter with a GPS sensor has the capability "fly to position"). This information about hardware modules allows the development of CASs without committing to a specific set of hardware. The goal is to create a system in which each hardware module supplies interfaces to capabilities, which can be executed by agents within a CAS, rather than creating an agent with a fixed set of capabilities. In order to realize such a system, an adapter for devices (i.e., sensors and actuators) is created to enrich the hardware with semantic annotations and a common interface to provide capabilities for the usage in CASs. We call these systems "Self-Descriptive Devices" (SDDs).

In sum, this paper contains the following contributions for facilitating the use of hardware devices within CASs:

(1) Storage and usage of distributed properties;
(2) Methods for the automatic provision of capabilities for agents;
(3) Determination of appropriate hardware for capabilities;
(4) Task fulfillment through combined capabilities;

As a running example various hardware modules (i.e., quadcopter that can be equipped with multiple modular sensor modules) are used to demonstrate the advantages of the developed technique. Each SDD, no matter if it is a quadcopter or a sensor module, provides a self-description. In case of a quadcopter, it has properties like "maximum payload" and "weight" and capabilities like "fly to position". Furthermore these simple capabilities of multiple SDDs can be composed to more complex capabilities of the whole system. For example capabilities like "sensor-based flight" can be created out of the capability "fly to position" of the quadcopter and the capability "measure sensor value" of a mounted sensor module.

This paper is structured as follows: Sect. 2 describes which objectives the paper pursues. In Sects. 3 and 4 the structure and realization of SDDs in CASs is described. Afterwards the architecture and realization are evaluated in a case study (Sect. 5). Section 6 shows related research fields and Sect. 7 finally concludes the paper.

2 Objectives and Challenges

The objective of our approach is to establish an architecture for multi-agent applications in the field of collective adaptive systems with real modular hardware. In view of the variety of possible applications, a common denominator must be found. In many projects, e.g., [5,7,24,25] the term capability is used for interactions with (simulated) hardware. For example the project of Preece et al. [25] use the term to define if a camera can be mounted on an UAS with the semantic annotation "can mount" for every camera type. This interpretation of capability is only indirectly coupled with real hardware and serves as descriptive information to define the *properties* of the UAS. From our point of view, a property qualifies static information like, e.g., physical specifications (e.g., geometric models, weight, ...) or hardware specific limitations (e.g., sensor accuracy, motor speed, ...). The storage, distribution and usage of properties is a fundamental challenge in this paper and serves to give an appropriate answer to the hardware device, regarding the question:

(1) "What am I?"

Projects like Knowrob [31] use the term capability to describe executable procedures. For example, the annotation "grab cup" can be executed and an industrial robot starts the appropriate procedure. This procedure in turn uses properties (e.g., pictures, geometric details and grasp pattern) to support the automated execution. This interpretation of capabilities with dependencies to properties is also used in this paper. For example, a UAS with the properties "payload" and a mounted sensor with the property "weight" has to determine if the capabilities "fly to position" or "fly direction" are feasible (e.g., weight is lower than the payload). This example illustrates the dependency between the capability "fly" and the property "weight". The capability "fly to position" can further use the "battery capacity" in combination with the "weight" and "power consumption" to estimate the "flight time". The challenge lies in linking the capabilities to executable processes with a common interface for the usage as well as providing a mechanism for the creation and usage of dependencies between properties and capabilities. These dependencies give an answer to the question:

(2) "Am I capable of doing it?"

The description of hardware with properties and the access of its functionality with capabilities have to be established on real hardware. Every hardware element should provide its self-description i.e., its properties and capabilities. One objective is to offer capabilities over several hardware parts. For example, a quadcopter must be equipped with a distance sensor to offer the capability "sensor-based flight". To realize such configurable robots with self descriptive hardware elements in real world applications, various challenges must be overcome. For a common physical interface, the hardware elements must on the one hand be able to handle multiple physical interfaces in order to support a large

set of components like sensors, actuators and combinations thereof. On the other hand the hardware elements must offer a common communication interface for the exchange of corresponding values between them. This communication interface should also be used for the exchange of capabilities and properties. For the programming of an agent, the capabilities and properties must be traceable to the corresponding hardware element and actual configuration of hardware elements (e.g., the quadcopter should use a mounted sensor for the capability "sensor-based flight"). This information answers the question:

(3) "With what should I do it?"

Projects like [15,17,29] combine the capabilities of multiple agents, e.g., several mobile robots pull a child, while one single robot can not apply the force to pull it. In this example the user task "carry child" can be divided into several agent tasks "carry subject" which the combination of agents has to solve. This task decomposition of user tasks into agent tasks with coordination mechanisms between agents is not in the scope of this paper. However, we want to establish a mechanism to enable the agent to solve the task with a combination of capabilities. The agent task should express the requirements in an abstract manner (e.g., sensor based flight to position). The information, which sensors and actuators (e.g., position sensor, quadcopter) are needed and the procedure how they interact (e.g., fly to position combines a position sensor with the flight capability of a quadcopter) is one challenge, which is focused on in this paper. The definition of abstract requirements for the instantiation of capabilities with distributed properties finally leads to the question:

(4) "What am I supposed to do?"

For simulated environments as well as real hardware.

3 Concept

This paper provides an architecture for the realization of multi-agent applications in the field of collective adaptive systems with real, modular hardware, as shown in Fig. 1. To give an overview over the proposed system, we start with the user of the multi-agent system, who is able to define so called *User Tasks*. These are tasks that can only be handled by a set of agents, for example "fly triangle formation to position 1 m over ground for 15 min". Such tasks are decomposed into multiple *Agent Tasks* that can be assigned to a single agent of the multi-agent system, e.g., "sensor-based flight to position 1 m over ground for 15 min". For the decomposition of the user tasks, a distributed multi-agent reasoning system is used in our overall architecture, presented in Kosak et al. [22]. After the agent task has been assigned to an agent the advantages of our proposed system come into play. The following paragraphs address the questions from the previous section with an analog equation.

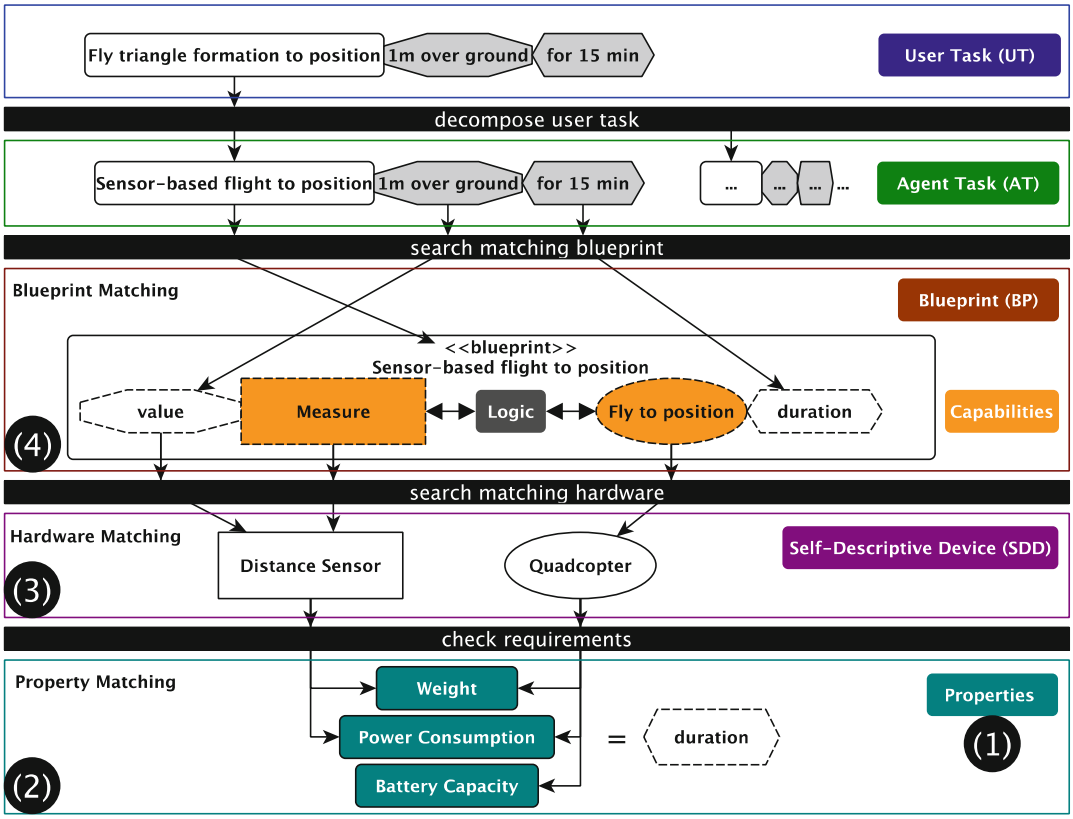


Fig. 1. Structure of the entire system from a given task of the user over the decomposition into capabilities to the selection of the required hardware.

With the requirements defined in the agent task, the agent searches for a blueprint in a set of predefined blueprints that can cover these requirements (*Blueprint Matching* (4)). *Blueprints* are a schematic representation of composed capabilities and depict how capabilities may be interconnected to more complex capabilities. For example, a task "sensor-based flight to position 1 m over ground" can be composed of the basic capabilities "fly to position" and "measure". With the help of the appropriate blueprints, the decomposition of a task into single basic capabilities is realized.

After the decomposition of a task into individual capabilities, the agent is capable of searching for hardware that possesses these capabilities and can fulfill the given task. This step is called *Hardware Matching* (3). In our example, we have a quadcopter that has a capability "fly to position" and a distance sensor that has a capability "measure distance to ground". From the information of the hardware's capabilities, the agent can determine with which hardware configuration it can fulfill the task.

Before execution of the task, a final check is made to ensure the agent is able to fulfill the task with the given hardware. For this check we use *Property Matching* (2), which guarantees that the constraints of each hardware module are met, for example if the task has the constraint that the sensor-based flight must last at least 15 min. Consequently, the system must check whether the desired flight duration can be achieved by the given set of hardware. For this check

several properties (i.e., "weight", "power consumption" and "battery capacity") of each attached hardware module are used.

After the concept of the entire system has been presented, the distribution of the concept to the individual hardware modules is described. Every agent in our CAS is composed of one Logic Device (LD) and a set of Self-Descriptive Devices (SDDs) as shown in Fig. 2. An SDD is a device which consists of the actual hardware, for example a distance sensor, and a Self-Description Adapter (SDA), which is responsible for the self-description of the hardware component. The SDA includes the communication interface, the semantic datasheet for self-description of the hardware with its capabilities and properties (1) and a driver to interact with the specific hardware for using its capabilities. Thus, SDDs provide the information used for the hardware and property matching of the system.

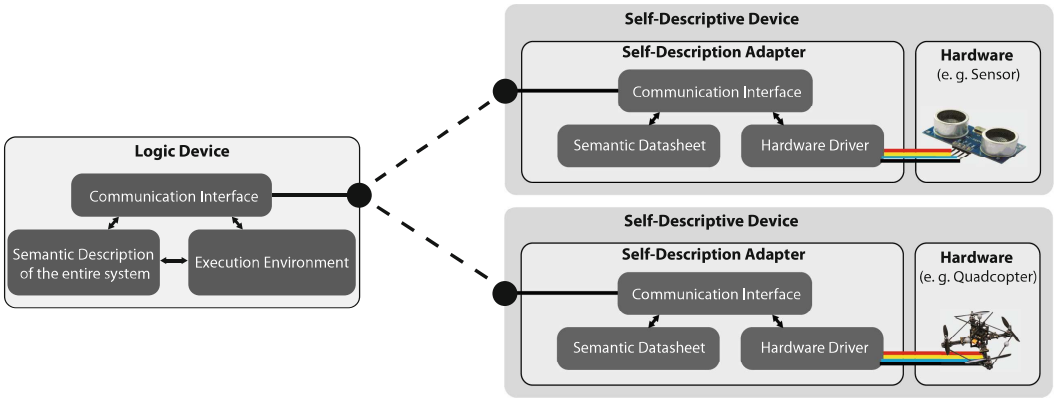


Fig. 2. Schematic structure of the system for our running example. This system is comprised of a quadcopter-SDD, a sensor-SDD and an Logic Device (LD).

An LD is a device which maintains a runtime environment for the execution of capabilities of multiple SDDs and provides an interface to the agent and thus represents an independent subsystem. LDs have a set of predefined blueprints that can be used to find a suitable hardware set and to utilize this blueprint with the real capabilities of the selected hardware for the execution of the agent task. Both LD and SDD communicate with each other over connections using wired or wireless interfaces. The communication interface is used on the one hand to transmit the semantic datasheets and on the other hand to query sensor values and set commands for actuators. One of the main components of an LD is the semantic description of the whole subsystem. This means all self-descriptions of each SDD are sent to the LD, where they are joined to form a complete knowledge base. For the execution of capabilities, we also need an execution environment, that can interact with mobile robots, actuators and sensors. For example, one agent of a CAS (e.g., mobile robot) consists of one LD and several sensor and actuator SDDs. In the quadcopter example, we use a single board computer attached to the quadcopter which runs an LD. This SDD has a connection to a quadcopter SDD and some sensor SDDs.

4 Implementation

The following section details the implementation of the four main concepts involved in the realization of SDD and LD: (Sect. 4.1) Distributed knowledge, (Sect. 4.2) Hardware and Property Matching, (Sect. 4.3) Blueprints and (Sect. 4.4) Deployment and Execution of Capabilities.

4.1 Distributed Knowledge

In order to create a common knowledge base of all SDDs connected to one LD, the distributed information of each SDD must be collected and processed. The information on properties and capabilities of each SDD is stored in form of a "Resource Description Framework" (RDF) ontology (e.g., the Semantic Sensor Network Ontology [3] for sensor data), where only the instances are specified in the self-description of the SDD. The abstract form of the ontology, which contains the classes and associations between them, is exclusively stored on the LD. To establish a common knowledge base, the abstract ontology is completed by the concrete instances of each SDD. To this end, the framework Jena [2] is used to merge the different ontologies. Therefore, a combination of multiple SDDs communicate with one LD which they are connected to by physical interfaces to transfer the knowledge. This collected knowledge is then available for the agent of a CAS. In addition to the merging of semantic data sheets, there is even the possibility of adding additional information to the ontology itself by inserting RDF-Triples with "Simple Protocol And RDF Query Language" (SPARQL) [26] `INSERT` statements. This extensibility is crucial due to the incredible variety of sensors and actuators (e.g., a new type of sensor is used). With such `INSERT` statements the user is able to add classes as well as instances to the ontology as needed.

4.2 Hardware and Property Matching

The created knowledge base can be used to find the required hardware to fulfill a given task. Therefore SPARQL queries are used to search for SDDs with specific capabilities or properties. By using SPARQL filter functions, the system is not only able to search for hardware devices which have a certain capability, but the search can be restricted even more precisely with help of the associated properties of a capability. For example if a task needs the capability "measure", it can be specified which value should be measured or with what accuracy the value is measured. These constraints of capabilities can be added by the user of a task by adding SPARQL filter functions to the task definition. After the search for and filtering of capabilities, the agent is able to use the found hardware for the execution of the task. If multiple SDD possibilities are found, the agent is even able to choose which hardware is most suitable for the task. If no suitable set of SDDs is found, the user is informed that no matching hardware devices were found for the execution of the given task.

If a set of SDDs is found, which is able to fulfill the given task, the property matching is executed. Therefore constraints of capabilities and constraints given by the user or agent tasks are considered. Constraints of capabilities may be for example that the capability "fly to position" can only be executed with a take-off weight of 1 kg. Task constraints may be for example that the sensor-based flight must last at least 15 min. This constraint check is realized through SPARQL queries. For the constraints of the take-off weight, the sum of the weight of all SDDs is queried and checked if its under 1 kg. The second example is more complex. For the calculation of the flight time, a function can be created that depends on "weight", "power consumption" and "battery capacity" of the entire system. Once all constraints are satisfied the execution starts, otherwise the user gets notified.

4.3 Blueprints

Between the abstract formulation of a requirement of a task (e.g., "sensor-based flight to position") and the decomposition into individual capabilities, a lack of information exists. This lack is eliminated by using blueprints to describe how to compose individual capabilities to more complex capabilities.

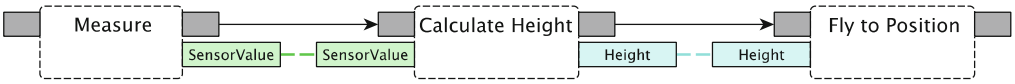


Fig. 3. Extract of the blueprint "sensor-based flight to position" with the capabilities "measure" and "fly to Position" and a logic component "calculate height"

Figure 3 shows an example of a blueprint for the composed capability "sensor-based flight to position". A blueprint consists of multiple semantic components (dashed boxes) which describe actions that can be executed. These actions can be capabilities or logical components, like calculation components (e.g., "calculate height") which in turn have control flow ports (boxes without description) as well as a data flow ports (boxes with names) and can be used to interconnect the semantic components. Control flow ports are used to set the order of execution while data flow ports serve to transfer data between multiple semantic components. In our example, the capability "measure" is used to influence the "height" parameter of the capability "fly to position". Each data flow port has a semantic description, which means it knows its content, its data type and its unit. For example, the capability "measure" in a blueprint, can be represented by a semantic component with a data flow port "sensor value" which measures distance values in meters.

Through this semantic description, the logical components are able to adjust their logic. To give an example, it makes a difference if a distance sensor or a temperature sensor (value must be mapped) is connected to the semantic component "calculate height". In both situations, the sensor value must be converted into a height in meters. As described in Sect. 3, the placeholder of the blueprint

is filled with concrete capabilities if capable SDDs are found in the hardware matching. After all capability placeholders of the blueprint are filled, the composed capability can be executed.

4.4 Deployment and Execution of Capabilities

Agents use blueprints to define which capabilities they need to execute their task. The blueprints must decide at runtime, which concrete hardware they use, thus a mechanism is necessary to load interfaces to enable the execution of capabilities. For this reason, a loading mechanism for hardware specific interfaces was created, that is capable of using semantically annotated code fragments. These fragments are Linked Open Data [34] compliant, stored in HTML sites and are linked to the appropriate self-description of the individual SDDs.

Figure 4 shows the deployment of a system with a quadcopter SDD, a sensor SDD and an LD. As described in Sect. 4.1 the LD includes the knowledge base consisting of the abstract ontology and the self-description of each SDD. The self-descriptions are sent to the LD when the SDD is added to the system. If the execution of a capability is required, a SPARQL query is used to get a URI for the code fragments of the SDD driver. Subsequently these code fragments are downloaded from the web or a snapshot, which is directly stored in an SDD, compiled at runtime and integrated into the Robotics API [30], which is responsible for the control of the sensors and actuators. Afterwards these fragments can be executed. All code fragments contain interfaces for the execution of capabilities within an SDD. For example, a sensor SDD has a function `getSensorValue()` or a quadcopter SDD has a function `flyToPosition(Position)`.

Because of the modularity, the system is able to exchange real SDDs for simulated SDDs and vice versa. So it makes no difference if it works on real or simulated hardware. Thereby the system is even capable of executing capabilities on a combination of simulated and real hardware devices.

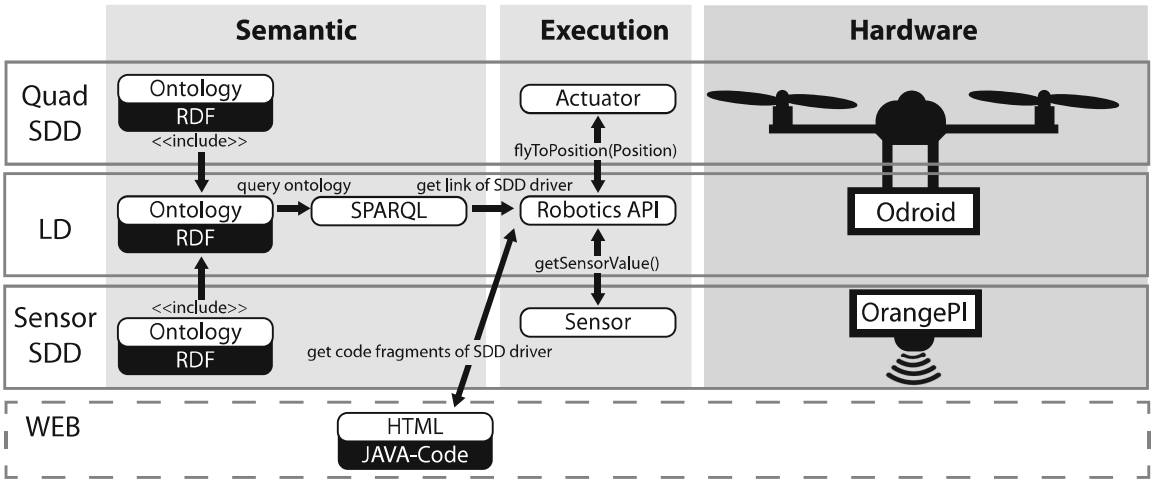


Fig. 4. Deployment of the running example with two SDDs and one LD

5 Proof of Concept

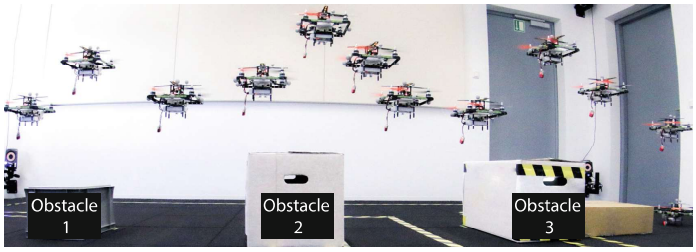
To show the feasibility of our approach and to implement a CAS in reality, several SDD prototypes were developed. These SDD prototypes consist of the actual hardware module (e.g., a sensor) and a single-board computer, which is responsible for the control of the hardware, the self-description of the hardware component and provides a wireless communication interface for the communication with the LD. These two components are encapsulated in a 3D printed case with a plug connection to enable the combination of several SDDs with an LD to an overall system (see Fig. 5).

(a) (b) (c)

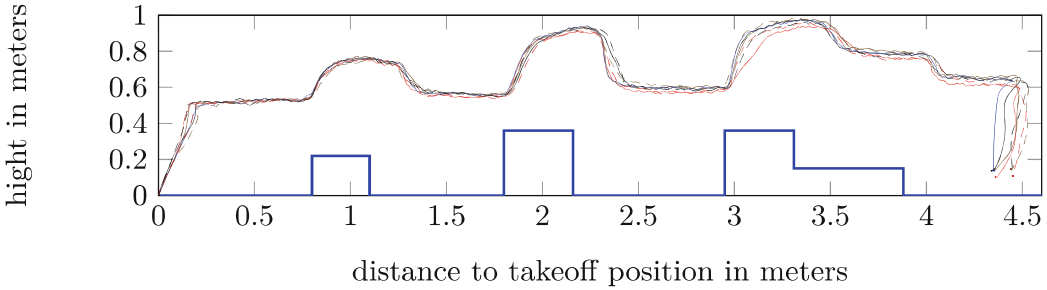
Fig. 5. Hardware prototype of an SDD adapter (a) for sensors and actuators. The used hardware is an Orange PI Zero with Wireless LAN in a custom 3D printed case with magnetic connectors. Prototype of a distance sensor SDD with an SR04 distance sensor (b). Prototype of a quadcopter SDD (c) with magnetic connectors for two SDD prototypes

The running example "sensor-based flight of a quadcopter" is used to determine whether our approach answers the questions mentioned in Sect. 2. By merging the distributed self-descriptions of each SDD to an overall knowledge base we can answer the question "*What am I?*". By using constraints in form of SPARQL queries and filters, the agent is able to define constraints for the execution of tasks. With these constraints the system is able to answer the question "*Am I capable of doing it?*". It has been shown that the system is even capable of selecting hardware or giving suggestions to the user which hardware should be used for the execution of a task. For example, if there are requirements like "measure temperature" the system will select a temperature sensor SDD or otherwise will inform the agent that no matching sensor SDD was found. This answers the question "*With what should I do it?*". The Question "*What am I supposed to do?*" is answered by blueprints, which are used for the decomposition of tasks into a set of capabilities. After all questions have been resolved and a suitable hardware configuration has been found, the task can be executed.

With the developed system it is possible to use values of a sensor SDD to influence the behaviour of actuators like a quadcopter SDD. Figure 6a shows the flight of a quadcopter that adjusts its height according to the measured distance to the ground with a distance sensor SDD, with the blueprint shown in Fig. 3. As a test setup a quadcopter SDD equipped with an LD and a distance sensor SDD flew along a specified route with obstacles. For the navigation of the route,



(a) time lapse recording of a quadcopter equipped with an ultra sonic distance sensor (SR-04) and obstacles on the ground for the sensor-based flight



(b) analysis of the sensor based flight (multiple runs) with obstacles within a indoor tracking system to get an accurate position of the quadcopter

Fig. 6. Case study: sensor-based distance flight

an indoor tracking system was used, which also recorded the exact position of the quadcopter, as shown in Fig. 6b. To validate the suggestion of SDDs, the capability "sensor-based flight" can be executed with distance sensor SDDs as well as with temperature sensor SDDs. The sensor-based temperature flight is comparable to a kind of thermometer. If the measured temperature increases or decreases the distance of the quadcopter to the ground matches the change.

By merging the self-descriptions of each SDD of an agent, it is possible to generate an added value for the overall system. Because the total weights, power consumptions and power reserves are known for each SDD, it is possible to calculate the average flight time of a quadcopter. Through the plug and play mechanism of the SDDs, it is possible to calculate the flight time dynamically depending on the current configuration of the system.

Figure 7 shows the comparison between the calculated flight time and the actually measured flight time with different takeoff weights of a quadcopter. From this data a function was derived, which predicts the flight time depending on the weight of the combined system with additional LDs. Hence, it is possible to evaluate the constraints whether a capability can be performed for a given duration as required in the example from Fig. 1.

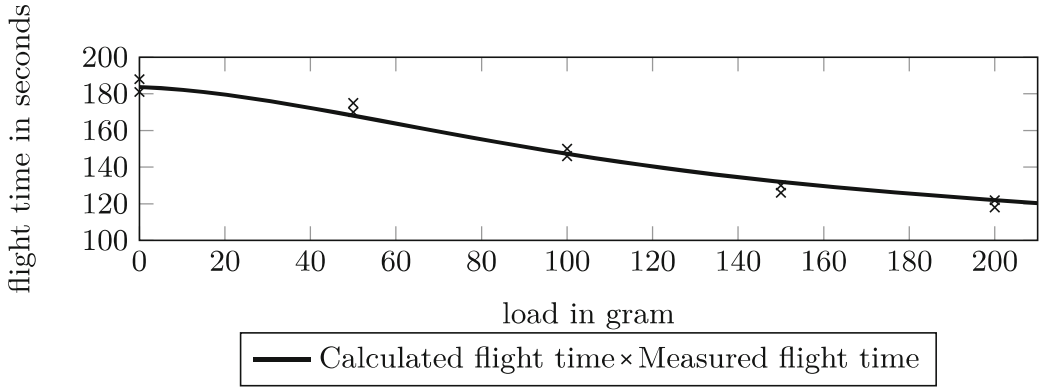


Fig. 7. Comparison between calculated and measured flight time with multiple runs with different weights

6 Related Work

The following section will investigate related work in relevant research fields emphasizing potentials as well as limitations in the context of Self-Descriptive Device (SDD). The presented architecture for SDDs addresses two areas: self awareness and modular hardware.

Due to many different protocols and storage possibilities for sensor data, common ontologies such as the Semantic Sensor Network Ontology [3] have been utilized. The advantage of the semantic storage in ontologies is primarily the usage of semantic reasoners (e.g., queries as described in Subsect. 4.2). In the field of geographic research, where many sensors are used to validate weather models, this ontology is used to convert stored binary data into semantically annotated data (e.g., [4,8,11,33]). The approach of Dibley et al. [13] goes one step further with a hardware adapter where this conversion takes place. In contrast to the architecture presented in this paper only sensors are semantically annotated and a transfer to capabilities is not in the focus of these approaches.

The project *Cubelets* [16] focuses the influence of measurements on corresponding actuators. Every sensor or actuator in this project is separated into one modular hardware element with a common physical interface. The communication is instead very primitive. Every sensor provides a value between 0 and 255 which is in turn used by the actuators. With this mechanism, primarily intended for educational purposes, reactive robots or systems can be built, however predefined processes (e.g., drive to position) are not possible. The idea of combined capabilities, derived from several hardware elements is focused on in the projects [9,29], in which homogeneous hardware is combined to gain locomotive capabilities. Heterogeneous aspects like e.g., combining a sensor with an actuator are not considered in these projects.

The robot operating system (ROS) [27] is a middleware for robots which allows publish-subscribe as well as service oriented communication mechanisms. In ROS, sensors and actuators can be integrated and tested within a simulation environment as well as on real hardware. The project *H-ROS* [1] aims to simplify the connection between hardware elements with a common interface for

communication in an ID-based plug and play manner. Each module must be plugged into a backbone, which is connected with a so called cognition. Within the cognition, the offered services of other connected H-ROS modules can be used in programs. Semantic annotations with dependencies on capabilities, as described in Sect. 3 are not in the focus of this project.

The project *Knowrob* [31] defines executable capabilities based on sensor information performed by (industrial) robots. Abstract capabilities (e.g., grab cup), geometric information (e.g., sensor position) as well as knowledge derived from observations of humans are stored within a common knowledge base built upon Semantic Web techniques. The abstract capabilities use this knowledge to subdivide themselves into executable robot motions. This subdivision is similar to the decomposition of capabilities. Modular self-descriptive devices as well as combined capabilities described in Subsect. 4.3 are not in the scope of this project.

To the best of our knowledge the combination of self description in a system of modular hardware with a capability interface for agents within a CAS is novel.

7 Conclusions

In this paper we have proposed an architecture for the realization of collective adaptive systems (CASs) on hardware devices for real world scenarios. With this architecture, we are able to fulfill tasks by analyzing the given requirements. Through the self-description and the plug and play mechanism of each Self-Descriptive Device (SDD), we are capable to compile a detailed description of the composed total system. Using this detailed description of properties and capabilities of each hardware component, it is possible to compose capabilities in a semantically correct way according to blueprints. The agent can use the blueprint on the one hand to execute the capabilities if they can be instantiated with the current set of hardware and on the other hand get a suggestion if the actual hardware can not handle it. If a system can handle a defined task it can execute this task in a simulated environment as well as on real hardware or even in a mixed reality. With an increasing amount of data the system can give further useful information about the system, like for example the rough estimation of the flight time mentioned in Sect. 5. This consistent and expandable architecture is in our point of view an important basis for the creation of CASs with real hardware in the real world.

Future research will focus on the autonomous reconfiguration of agents through the automatic exchange of SDDs with recommender techniques. Associated with this, predictive maintenance of SDDs will be examined. So the SDD can give information about its condition and if it is defective, it can be replaced autonomously e.g., to facilitate long term measurements with quadcopters and replaceable intelligent batteries. A first approach for the resource allocation on agent level is presented in Hanke et al. [19]. Nevertheless, future work will focus on the parallel execution of blueprints allocated to the same resources based on a previous work [32].

References

1. Hardware ROS (2018). https://www.h-ros.com/
2. JENA Framework (2018). https://jena.apache.org/
3. Semantic Sensor Network Ontology (2018). https://www.w3.org/TR/vocab-ssn/
4. Barnaghi, P., Wang, W., Dong, L., Wang, C.: A linked-data model for semantic sensor streams. In: 2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing, pp. 468–475 (2013)
5. Barreiro, J., Boyce, M., Do, M., Frank, J., et al.: EUROPA: a platform for AI planning, scheduling, constraint programming, and optimization. In: 4th International Competition on Knowledge Engineering for Planning and Scheduling (ICKEPS) (2012)
6. Bonabeau, E., Dorigo, M., Theraulaz, G.: Swarm Intelligence: From Natural to Artificial Systems, 1st edn. Oxford University Press, Oxford (1999)
7. Braubach, L., Pokahr, A., Lamersdorf, W.: Extending the capability concept for flexible BDI agent modularization. In: Bordini, R.H., Dastani, M.M., Dix, J., El Fallah Seghrouchni, A. (eds.) ProMAS 2005. LNCS (LNAI), vol. 3862, pp. 139–155. Springer, Heidelberg (2006). https://doi.org/10.1007/11678823_9
8. Bröring, A., Maué, P., Janowicz, K., Nüst, D., Malewski, C.: Semantically-enabled sensor plug & play for the sensor web. Sensors **11**(8), 7568–7605 (2011)
9. Cao, Y., Leng, Y., Sun, J., Zhang, Y., Ge, W.: 360botG2 - an improved unit of mobile self-assembling modular robotic system aiming at exploration in real world. In: 41st Annual Conference of the IEEE Industrial Electronics Society, IECON 2015, pp. 001716–001722. IEEE (2015)
10. Casbeer, D.W., Kingston, D.B., Beard, R.W., McLain, T.W.: Cooperative forest fire surveillance using a team of small unmanned air vehicles. Int. J. Syst. Sci. **37**(6), 351–360 (2006)
11. Compton, M., Henson, C., Lefort, L., Neuhaus, H., Sheth, A.: A survey of the semantic specification of sensors. In: Proceedings of the 2nd International Conference on Semantic Sensor Networks, vol. 522, pp. 17–32. CEUR-WS.org (2009)
12. Daniel, K., Dusza, B., Lewandowski, A., Wietfelds, C.: AirShield: a system-of-systems MUAV remote sensing architecture for disaster response. In: Proceedings of 3rd Annual IEEE Systems Conference (SysCon) (2009)
13. Dibley, M., Li, H., Rezgui, Y., Miles, J.: An integrated framework utilising software agent reasoning and ontology models for sensor based building monitoring. J. Civ. Eng. Manag. **21**(3), 356–375 (2015)
14. Dorigo, M., et al.: Swarmanoid: a novel concept for the study of heterogeneous robotic swarms. In: IEEE Conference on Robotics, Automation and Mechatronics (RAM), vol. 20, no. 4, pp. 60–71 (2013)
15. Dorigo, M., et al.: The SWARM-BOTS project. In: Şahin, E., Spears, W.M. (eds.) SR 2004. LNCS, vol. 3342, pp. 31–44. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-30552-1_4
16. Gross, M.D., Veitch, C.: Beyond top down: designing with cubelets. Tecnologias, Sociedade e Conhecimento **1**(1), 150–164 (2013)
17. Gross, R.: Self-assembling robots. KI **22**(4), 61–63 (2008)
18. Gross, R., Dorigo, M.: Towards group transport by swarms of robots. Int. J. Bio-Inspired Comput. **1**(1–2), 1–13 (2009)
19. Hanke, J., Kosak, O., Schiendorfer, A., Reif, W.: Self-organized resource allocation for reconfigurable robot ensembles. In: 2018 IEEE 12th International Conference on Self-Adaptive and Self-Organizing Systems, September 2018

20. Kernbach, S.: Structural Self-organization in Multi-agents and Multi-robotic Systems. Logos Verlag Berlin GmbH, Berlin (2008)
21. Kernbach, S., Schmickl, T., Timmis, J.: Collective adaptive systems: challenges beyond evolvability. arXiv preprint arXiv:1108.5643 (2011)
22. Kosak, O.: Facilitating planning by using self-organization. In: 2017 IEEE 2nd International Workshops on Foundations and Applications of Self* Systems (FAS*W), pp. 371–374, September 2017
23. Kovacina, M.A., Palmer, D., Yang, G., Vaidyanathan, R.: Multi-agent control algorithms for chemical cloud detection and mapping using unmanned air vehicles. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, vol. 3, pp. 2782–2788 (2002)
24. Morgan, D., Subramanian, G.P., Chung, S.J., Hadaegh, F.Y.: Swarm assignment and trajectory optimization using variable-swarm, distributed auction assignment and sequential convex programming. Int. J. Robot. Res. **35**(10), 1261–1285 (2016)
25. Preece, A., et al.: Matching sensors to missions using a knowledge-based approach. In: Proceedings of SPIE: Defense Transformation and Net-Centric Systems, vol. 6981, p. 698109–1 (2008)
26. Prud, E., Seaborne, A., et al.: SPARQL query language for RDF (2006)
27. Quigley, M., et al.: ROS: an open-source Robot Operating System. In: ICRA Workshop on Open Source Software, Kobe, vol. 3, p. 5 (2009)
28. Reid, C.R., Lutz, M.J., Powell, S., Kao, A.B., Couzin, I.D., Garnier, S.: Army ants dynamically adjust living bridges in response to a cost–benefit trade-off. Proc. Natl. Acad. Sci. **112**(49), 15113–15118 (2015)
29. Romanishin, J.W., Gilpin, K., Rus, D.: M-blocks: momentum-driven, magnetic modular robots. In: 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 4288–4295, November 2013
30. Schierl, A.: Object-oriented modeling and coordination of mobile robots. Doctoral thesis, Universität Augsburg (2017)
31. Tenorth, M., Beetz, M.: KNOWROB - knowledge processing for autonomous personal robots. In: 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 4261–4266 (2009)
32. Vistein, M., Angerer, A., Hoffmann, A., Schierl, A., Reif, W.: Flexible and continuous execution of real-time critical robotic tasks. Int. J. Mechatron. Autom. **4**(1), 27–38 (2014)
33. Xue, L., Liu, Y., Zeng, P., Yu, H., Shi, Z.: An ontology based scheme for sensor description in context awareness system. In: 2015 IEEE International Conference on Information and Automation, pp. 817–820 (2015)
34. Yu, L.: Linked open data. In: Yu, L. (ed.) A Developers Guide to the Semantic Web, pp. 409–466. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-15970-1_11