An Architectural Approach for Enabling and Developing Cooperative Behaviour in Diverse Autonomous Robots

Linkola, Simo

Springer
2022-08-19

Linkola , S , Mäkitalo , N , Laurinen , T , Kantosalo , A & Männistö , T 2022 , An Architectural Approach for Enabling and Developing Cooperative Behaviour in Diverse Autonomous Robots . in P Scandurra , M Galster , R Mirandola & D Weyns (eds) , Software Architecture. ECSA 2021 . Lecture Notes in Computer Science , vol. 13365 , Springer , pp. 181-204 , European Conference on Software Architecture , Växjö , Sweden , 13/09/2021 . https://doi.org/10.1007/978-3-031-15

http://hdl.handle.net/10138/353419
https://doi.org/10.1007/978-3-031-15116-3_9

acceptedVersion

An Architectural Approach for Enabling and Developing Cooperative Behaviour in Diverse Autonomous Robots

Simo Linkola, Niko Mäkitalo, Tomi Laurinen, Anna Kantosalo, and Tomi Männistö

Department of Computer Science
University of Helsinki, Finland
{first.last}@helsinki.fi

Abstract. The paper introduces an architecture for robot-to-robot cooperation which takes into consideration how situational context augmented with peer modeling fosters cooperation opportunity identification and cooperation planning. The presented architecture allows developing, training, testing, and deploying dynamic cooperation solutions for diverse autonomous robots using ontology-based reasoning. The architecture operates in three different worlds: in the Real World with real robots, in a 3D Virtual World by emulating the real environments and robots, and in an abstract Block World that enables developing and studying large-scale cooperation scenarios. We describe an assessment practice for our architecture and cooperation procedures, which is based on scenarios implemented in all three worlds, and provide initial results of stress testing the cooperation procedures in the Block World. Moreover, as the core part of our architecture can operate in all the three worlds, development of the robot cooperation with the architecture can regularly accommodate insights gained from experimenting and testing in one world as improvements in another. We report our insights from developing the architecture and cooperation procedures as additional research outcomes.

Keywords: Robot software architecture · Robot cooperation · Ontology-based reasoning · Peer modeling · Autonomous robots.

1 Introduction

Understanding the context of the robots plays a key role in autonomous robot cooperation. Situational context is a term used to describe why some phenomenon occurs in a specific situation and what actions can be associated with this situation [4]. This paper presents an extended and revised version of an architecture that fosters the situational awareness of cooperative robots, originally presented in 2021 CASA Workshop [14].

Information relevant to autonomous cooperation is pivotal in our approach to cooperation planning: A robot must be able to form an understanding of both

(a) the other robots and their resources and (b) the environment where the co-operation is intended to take place. Our architectural approach does not provide a solution to form a complete or joint contextual understanding between the robots. Instead, the architecture enables each robot to form its own view of the situation. The robots then use their situational context model and understanding as a basis for forming joint action plans for meeting their personal goals.

In most autonomous robot approaches, the goal of the individual robot and its cooperation behavior is fixed during the design. This leaves little room for novel dynamic cooperation where new (joint) actions and plans could be formed or goals adjusted after deployment in heterogeneous encounters with diverse peers or other computational actors. Nonetheless, this kind of creative use of complementary capabilities could highly benefit the whole robot population associated with a specific location, especially when the population is sparse and consists of low-end consumer robots built for singular tasks, e.g., cleaning, with ample idle time to allocate to other goals.

To optimize the use of context, and to develop the capabilities of the robots to understand their situation and cooperation possibilities, the architecture enables development in three conceptually and operationally different worlds. The development approach involves the Real World, a 3D Virtual World, and a 2D Block World, and a shared associated software architecture and frameworks, which can operate in all of the three worlds. Each of the worlds allows the designer to focus on different aspects of the development effort.

The 2D Block World works as a test bed for developing an ontology-based understanding of the cooperation context for the robots as it allows the simulation of large number of diverse robots in different cooperation scenarios. Ontology-based reasoning and planning provide robots a shared understanding of "how the world works" and thus are crucial in our approach for multi-robot cooperation.

We adopt DUL (DOLCE+DnS Ultralite)[1] as our base ontology. DUL is well suited for autonomous robot reasoning (see, e.g., KnowRob 2.0 [3]), where it serves as a *top-level ontology*, which specific applications are supposed to extend through their own ontology classes and relations. For this work we have developed a preliminary extension to DUL to showcase the applicability of our general approach.

In the Real World and the 3D Virtual World implementations, we have focused on robots based on the Robot Operating System (ROS), specifically ROS2 [18]. Briefly put, ROS2 is an open-source robot development framework where different nodes, or programs, communicate asynchronously using DDS, allowing nodes to *subscribe* and *publish* to *topics* shared over a network. Being a leading open-source project in robotics, ROS (and ROS2) has an active development community.

Our approach to autonomous robot cooperation aims to support ad hoc encounters between heterogeneous autonomous robots, each of which have their own individual goals. These goals can be used to define various plans, or workflows, which include different types of tasks. Typically, the cooperation tasks can

[1] http://ontologydesignpatterns.org/wiki/Ontology:DOLCE+DnS_Ultralite

be categorized into loosely and tightly coupled cooperation tasks [7]: *Tightly coupled tasks* cannot be performed by one robot but require multiple robots to work cooperatively; *Loosely coupled tasks*, on the other hand, can be performed by a single robot but the task can be performed more efficiently with cooperation.

The proposed software architecture enables cooperation in both tightly coupled and loosely coupled tasks mainly through *peer modeling*, which has been argued to be a requirement for cooperation [5]. The robots can exchange, learn, use and evaluate models of themselves and their peers to identify and exploit cooperation opportunities. Although the architecture proposes means for coordination and communication, implementing tightly coupled tasks, however, requires more work from the developer.

The rest of this paper is structured as follows. In Section 2, we introduce the relevant cooperation concepts related to our architecture. In Section 3, we describe our solution – a software architecture that enables the development, training, and testing of cooperation between autonomous robots. In Section 4, we explain the current status of two core elements of our cooperation solution: the ontology extension and the planner which uses the ontology. In Section 5, we describe the scenario-based assessment practice of our architecture, and results of preliminary stress tests of the architecture in the 2D Block World. In Section 6, we report our insights on developing the autonomous robot cooperation with the three-world approach. In Section 7, we cover related work and discussion. Finally, in Section 8, we draw the conclusions for this work.

2 Cooperation Concepts

To understand our architecture, we first introduce the ontological concepts we use to enable cooperation. The basic concepts introduced here are part of the DUL ontology, but we extend them in our work to provide concrete solutions and a better understanding of the situational context at hand (see Section 4.1).

The robots' essential operation revolves around *goals*, which we model as *environment states*, describing desirable situations the robot should find itself in. A goal can be, e.g., to keep a room clean or deliver a package to a specific place. A robot may have multiple or even conflicting goals.

To achieve its goals (either by itself or in cooperation), a robot forms *plans* which consist of *tasks*. A plan describes how a certain goal is achieved, i.e., which tasks should be done and their (partial) order. To make a plan concrete, each task needs to be assigned to a robot (or a set of robots). In DUL this type of plan, where each task has a designed executor(s), is called a *workflow*.

Tasks are the individual elements from which plans and workflows are composed of. Each task includes some objective(s) to be achieved, e.g., open a particular door, move to a specific place, etc. Tasks can be hierarchically nested in two ways. First, there can be general tasks (open a door) and refinements of those tasks (open a door by pulling the handle). Second, lower-level tasks may be combined to compose higher-level tasks, e.g., moving, opening a door, and moving again can be seen as one higher-level moving task. These task structures are used when generating and communicating workflows.

Tasks have defined start and end conditions. However, the actions (see below) can be partly responsible for checking these conditions. The start condition is checked before the task can be attempted, e.g., to open a door manually, the robot must be next to it. The end conditions are checked to see if the task was completed successfully, e.g., if the door is open. The task end conditions can be modeled as individual, low-level goals.

To achieve tasks, each robot has *actions* by which the tasks can be completed. The robot may have multiple (sets of) actions that achieve the same task, and an action may be utilized in multiple tasks. Where goals, plans, workflows, and tasks are platform-independent, actions need to be implemented on each platform (and world) separately.

To allow cooperation, robots communicate their goals, suggested workflows, and tasks to develop workflows, including multiple robots. To make this communication more fluent, robots maintain a model of themselves and each of the peers they have encountered. In general, these models may hold any important information of the robot in question, such as their physical properties, *capabilities*, i.e., which tasks they can perform, the robot's goals, and the history of the workflows they have been included in and their success.

3 Software Architecture for Autonomous Robot Cooperation

At the core of our research is the *CACDAR architecture* (Creative and Adaptive Collaboration between Diverse Autonomous Robots). The architecture, with its components and the leveraged services, is depicted in Figure 1. The architecture can operate in all three worlds, the 2D Block World, the 3D Virtual World, and the Real World, and it also enables feedback loops for the developers between these three different worlds, allowing them to manually – or automatically – incorporate insights gained from one world to another in order to advance the situational context awareness that fosters the robot cooperation (see Section 6).

Next, we introduce the high level descriptions of all the components and their development end goals. The current status of (some of) the main elements enabling cooperation are explained in the next section.

3.1 Cooperative Brain Service

The critical enabling service for the novel and valuable cooperation is the platform-agnostic *Cooperative Brain Service*, which encloses several components. The *Cooperative Brain Service* is responsible for the high-level functionality of the robot such as the planning of future tasks and cooperation (see *Planner)*, scheduling of tasks to be executed (see *Scheduler*) by *Task Runtime*, and it gathers information from sensors, the operation and communication of the robot with

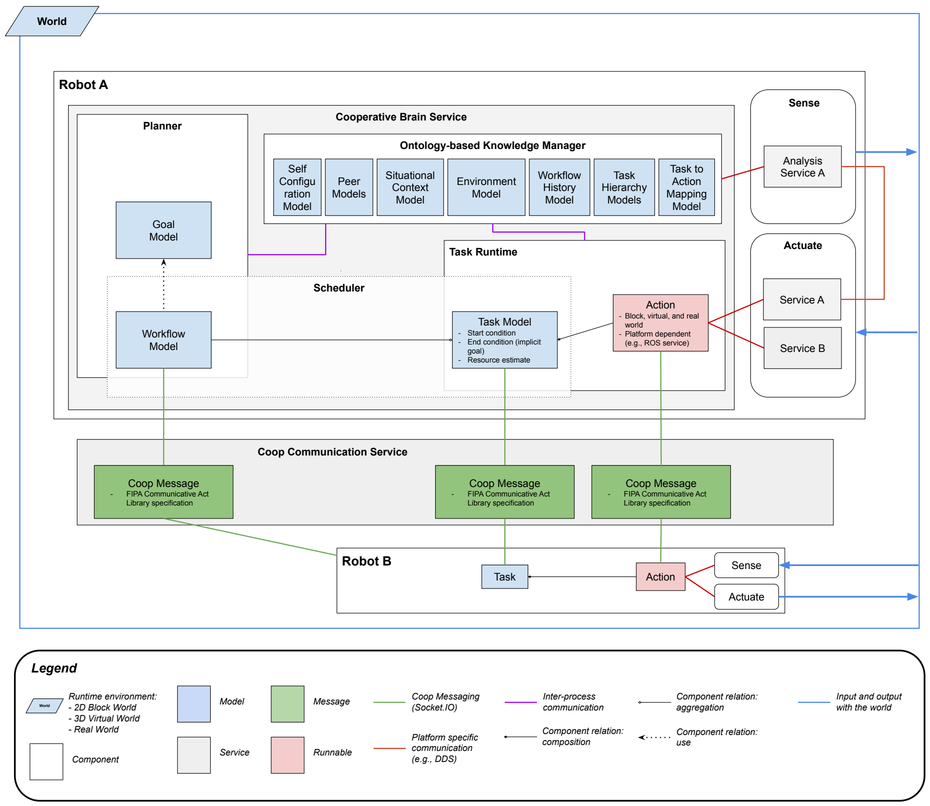

World
Robot A
Cooperative Brain Service
Planner
Goal Model
Workflow Model
Ontology-based Knowledge Manager
Self Configuration Model
Peer Models
Situational Context Model
Environment Model
Workflow History Model
Task Hierarchy Models
Task to Action Mapping Model
Task Runtime
Scheduler
Task Model
- Start condition
- End condition (implicit goal)
- Resource estimate
Action
- Block, virtual, and real world
- Platform dependent (e.g., ROS service)
Sense
Analysis Service A
Actuate
Service A
Service B
Coop Communication Service
Coop Message
- FIPA Communicative Act Library specification
Coop Message
- FIPA Communicative Act Library specification
Coop Message
- FIPA Communicative Act Library specification
Robot B
Task
Action
Sense
Actuate
Legend
Runtime environment:
- 2D Block World
- 3D Virtual World
- Real World
Component
Model
Service
Message
Runnable
Coop Messaging (Socket.IO)
Platform specific communication (e.g., DDS)
Inter-process communication
Component relation: composition
Component relation: aggregation
Component relation: use
Input and output with the world

Fig. 1: CACDAR architecture.

other robots into *Knowledge Manager* which it uses in its reasoning. For cooperation, the service needs to be able to understand the requests of another robot for help, and then try to reason if it would (a) have the missing resources, or (b) would have free resources or less important tasks so that it could free up the resources for the cooperation. The availability of such resources (e.g., time and battery) are estimated in collaboration with the *Scheduler* and the *Task Runtime* components.

However, the most crucial responsibility of the service is to estimate whether it will meet its own goals. It constantly keeps track of its resources and what resources other robots have allocated for helping it to meet its goals. Hence, it leverages *Knowledge Manager* and *Task Runtime* components by observing changes in the models that represent the other robots and environment, and then notifies the *Planner* which can alter its workflow and tasks (e.g., by replanning tasks with missing resources or reorganizing tasks in its workflow).

3.2 Knowledge Manager

Knowledge Manager takes care of maintaining the robot's understanding of the world and the information associated with the cooperation. The main input source for the component is the robot's (platform-dependent) service components that the robot uses for observing and sensing. *Knowledge Manager* may also exchange information with the components of other robots *Knowledge Manager* via respective *Cooperative Brain Services* with *Coop Messages*.
Knowledge Manager maintains the following models that enable novel and valuable cooperation as well as individual goal-oriented behavior of the robot:

Situational Context Model captures information considering the current situation of the robot, e.g., where it and other robots currently are, what is the state of the environment objects near it, and other dynamic properties. The model's contents can be updated using feedback from sensors, *Environment Model* (e.g., by making queries of possible state changes in the physical objects represented in the ontology if they are not directly perceived), *Self* and *Peer Models*, and direct communication with other actors, such as robots, through *Coop Messages*. To this extent, *Situational Context Model* operates in tandem with the environment and peer models to provide a unified view of the most current understanding of the situation. This model can be used directly in *Planner*, whereas other models provide more fractured view of the situation.

Environment Model connects actions in the operating environment, e.g., moving or object manipulation, into state changes in the ontological objects. The model should represent the environment and the objects in it in sufficient detail so that it can be used to derive reasonable *Situational Context Model* and reason about possible consequences of actions in particular situations. It can be updated using feedback from the environment (either perceived or received through communication). The level of detail in the *Environment Model* varies across the different *world types*. In 2D Block World, the model is sufficient to possess simple logical states, e.g., is the door open or closed, while in Virtual and the Real World the model may be more elaborate, e.g., a door can be partially closed and currently opening. However, to keep the "backward functionality" intact from Real World back to 2D Block World, individual object states and actions that manipulate them in Real World model should be mappable into the 2D Block World model.

Self Model and Peer Models contain information about the robot itself and its peers. In general, each peer has its own model, but aggregate models, e.g., considering certain classes of robots, are possible. Robots exchange information considering themselves (drawn from their *Self Model* and other knowledge sources) when they first meet their peers and update and replace this information through communication and observations. Where *Situational Context Model* offers current information of the state of the world, and *Environment Model* offers an understanding of how the world works, these models provide knowledge of what are the goals of each robot, which tasks are possible for the robot, and what restrictions the robot may have for performing specific tasks, e.g., if the robot can only open specific types of doors. From the cooperation perspective, these models are highly relevant, as their information is needed in *Planner* when determining which peers can perform a particular *Task*.

Task to Action Mapping Models contains knowledge about mapping the task realizations to actions. This knowledge is mainly about the robot's tasks, but peers' tasks to action mapping information can also be partially stored. This applies especially to cases if the robots are of the same type. Additionally, other peers may provide some information about their action mapping for a particular task, e.g., resource estimates, timing information, or constraints that can be used in reasoning.

Workflow History Model contains the information on earlier cooperation situations, such as performed task sequences, their configurations, and execution results. The information is used for improving the quality of the cooperation by analyzing which workflows have previously worked well and which ones have failed.

Task Hierarchy Models are used as configuration models for creating task hierarchies (consisting of task-goal-plan nodes), e.g., options for decomposing tasks or goals and constraints for valid hierarchy configurations. The model can be used to determine whether a particular task hierarchy configuration is valid, and the hierarchies can, then, be used by *Planner* or other components in *Knowledge Manager*, e.g., to represent aggregated high-level capabilities of the peers.

3.3 Planner

Planner is responsible for constructing *Workflows* which are then, e.g., passed to *Scheduler* for execution or stored for later use. As input, *Planner* is given some starting situation, e.g. the current *Situational Context*, a desired end condition, e.g. the current *Goal*, and other related parameters, e.g. restrictions for the workflow. *Planner* leverages the information maintained by *Knowledge Manager* in its attempts to select the robot and its peers to specific roles and to assign them *Tasks*. For actually assigning *Tasks* for its peer robots, *Planner* negotiates with the *Planner* components of different robots. The purpose is to ensure that the robot has a correct understanding of the capabilities of its peer (i.e., *Tasks* it can perform) and that the peer has sufficient resources, e.g., time and battery power, to participate in the workflow.

Goal Model defines a single mission (e.g. a task) that is expected to be carried out by a single robot or a set of robots. However, it does not define how the actual plan and the mission is expected to be performed. Instead, a *Goal Model* can set some ground rules for the robot behavior, like time constraints or quality attributes. A *Goal Model* is used for deriving start and end conditions for specific tasks. It may also affect what types of robots get selected into the roles of the cooperation.

Workflow Model consists of a *Goal Model* and a partially ordered list of *Task Models* where each task is assigned to a (set of) robots. By default, *Planner* tries to put together a *Workflow Model* where the robot itself is in the primary role, and its peers are assigned only if the robot cannot meet the *Goal*. However, the *Goal Model* can affect how the workflow is put together: As the *Goal Model* contains information regarding the mission of a single robot, it can then define the mission to be highly cooperative or act as a leader. For example, consider that one robot is expected to act as a supervisor for the other robots – its mission is then defined to coordinate the others and their cooperation.

3.4 Task Runtime

Different types of robots can feature very differing underlying platforms for development and interfacing in general. Therefore, the platform is essentially what dictates how actions have to be implemented. The *Task Runtime* is accordingly designed so that support for new platforms can be added at will, in the form of *platform modules*. However, special care needs to be taken when implementing 2D Block World platforms, as they operate in discrete time and not in continuous time. Currently supported platforms are ROS2 and a simple iterative simulation platform for 2D Block World built on top of Creamas[2].

Task. The self-adaptive aspects of the architecture come into play when the autonomous operation or cooperation requires certain resources. Each robot describes its capabilities by communicating to others what kind of *tasks* they can execute. A task may consist of other tasks, that is, a task may group together other tasks to obtain a higher-level behavior. As an example, consider that a robot can perform a task Guide. Such task then consists of other tasks, like Move, Turn, Navigate, etc.

Action is the mapping from the behavior modeled with tasks to the actual implementation of a specific task. Actions are generally platform-specific, but there can be alternative versions of actions for different robots even within the same platform. Similar to the tasks, also actions can be composed of a set of other actions. For instance, conforming Action: Guide may leverage various other action implementations.

3.5 Robot's Services

For actuating and sensing the events coming from the world, the architecture enables leveraging various services and communication between them. In Figure 1, such services have been illustrated: an imaginary actuating *Service A* is used, for example controlling the robot, and at the same time, it sends data to *Analysis Service A*. While we have mainly used ROS2 based services in our current implementation, the Cooperation Brain is not tied to any specific robot technology. Hence the services may also be realized as ROS1 services or any other type of service technology (e.g., as a Docker-based microservice), or, in case of 2D Block World, simple asynchronous function calls.

[2] https://creamas.readthedocs.io/en/latest/

3.6 Scheduler

Scheduler component is part of the Task Runtime component. Scheduler's main duty is to fetch tasks to be executed in Task Runtime. To this end, it queries from the Brain if there are tasks to be executed, which may cause the Planner to plan a new workflow, converts the responded tasks to their platform specific implementations, and delivers the runnable Actions to the Task Runtime. The Scheduler also ensures that the situation is correct for running the task, i.e. the task's start conditions are satisfied, and it communicates back to the other Brain components if that is not the case. The Scheduler can also use the resource estimates to ensure that the robot has the promised resources for performing the task.

3.7 Coop Communication Service

In order to cooperate effectively in varying situations and environments, the robots require a communication platform that can relay messages between the components deployed on various robots. The base technology for inter-robot communication is Socket.IO. It provides a relatively reliable and fast enough communication channel for negotiating about the cooperation-related activities, like tasks and roles in workflows, and providing feedback.

In our present research, we mainly leverage ROS2-based robots. ROS2, on the other hand, leverages DDS technology for communication between the ROS2 services. Hence, in the future, our implementation may change using DDS also for the cooperation communication to make the architecture more streamlined. The downside, however, is that setting up a DDS-based communication infrastructure can be challenging for robots that lack the required resources, and as there are several different DDS implementations, incompatibility issues may emerge and issues with licensing. For this reason, the implementation yet relies on our service and Socket.IO technology. Additionally, to support also non-ROS2 based robots, we have been discussing implementing a communication bridge that would allow ROS and other types of robots and smart objects and resources (e.g., sensors, existing facility service systems, smart home systems, etc.) in the environments to participate and enhance the cooperation.

Coop Message is the base unit of the communication in the *CACDAR architecture*. Two other base message types – *BroadcastMessage* and *Direct Message* – are inherited from the base. The idea is that the communication language is extended by inheriting new subtypes. The only requirement is that each message has a sender. The actual communication messages are based on FIPA Communicative Act Library Specification [8] from which we use a subset.

Broadcast Messages are sent publicly to all robots and services connected to the Coop Communication Service. Typical use cases for these messages are when a new robot arrives at a specific venue and then gets connected to the Coop Communication Service located at this venue. The robot may then greet other connected robots by broadcasting its name and the tasks it considers capable of performing. The robot may also request help from other robots by trying to describe its goal to other robots.

Direct messages, on the other hand, are sent directly from one robot to a set of recipients. These messages are mainly used for negotiating a cooperation plan and communicating during the execution of the plan.

4 Current Status

In this section, we present the current implementation status of our ontology extension and the *Planner* component, two of the main elements enabling the cooperative behaviour.

4.1 Ontology

The collaboration of our robots is based on ontological reasoning. While rigorous ontology development is not the main focus of our study, developing an ontology based on a well-known general ontology enables new collaboration possibilities for the robots: As long as the robots are familiar with the top level ontology, they can reason about the concepts presented to them, even if they are not familiar with the exact ontological classes used by another robot. Here we explain our ontology development process and illustrate a few classes from our ontology.

In our ontology development effort we have consulted the approach advocated by Noy and McGuinness [15]. They describe an iterative process of creating an ontological model of the world in a specific domain. After considering the scope and domain of the ontology, Noy and McGuinness encourage defining competency questions[3] to guide the ontology development before considering existing ontologies, enumerating terms, defining classes and hierarchies, properties, their value types, and finally creating instances of the ontology classes.

As stated before, we decided to extend the domain general DUL (DOLCE+DnS Ultralite) ontology to fit collaboration in the intended sample use cases. The domain of our work thus is general robot-to-robot collaboration and from the use case we derive that the collaboration in this case considers planning of navigation in a physical space. To aid our design we considered the following competency questions:

1. What objects in the environment are important for the robot to recognize and how will the robot tell them apart? (e.g. a door or a room in our use case of package delivery)
2. Which concepts are necessary for the robot to conduct a simple task alone? (e.g. moving from one room to another)
3. Which concepts are essential for collaborative plan construction?
4. How will the robot ask for help?

[3] After Gruninger, M. and Fox, M.S. (1995). Methodology for the Design and Evaluation of Ontologies. In: Proceedings of the Workshop on Basic Ontological Issues in Knowledge Sharing, IJCAI-95, Montreal.

5. How will the robot maintain an understanding of its capabilities?
6. How will the robot maintain an understanding of its peers' capabilities?

The first two questions (1-2) deal directly with the physical properies of the domain and suggest additions to the ontology supporting the description of spaces and their relations through doorways. The two following questions (3-4) relate to the ability of the robot to plan and initiate collaboration. Finally the last two questions (5-6) deal with the robots ability to recognize and model itself and its peers as agents in the world. In our current ontology development effort we have focused on questions 1-4.

The existing DUL ontology has general classes which can be used to describe any domain. The full DUL ontology is out of the scope of this paper, but we describe some general principles that support our extension. The DUL ontology offers several classes to represent physical objects and agents, which we extend in our own implementation. Examples of our extensions can be seen in Table 1. The DUL ontology also has several classes for representing social concepts, such as the class `Task`, which is inherited by our extensions describing specific kind of tasks in the domain, such as `OpenDoorTask`. Finally some classes represent information rather than events or physical objects.

The separation between `PhysicalObject` and subclasses of the `InformationEntity` class in the DUL Ontology means that some aspects of our ontological extension need to be represented by two separate classes to connect the information and the physical object representing it in the simulations or the real world. This is linked to the problem of symbolic grounding. We chose to circumvent it by using QR codes to tag the physical objects in our simulations and the real world. In our ontology the `QRCode` class represents the information stored in a QR Code tag, while the `QRCodeTag` represents the physical or virtual entity. These further link to other ontological objects, such as doors. The grounding problem could also have been solved by using for example machine learning to recognize the objects, but as it is not the focus of our project, we chose the QR code + tag approach.

Our ontology development effort is a living project and it is developed further to raise to the challenge of creative collaboration as we move on to investigate the ontological questions 5-6.

4.2 Planner

We have implemented the first working version of the planner, where emphasis has been given to forming operational plans for physical navigation in enclosures consisting of rooms which are connected with doors, i.e. floor plans resembling typical offices and other buildings. We chose to implement our own planner software as it needs to constantly communicate with the peers in the context, therefore making the existing software solutions for ontology-based reasoning, e.g. KnowRob 2.0 [3], only partially suitable for our needs. In the future, some of the subroutines of the planner may be refactored to utilise existing solutions.

Class (DUL inheritance)	Description	Current Usage
NavigateToTask (DUL:Task)	General task to move around in the world.	Used to move between two points.
DeliverObjectTask (DUL:Task)	Task to deliver objects to their destinations.	Used to deliver packages to their destination locations. The robot must check it is next to the destination before it can deliver an object.
Point2D (DUL:SpaceRegion)	Singular point with (x, y) coordinates.	Represents the location of any DUL:PhysicalObject. The coordinates are internal to the robot.
Room (DUL:SpaceRegion)	General area construct. Rooms are connected to other rooms (or areas) by doors (or other portals).	The robot can move between any two points in the same room using only NavigateToTask.
Door (DUL:DesignedArtifact)	Any object which can act as a closeable portal between two (or more) rooms	A robot with proper capabilities can open a door between two rooms to move between them, but it must be next to the door to do this.
DeliveryObject (DUL:DesignedArtifact)	Objects to be delivered	Used to represent the real objects with knowledge of their destinations, etc..
QRCode (DUL:InformationObject)	Represents a QR code which can be attached to any DUL:PhysicalObject	Robot identifies objects, e.g. doors and delivery locations, in its environment by recognising and reading QR codes.

Table 1: Examples of ontological concepts used in planning and other reasoning.

Next, we briefly describe the operation of the current planner. The full description of the component is out of the scope of this paper.

The current planner closely follows the famous A* heuristic search algorithm [10], which is quaranteed to find the shortest path in a graph (for us, a sequence of tasks) if the search's properties conform to some general assumptions. For now, the path cost is simply the time estimated to complete all the tasks in it, but it can be expanded to take into account other resources as well. A* search uses a heuristic function to guide the search to more promising directions. In our current implementation, with the focus for physical navigation, the heuristic is the time estimated for travel from current position to the goal position via straight line distance.

The A* algorithm operates on (directed) weighted graphs. We can construct the search graph using the instances of ontology classes and their relations.

Starting from a single node, the starting location, the algorithm considers which tasks are possible to execute from the current node and adds them to the task graph as edges between nodes representing states. For example, in a room we can ask which all other points are in this room and expand the search with instances of `NavigateToTask` to those points, or in points next to doors, we can expand the search with `OpenDoorTasks` allowing the search procedure to cover also other rooms.

To better accommodate our needs, we have made a few modifications to the basic A*. First, for some of the tasks it is crucial to verify if their starting conditions are satisfied before the task can be expanded, e.g. the manual `OpenDoorTask` can be added as an edge to the graph only, if the robot is next to a door it can open manually. Second, the planner communicates with peers to find executors for tasks which the robot can not do itself. In such cases it can use the time estimates of the peers to compute the path cost estimates. Third, for some closely associated task sequences, the planner utilises task hierarchies to achieve more complete behaviour. For example, if the current goal is to find a route to a certain point, and the robot is not guiding anybody, then `OpenDoorTask`, `NavigateToTask` and `CloseDoorTask` are added automatically to the task graph as tasks related to entering a new room before a new search expansion is done.

After a task sequence required to achieve the goal task is found, the planner communicates to the peers which tasks are done by which peers based on their request responses. The planner also inserts into the robot's own task sequence `WaitTask` with proper end condition before any task that (1) is done by a peer and (2) is required to be completed before the robot can continue to its next task towards its goal.

5 Asessing Cooperative Behaviour in the Three Worlds

To evaluate the implementation of our architecture we developed scenarios that represent the desired qualities of the system: autonomy, cooperation with diverse peers, self-adaptation and functioning in changing or uncertain environments. First, we elaborate on our scenario development process, and then we discuss the implementation and results of testing our architecture using randomised scenarios in the 2D Block World.

5.1 Scenario Development

In this section we describe the assessment practice of our architecture revolving around *scenarios*, and the properties we have identified as preliminary requirements for the scenarios. We report our insights on developing the architecture using an example scenario in Section 6.3.

We defined a scenario to consist of a context and an activity (associated with some goals) in one of the worlds. Both the context and the activity can have variable properties. Variables of the context include the physical context, e.g. a floor plan and objects residing in it, and the participants of the scenario, e.g. the robots operating in it. The variables for the activities include parameters specific to that activity, such as goal states and start locations for the participants. A scenario that is executed forms a situation. A situation describes how things are at a particular time in a particular context when the robots are performing particular activities. The situation defines the current status of the system and can consider also the inner states of the robots.

These scenarios were developed to be implementable in the three different worlds: 2D Block World, 3D Virtual World, and Real World. The main purpose of developing the system for all three platforms was to evaluate its feasibility similar to building a skeleton system for evaluating software architecture (see e.g. Rozanski and Woods [17, p.225]). The use of the different worlds allows us to evaluate further aspects of the system: Through building a sample system for Real World we evaluate the feasibility of our system. Simulating the system in the 3D Virtual World allows us to test the variability and deployability of the system as we can change the simulated hardware and the environment of the system easily. Finally evaluating the system in the 2D Block World allows us to test the logic with different single-agent service configurations, as well as the robustness of the system with large sets of agents. Together the multiple worlds, multiple scenarios approach allows us to achieve reliability through iterative development and experimentation.

We conducted our scenario development iteratively starting from a simple 'follow me' scenario in which one robot follows the other to a target (see Mäkitalo et al. [14]). From this simple scenario we derived alternative activities and contexts to enable for more robust testing that would allow us to validate the capability of the system in working autonomously under variable scenarios including changes and uncertainty.

After implementing the initial scenario in the three worlds, we decided on the materials for the physical context of the scenarios. We selected 3mm thick cellular board as the physical building material and designed the physical scenario creation based on 50cm x 50cm modular wall pieces supporting easy implementation in all three worlds. The modularity allows for building various enclosures, which we also call floor plans. The enclosures can be used to test how cooperation and the base software function in changing or uncertain environments, supporting our goals of self-adaptation and autonomy. In our scenarios the enclosures resemble house floor plans, which typically include rooms. The different rooms (as well as other physical objects relevant for robots) are labeled using QR codes to facilitate their recognition.

Next we defined a number of activities to be implemented in an enclosure built from the physical or simulated modular squares. These activities focused on the collaboration aspects of testing. These activities are described in detail in Table 2. The first three tasks are variations of the same theme. The value of separating the guiding task into three different activities becomes apparent if additional restrictions are posed on the system, such as access rights. The first three activities are suitable for *static* scenarios, in the sense that the properties

Label	Task	Description
Guide 1	Guide robot from point A to point B	In this task one robot guides another from a point to another in a situation where the first robot does not know the location or route to point B. The robots need to communicate using ontological concepts. The guidance could be for example related to a package delivery scenario, or regular maintenance performed by a robot visiting a new environment.
Guide 2	Guide robot from room A to room B where there are two or more routes available that are equally good.	In this task one robot guides another from one room to another, but there are more than one optimal route. This can be used to test additional constraints, e.g. situations in which access rights required to complete the two routes are different.
Guide 3	Guide robot from room A to room B, when there are several routes available with different assessments of route quality or cost.	In this task one robot guides another from one room to another, but there are routes with varied levels of optimality. This can be used to test additional constraints, e.g. situations in which the required access to complete the routes is different.
Pickup	Pick up a package in room A and bring it back to room B.	In this task one robot picks up a package stored in a room. It can be combined with the guiding task. As the robot picks up the package its weight and dimensions may change, meaning it may need to select an alternative route back.

Table 2: Set of test activities for our scenario-based assessment practice.

of the participating robots or the environment do not change during the scenario involving the activity. They can still be used to test for autonomy, cooperation with diverse peers, self-adaptation and functioning in uncertain environments, as the participating robots and the context, or the start or end locations of the guiding activity can be changed before each test run. Any of these activities can be used to construct a *dynamic* scenario by blocking the route from one room to another during the activity, or changing access rights. However, the fourth activity, 'Pickup' has some inherent dynamism as the properties of the participating robot can change if it picks up a large delivery. This can for example prevent the use of certain doors on the way back.

In our scenarios the differences between participating robots are defined through *robot capabilities*. E.g. one robot is able to follow, one to guide and a third to open doors. It is however possible also to further derivate between robot properties by allowing them different access levels to different areas of the physical enclosure. This can enable more complex collaborative tasks, such

Size	#DL	#DO	#Objects	Delivered%	Steps	Time
20 x 20	4	10	440	100%	129	0.279 (0.002)
50 x 50	10	10	2720	100%	351.5	2.326 (0.007)
200 x 200	20	10	43280	100%	1487.5	112.3 (0.076)
400 x 400	50	10	172960	100%	3084.7	953.6 (0.309)

Table 3: Results of tests in 2D Block World simulations. The setting on each row has been run 10 times and run averages are reported. Size is the scenario's floor plan size, #DL is the number of delivery locations, #DO is the number of delivery objects, #Objects is the number of `Point2D` and `Door` objects in the scenario, Delivered% is the percentage of successful deliveries, Steps is the average number of simulation steps required to deliver all the objects, and Time is the average time the simulation took to complete with time per step in parentheses.

as a robot having to be lead by two others to reach a final destination going through areas the two other robots are not allowed to be in alone. The use of differently capabled robots in our scenarios makes them suitable for evaluating tightly coupled cooperation, but they could be used for evaluating loosely coupled cooperation, if the same capabilities were given to all robots.

5.2 Stress Testing in 2D Block World

We conducted stress testing of our architecture, and especially the *Planner* component, by building on top of the basic scenarios described above in 2D Block World iterative simulations. We created a diverse set of floor plans consisting of different sized rooms, doors between them, and a few delivery points spawned in random locations across the floor plan. Specifically, we created 10 floor plans for each 2D Block World size: 20 x 20, 50 x 50, 200 x 200 and 400 x 400. For each floor plan, we spawned two robots to random places with different capabilities: one that could open doors and guide, and one that could follow and deliver objects. The delivery robot did not have an understanding of the floor plan, i.e. it had to ask for guidance to the delivery points, and it was initialised with 10 delivery objects with random, specific delivery point destinations.

The purpose of the expanded scenario was to verify that (1) the planner was able to find a proper cooperation plan for each `DeliveryTask`, (2) the robots were able to execute the plan so that the object was delivered in the end, and (3) to ensure that the execution time of the planner did not grow too much as a function of the floor plan size. We report our results in Table 3.

We make two main observations from Table 3. First and foremost, the planner is able to construct a proper cooperative plan and the agents are able to execute it in all the randomly initialised scenarios (see Delivered% for each row). This is strong evidence that both the planner is working correctly and that the

2D Block World implementations of the actions are properly implemented. Second, we see that the time to execute each simulation step grows as the scenario size grows (both the size and the number of ontological objects). However, the growth is non-existent when dividing the step time with the number of ontological objects in the scenario. This indicates that the planner is able to consider the plan candidates in an efficient way and expand the search to the most promising directions, and that the number of ontological objects in the agent's knowledge does not affect (on this scale) the performance too much. However, based on these observations, we have already identified certain bottlenecks in our architecture implementation where inefficient list lookups of ontology objects could be refactored to more efficient solutions.

6 Three-world Development Process

We chose to develop our architecture for three different worlds the Real World, the 3D Virtual World and a simple 2D Block World to facilitate different development targets. The 2D Block World offers a simple environment to test the architectural adaptation logic, as well as for stress testing. The 3D Virtual World offers a way to progressively transfer the implementations to the Real World via controlled platforms featuring 3D physics simulation. Finally, the Real World helps us to identify potential gaps left by the simulations as well as test the applicability of the architecture in real scenarios. We began with simple tests run in the Real World, and now conduct our development concurrently on the different worlds, creating feedback loops from one world to the other. The effect of this feedback loop on our development effort is illustrated in Figure 2. In this section we describe some of the lessons learned in our development efforts on the three worlds.

6.1 Introducing ROS and Real-World Robots

We deemed it beneficial to familiarize ourselves with physical robot development from early on and started by prototyping with robots in the Real World. Our initial development was done on Rosbot 2.0 and TurtleBot3, both affordable robotic platforms using the Robot Operating System (ROS) as their development platform. However we soon moved on to using the newer version of ROS, named ROS2, as it offered better functionality, and we also started using exclusively the TurtleBot3, as it had much better ROS2 support at that point.

TurtleBot3 is an economical robot intended primarily for educational and research use. It is equipped with a pair of wheels and a LIDAR for scanning surroundings. The system is modular and the composition of different robots can be changed with relative ease. For example, cameras can be added to the robots to enable more complex sensing of the world. This proved important, as in our project the robots do not share any common understanding of the world, such as a common map or even a common coordinate system. Therefore one

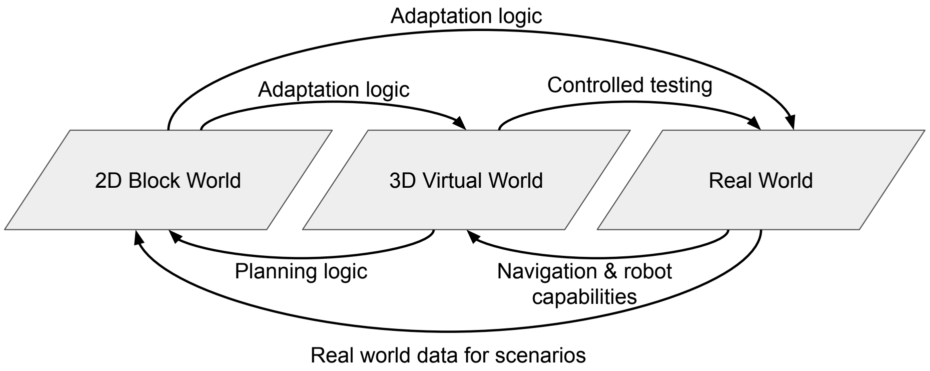


Fig. 2: Examples of lessons learned feedback loops between three development and evaluation worlds. The three worlds each bring their own development properties, but some of these properties also affect other worlds. For example data from the Real World, including robot types and configurations, guide the work on the simulated worlds.

of the preliminary coordinated Actions we implemented consists of one robot following another with the help of QR codes [14].

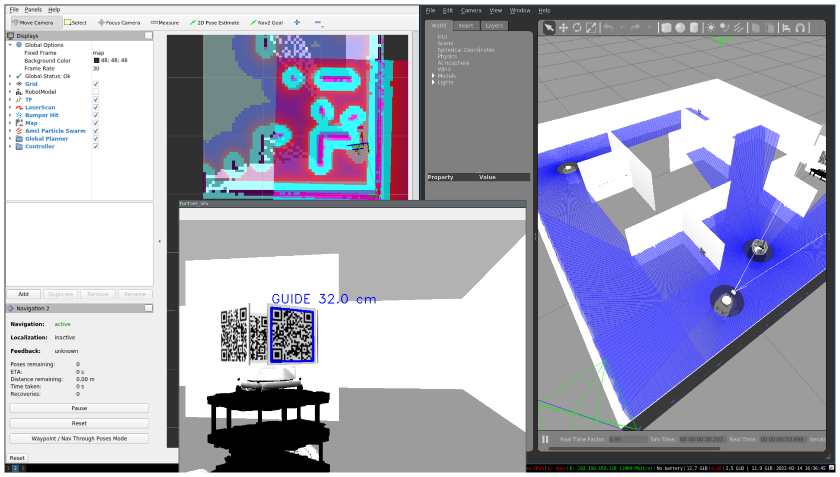
During experimentation in the Real World, several realities of robot development became apparent: Lighting conditions would affect the detection of QR codes greatly, even the slightest of obstacles such as cables were insurmountable for the TurtleBot3, and having to reset the positions of the robots manually every attempt was also rather inconvenient in the long run. We also did not have the equipment or means for complicated feats such as having the robots carry objects. Finally, the worsening COVID-19 pandemic shut down Real World development, so we focused on the 3D Virtual World environment next.

6.2 Bringing Cooperation from Real World to 3D Virtual World

As the intermediate world of the three-world approach, we chose Gazebo[4] for our first simulation environment, as it integrates well with ROS2, supporting the reuse of the implementation of our architecture in the three worlds. Gazebo also supported a smooth transition from the Real World to the 3D Virtual World, as it already includes a well implemented, accurate model for Turtlebot3. Therefore the QR code based robot following method, for one, worked in the 3D Virtual World as-is (see Figure 3 for cooperation development in Gazebo with Turtlebots).

However, we did face some challenges with the 3D Virtual World implementation when trying to run multiple robots in a single Gazebo simulation. When using ROS2 in Real World, each robot is typically assigned its own ROS2 domain. However, when using Gazebo with ROS2, all the simulated robots have to

[4] http://gazebosim.org/



```
Global Options
  Fixed Frame          map
  Background Color     48; 48; 48
  Frame Rate           30
Global Status: Ok
Grid                   ✓
RobotModel
TF                     ✓
LaserScan              ✓
Bumper Hit             ✓
Map                    ✓
Amcl Particle Swarm    ✓
Global Planner         ✓
Controller             ✓

Navigation:    active
Localization:  inactive
Feedback:      unknown

Poses remaining:      0
ETA:                  0 s
Distance remaining:   0.00 m
Time taken:           0 s
Recoveries:           0
```

Fig. 3: Cooperation scenario running in the 3D Virtual World: Gazebo.

use the same ROS2 domain. Unfortunately, this means the ROS2 topics used by the simulated robots would overlap by default, making it impossible to communicate with the robots separately. To avoid this overlap issue, separate namespaces had to be assigned to differentiate the topics of the robots. However, while this namespace approach is commonly used in ROS1, support for it is much more limited in ROS2, so the platform change required developing some code workarounds.

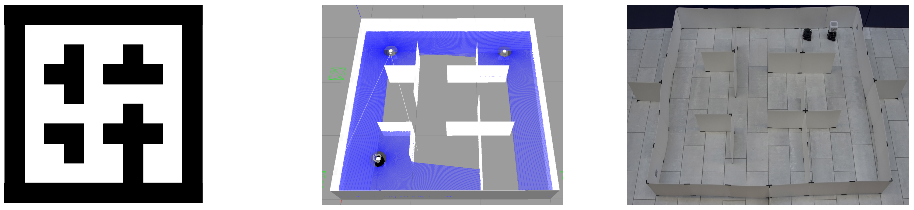
Currently we can run several Turtlebots in the same simulation in Gazebo. However increasing the number of robots increases the processing power required to run the simulation significantly. This limits the current simulation run on a virtualized Ubuntu 20.04 on a business laptop to 3 robots, and even then the simulation runs between 0.4–0.7 times of the normal speed.

Despite the initial challenges, the benefits of the 3D Virtual World approach are clear: The simulation environment can be edited and reset at will, although we are currently experiencing some Gazebo-specific bugs with these features. Troublesome aspects of the real world such as light conditions causing problems with QR code detection are not an issue in Gazebo, and delivery of objects can be simulated with ease by spawning and despawning items.

6.3 Building Example Scenario for Cooperation Development

After getting the first scenario in Gazebo working, we built an example scenario following the ideas explained in Section 5.1 in all three worlds (see Figure 4). After building the scenario we soon observed that the following method, which worked in our previous scenario, was bound to break in the 3D Virtual Worldand the Real Worldin the new scenario due to two main factors.

First, the interoperation of navigation and SLAM (simultaneous localisation and mapping) implementations in ROS2 were prone to create artifacts from



(a) Block world (b) Gazebo (c) Physical world

Fig. 4: An example enclosure in the three worlds: 2D Block World with iterative discrete time simulation, Gazebo 3D Virtual World, and Real World.

previous observations in the follower's navigation map. That is, the follower continuously observed spaces previously occupied by the guide still as occupied even though the guide had already moved away. This resulted in poor behaviour in tight spaces as the follower could not navigate around the artifacts, and the underlying implementations did not clear the artifacts if the follower was nearly stationary. Second, the sharp navigation angles of the guide accompanied with the thinness of the walls was causing problems for the following method, which was implemented by keeping the guiding robot's QR code as close to the center of the camera stream as possible. Due to this the follower repeatedly bumped into walls.

Fortunately, due to our three world development approach, we could still continue the development of other parts of the architecture and cooperation procedures even though a single platform (Turtlebot3 with ROS2) *Action* achieving single *Task* was temporarily broken. It was easy to verify that the problem was in the underlying action implementation and not in the core workings of the architecture. Moreover, we could begin to test how to circumvent these kind of problems in the future by making the following method more robust. For this, we could temporarily easen the properties of the 3D Virtual World, e.g. by making the corridors of the scenario wider and making the walls thicker.

7 Related work and Discussion

In this section, we discuss related research on architectures enabling autonomous robot cooperation, leveraging ontologies for forming an understanding of the cooperation possibilities and situations, task planning and decision making in the context of autonomous robot cooperation, as well as changing environment in our development approach.

7.1 Architectures for Autonomous Robot Cooperation

Autonomous robots cooperating in uncertain and constantly changing environments have been studied for many years. The general interest in the overall topic has spawned several research subfields, e.g., swarm robotics [1], collaborative robotics (cf. [9]) and unmanned autonomous vehicles (UAV) (cf. [13]).

We find that the closest works related to our work from the architectural perspective are related to tightly coupled multi-robot cooperation. For example, Chaimowicz et al. [6] have studied an architecture in which the key feature is flexibility, which enables changes in leadership and assignment of roles during the execution of a task. While their approach allows dynamical behavior, the cooperation is still tightly coupled. In our approach, each robot is expected to individually execute their tasks and then ask for help when needed. Hence, the cooperation is less tightly coupled. In addition, the aim is not to jointly execute predefined tasks but instead, enable the robots to model their environment and their peers so that they could independently form new cooperative plans and meet their personal goals.

The use case of transportation of objects has been studied by several researchers over the years, including Chaimowicz et al. [6]. Recently, Zhang et al. [20] as well as Manko et al. [12] have studied control architecture that is using deep reinforcement learning in the transportation of large or heavy objects with a particular focus on decentralized decision making. While these approaches have similarities to our work, our work aims to enable individual robots to fulfill their personal goals instead of the group's goal. Hence our architecture would likely not be well-suited for such tightly coupled cooperation. However, we can learn from their experiences on how they use deep learning technologies and Q-learning-based algorithms for training the robots to execute a tightly coupled task, and in the future, we could try a similar approach in our 2D Block World.

7.2 Ontologies for Cooperation

Ontologies have been widely used to make agents and robots understand the structures of the physical and social world around them [16, 3], and initiatives considering their usage to build robot collectives that can communicate and cooperate have been suggested before, e.g., RoboEarth [2]. In contrast to RoboEarth, where most of the reasoning happens in the cloud, cooperation understanding and planning in our architecture take place inside the individual robots. The robots do not share their world views in general as they are assumed to hold also information that should not be shared with others, such as maps of restricted areas or passwords. Instead, they will only exchange information relevant to the current situation and goals directly with each other. That said, cloud-based solutions, such as RoboEarth, could be integrated into the architecture as optional components.

Mainly due to the advent of IoT, ontologies prove to be an exciting starting point for robots to understand the world as the built environment is getting populated with intelligent devices capable of communicating with other computational actors. This means that, e.g., a door can be opened using software communication alone and does not have to rely on physical door manipulation, and that sensors and other IoT devices may send information of their physical composition, purpose, and capabilities using ontological representations. Especially on low-end robots this benefits cooperation, as the robot does not need to perceive these attributes from its raw sensor inputs such as camera streams.

7.3 Planning for Agents and Robots

Single robot planning may be approached from multiple perspectives. Two often used ones are heuristic shortest path search, such as the famous A* algorithm [10] and its dynamic counterparts, and solutions used for logical optimization problems, e.g. (weighted) maximum satisfiability solvers. The shortest path search provides (estimates) for moving from one node to another in a graph and aims to find the path of nodes with the shortest length, and logical optimization aims to find a (maximal or minimal) set of clauses that satisfy certain conditions. Dynamic shortest path algorithms fit well in environments where the robot may not fully understand its situation, e.g., the robot does not have a complete map or the map is bound to change, and logical optimization excels in cases where it is crucial to ensure the correctness of the solution beforehand.

However, our goal is to provide a planner that uses both logical verifications of the workflows through the fulfillment of each tasks' start and end conditions and a heuristic estimate of its execution resources through peer models and communication. Our approach differs from typical multi-robot task planning (see, e.g., [19]) in that one robot initiates the planning of the workflow phase (task decomposition), and it communicates, based on its peer models, with other robots to find suitable members to execute the tasks (task allocation).

7.4 Changing Environment and Three-world Develoment

In Real World, there are innumerable factors that can potentially affect the robots' ability to perform, such as the lighting conditions and cables on the floor. Of course, for our project's purposes, this would not seem a significant issue, as we can perform our tests also in carefully designed, controlled environments. However, this does not remove the fundamental issue of unexpected factors. How would this uncertainty be dealt with within a hypothetical practical environment? One possible approach would be introducing some degree of self-healing (see, e.g. Kounev et al. [11]) properties in the design, both in terms of the robots' performance and the cooperation context. Currently, extensive work on this aspect is beyond the scope of this project, however.

In contrast to the unpredictable Real World, the simulated 3D environments are inherently about control and thus easier to work with. Nevertheless, there can still be considerable effort to set up a simulation the desired way, as seen with the difficulties in deploying multiple robots simultaneously in Gazebo. It also became apparent that multi-robot simulation can involve substantial hardware requirements. Still, we have found the Gazebo 3D simulation fulfills its purpose satisfactorily as a platform where cooperative actions can be developed for Real World (ROS2) robots in a more controlled manner.

However, it could also be noted that while the usage of 3D simulation does simplify some aspects, designing *Actions* for the *Task Runtime* remains an endeavor that relies on detailed knowledge in leveraging a particular robot's inner workings. Contrary to how the primary interests of this project are in the dynamic and creative aspects of robot cooperation, there remains a nontrivial effort necessary in creating the actual units of implementation, *Actions*. For this purpose, ROS2 has been of great help with its high quality open source solutions for features such as robot navigation. Yet conversely, it became apparent that it can be even impractically laborious to implement new *Actions* where the ROS2 support falls short.

8 Conclusions

In this paper, we presented a new software architecture and development approach for diverse multi-robot cooperation. The core ideas of our approach include (a) improving the robot's understanding of its situational context for cooperation by peer modeling, (b) an ontology that enables the robots to understand and share information about the world and (c) three conceptually and operationally different worlds where the development of cooperative behaviour takes place: 2D Block World, 3D Virtual World, and Real World. The peer models enable the robots to take the capabilities and goals of their peers into account in their reasoning, the ontology can be used as a shared basis for communication and forming cooperation plans, and the different worlds serve different development purposes, e.g. testing different components and services, in the overall cooperation development process.

To test the feasibility of our architecture's core, we conducted stress testing in 2D Block World and verified that the ontology-based reasoning and planning is able to find and execute suitable cooperation plans in all the random scenarios encountered. This gives our architecture implementation a stable starting point to improve its other aspects, e.g. situational context awareness through peer modeling, which in turn should result in better suited cooperation plans.

Acknowledgments The work was supported by the Academy of Finland (project 328729).

References

1. Bayındır, L.: A review of swarm robotics tasks. Neurocomputing **172**, 292–321 (2016)
2. Beetz, M., Civera, J., D'Andrea, R., Elfring, J., Galvez-lopez, D.: Roboearth: a world wide web for robots. IEEE Transactions on Robotics and Automation **6**(14), 69–82 (2011)
3. Beetz, M., Beßler, D., Haidu, A., Pomarlan, M., Bozcuoğlu, A.K., Bartels, G.: Know rob 2.0 — a 2nd generation knowledge processing framework for cognition-enabled robotic agents. In: 2018 IEEE International Conference on Robotics and Automation (ICRA). pp. 512–519 (2018)

4. Berrocal, J., Garcia-Alonso, J., Galán-Jiménez, J., Murillo, J.M., Mäkitalo, N., Mikkonen, T., Canal, C.: Situational context in the programmable world. In: 2017 IEEE SmartWorld. pp. 1–8 (2017)
5. Castelfranchi, C.: Modelling social action for AI agents. Artificial Intelligence **103**(1), 157–182 (1998)
6. Chaimowicz, L., Sugar, T., Kumar, V., Campos, M.: An architecture for tightly coupled multi-robot cooperation. In: Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164). vol. 3, pp. 2992–2997 (2001)
7. Chaimowicz, L., Campos, M., Kumar, V.: Simulating loosely and tightly coupled multi-robot cooperation. In: V Brazilian Symposium on Intelligent Automation. The Electrical Engineering Department of the Federal University of Rio Grande do Sul (2001)
8. Edwardes, A., Burghardt, D., Neun, M.: Fipa communicative act library specification. foundation for intelligent physical agents. In: University of Maine: Orono. pp. 377–387. John Wiley & Sons (2000)
9. El Zaatari, S., Marei, M., Li, W., Usman, Z.: Cobot programming for collaborative industrial tasks: An overview. Robotics and Autonomous Systems **116**, 162–180 (2019)
10. Hart, P.E., Nilsson, N.J., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions on Systems Science and Cybernetics **4**(2), 100–107 (1968)
11. Kounev, S., Kephart, J., Milenkoski, A., Zhu, X.: Self-Aware Computing Systems. Springer International Publishing (2017)
12. Manko, S.V., Diane, S.A.K., Krivoshatskiy, A.E., Margolin, I.D., Slepynina, E.A.: Adaptive control of a multi-robot system for transportation of large-sized objects based on reinforcement learning. In: 2018 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus). pp. 923–927 (2018)
13. Mathew, N., Smith, S.L., Waslander, S.L.: Planning paths for package delivery in heterogeneous multirobot teams. IEEE Transactions on Automation Science and Engineering **12**(4), 1298–1308 (2015)
14. Mäkitalo, N., Linkola, S., Laurinen, T., Männistö, T.: Towards novel and intentional cooperation of diverse autonomous robots: An architectural approach. In: Proceedings of the Context-aware, Autonomous and Smart Architecture Workshop. pp. 1–10. CEUR Workshop Proceedings (2021)
15. Noy, N.F., McGuinness, D.L.: Ontology development 101: A guide to creating your first ontology (2001)
16. Olivares-Alarcos, A., Beßler, D., Khamis, A., Goncalves, P., Habib, M.K., Bermejo-Alonso, J., Barreto, M., Diab, M., Rosell, J., Quintas, J., et al.: A review and comparison of ontology-based approaches to robot autonomy. The Knowledge Engineering Review **34**, e29 (2019)
17. Rozanski, N., Woods, E.: Software systems architecture Second Edition : working with stakeholders using viewpoints and perspectives. Addison-Wesley (2012)
18. Thomas, D., Woodall, W., Fernandez, E.: Next-generation ROS: Building on DDS. In: ROSCon Chicago 2014. Open Robotics, Mountain View, CA (sep 2014)
19. Yan, Z., Jouandeau, N., Cherif, A.A.: A survey and analysis of multi-robot coordination. International Journal of Advanced Robotic Systems **10**(12), 399 (2013)
20. Zhang, T., Liu, G.: Design of formation control architecture based on leader-following approach. In: 2015 IEEE International Conference on Mechatronics and Automation (ICMA). pp. 893–898 (2015)