Online self-adaptive behavior of mobile robots through skill knowledge graphs

Lars Vanderseypen[1,2,*], Senne Van Baelen[1], Maria I. Artigas[1,2], Nikolaos Tsiogkas[3] and Herman Bruyninckx[1,2,4]

[1]Department of Mechanical Engineering, KU Leuven, Leuven, Belgium
[2]Flanders Make, Leuven, Belgium
[3]Department of Computer Science, KU Leuven, Leuven, Belgium
[4]Department of Mechanical Engineering, TU Eindhoven, Eindhoven, the Netherlands

Abstract

Autonomous mobile robots involved in traffic situations with other robots must be able to autonomously adapt their control and perception behavior to the overall traffic situation in those shared environments. This adaptation requires each robot to (1) be able to reconfigure its control and perception algorithms to the current situation around the robot, and (2) have the knowledge about which control and perception algorithms it has available at any instant and select the appropriate ones. This paper introduces the mereo-topology of a skill knowledge graph as the formal representation of the knowledge needed for the above-mentioned online self-adaptation, in addition to the knowledge about how to execute navigation maneuvers. It explains how and why a skill graph must connect to four other knowledge graphs, about tasks, resources, environments and object affordances. The approach is validated in a multi-robot navigation case with dynamic traffic layout inside an indoor corridor.

Keywords

Knowledge graphs, mobile robot navigation, runtime reconfiguration

1. Introduction

When involved in traffic situations with others, autonomous mobile robots (AMRs) must be able to adapt their behavior to the overall traffic situation. Robots realize their behavior by executing *skills*: compositions of *software activities* providing control, perception, world modeling and decision making functionality that work together to perform a specific *task* [1]. For AMRs, these are navigation tasks such as cruising, crossing a junction, overtaking, or parking. For robotic arms, these are manipulation tasks such as inserting, aligning, or polishing. In addition to these task-realizing activities, skills also run *monitoring* activities for, among other things, checking continuously if the robot's hardware and the activity software architecture are behaving correctly, or checking that the currently selected skill is still appropriate.

The functionality of all activities inside the skill must be *configured* based on the *task* the skill must perform, the *environment* in which the it executes, the *objects* it must deal with, and the *resources* it has available. In addition, each robot that shares resources and space with other robots must *coordinate* its own behavior with the other robots to make efficient and deterministic use of these resources and space. That coordination typically has multiple configuration options as well. For example, a threshold on what is a safe closest distance to other AMRs, or a level of assertiveness in navigation maneuvers that involve priority. More often than not, such configuration is done *manually* at *design*, *implementation* or *deployment time*. However, robots that autonomously adapt their behavior to their situation (that is, robots that are *situation aware*), should be able to configure their skills *themselves* at *runtime*.

RobOntics'25: International Workshop on Ontologies for Autonomous Robotics, December 10, 2025
*Corresponding author.
lars.vanderseypen@kuleuven.be (L. Vanderseypen); senne.vanbaelen@kuleuven.be (S. Van Baelen); mariaisabel.artigasalfonso@kuleuven.be (M. I. Artigas); nikolaos.tsiogkas@kuleuven.be (N. Tsiogkas); herman.bruyninckx@kuleuven.be (H. Bruyninckx)
0009-0000-7815-8519 (L. Vanderseypen); 0000-0003-4624-3107 (S. Van Baelen); 0000-0001-7433-559X (M. I. Artigas); 0000-0003-2842-7316 (N. Tsiogkas); 0000-0003-3776-1025 (H. Bruyninckx)
© 2025 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073
published 2026-02-15

CEUR-WS.org/Vol-4169/paper2.pdf

This paper presents a methodology to allow robots to configure and coordinate their skills by *reasoning* on *knowledge graphs*. Due to size constraints, only the *mereo-topological* levels [2] of the ontology are discussed. A knowledge graph has *entities* with a set of *properties*, connected through *edges* with the edge label indicating the *meaning* of the relation between the connected entities. The *skill graph* is key in the presented approach, because it connects knowledge about the robot's behavior to knowledge of the task, environment, resources and object affordances.

Two software activities are proposed to support runtime skill (re)configuration: (1) a skill *selection* activity which is responsible for *querying* the skill graph, at runtime, to find which skill configurations a particular robot can use to perform a particular task in a particular environment, and (2) a skill *execution* activity which is responsible for *(re)configuring* and *coordinating* the running activities inside the robot's software architecture, at runtime, based on the selected skill configuration.

The presented methodology is validated using a multi-robot navigation scenario where a *corridor* area is shared between different AMR agents. The corridor always has an active *traffic layout* which is mediated by a *traffic layout mediator*, that can suggest traffic layout changes based on the number of robots that want to use the corridor. It is shown that by utilizing the proposed skill graph and activities, a robot is able to autonomously adapt its control behavior to the active traffic layout.

2. Related work

Self-adaptation of robot software has been studied in a number of works in the literature [3]. A common pattern is to divide the robot's software into a *managed* and *managing* subsystem, where the managed subsystem is responsible for the functional behavior of the robot, and the managing subsystem is responsible for monitoring the execution of the managed subsystem and reconfiguring it when needed.

Managing subsystems are often, but not always, implement using a MAPE-K loop [4], which consists of four different steps: (1) *monitoring* the data produced by the managing subsystem, (2) *analyzing* the data to determine whether a reconfiguration is necessary, (3) *planning* the reconfiguration, and (4) *executing* it. Each of these steps makes use of a central *knowledge base*, which contains models to support the runtime reconfiguration. While the MAPE-K concept is not used in this work, the presented methodology could be used as a way of structuring the knowledge base in a software system that does.

Previous work presenting models for runtime reconfiguration typically focuses on modeling the components in the robot's software architecture. TOMASys [5] is a metamodel to represent the software components of a robot and their possible configurations. These models are used at runtime by a managing subsystem, the metacontroller, to reconfigure a robot's software architecture to recover from faults such as sensor failure.

Likewise, MARTE::ARM-Variability is a UML-based metamodel for representing robot software variability [6]. This model focuses on abstracting the reconfiguration mechanisms supported by the robot software, thus creating a standard interface between the managing and managed subsystems.

The ROSA model presented in [7] represents the robots software components and a set of reconfiguration plans, and is used to support task-and-architecture co-adaptation [8, 9]. This allows the robot to adapt its task plan in case no valid software architecture reconfiguration can be found.

This work departs from a similar model of the skill's software architecture and connects them to the knowledge of particular situations which the robot can handle, and shows how the robot's software can be reconfigured when the robot's situation changes.

3. Methodology

Robot skills are compositions of concurrently running software activities for perception, motion control, monitoring and decision making that work together to realize a specific task. For robots to (re)configure their skill activities at runtime themselves, they must know how different skill configurations are related to different tasks, resources, environments and objects. This section introduces (1) the *knowledge graphs* needed to formalize this knowledge, and (2) a skill *selection* activity to query skill configurations based

on a given situation, and a skill *execution* activity to configure and coordinate the skill's activities at runtime.

3.1. Skill graph: from knowledge relations to model instances

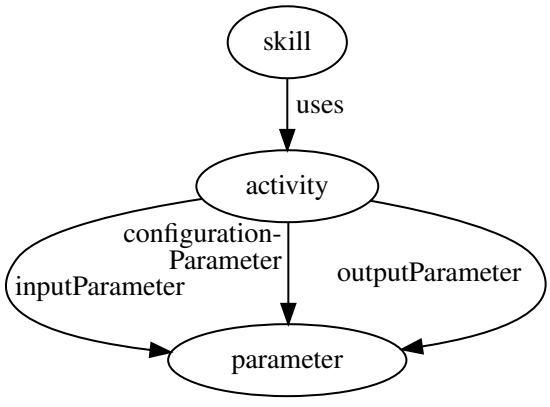
The *skill software architecture model* is shown in Fig. 1a, which defines the *software* relations between *skills*, their *activities* and the activity's *parameters*. We define three possible relations between activities and parameters: *input parameter*, *output parameter*, and *configuration parameter*. Input and output parameters represent the input and output data streams that are consumed and produced by an activity, respectively.

Configuration parameters influence the algorithms inside an activity that compute the input-output relations and must be configured before the skill can be executed. The value of configuration parameters depends on the robot's situation. Therefore, when the robot's situation changes, through a change in task, environment or resources, then the value of the configuration parameters should be changed accordingly. In contrast, the value of input parameters can change every time the activity is run.

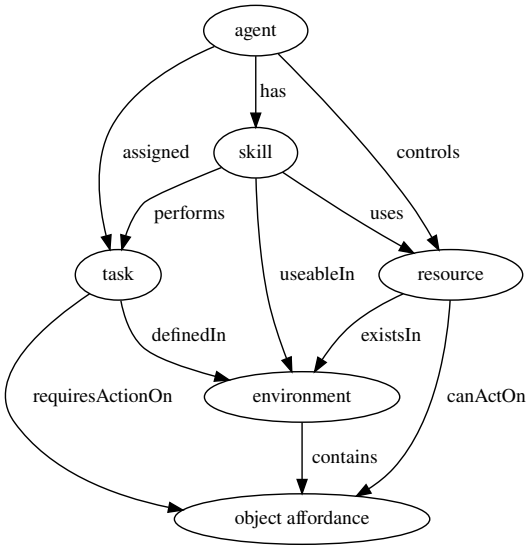
Changing a configuration parameter's value often requires a coordination to be executed to not disturb the determinism of the input-output relations of the running activities. Input, output, and configuration parameters are modeled as relations, as a parameter can be an input parameter in one activity while it is an output parameter in another.

The *skill behavior model*, shown in Figure 1b, links a skill to the agents that execute skills, the tasks they perform, the resources they use, the environments they can be used in and the object affordances they act on. The behavior model is formed as a *composition* of *sub-graphs*, represented by the nodes in Figure 1b. The agent graph represents agents, which are software entities that autonomously perform assigned tasks by executing skills. Each agent controls a set of software resources (activities), and hardware resources that are either *atomic* (sensors and actuators) or *composed* (robot body composed of sensors and actuators). For example, a robot is considered an agent because it autonomously performs tasks by controlling its hardware resources to interact with the environment.

Environments are represented using *semantic maps* [10], which add semantic labels on top of a geometric map. These labels add additional meaning on top of the base map such as: areas, traffic



(a) Software architecture model of a skill. A skill uses activities that contain input, output and configuration parameters. The configuration parameters must be configured before the skill can be executed.



(b) Behavior model of a skill. The nodes represent *sub-graphs* of agents, skills, resources, tasks, environments, and object affordances.

Figure 1: Top-level skill knowledge graphs for software (a) and behavior (b).

layouts and objects. Each object in the semantic map has object affordances, which define the actions that can be taken on them by resources. Semantic maps can be dynamic [11], meaning that the map can be updated at runtime based on changes in the environment. During the validation case in Section 4, the semantic map is updated to change the active traffic layout in the corridor.

A task is specified as a set of actions that must be taken on specific objects to reach a goal. The object affordances therefore link the environment, task and resources together.

The behavior skill model is a *metamodel* in the sense that each of the sub-graphs in the behavior skill model can have different *instances*. Each instance of the sub-graphs represents a *model* of a particular type of agent, skill, environment, etc. For example, two different instances of the environment graph could be a *corridor* graph and a *junction* graph, representing the two types of areas.

A concrete instance of the behavior skill model, called a *situation graph*, defines how a particular software skill model is linked to particular task, resource, environment, agent and object affordance models. A situation graph for the *corridor navigation situation* for the validation case in Section 4 is shown in Figure 2.

A navigation skill is modeled as in Figure 2a, which defines that this particular skill uses three activities: a lane control, a lidar localization, and a position monitoring activity. Each activity has its own set of input, output and configuration parameters. The skill is usable in corridors and can perform navigation tasks, which define what areas to navigate in order to reach a goal area as represented in Figure 2e. The skill can be used by a robot whose agent graph is shown in Figure 2d.

Beside the navigation skill, the robot controls hardware resources whose model is shown in Figure 2c. These contain a lidar sensor, which can observe wall surfaces and produces point clouds, and a wheel drive, which can navigate lanes and consumes velocity setpoints. The corridor itself is modeled as in

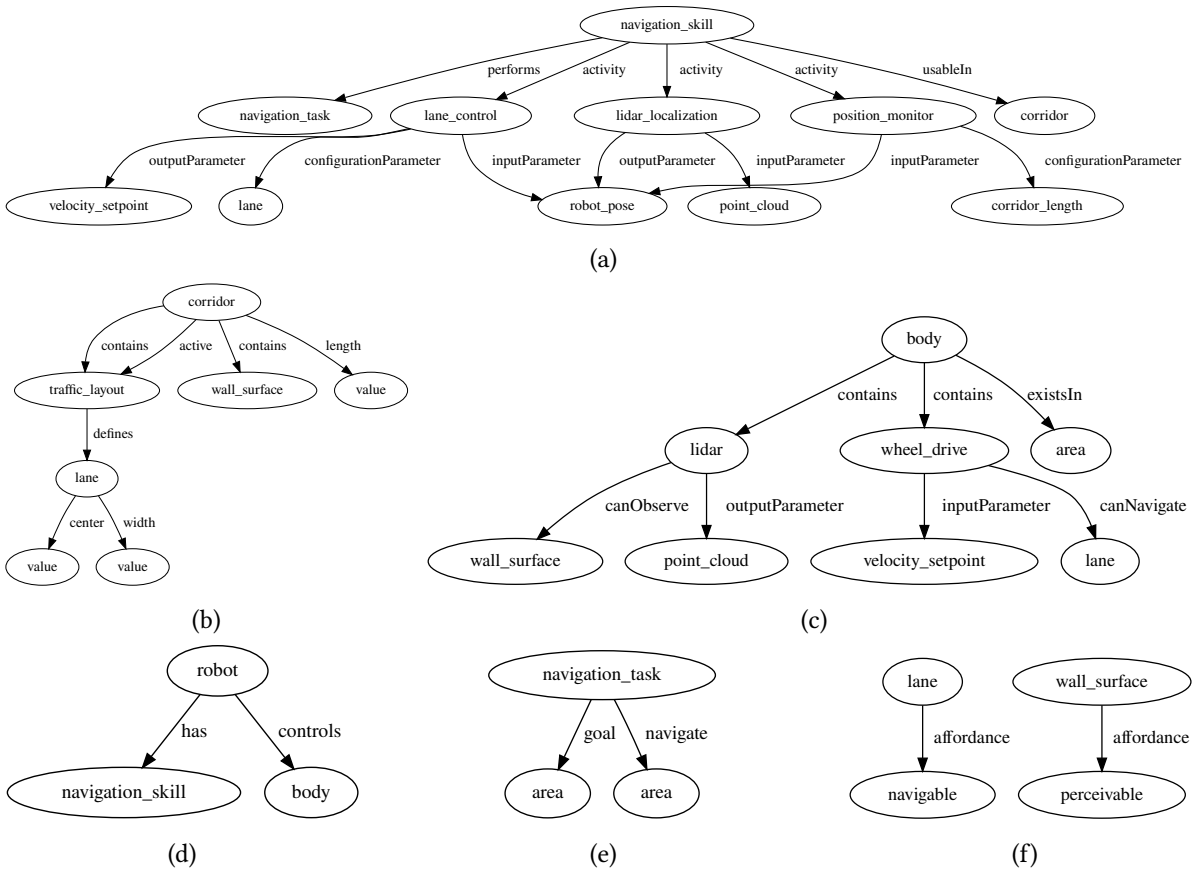


Figure 2: The sub-graphs of the corridor navigation situation graph *model*, representing: (a) the navigation skill, (b) the corridor, (c) the robot resources, (d) the robot agent, (e) the navigation task, and (f) the object affordances. Nodes with the same label in different sub-graphs connect the sub-graphs in the situation graph.

Figure 2b, which defines that the corridor has a length, and contains wall features and traffic layouts which define lanes. The fact that lanes are navigable and wall surfaces are perceivable is defined in the object affordance graph in Figure 2f.

The sub-graphs in the situation graph can again be instantiated to define a *situation graph instance*. In the validation case, the corridor navigation situation graph is instantiated, as shown in Figure 4 in Section 4, which defines instances of the sub-graphs in Figure 2. For example, the corridor graph is instantiated by assigning a numeric value to the corridor length, and defining concrete traffic layouts in the corridor based on the physical corridor the robots have to navigate, as shown in Figure 4b.

To summarize, there are three layers of abstraction. The behavior skill model defines the metamodel, as shown in Figure 1b. The situation graph defines the model, as shown in Figure 2, which conforms to the metamodel in the behavior skill model. The situation graph instances define instances which conform to the model in the situation graph, as shown in Figure 4 in Section 4.

3.2. Online skill (re)configuration: from model instances to executable software

To (re)configure a skill at runtime, two activities are proposed: (1) a skill *selection* activity and (2) a skill *execution* activity. When an agent is given a new task, the skill selection activity uses a skill selection query on the knowledge graph to find what skills the agent has to perform its new task in its current environment. The query starts from the relevant agent, task and environment graphs and traverses the edges defined in the skill behavior model in Figure 1b to check if a skill can be found that uses the agent's resources to perform the given task in the given environment. If such a skill can be found, the activity performs a skill configuration query.

The skill configuration query traverses the graph to find values for the different parameters defined in the given skill. The query traverses the edges defined in the situation graph which links the agent's current situation to the given skill. For example, for the navigation skill in Figure 2a, to find a value for the lane parameter in the lane control activity, the query would traverse from the corridor node to the traffic layout node to the lane node. Once an executable skill configuration is found, it is send to the skill execution activity.

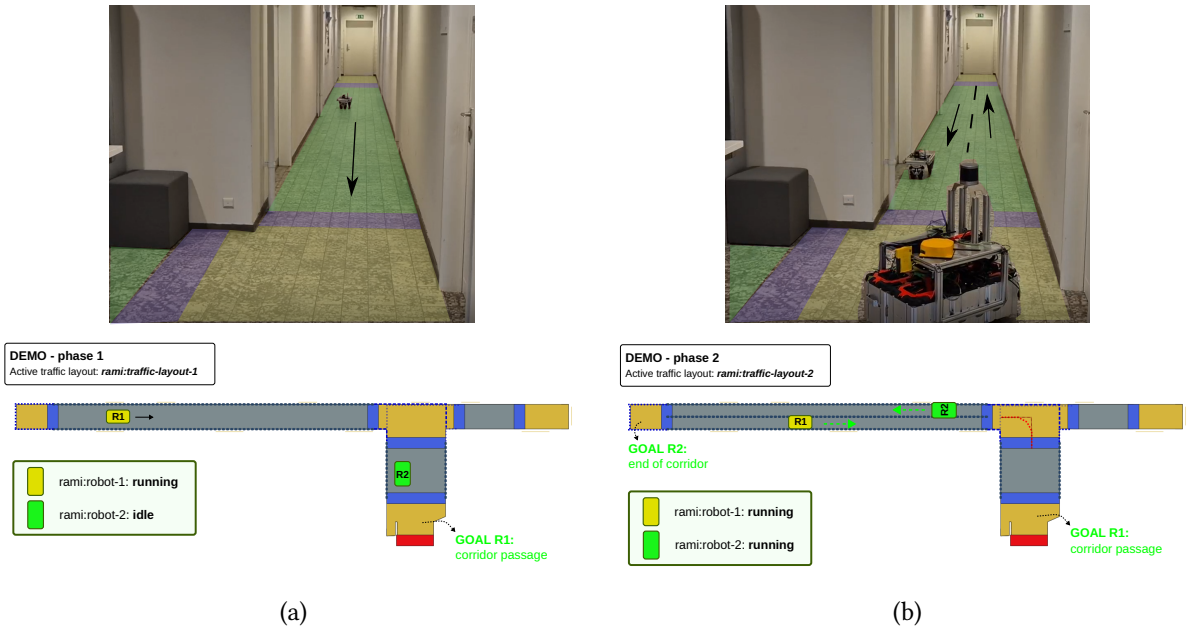
The skill execution activity *(re)configures* and *coordinates* the *running* activities in an agent's software architecture to execute a desired skill. There are two reconfigurations which the software architecture should support: (1) enabling and disabling activities, and (2) changing the values of activity parameters at runtime. The software activities used during the validation in Section 4 were designed according to the "5C" paradigm [1, 12], and support both types of reconfiguration. During the validation, reconfigurations of type 2 are performed, but not of type 1.

Skill execution is *preempted* whenever there is a change in the task, the resources, or the environment. When this occurs, the skill selection activity first performs a skill configuration query to determine whether the currently executing skill can be reconfigured according to the new situation. If this is the case, the skill execution activity can reconfigure the software architecture. Otherwise, a skill selection query is performed to search for a new skill to execute. If no such skill can be found, task execution should be aborted.

The two presented activities show a form of *separation of concerns* in the proposed methodology. The skill selection activity only needs to know *what* skill configuration are valid for a particular agent and situation, without needing to know *how* those skills are executed in a particular software architecture. The skill execution activity does need to know *how* a skill should be executed so it can (re)configure and coordinate the activities of a particular software architecture. Therefore the proposed activities consist of a software architecture *independent* skill selection activity, and a software architecture *dependent* skill execution activity.

4. Validation

The validation of the proposed methodology is done using the following use case. Two mobile robots must each complete a navigation task which requires them to drive through the same corridor. The



(a) (b)

Figure 3: (a) One robot navigates alone in a corridor with a *one-lane* layout. (b) The corridor's layout has changed to a *two-lane* version, to accommodate a second robot to enter the same corridor. The top figures are snapshots from the demonstration; the bottom ones represent the corresponding status of the semantic map of the T-junction situation used in the demonstration.

corridor is divided into lanes by a traffic layout, which is mediated by an external *traffic layout mediator*, that can change the *active* traffic layout based on the number of robots that want to use the corridor.

Initially only a single robot (robot 1) is using the corridor, as shown in Figure 3a. The active traffic layout defines a single lane, allowing robot 1 to use all available space in the corridor. After a period of time, a second robot (robot 2) approaches the corridor, but cannot enter because of the active traffic layout. Therefore, it contacts the traffic layout mediator to request a layout change. The mediator then contacts robot 1, proposing to change to a two lane layout. Robot 1 must then decide whether it can reconfigure its executing skill to the proposed traffic layout. If this is the case, the traffic layout is changed and robot 1 reconfigures its skill, such that robot 2 can enter, as shown in Figure 3b.

A knowledge graph was developed containing an instance of the corridor navigation situation graph, whose sub-graphs were shown in Figure 2. The instance of the agent model of robot 1 is shown in Figure 4c. The agent now has an assigned navigation task, whose graph instance is shown in Figure 4e. The task defines only a single corridor that must be navigated to reach a junction. The instance of the corridor model is shown in Figure 4b. The corridor length has been instantiated with a value, and two traffic layouts are defined in the corridor.

Traffic layout 1 corresponds to the traffic layout in Figure 3a, and defines one lane. Traffic layout 2 corresponds to the traffic layout in Figure 3b, and defines two lanes. Traffic layout 1 is defined as active, corresponding to the situation at the beginning of the demonstration.

The instance of the navigation skill graph is shown in Figure 4a. The configuration parameters in the skill's activities are now connected to values that come from the other sub-graphs. For example, lane 0, lane 1 and lane 2 in the navigation skill are the ones defined in the corridor instance in Figure 4b. Likewise, the resource graph instance in Figure 4d is now connected to the lanes from the corridor graph.

An implementation of the skill selection, and skill execution activities was made to support the runtime reconfiguration of robot 1's navigation skill. At the start of the demonstration, robot 1 receives its navigation task and the skill selection activity performs a skill selection query to find what skill it can use to navigate the corridor.

The query returns the navigation skill of Figure 4a. The skill selection activity then performs a skill

configuration query to find an executable instance of the navigation skill. At the start of the experiment traffic layout 1 is active, that knowledge is used to query for the lane reference. The resulting query configuration is send to the skill execution activity which configures the robot's software architecture. This results in the behavior shown in the top panel of Figure 3a.

When robot 1 receives a message from the traffic layout mediator to update to the two lane layout, the skill selection activity is triggered to determine whether a skill configuration can be found for that layout. A skill configuration query is used again, this time looking for a valid configuration for traffic layout 2. The query returns a valid configuration, and robot 1 accepts the traffic layout change proposal.

When the traffic layout is changed, the skill selection activity sends the new configuration to the skill execution activity. This activity reconfigures robot 1's software architecture. This results in the behavior shown in the top panel of Figure 3b.

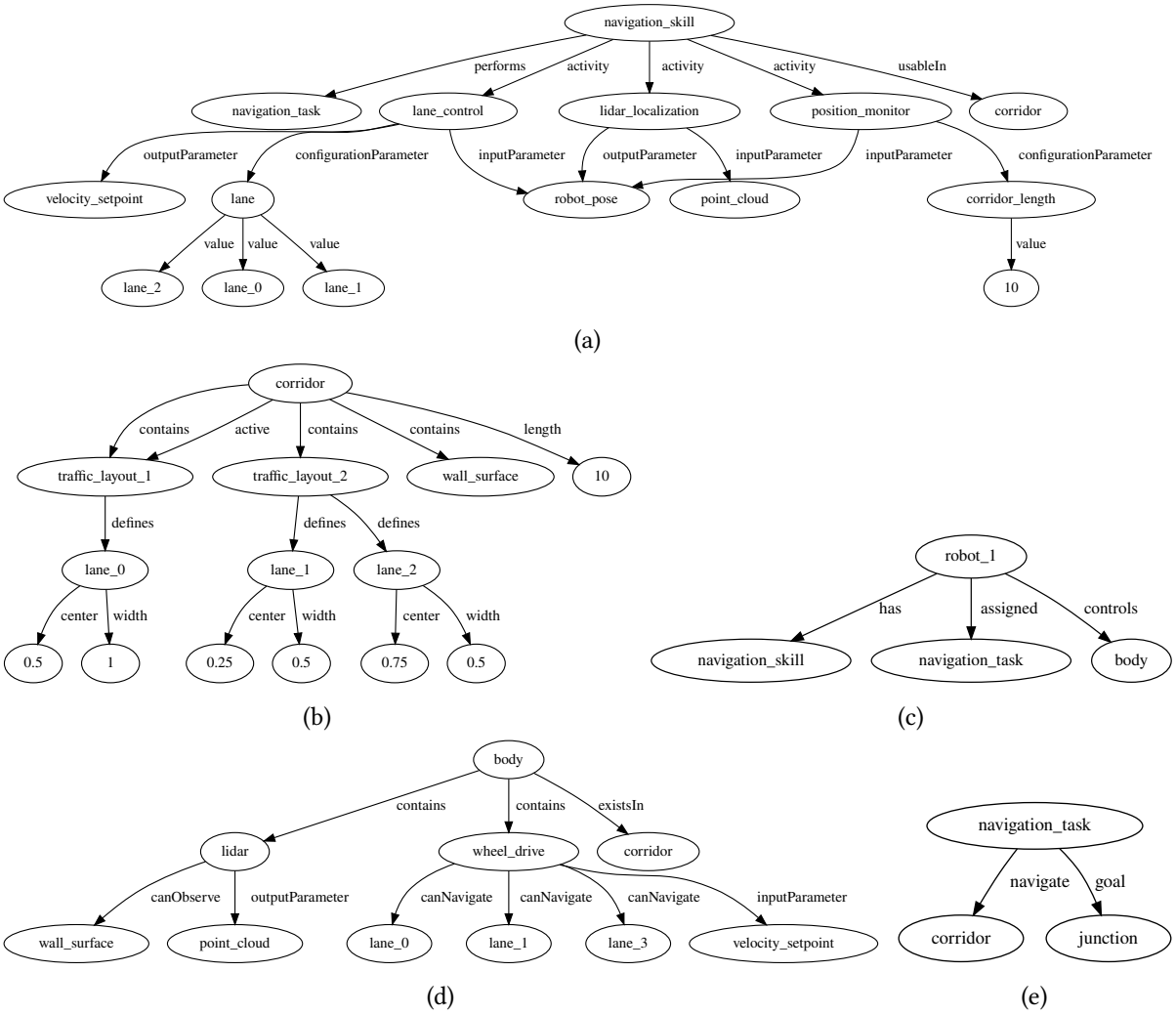


Figure 4: The sub-graphs of the situation graph *instance*, representing: (a) the navigation skill instance where `value` relations point to nodes in different sub-graphs, (b) the corridor instance with concrete traffic layouts and a value for the corridor length assigned, (c) the agent instance, (d) the resource instance with the wheel drive and lidar linked to the lane and wall features in the corridor graph, (e) the task instance defining that robot 1 needs to navigate the corridor to reach the junction.

5. Conclusion

This paper proposed to use *skill knowledge graphs* to support runtime (re)configuration of robot skills, where a skill is the composition of concurrently running software activities that perform a specific task.

The major contribution is the *mereo-topology* [2] of the knowledge representation: the mereology of the relevant terms, and the topology of the structural relations between these terms. The full semantics of all terms and relations is beyond the scope of this paper, and the subject of ongoing work.

The *skill software architecture* and *skill behavior graphs* are at the core of the mentioned separation. The skill software architecture graph defines what software activities must run to execute a specific skill, and what parameters must be configured. The skill behavior graph is a metamodel that links the skill representation to sub-graphs of agents, tasks, resources, environments and object affordances.

Instances of the behavior skill graph, called *situation graphs*, define how a particular skill is linked to models of particular task, agent, environment, resource and object affordance types. Situation graph *instances* are defined by instantiating the sub-graphs in the situation graph for a concrete application. These sets of graphs contain the relevant knowledge to find configurations of the robot's activities, which was demonstrated in the validation case.

Two activities, the skill selection activity and skill execution activity, where proposed that query the knowledge base at runtime to find executable skill configurations, and reconfigure a robot's software architecture based on the selected skill.

A validation case was presented where the traffic layout inside a corridor was changed from a one-lane layout to a two-lane layout based on the number of robots that needed access to the corridor. In the validation case, the presented graphs allowed the skill selection activity to select the right control activity configuration when the traffic layout was changed from a one-lane to a two-lane layout. The skill execution activity was able to reconfigure the robot's control activity according to the selected configuration.

The advantage of the presented topology is a pragmatically scalable *separation of concerns*: (1) specialists in the different sub-knowledge domains can *independently* develop and extend the sub-graphs relevant to their domain, and (2) the evolving sub-knowledge domains remain *composable* by the topology provided in this paper.

A limitation of the current approach is that the robot can only configure skills in *known* situations, since it can only query the situation models in its knowledge base. If a situation were to occur for which there is no situation model available, then no skill configuration can be found. However, chances are low that the available knowledge and software activities would suffice to deal with those situations.

Future work will focus on formalizing an ontology based on the presented approach, and on exploring how the methodology and software activities can be extended to support multi-agent skill execution, where different agents must coordinate their skill execution for a particular task.

Acknowledgments

This work was supported by the Flanders Make project Sitanav (*Situational Aware Navigation and Mapping*), and by the Agribot project with grant agreement id: 101183158.

Declaration on Generative AI

The author(s) have not employed any Generative AI tools.

References

[1] H. Bruyninckx, Situational aware robotic and cyber-physical multi-agent systems, Technical Report, KU Leuven, Department of Mechanical Engineering, 2025.

[2] P. Borst, H. Akkermans, J. Top, Engineering ontologies, International Journal on Human-Computer Studies 46 (1997) 365–406.

[3] E. Alberts, I. Gerostathopoulos, I. Malavolta, C. H. Corbato, P. Lago, Software architecture-based self-adaptation in robotics, Journal of Systems and Software 219 (2025) 112258.

[4] J. Kephart, D. Chess, The vision of autonomic computing, Computer 36 (2003) 41–50.

[5] C. Hernández, J. Bermejo-Alonso, R. Sanz, A self-adaptation framework based on functional knowledge for augmented autonomy in robots, Integrated Computer-Aided Engineering 25 (2018) 157–172. doi:10.3233/ICA-180565, publisher: SAGE Publications.
[6] D. Brugali, Modeling variability in self-adapting robotic systems, Robotics and Autonomous Systems 167 (2023) 104470.
[7] G. Rezende Silva, J. Päßler, S. L. Tapia Tarifa, E. B. Johnsen, C. Hernández Corbato, Rosa: a knowledge-based solution for robot self-adaptation, Frontiers in Robotics and AI 12 (2025).
[8] V. Braberman, N. D'Ippolito, J. Kramer, D. Sykes, S. Uchitel, An extended description of morph: A reference architecture for configuration and behaviour self-adaptation, in: R. de Lemos, D. Garlan, C. Ghezzi, H. Giese (Eds.), Software Engineering for Self-Adaptive Systems III. Assurances, Springer International Publishing, Cham, 2017, pp. 377–408.
[9] J. Cámara, B. Schmerl, D. Garlan, Software architecture and task plan co-adaptation for mobile service robots, in: Proceedings of the IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS '20, Association for Computing Machinery, New York, NY, USA, 2020, p. 125–136. URL: https://doi.org/10.1145/3387939.3391591. doi:10.1145/3387939.3391591.
[10] A. Nüchter, J. Hertzberg, Towards semantic maps for mobile robots, Robots and Autonomous Systems 56 (2008) 915–926.
[11] S. Van Baelen, G. Peeters, H. Bruyninckx, P. Pilozzi, P. Slaets, Dynamic semantic world models and increased situational awareness for highly automated inland waterway transport, Frontiers in Robotics and AI 8 (2022) 739062:1–25.
[12] D. Vanthienen, M. Klotzbuecher, H. Bruyninckx, The 5c-based architectural composition pattern: lessons learned from re-developing the itasc framework for constraint-based robot programming, JOSER: Journal of Software Engineering for Robotics 5 (2014) 17–35.