UNIVERSITY OF BREMEN
INSTITUTE FOR
ARTIFICIAL INTELLIGENCE

Ontological Representation of Activity Context for Flexible Robot Task Execution

Daniel Beßler

Vollständiger Abdruck der vom Fachbereich Mathematik und Informatik der Universität Bremen genehmigten Dissertation zur Erlangung des Grades

Doktor-Ingenieur (Dr. Ing.)

Vorsitzender:	Prof. John A. Bateman, PhD *Universität Bremen*
1. Prüfer:	Prof. Michael Beetz, PhD *Universität Bremen*
2. Prüfer:	Prof. Frank van Harmelen *Vrije Universiteit Amsterdam*
Beisitzer:	Dr.-Ing. Robert Porzel

Die Dissertation wurde am 30.05.2022 bei der Universität Bremen eingereicht und durch den Prüfungsausschuss am 15.09.2022 angenommen.

Abstract

It has been demonstrated many times that modern robotic platforms can generate competent bodily behavior comparable to the level of humans. However, the implementation of such behavior requires a lot of programming effort, and is often not feasible for the general case, i.e., regardless of the situational context in which the activity is performed. Furthermore, research and industry have an enormous need for intuitive robot programming. This is due to the high complexity of realizing an integrated robot control system, and adapting it to other robots, tasks and environments. The challenge is how a robot control program can be realized that can generate competent behavior depending on characteristics of the robot, the task it executes, and the environment where it operates. One way to approach this problem is to specialize the control program through the context-specific application of abstract knowledge.

In this work, it will be investigated how abstract knowledge, required for flexible and competent robot task execution, can be represented using a formal ontology. To this end, a domain ontology of robot activity context will be proposed. Using this ontology, robots can infer how tasks can be accomplished through movements and interactions with the environment, and how they can improvise to a certain extent to take advantage of action possibilities that objects provide in their environment. Accordingly, it will be shown that parts of the context-specific information required for flexible task execution can be derived from broadly

applicable knowledge represented in an ontology. Furthermore, it will be shown that the domain vocabulary yields additional benefits for the representation of knowledge gained through experimentation and simulation. Such knowledge can be leveraged for learning, or be used to inspect the robot's behavior. The latter of which will be demonstrated in this work by means of a case study.

Zusammenfassung

Es wurde vielfach prominent demonstriert, dass moderne Roboterplattformen kompetentes körperliches Verhalten erzeugen können, welches mit dem Niveau von Menschen vergleichbar ist. Die Implementierung eines solchen Verhaltens erfordert jedoch einen hohen Programmieraufwand und ist für den allgemeinen Fall, d.h. unabhängig vom situativen Kontext in dem die Aktivität ausgeführt wird, oft nicht realisierbar. Forschung und Industrie haben desweiteren einen enormen Bedarf an intuitiver Roboter Programmierung. Dies liegt an der hohen Komplexität ein integriertes Robotersteuerungssystem zu realisieren und es an andere Roboter, Aufgaben und Umgebungen anzupassen. Die Herausforderung besteht darin, wie ein Robotersteuerungsprogramm realisiert werden kann, das in Abhängigkeit von Eigenschaften des Roboters, der von ihm ausgeführten Aufgabe und der Umgebung in der er arbeitet, kompetentes Verhalten erzeugen kann. Eine Möglichkeit dieses Problem anzugehen besteht darin, das Steuerprogramm durch die kontextspezifische Anwendung von abstraktem Wissens zu spezialisieren.

In dieser Arbeit wird untersucht, wie derartiges abstraktes Wissen, welches für eine flexible und kompetente Aufgabenerledigung benötigt wird, durch eine formale Ontologie repräsentiert werden kann. Zu diesem Zweck wird eine Domänenontologie des Roboteraktivitätskontexts vorgeschlagen. Mithilfe dieser Ontologie können Roboter ermitteln, wie Aufgaben durch Bewegungen und Interaktionen mit der Umgebung bewältigt werden können und wie sie bis zu einem gewissen Grad

improvisieren können um Handlungsmöglichkeiten zu nutzen, die Objekte in ihrer Umgebung bieten. Dementsprechend wird gezeigt, dass Teile der für eine flexible Aufgabenausführung erforderlichen kontextspezifischen Informationen aus breit anwendbarem Wissen abgeleitet werden können, welches in einer Ontologie repräsentiert wird. Weiterhin wird gezeigt, dass das Domänenvokabular zusätzlichen Nutzen für die Darstellung von Wissen bringt, welches durch Experimentieren und Simulation gewonnen wurde. Solches Wissen kann zum Lernen genutzt oder verwendet werden um das Verhalten des Roboters zu inspizieren. Letzteres wird in dieser Arbeit anhand einer Fallstudie demonstriert.

Acknowledgments

Throughout the time as a doctoral student I greatly enjoyed working together with many talented people in the scope of numerous exciting research projects. In the following, I want to express my highest acknowledgments to the people who helped me during the effort of writing this thesis.

First of all, I would like to express my sincere gratitude to my supervisor, Michael Beetz, for giving me the opportunity to write my doctoral thesis in his robotics lab, and to investigate and work on my own research ideas during that time. His visionary ideas and outstanding expertise have been a great source of inspiration, and his support has helped me a lot in writing this thesis.

I would further like to thank all of my colleagues from the Institute for Artificial Intelligence at the University of Bremen for contributing to a productive and inspiring atmosphere at the institute, and for the enjoyable time we spent outside the lab. I would especially like to thank Mihai Pomarlan for being a great colleague during the many collaborative projects we have worked on together, Asil Kaan Bozcuoglu for being a great office mate over the years, and Hagen Langer, Mona Abdel-Keream, Abhijit Vyas, Sascha Jongebloed and Kaviya Dhanabalachandran for their input regarding this thesis.

I would also like to thank all colleagues with whom I got the honor to work with in the scope of research projects. I would especially like to thank Robert Porzel for his continuous support, and for the many discussions that helped me a lot developing ideas, and putting them onto paper. It has further been a

great experience to work together with researchers in the scope of an IEEE work group. My special thanks goes to Alberto Olivares-Alarcos with whom it was a particularly great experience to work together. Finally, I would like to express my sincere gratitude to Maria Hedblom and Daniel Leidner for their detailed comments on this work, which helped me a lot in the final steps leading up to the submission of my doctoral thesis.

October 10, 2022,
Daniel Beßler

This work has received funding from the German Research Foundation (DFG) through the Collaborative Research Center (CRC) 1320 EASE (Everyday Activity Science and Engineering), from the German Federal Ministry for Economic Affairs and Energy (BMWi) project Knowledge4Retail (grant number 01MK20001G), and from the European Union Seventh Framework Programme FP7 project SAPHARI (grant number 287513).

Contents

1 Introduction **1**

1.1 Logical Formalization of Robot Problems 6

1.2 Ontologies in Robot Knowledge Bases 9

1.2.1 Query Answering 10

1.2.2 Abstract Knowledge 10

1.2.3 Experience Knowledge 11

1.3 Contributions 13

1.4 Outline 18

2 Ontology-based Approaches to Robot Autonomy **21**

2.1 Basics of Ontology and Autonomous Robotics 22

2.1.1 Types of Ontologies 22

2.1.2 Logical Foundation of Ontologies 26

2.1.3 The Web Ontology Language 27

2.1.4 Autonomous Robotics 30

2.2 A Classification of Ontologies for Autonomous Robots 32

2.2.1 Ontology Scope 32

2.2.2 Reasoning Scope 39

2.2.3 Application Domain Scope 44

2.3 Ontologies to support Robot Autonomy 45

2.3.1 Discussion of Frameworks and Projects 47

2.3.2 Comparison of Frameworks and Projects 53
2.4 Discussion 64
2.5 Conclusion 68

3 An Ontological Framework for Robot Knowledge 69
3.1 The Socio-physical Model of Activities (SOMA) 70
3.1.1 Related Work 72
3.1.2 Overview 73
3.1.3 Object Representation 76
3.1.4 Event Representation 79
3.1.5 Evaluation 85
3.1.6 Summary 87
3.2 The Descriptive Affordance Ontology 88
3.2.1 Related Work 90
3.2.2 Formalization and Implementation 91
3.2.3 Evaluation 100
3.2.4 Summary 107
3.3 Conclusion 108

4 Ontology-enabled Planning in Robot Manufacturing 109
4.1 Flexible Assembly Planning 110
4.1.1 Related Work 112
4.1.2 Modeling Assemblages 113
4.1.3 Reasoning Methods 117
4.1.4 Materialization of Restrictions 122
4.1.5 Assembly Actions 130
4.1.6 Evaluation 132
4.1.7 Summary 136
4.2 Assembly Planning with Geometric Relations 137
4.2.1 Related Work 139
4.2.2 Overview 140
4.2.3 Modeling Assembly Tasks 141
4.2.4 Reasoning Methods 147
4.2.5 Planner Extensions 148

4.2.6 Evaluation . . . 151
4.2.7 Summary . . . 153
4.3 Conclusion . . . 154

5 Ontology-enabled Inspection of Robot Behavior **155**
5.1 Related Work . . . 157
5.2 Representation of Experience Knowledge . . . 158
5.2.1 Ontological Characterization of Experience Data . . . 160
5.2.2 Ontological Representation of Actions . . . 162
5.2.3 Ontological Representation of the pHRI Domain . . . 164
5.3 Monitoring Experiences . . . 166
5.3.1 Storage Infrastructure . . . 167
5.3.2 Acquisition Infrastructure . . . 167
5.3.3 Task Execution . . . 169
5.4 Querying Experiences . . . 170
5.4.1 Querying Language for Experience Data . . . 172
5.4.2 Construction of Data Sets for Machine Learning . . . 176
5.5 Evaluation . . . 178
5.5.1 Inspection of Safety-Relevant Situations . . . 178
5.5.2 Classification of Safety-Relevant Situations . . . 181
5.6 Conclusion . . . 182

6 Discussion and Conclusion **183**
6.1 Summary . . . 183
6.2 Discussion . . . 185
6.3 Conclusion . . . 191

References **193**

Appendix A Prior Publications **219**

List of Figures

1.1 Context-specific robot behavior 2
1.2 Knowledge-based contextualization 4
1.3 Different types of robot problems 7
1.4 Knowledge-based query answering 9

2.1 Different classifications of ontologies 24

3.1 Physical and social context in SOMA 71
3.2 The modular organization of SOMA 75
3.3 An example of how object classes are represented 76
3.4 An example of how plans are represented 80
3.5 Object selection through affordance reasoning and testing 89
3.6 Relationships between affordances and dispositions 94
3.7 An example of how dispositions are represented 100
3.8 Pairs of items in the coverage affordance test 106

4.1 An architecture for flexible assembly planning 111
4.2 Concepts and relationships used to represent agenda items 123
4.3 Concepts and relationships used to represent planning strategies . 125
4.4 An example selection strategy 126
4.5 An action designator generated during planning 131
4.6 Runtime performance for toy plane assembly planning 136

4.7 An architecture for hybrid assembly planning 138
4.8 Different initial workspace configurations 139
4.9 Concepts and relationships in the assembly domain 141

5.1 An architecture for robot behavior inspection 156
5.2 Experience knowledge about a safety aware robot 159
5.3 Hierarchical representation of actions 164
5.4 A motion designator retrieved from experience memory 170
5.5 A visualization of the belief state for different time instants 171
5.6 Example scenes generated using user queries 178
5.7 Statistics computed over events occurring in the case study 181

List of Tables

2.1 Concept constructors of the $\mathcal{ALC}$ fragment of DL. 28
2.2 Coverage of relevant notions in related systems 54
2.3 Coverage of cognitive capabilities in related systems 65
2.4 Application domain of related systems 65

3.1 The execution context for testing a *cover* affordance. 107

1 Introduction

We humans are remarkably apt at adapting ourselves to various kinds of common tasks and environments in a way that allows us to perform such tasks competently and well. The reason for this is that we have gathered considerable experience and knowledge about tasks that are common, and variations thereof where different combinations of objects are involved. This allows us to fine-tune our plan how a task should be executed depending on the concrete set of objects that is available to us. For example, we are able to cope with situations where required objects are missing, by repurposing other items for our goals. In general, it is sufficient for us humans to receive an abstract task request in order to come up with a detailed workflow allowing us to execute the task successfully. This is a remarkable capability as the particular behavior required for successful task execution is substantially influenced by the concrete combination of task and environment.

An example of a task that is common to us humans is the pouring of liquids from one container to another. We humans are able to perform this task with a huge variety of different liquids and types of containers, and we are able to adapt to the context in which we execute the task. For example, the skilled pouring of beer into a glass requires the tilting of the glass to produce the right amount of foam for quick drinkability of the beer. We further make use of tools such as measuring cups for the transportation of exact quantities, and whisks for blending the liquid with the substance in the destination container in order to avoid clumps. More rarely humans also learn how to master extremely difficult variations of the task, such as performing artistic movements while mixing a cocktail.

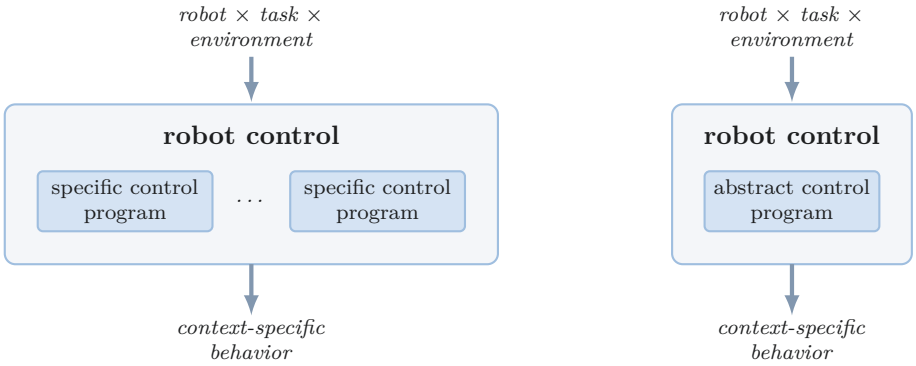

Figure 1.1: Context-specific behavior is required to cope with different combinations of tasks, environments and robots. Specialized control programs are often created for specific combinations (left); but more general programs are desirable that can cope with many variations (right).

Today's robot hardware can be programmed to perform similarly challenging tasks. This has been demonstrated prominently, e.g., by Boston Dynamics and their *Atlas* robot which was programmed to run through a Parkour track with surprisingly skilled motions (Ackerman, 2021). However, a substantial programming effort is required to implement such robot behavior, and the resulting program is often specific to the demonstrated task, and the environment in which the robot operates. The level of adaptability in robot programs is often rather low: the same robot program might not be able to generate the desired behavior in another environment, or for a slightly different task. Accordingly, robots deployed for manufacturing tasks are nowadays programmed for a specific task, and need to be re-programmed whenever the assembly procedure is altered, e.g., in case of product modification. This re-programming causes tremendous costs in manufacturing as the production cannot be continued at full scale in the meantime.

The demand for intuitively programmable robots that can be configured by non-experts for a wide variety of tasks and environments is correspondingly high, as pointed out by Pan et al. (2010), for example. This includes, e.g., domestic service robots preparing meals, setting the dinner table, and cleaning it up; agriculture robots monitoring fields, analyzing the growth of plants, cultivating the fields, and collecting the harvest; and robots in retail stores doing the inventory, and replenishing the shelves if needed. As indicated in Figure 1.1, one of the biggest challenges is that **every combination of task, robot, and environment requires a context-specific robot behavior**.

One way to make the resulting programming effort feasible is to adopt a *knowledge-enabled robot programming* paradigm (Beetz et al., 2012). The basic idea of this paradigm is to separate knowledge from the program, and to modularize it into broadly applicable chunks, such as *grasp objects by their handles*, *hold filled containers upright*, *do not squeeze breakable items*, etc. This kind of abstract knowledge can be organized in a knowledge base system that provides a querying interface for the contextualization of abstract control programs that are not specifically written for a particular robot, task, or environment. For example, an abstract plan for picking up an object can be formulated independently from the type of the object when the way to grasp is determined based on knowledge about grasping. That is, e.g., that heavy objects should be grasped with two hands, or that breakable objects should be grasped carefully without applying too much force. Thus, certain environment-specific aspects can be factored out from the program as pieces of abstract knowledge, and these knowledge pieces can then be used by a robot to constrain the generation of context-specific behavior.

Generally speaking, the main advantage of knowledge-based approaches to robot programming is that these knowledge pieces apply to many scenarios and can dramatically accelerate the realization of new robot applications. For instance, the knowledge about physical characteristics of a material type is general, and applies to several different objects that are made of this material. Therefore, knowledge-enabled programming empowers programmers to write control programs that are fairly independent of tasks, robots, and environments, and to state the necessary knowledge as separate modular knowledge bases.

Another advantage of knowledge-based approaches is that knowledge pieces can be shared easily between different robots, robots and humans, and different subsystems of a robot as long as concepts used for communicating knowledge are machine-understandable and consistent. Consider, e.g., the knowledge chunk *grasp objects by their handles*. This knowledge chunk can only be effectively transferred between different robot applications if the robot generating the knowledge chunk, and the one using it agree on the meaning of *grasping*, *objects*, and *handles*. Thus, it is needed to define these terms in a machine-understandable form that is accepted by robots generating and using this knowledge. Particularly promising, in this context, is the concept of web-based knowledge services, such

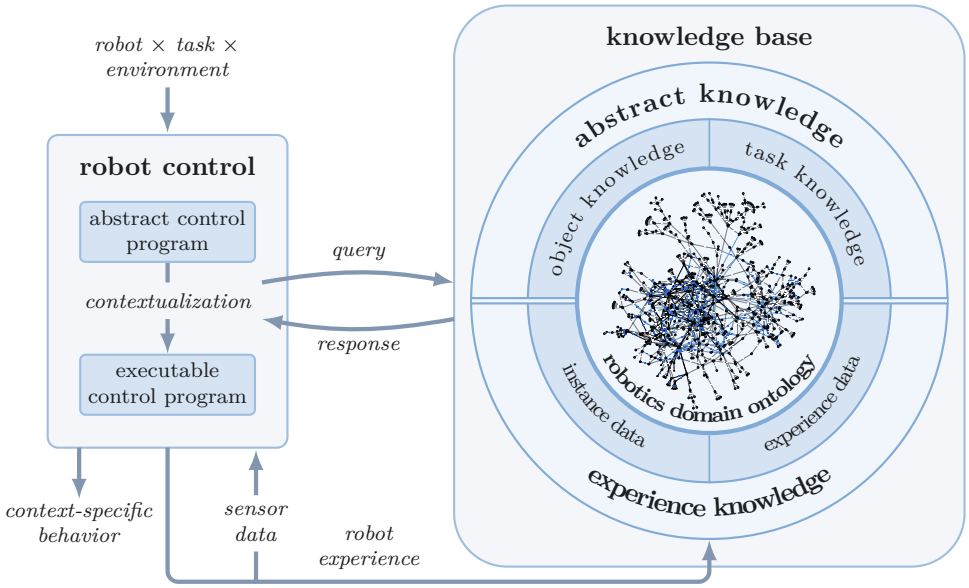

Figure 1.2: Knowledge-based contextualization of abstract robot control programs reduces the programming effort to cope with different combinations of tasks, environments, and robots.

as RoboEarth (Waibel et al., 2011) and openEASE (Beetz et al., 2015c) that enable robots to share knowledge via a dedicated web-service. Such cloud robotics services can be leveraged in machine learning applications which could cause a *virtuous cycle of explosive growth* in robot capabilities as argued by Pratt (2015).

In this work, a specialization of the knowledge-enabled robot programming paradigm will be investigated which is also based on the separation of re-usable knowledge from the control program, but commits to using a formal ontology language for knowledge representation. More specifically, the *Description Logic* (DL) fragment of the *Web Ontology Language* (OWL) will be used to realize more flexible robot behavior where the robot improvises, to some extend, given the action possibilities it detects in the environment. The conceptualization of activity context will also be employed for the characterization of experience data, and to represent what the robot did, why and how. This general approach is displayed in Figure 1.2. The overarching hypothesis of this work is that:

> **a linked ontological characterization of the robotics domain is an integrative framework for robot query answering to support flexible execution and monitoring of tasks, and categorization of robot experiences based on situational context.**

The OWL DL language was selected for several reasons. First of all, the language is a recommended W3C standard (Bechhofer et al., 2004). Since its initial specification, the language was adopted in many application domains, and several useful tools were created including ontology editors, and highly optimized systems for reasoning with OWL ontologies. OWL is also used in the *Semantic Web* (Antoniou et al., 2012). The goal of it is to enhance the accessibility of the Internet with the help of OWL ontologies that are distributed in the Internet. Thus, OWL is also suitable for the sharing of knowledge between robots. OWL ontologies can further be organized in a hierarchical and modular manner such that ontology modules of different generality, and with different scope can be combined with each other. Finally, formal concepts defined in ontologies usually correspond to natural language terms that can be understood by humans, and, thus, contribute to the explainability if used by a robot to characterize its actions.

The general approach of this work is to define different characteristics of robot activity context in OWL DL ontologies to support certain types of reasoning queries that provide the robot with task-relevant knowledge as response to these queries. To this end, a domain ontology for the robotics domain will be proposed that fixes general notions in the domain, and makes fundamental distinctions that are widely motivated by findings in cognitive science. Based on the core ontology, an additional ontology will be proposed to cover the cognitive task of finding and combining appropriate objects that are suitably disposed for a task at hand. The proposed ontologies will further be employed in two application domains. The first considered application domain is an industrial assembly task where an ontological representation of the end-product will be used to realize a mechanism for flexible task planning. The second considered application domain is a service task where the behavior of the robot is characterized in terms of the ontology, and inspected by a human operator through abstract queries.

In the remainder of this chapter, it will first be discussed how robots can employ logical formalization even though the problems they face are often too complex to be solved entirely in a logical framework (Section 1.1). Second, the use of an ontology language for a robot querying system will be motivated (Section 1.2). Finally, the contributions of this work will be summarized (Section 1.3), and, a more detailed outline of the remaining chapters will be provided (Section 1.4).

1.1 Logical Formalization of Robot Problems

Realistic problems faced by robots are too complex to be solved solely based on an axiomatization of the problem domain. It would further be redundant as a lot of the information needed to tackle those problems is already present in the internal data structures of the robot, or can be obtained through computational methods implemented in its control system. However, abstract theories formalized in logic can be employed when they are combined with additional information that links these abstract concepts with the sensorimotor system of the robot.

Early research in the AI field was focused on symbolic reasoning as a means of creating intelligent systems. The hope was that intelligence can be achieved by equipping an agent with a huge corpus of commonsense knowledge formalized in logic. One of the foundational works was presented by Hayes (1979). Hayes suggested the use of *First-Order Logic* (FOL) to formally define the intuitive understanding humans have about objects in the physical world (naive physics). Short after, McDermott (1982) has proposed a temporal logic for reasoning about processes and plans. Other notable models for the planning problem are the *Stanford Research Institute Problem Solver* (STRIPS) (McDermott, 2003), the *Planning Domain Definition Language* (PDDL) (Fox and Long, 2003), and systems using probabilistic representations (Kaelbling et al., 1998).

These approaches are elegant in uniformly handling different problems, but tend to have high computational costs due to exponentially growing number of axioms for more complex problems. A common benchmarking problem for commonsense reasoning is the *egg-cracking-problem*. The problem is how the activity to crack an egg can be formally described in a logic theory. The theory must guide the agent in its decisions that transform an intact egg into a cracked one, while its yolk is still intact and separated from the white in some bowl. Embodied agents would only be able to employ such a theory if it comes with a notion of space and physics – an extremely difficult task in logical formalism (Shanahan, 1997). The challenge is to reason about how much force to apply to crack the egg, how to move the arm to separate yolk and white, and so on. Furthermore, the reasoning must be relatively fast such that it does not slow down the robot too much. A restriction that cannot be fulfilled by FOL for the general case.

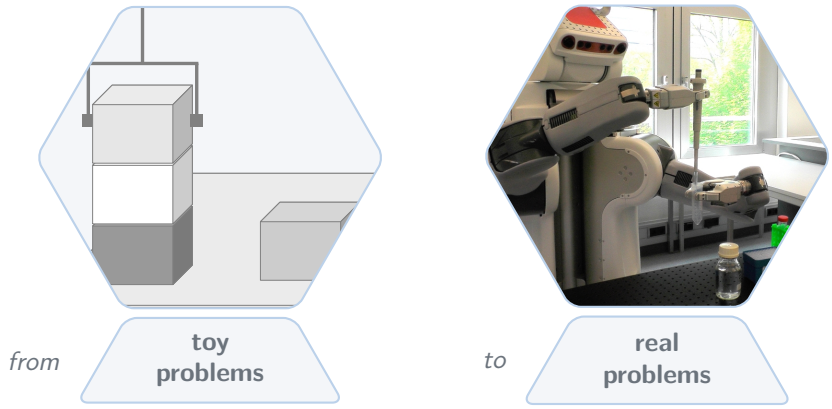

Figure 1.3: Different types of robot problems: a non-realistic scenario based on the well-known *blocks world* (left), and a realistic scenario in a laboratory environment (right).

This rapid growth in complexity is often tackled by an organization of knowledge into *microtheories* that capture a limited range of phenomena (Davis, 1998). For example, Morgenstern (2001) presented an axiomatization for the *egg-cracking-problem* with a few simplifications such as that hand positions are ignored. More recently, Hedblom et al. (2019) presented an axiomatization of the problem based on image schema profiles. An image schema is seen as a categorical mental representation learned from early sensorimotor experiences such as the *containment* concept. Image schema profiles capture image schematic changes such as that, when an egg is dropped onto a floor, that the egg is initially supported by the hand, then looses this support before falling, hitting the ground, and finally breaking on the floor and losing the yolk it contains. However, physical properties of objects cannot be ignored by a robot as they are important for reasoning about action feasibility such as reachability and stability of object placements. Hence, a formalization based on, e.g., image schema profiles can only be employed by a robot in combination with additional information about low-level characteristics.

Much of the research in symbolic AI has, in contrast to the egg-cracking-problem, been validated along toy problems such as the *block-stacking* scenario which is depicted in the left part of Figure 1.3. The right part of the figure depicts a more realistic scenario in which a robot operates a pipette to transfer liquid from one tube into another. It has to reason about where to grasp the pipette, how to hold it, how deep the pipette should be injected into the tube, and so on.

A complete formalization of such a realistic scenario is rather unfeasible due to the complexity of the necessary theories, i.e., the high effort needed to formulate them, and the computational complexity needed to employ them. However, much of the information needed is already present in the robot's internal data structures: the robot needs to localize itself in its environment, it needs to know the pose of its body parts and actuators to coordinate movements, it needs to know how to parameterize motions according to the object's spatial and physical properties, and it needs to be aware of environmental structures and functionality of appliances and tools in order to use them during its activity. This has also been recognized by Brooks (1991) who stated that it is best to use *the world as its own model* without relying on complete representations of the environment. Brooks proposes an evolutionary-inspired approach, called *behavior-based robotics*, which is based on layers of reactive behavior organized in a subsumption architecture where no explicit representations are used. However, Brooks has relaxed this controversial anti-representational standpoint later by admitting that memory of previous events is necessary for some aspects of higher-level cognition (Jordanous, 2020).

Another aspect is that needed information can be obtained through existing methods that, e.g., compute the visibility of objects, i.e., by virtually rendering them from a given viewpoint (Mösenlechner and Beetz, 2011), or the reachability of locations, i.e., using inverse kinematics (Siciliano and Khatib, 2008). Such algorithms often work on raw data and thus can generate more detailed answers than would be possible if the data is abstracted a priori.

To conclude, instead of using a complete axiomatization, robots shall reuse existing data and methods, and only employ microtheories given a task at hand. By doing so, robots can use a suitable level of abstraction with respect to their current situation while avoiding rapid growth in complexity. Reasoning can then be performed using a rather shallow axiomatization of the domain and knowledge about how symbols can be linked with the control system of the robot, e.g., through an object detector that establishes correspondence between symbols in the knowledge base and segments of camera images. The symbolic knowledge base can further be seen as an integration layer for different specialized methods used in the control program of the robot where information is exchanged between methods using a common language which can be defined by an ontology.

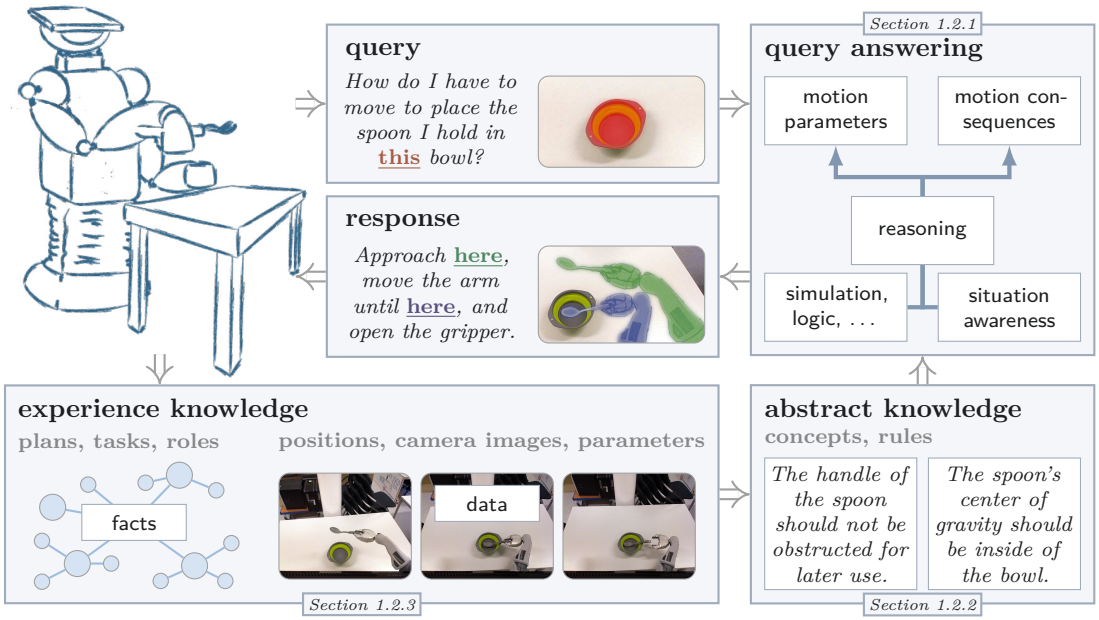

Figure 1.4: A robot knowledge base that employs experience and abstract knowledge to supply the robot with task-relevant knowledge via a query answering interface.

1.2 Ontologies in Robot Knowledge Bases

Ontology languages are often used for the characterization of abstract knowledge. But robots further need to consider knowledge obtained from experimentation and simulation, and must link this knowledge with their sensorimotor system. The hypothesis of this work is that a linked ontological specification of these aspects is a powerful, elegant, and integrative framework for robot query answering.

Such a query answering framework can be designed similar to the human memory system which allows humans the use cognitive abilities such as reasoning, recalling and learning to adjust themselves to new conditions. To this end, humans heavily use episodic memories, and extensive amounts of activity-specific knowledge (Anderson, 1995). Another impressive aspect is the persuasive presence and use of commonsense and naive physics knowledge in the human memory system (Hegarty, 2004). For example, telling a person to set the table is enough for having a detailed version of the intended arrangement. An example of a robot memory system with capabilities similar to those of the human memory system is outlined in Figure 1.4. Its components will be described in the following sections.

1.2.1 Query Answering

One of the main components of robot knowledge bases is the query answering interface. The purpose of it is to answer certain types of questions to bridge the gap between vague and shallow instructions such as *prepare a breakfast table*, and the context-specific realization of such an abstract task. To bridge the gap, a substantial amount of abstract and experience knowledge is required. In the shown example, for instance, the robot has to determine how to hold the spoon, how much force to apply, how to approach the bowl, and how fast to move to execute the task successfully without causing unwanted side effects. It also needs to reason about how the spoon should be placed such that the placement is stable, and provides required possibilities for future actions. As argued earlier, the reasoning required to obtain such detailed answers is beyond of what can be handled effectively in a purely logical framework. However, the answers can be constrained through ontological reasoning which will be investigated in this work.

Ontology languages commonly support reasoning via means of an automated reasoner that can solve decision problems such as class membership of instances, and subsumption of classes. Such reasoning can be employed to make a robot control program more generally applicable, and it is also useful to detect errors during modeling (Kalyanpur, 2006). However, whether a problem can be accounted for via automated reasoning depends on the expressivity of the logic formalism underlying the ontology language which is often based on DL which is a fragment of FOL. Another aspect is that of whether there exists an effective method for performing the inference. The logic is called *decidable* if this is the case. FOL is only semi-decidable, but reasoning with most DL languages is decidable. Thus, if run-time reasoning is required, DL should be preferred.

1.2.2 Abstract Knowledge

Ontologies are widely used for the representation of *abstract knowledge*, i.e., knowledge encoded as general rules and concepts that are used for the classification of specific examples (also called *instances* or *entities*). Abstract knowledge can be used to conceptualize the environment, task and robot, and restrict which relations might hold between entities. A robot working as assistant in a chemical laboratory,

for example, may receive an instruction to *neutralize the alkaline substance in some tube*. First, the robot needs abstract knowledge about the phrases *neutralize*, *alkaline*, and *tube*. That a *tube* is a container for liquid substances with an opening to pour substances into it, that *alkaline* is a chemical substance and the counter part of acid, and that *neutralization* is a chemical process caused by mixing acid and alkaline substances. The robot further needs to infer that an acid substance should be used for the neutralization, and that a pipette should be used to transport small quantities of acid into the tube. It also has to infer that pipetting is a motion that involves ingesting some acid from a source container into the pipette, and dripping it into the tube by pressing a button on the pipette. This type of abstract knowledge is often accounted for in ontology representations.

In addition, the robot has to draw on substantial amounts of *experiential knowledge*, i.e., knowledge which was obtained through experiences. This is, for instance, that the tip of the pipette should be inserted into the tube, or at least held close above it to avoid spillage. This type of abstract knowledge can be learned through practical experimentation and simulation. It is important for finding appropriate motion parameters that are likely to achieve desired effects while avoiding unwanted side effects. Representations for integrated robot control systems should therefore take into account both a priori and experiential knowledge. In this work, the experiential knowledge required for intelligent behavior realization is considered as a component of the ontological characterization of the robot's experiences. Robots shall build up a knowledge base that grows with experience whenever the robot performs an activity, and gather information about previous experiences, generalize over specific situated experiences, and adapt them into the situated context ahead. The hypothesis is that a linked ontological specification of the distinct kinds of knowledge required is a powerful, elegant, and highly integrative framework for query answering where concrete experiences of the robot can be selected through the evaluation of abstract queries.

1.2.3 Experience Knowledge

Experience knowledge is very detailed and specific information that represents events and situational context of particular activity episodes, and from which experiential knowledge can be learned. Action representations used in AI and

autonomous agents research typically represent the agent control system at a coarse level of detail, at which actions are described by blackbox models. Instead, the approach taken in this work is to put ontologies at the core of the robot control system so that data structures, even those used at lower levels, can be semantically annotated with their meaning according to the core ontologies.

Accordingly, the knowledge base consists of symbols that need to be linked to the sensorimotor system of the robot. The problem of establishing this link between symbols and real-world entities is called *symbol grounding problem* (Harnad, 1990). Symbols that are not grounded are meaningless for robots. A symbol such as *this spoon*, for example, can only be used by a robot if it can establish a link between the symbol and the actual spoon in the real world it refers to. The label used to denote a symbol is not relevant for computational methods. However, it is crucial when other agents are using the same symbol, e.g., when a robot speaks with a human both need to have the same grounding for the linguistic forms to understand each other. In the context of robot programming, symbols can be provided ungrounded, e.g., as abstract instructions, and grounded at run-time by the robot; or generated by the robot in a grounded form, and, e.g., be accessed through log files for the purpose of debugging.

In order to gain a better understanding of the advantages of putting ontologies at the core of robot control systems, consider, for example, the concept of a dynamically changing robot pose. Assume this is represented in the form of the *holds* predicate *holds(pose(r,⟨x,y,z⟩), ti)*, which asserts that the robot believes its pose with respect to the origin of the environment map at time instant *ti* is ⟨*x,y,z*⟩. Typically, the robot control system would estimate the pose of the robot over time using a Bayesian filter — such as a particle filter for robot self-localization. In this case, the robot's belief about where it is in the probability distribution over the possible robot poses in the environment is estimated through the filter. Then, one can identify the pose where the robot believes to be as the pose ⟨*x,y,z*⟩ with the maximal probability, and use this data for the computation of spatial relations via rules. By specifying rules that ground concepts and relations defined in the ontology into the data structures of the control systems, substantial parts of the data structures can be turned into *virtual knowledge bases* where the relevant relations are grounded in the data structures.

1.3 Contributions

This work examines the use of ontologies as an integrative framework for robot query answering to support flexible execution and monitoring of tasks, and categorization of experiences. To this end, an ontological framework will be proposed, and its use for robot query answering will be investigated. In the following, a summary of the contributions of this work will be provided.

1. **The *Socio-physical Model of Activities* (SOMA)** is a domain ontology that connects the observable physical domain with conceptualized interpretations thereof. Using this ontology, robots can reason about how tasks can be accomplished in different ways through motions and interactions with the environment. Thus, compared to current models that treat actions as atomic events, additional knowledge about action phases can be formalized. The modeling of tasks at the level of interactions further enables the robot to organize and classify actions performed by other agents based on distinct patterns of contact events caused by the interactions.
2. **The *Descriptive Affordance Ontology*** is designed for reasoning about finding and combining appropriate objects and discovering possible dispositions at hand. Using this ontology, robots can to some degree improvise, and take advantage of action possibilities that objects provide in their environment. To this end, the current object-centric reasoning in robot control programs is reformulated around object uses and their roles, and thereby the control program is formulated in a more general and reusable manner. The ontology further enables robots to anticipate object arrangements that enable future actions by exposing the required dispositions.
3. **The *Assembly Ontology Framework*** provides a mechanism for flexible task planning in the assembly domain that is based on an ontological model of the end-product. Using this mechanism, robots can determine what still needs to be done to complete an assembly task through efficient ontological reasoning. Compared to the status quo in industrial assembly, this approach presents a step towards more versatile and reconfigurable assembly cells which is achieved through more flexible robot programming by means of altering the ontological model instead of the program code.

4. **The *Narrative-enabled Episodic Memory* (NEEM) querying language** for robot experience knowledge that exploits ontological categories to retrieve episodes from the experience memory of a robot. Such experienced episodes include a conceptualization of the activity, and also the sensorimotor experience data of the robot. Using this querying language, particular situations of interest can be queried, and the corresponding experience data can be retrieved for the purpose of debugging, and data sets can be created for the purpose of conducting a statistical analysis of the behavior of a robot, or to learn behavior based on examples in the data set.

The contributions of this work will primarily be validated through an implementation of the proposed ontologies in the OWL DL language, and by demonstrating that a standard reasoner can use these ontologies in order to infer information that is needed by the robot to fulfill its tasks. In more detail, it will be highlighted how different types of task-relevant questions, such as what tasks an object can be used for, can be formulated as reasoning queries such that these queries can be answered efficiently using DL reasoning. It will further be shown that the proposed ontologies enable the realization of a flexible mechanism for robot assembly tasks where ontological reasoning is used for the configuration of the product. Finally, the proposed ontologies will be validated for the representation of experience knowledge by means of a case study where the behavior of a robot is inspected through the evaluation of abstract queries formulated around the ontology that yield particular experienced situations, and the associated experience data.

Additionally, a few non-technical contributions to the robotics research field have been achieved in the scope of this work. First of all, the work has influenced the definition of an IEEE standard for robot ontologies (Olszewska et al., 2017), which is by and large complementary to the proposed ontologies. The technical contributions of this work have also been implemented as part of the open-source framework KnowRob (Tenorth and Beetz, 2013). Finally, an annual workshop has been established with the purpose to encourage discussions connecting the two fields of ontology and robotics, and to consolidate scientific work in the intersection of both fields (Hammar et al., 2020; Sanfilippo et al., 2021). In the remainder of this section, a listing of related scientific publications will be provided, and the previously mentioned non-technical contributions will be described in more detail.

Publications The aforementioned technical contributions have been the subject of several publications in international conferences. The most important ones are:

- D. Beßler, R. Porzel, M. Pomarlan, A. Vyas, S. Höffner, M. Beetz, R. Malaka, and J. Bateman. Foundations of the Socio-physical Model of Activities (SOMA) for autonomous robotic agents. In *FOIS*, volume 344 of *Frontiers in Artificial Intelligence and Applications*, pages 159–174. IOS Press, 2021. doi: 10.3233/FAIA210379

- D. Beßler, R. Porzel, M. Pomarlan, M. Beetz, R. Malaka, and J. A. Bateman. A formal model of affordances for flexible robotic task execution. In *ECAI*, volume 325 of *Frontiers in Artificial Intelligence and Applications*, pages 2425–2432. IOS Press, 2020b. doi: 10.3233/FAIA200374

- D. Beßler, M. Pomarlan, and M. Beetz. OWL-enabled assembly planning for robotic agents. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '18, pages 1684–1692, Richland, SC, 2018c. International Foundation for Autonomous Agents and Multiagent Systems. Finalist for the Best Robotics Paper Award

- A. Olivares-Alarcos, D. Beßler, A. Khamis, P. Gonçalves, M. Habib, J. Bermejo, M. Barreto, M. Diab, J. Rosell, J. Quintas, J. Olszewska, H. Nakawala, E. Pignaton de Freitas, A. Gyrard, S. Borgo, G. Alenyà, M. Beetz, and H. Li. A review and comparison of ontology-based approaches to robot autonomy. *The Knowledge Engineering Review*, 34, 12 2019. doi: 10.1017/S0269888919000237

- M. Beetz, D. Beßler, A. Haidu, M. Pomarlan, A. K. Bozcuoglu, and G. Bartels. Knowrob 2.0 – A 2nd generation knowledge processing framework for cognition-enabled robotic agents. In *2018 IEEE International Conference on Robotics and Automation, ICRA 2018, Brisbane, Australia, May 21-25, 2018*, pages 512–519, 2018. doi: 10.1109/ICRA.2018.8460964

These are only the most relevant publications. A full list of my prior publications can be found in Appendix A. The following chapters also contain references to the prior publications they are based on, and additional information about my own contribution in case I am not the first author of the publication.

Open-source Software In the scope of this work, substantial contributions to the open-source robotics community have been made. This has been achieved through integration of technical results into the open-source knowledge processing framework KnowRob[1]. KnowRob is one of the most influential open-source knowledge processing for robots nowadays (Olivares-Alarcos et al., 2019). The most noteworthy step change that was achieved in the scope of this work is that KnowRob was rebuild on a more principled basis, using a well established foundational ontology that models different phenomena of human cognition. One of the consequences is that it makes it easier to integrate knowledge from different sources, and to cooperate across the disciplines of robotics and ontologies. It further means that KnowRob covers a wider set of competency questions that can be answered by robots that are equipped with the KnowRob system.

The ontologies that have been developed in this work have also been published as open-source artifacts[2]. My colleagues and I have further established best practices of ontology engineering for the development and maintenance of the SOMA ontology. This includes, e.g., that the quality of the ontology is continuously evaluated using common infrastructure for the computation of ontology metrics, and that a reasoner is used in a continuous integration pipeline to ensure that no inconsistent modifications occur. The SOMA ontology is further accompanied by rich documentation in form of a handbook (Beetz et al., 2020) and a website[3].

The third open-source contribution of this work is the further development of openEASE, a cloud-service for experience knowledge in the scope of robot task execution[4], and in particular the models and infrastructure that are used for representation, visualization and processing of symbolic data that is stored as part of an experience knowledge repository. An instance of the openEASE service is publicly accessible[5]. The website provides access to experience knowledge of different agents performing everyday activities where the experience data is annotated by concepts defined in the SOMA ontology.

[1] `https://github.com/knowrob/knowrob` (accessed 22 May 2022)
[2] `https://github.com/ease-crc/soma` (accessed 22 May 2022)
[3] `https://ease-crc.github.io/soma` (accessed 22 May 2022)
[4] `https://github.com/ease-crc/openease` (accessed 22 May 2022)
[5] `https://open-ease.org` (accessed 22 May 2022)

RobOntics Workshop The RobOntics workshop has been established in 2020 by a group of researchers including me with the purpose of establishing a framework where researchers in robotics and ontologies can exchange ideas and present results that are located in the intersection of the fields autonomous robotics and formal ontologies. The goal of RobOntics is to foster the interaction across these domains, to identify and discuss promising approaches, and to consolidate the progress in knowledge-based robotics. Topics of main interest include: ontology foundations, robustness of task execution, human-robot interaction, normed robot behavior, and explainability of robot behavior.

The annual workshop has called for papers in 2020 for the first time[6]. It was planned as part of the *Joint Ontology Workshops* (JOWO) event (Hammar et al., 2020), which was associated to the international conference on *Formal Ontology in Information Systems* (FOIS) in 2020. However, due to traveling restrictions during that time the first gathering of workshop participants was postponed to 2021. In 2021, the RobOntics workshop was organized as a hybrid event, and as part of the FOIS conference which was co-located with the *Bolzano Summer of Knowledge* (Sanfilippo et al., 2021)[7].

IEEE-SA P1872.2 Standard for Autonomous Robotics Ontology The IEEE standardization organization has started an effort to standardize ontologies for robotics in the framework of the IEEE workgroup *Ontologies for Robotics and Automation* (ORA)[8]. The ORA workgroup was divided into several sub-groups investigating different specific topics in the intersection of robotics and ontologies. The core ontology standard has been presented by Schlenoff et al. (2012).

One of the sub-groups of this effort is called *Ontologies for Autonomous Robots* (AuR) (Olszewska et al., 2017). I was part of this group for several years. The AuR workgroup was dedicated to the topic of robot autonomy. It was initiated in 2011, and became an official workgroup by the end of the year. The effort was successfully concluded in 2021 when the proposed standard P1872.2 was approved by the standardization committee, and finally published in 2022[9].

[6] `https://robontics2020.github.io` (accessed 22 May 2022)
[7] `https://robontics2021.github.io` (accessed 22 May 2022)
[8] `https://standards.ieee.org/ieee/1872/5354` (accessed 22 May 2022)
[9] `https://standards.ieee.org/standard/1872_2-2021.html` (accessed 22 May 2022)

1.4 Outline

The remainder of this work is organized in different chapters that all contribute to the overall goal of investigating flexible robot task execution through the use of formal ontologies. In the following, an abstract will be provided for each of the remaining chapters that contribute to this field.

Chapter 2 In the second chapter of this work, different software projects that use ontologies to support robot autonomy will be reviewed. Considered software projects are systematically identified based on whether they fulfill a set of inclusion criteria, and compared with each other with respect to the scope of ontologies used in the system, what types of cognitive capabilities are supported by the use of ontologies, and for which application domain the system was designed.

Chapter 3 In the scope of the third chapter, a domain ontology for the robotics domain will be proposed, and one of its extensions will be presented. The ontology, called SOMA, conceptualizes social as well as physical activity context, and establishes relations between both branches, i.e., between occurrences of events and generalizations of them. The extension of SOMA that will be presented is concerned with inferring possible actions that can be accomplished with a given object at hand which is commonly referred to as inferring the affordances of objects.

Chapter 4 In the fourth chapter of this work, a method for flexible assembly planning will be proposed. The method is based on an ontological conceptualization of what an assembled product should look like. The ontology is aligned with the SOMA ontology, and uses its notion of affordance as one of the central concepts for modeling assembly tasks. Action selection is done based on a comparison of the belief state of the robot, and a dedicated goal state, i.e., the assembled product. The chapter also investigates the grounding of ontological symbols that represent geometric relations through common procedures for spatial computation in order to be able to perform assembly tasks in cluttered environments where occlusions might occur.

Chapter 5 In the scope of the fifth chapter, a querying language for knowledge obtained through experimentation and observation will be investigated, and how such knowledge can be characterized ontologically. To this end, a case study will be presented that shows the feasibility of automated acquisition for a case where the robot control program can be monitored during task execution, and it will further be demonstrated that the acquired knowledge is a powerful vehicle to inspect the behavior of a robot through the evaluation of abstract queries that refer to concepts defined in the ontology.

2 Ontology-based Approaches to Robot Autonomy

The robotics domain is fairly challenging due to the high versatility of tasks, environments, and robot hardware that can be considered in robot applications. It is further non-trivial to realize the necessary capabilities for sensing, planning, and acting if the robot is supposed to operate autonomously. This holds true especially for the general case, i.e., when these capabilities are implemented independently of the application domain. Nevertheless, some general notions such as *plan* and *affordance* can be identified that are relevant in almost all application areas in the robotics domain. However, the meaning of such basic terms underlying the robotics domain is often not generally agreed upon. Hence, it is worth investigating how these terms are commonly defined in literature, and how they relate to the capabilities of the robot. To this end, a set of central notions in the robotics domain will be identified in the scope of this chapter, their intuitive meaning in the robotics domain will be discussed, and how they are commonly defined in ontologies. It is further of interest how these notions are pragmatically used in control programs, i.e., how they are defined, in the scope of which capabilities they are used, and in which application domain. This will be investigated in this chapter for a set of selected software systems that equip robots with ontologies. The result of this investigation is further condensed in a publicly accessible website[1]. The literature survey presented in this chapter is based on a previously published article by Olivares-Alarcos et al. (2019)[2].

[1] `https://ease-crc.org/ontology-survey-2019` (accessed 06-04-2022)

[2] First author A. Olivares-Alarcos and second author D. Beßler have contributed equally to the work which is also indicated in the cited journal paper.

2.1 Basics of Ontology and Autonomous Robotics

In this section, some basics about ontologies will be discussed which are necessary in the context of this work, and notations will be introduced that are used throughout this work. Furthermore, the concept of autonomy in the domain of robotics will be discussed.

2.1.1 Types of Ontologies

One of the first attempts to define ontology was made by Gruber (1993) stating that *an ontology is an explicit specification of a conceptualization*. Gruber's definition was informal and several authors tried to refine it either by specifying what a conceptualization is (Guarino and Giaretta, 1995), or by adding further requirements like being *formal* and *shared* (Borst et al., 1997; Studer et al., 1998). We had to wait another decade to settle the problem. Guarino et al. (2009) presented a technical discussion of the terminology, and a formal definition of ontology was finally given.

Starting from Gruber's intuition that an ontology depends on a conceptualization, Guarino et al. explain that a conceptualization is (mathematically speaking) an intentional relational structure, i.e., a domain of discourse (a set of entities), a set of possible worlds (possible layouts of the entities), and a set of relations (stating which properties entities have in each possible world). Once the formal notion of conceptualization is given, it suffices to fix a vocabulary and the ontological principles one commits to, the so called *ontological commitments*. The purpose of the commitments is to constrain the interpretation of the vocabulary to the given conceptualization. At this point, an ontology is defined to be a *logical theory* consisting of a set of formulas whose models approximate the intended models, i.e., those models that satisfy the conceptualization and the ontological commitments.

Practically speaking, since an ontology is a logical theory, it consists of individuals, classes, relations and axioms. The exact list changes depending on the specific logic language one adopts. Often, ontology languages correspond to fragments of FOL. Roughly speaking, *individuals* are the objects in the ontology, i.e., the entities the ontology is about. *Classes* are categories the individuals may

belong to. *Relations* are connections that hold among individuals. *Axioms* are expressions in the language that use the previous elements to state what is true in the ontology.

In domain studies, the term ontology is used to refer to a variety of things. For instance, for Chandrasekaran et al. (1998), an ontology is a *representation vocabulary* specialized to some domain and constrained by a conceptualization. It is also understood as a *domain theory* about objects, properties and relationships among those objects that are possible within a specified domain of knowledge. The purpose of an ontology in this sense is to provide the knowledge structure for a particular domain, therefore, it focuses only on the viewpoint taken within the domain, and it includes the relevant concepts for working in such domain. In the literature, this latter use of the term is known as *domain ontology* while the characterization introduced by Guarino and colleagues (Guarino et al., 2009) is general encompassing domain as well as foundational ontologies that contain very general terms applicable across all domains.

Developing a domain ontology, one provides the description of a particular domain without challenging the ontological perspective. The purpose is to make the domain knowledge explicit and formal (at least partially), i.e., to fix in a formal language the vocabulary and what the experts' consider its correct interpretation including the valid assertions in the domain. Generally speaking, a domain ontology can help to achieve data and model interchangeability within and across communities. One should bear in mind that there is no unique conceptualization of a domain. Having suitable domain ontologies helps to clarify the differences as well as to compare the conceptualizations.

Ontologies can be classified along many dimensions. Figure 2.1 depicts a graph containing the two classifications of ontologies considered along this work. A classification based on the characteristics of the language used for the ontology is presented by Uschold and Grüninger (1996). It shows that the term *ontology* is sometimes used vaguely. That classification divides ontologies in four classes: (a) highly informal, (b) semi-informal, (c) semi-formal and (d) rigorously formal. However, following the previous discussion, we observe that the class (a) is not talking about ontologies: it refers to linguistic resources or to knowledge repositories in an early phase of ontology construction.

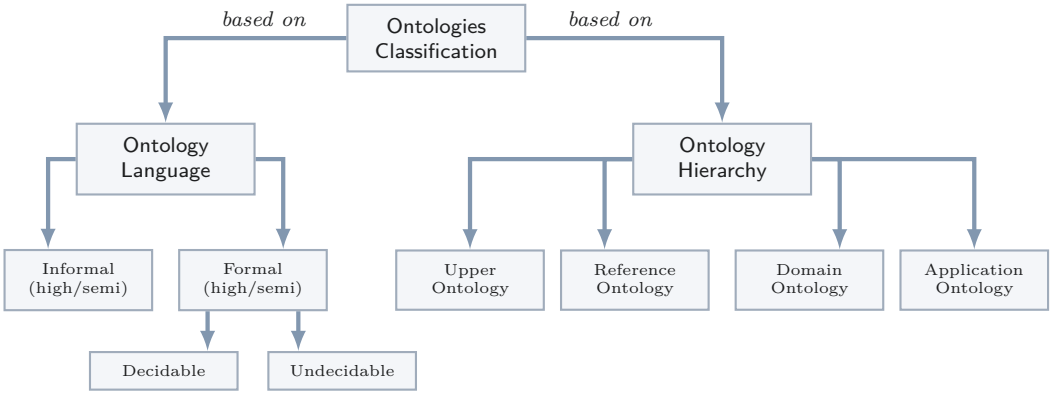

Figure 2.1: Different classifications of ontologies. They are based on the language used to write the ontology (left), and the hierarchical level of the ontology (right).

Ontology Language Since the language of the ontology constrains how the ontology can be used in an information system, this kind of classification is of relevance in the context of this work. The first distinction that can be drawn is between *informal* and *formal* languages. *Informal* languages are the ones which do not have an associated formal semantics, like *Resource Description Framework* (RDF), and (part of) *Unified Modeling Language* (UML). They are mostly dedicated to representation tasks and syntactic manipulation. Automatic reasoning on these languages is not reliable because there is no systematic way to constrain their interpretation. *Formal* languages are the ones endowed with formal (e.g., Tarskian) semantics, that is, languages whose interpretation is formally established. These languages, among which *Knowledge Interchange Format* (KIF) and OWL are popular examples, are suitable for knowledge representation and reasoning since they are based on clear and exhaustive syntactic and semantic rules.

Among the formal languages, a further distinction is of interest, that is, whether a language is decidable or undecidable. Here decidable means that, given a logical theory expressed in that language, there exists a method for determining whether an arbitrary formula is derivable or not in the theory. Since ontologies are special logical theories, an ontology written in a decidable language is decidable. A language is called undecidable if it is not decidable. An ontology that uses a decidable language is also called *computational* since it can be used at runtime for information extraction and verification, e.g., OWL DL. A formal ontology,

which is not computational, like OWL Full, is not appropriate for such role since, when queried, it might not return an answer. Since, depending on the application domain, the answer to a query might need to be available very quickly, decidability is enriched with computational complexity considerations (Papadimitriou, 2003).

Ontology Hierarchy Ontology classifications usually divide ontologies into *upper-level*, *reference*, *domain* and *application ontology*.

An *upper-level ontology* is an ontology that focuses on widely applicable concepts like object, event and quality, and high-level relations like part-hood, constitution, participation and dependence. Examples are *Suggested Upper Merged Ontology* (SUMO) (Niles and Pease, 2001), Cyc ontology (Lenat and Guha, 1990), *Basic Formal Ontology* (BFO) (Arp et al., 2015), and *Descriptive Ontology for Linguistic and Cognitive Engineering* (DOLCE) ontology (Masolo et al., 2003). These foundational ontologies may differ in their ontological commitments, and also in expressiveness of the language.

A *reference ontology* is an ontology that focuses on a discipline with the goal of fixing the general terms in it. It is highly reusable within the discipline, e.g., medical, engineering, enterprise (Guarino, 1998). When the ontology focuses on a more limited area, e.g., manufacturing or tourism, we instead call it a *domain ontology*. This kind of ontology provides vocabulary about concepts within a domain and their relationships, about the activities taking place in that domain, and about the theories and elementary principles governing that domain. The concepts in domain ontologies are mostly specializations of concepts already defined in upper-level and in reference ontologies, and the same might occur with the relations (Gómez-Pérez et al., 2004). An *application ontology* contains all the definitions needed to model the knowledge required for a particular application, e.g., a CAD/CAM system or an ERP. Another major classification system is useful to differentiate reference and domain ontologies, and focuses on topic coverage. The *Enterprise Ontology* (Uschold et al., 1998), the *Process Specification Language* (Grüninger, 2004) and *Core Ontology for Robotics and Automation* (CORA) (Schlenoff et al., 2012) are examples that fall into this classification. There are different ways to classify topics as shown by the variety of library classification systems around the world.

2.1.2 Logical Foundation of Ontologies

The logical foundation of ontology languages is usually some fragment of FOL. A FOL theory consists of an alphabet over symbols used to construct terms and formulas. Terms are used to identify objects, and formulas to state what is true in the theory. The symbols in a FOL theory can be divided into logical and non-logical. Logical symbols include the logical connectives for conjunction ($\wedge$), disjunction ($\vee$), implication ($\rightarrow$), biconditional ($\leftrightarrow$), and negation ($\neg$). This allows, for example, to express conditional statements of the form *if a is true, then b is also true* as $a \rightarrow b$ using logical implication, where a and b are formulas. FOL further allows the use of free or bound variables. Variables can be bound through universal quantification (using the $\forall$ symbol), or existential quantification (using the $\exists$ symbol). For example, x is a quantified variable in the formula $\exists x P(x)$. The symbol P in the example formula is a non-logical symbol. The set of non-logical symbols is often fixed through a *signature*, and consists of predicate (also called relation) and function symbols of certain arities. A n-ary predicate symbol P is used to form formulas of the form $P(t_1, \ldots, t_n)$ where each t_i is a term. For example, $P(x)$ is a formula formed using the 1-ary predicate symbol P. A term is either a variable, or function expression $f(t_1, \ldots, t_n)$ where each t_i is a term, and f a n-ary function symbol. Constants are the nullary function symbols.

The meaning of FOL formulas is defined with respect to a domain $\mathcal{D}$, and an interpretation function $\mathcal{I}$. The domain $\mathcal{D}$ is a set consisting of the objects in consideration. Quantification of variables in formulas ranges over elements of this set. The interpretation function maps each n-ary function symbol f to a function from $\mathcal{D}^n$ to $\mathcal{D}$, and each n-ary predicate symbol P to the set $P^{\mathcal{I}} \subseteq \mathcal{D}^n$ consisting of predicate tuples in the domain. Each FOL formula is either true or false under an interpretation. Logical connectives are simply evaluated based on their truth tables, i.e., their meaning does not depend on the interpretation. The truth of a formula $P(t_1, \ldots, t_n)$, on the other hand, is determined by testing whether a tuple $(v_1, \ldots, v_n) \in P^{\mathcal{I}}$ exists such that the evaluation of t_i yields v_i for every $0 < i \leq n$. The evaluation of terms is needed to replace free variables appearing in them with elements of the domain. Furthermore, a formula $\exists x \phi(x)$ holds true if there exists an assignment $v \in \mathcal{D}$ for the variable x such that the formula $\phi(v)$ holds true, and a formula $\forall x \phi(x)$ holds true if $\phi(x)$ holds true for each possible assignment of x.

2.1.3 The Web Ontology Language

OWL is a family of formal languages consisting of different dialects with varying expressiveness and computational properties. The first W3C recommendation for OWL includes three variants: OWL Lite, OWL DL and OWL Full (Bechhofer et al., 2004). OWL DL is more expressive than OWL Lite while retaining desired properties such as decidability. Effective reasoning methods exist for OWL Lite and OWL DL. OWL DL allows the use of all syntactic structures of OWL, but some constructs can only be used under certain conditions. Such restrictions do not exist for OWL Full. However, at the cost of undecidability of the dialect. In 2009, an updated version of the OWL specification has been recommended by the W3C (Motik et al., 2008). It includes the sub-languages OWL2 EL with polynomial time reasoning complexity, OWL2 QL which is designed for database systems, and OWL2 RL with support for rules in the language.

Dialects of the OWL family have a strong logical foundation, and most of them correspond to a fragment of DL which in turn is a decidable fragment of FOL with well understood computational properties. Several fragments of DL have been investigated with varying computational properties. OWL DL, for example, corresponds to the $\mathcal{SHOIN}^{(D)}$ fragment (Horrocks and Patel-Schneider, 2004) while OWL2 corresponds to $\mathcal{SROIQ}^{(D)}$ (Horrocks et al., 2006). Symbols in the name of the fragment indicate which extensions are allowed in expressions of the language. The symbol $\mathcal{I}$, for example, indicates the inclusion of inverse properties. The basic elements of DL are concepts, roles and individuals. Concepts correspond to OWL classes, roles to OWL properties, and individuals to OWL individuals.

OWL classes are organized in a hierarchy where characteristics of the parent classes are inherited to their children. This corresponds to logical subsumption in DL which is written as $C \sqsubseteq B$ where D denotes the child and D the parent class. If, in addition, $B \sqsubseteq D$, then both concepts are said to be equivalent which is written as $D \equiv B$. Such statements are called *terminological axioms*. They form a component of the knowledge base called *Terminological Box* (TBox). The TBox consists of two special concepts called *top* (written as $\top$) and *bottom* (written as $\bot$). Every individual is an instance of, and every concept is a sub-concept of $\top$, while no individual is an instance of $\bot$, and $\bot$ is a sub-concept of every concept. These concepts correspond to the OWL classes `Thing` and `Nothing` respectively.

DL Syntax	OWL Manchester Syntax	Semantics
$\top$	`Thing`	$\mathcal{D}$
$\bot$	`Nothing`	$\emptyset$
$C \sqcap D$	C **and** D	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
$C \sqcup \mathtt{B}$	C **or** D	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
$\neg C$	**not** C	$\mathcal{D} \setminus C^{\mathcal{I}}$
$\forall R.C$	R **only** C	$\{x \in \mathcal{D} \mid \forall y((x,y) \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}})\}$
$\exists R.C$	R **some** C	$\{x \in \mathcal{D} \mid \exists y((x,y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}})\}$

Table 2.1: Concept constructors of the $\mathcal{ALC}$ fragment of DL.

A statement about individuals, on the other hand, is called *assertional axiom*, and is stored in the *Assertional Box* (ABox) of the knowledge base. Such an axiom is either a concept assertion $x : C$, or a role assertion $(x, y) : R$ where x, y are individuals and R a role. If the ABox consists of a role assertion $(x, y) : R$, then x is said to be R-related to y.

DL concepts can be constructed from syntactic expressions of the language. The particular set of syntactic structures that can be used varies in different fragments of DL. The $\mathcal{ALC}$ fragment is often used as a basis for more expressive DLs. Its concept constructors are shown in Table 2.1. The language includes constructs for the intersection ($\sqcap$), union ($\sqcup$), and negation ($\neg$) of concepts, and for universal ($\forall$) and existential ($\exists$) restrictions. Restrictions are concepts whose instances must satisfy the constraint defined by the concept. This is, for a universal restriction $\forall R.C$, that instances of the concept may only be R-related to instances of the concept C, and for an existential restriction $\exists R.C$, that instances have at least one R-relation to an instance of C.

The meaning of statements in the TBox is defined by interpreting DL concepts as sets of individuals, and roles as ordered pairs of individuals. To this end, an interpretation $\mathcal{I}$ is defined as a tuple of a set $\mathcal{D}$ consisting of individuals (the domain), and an interpretation function that maps each individual x to an element $x^{\mathcal{I}} \in \mathcal{D}$ of the domain, each concept C to a sub-set $C^{\mathcal{I}} \subseteq \mathcal{D}$ of the domain, and each role R to a set of tuples $R^{\mathcal{I}} \subseteq \mathcal{D} \times \mathcal{D}$. The interpretation is defined over a signature (N_C, N_R, N_O) fixing the set of concept names (N_C), role names (N_R), and individual names (N_O) in an interpretation. The semantics of concept

constructors is then defined based this interpretation of atomic concepts and roles as shown in Table 2.1. Finally, an interpretation $\mathcal{I}$ satisfies (models) a concept inclusion $C \sqsubseteq D$ if and only if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$, a concept assertion $x : C$ if and only if $x^{\mathcal{I}} \in C^{\mathcal{I}}$, and a role assertion $(x, y) : R$ if and only if $(x^{\mathcal{I}}, y^{\mathcal{I}}) \in R^{\mathcal{I}}$.

Reasoning with OWL languages is often based on the tableau algorithm which uses *refutation* as a proof mechanism (Carnielli, 1987), i.e., a formula holds true if the reasoner can proof that its negation cannot be satisfied (*proof by contradiction*). The reasoning tasks that can be tackled with OWL ontologies include consistency of concepts, building subsumption hierarchies, classification of individuals, and finding relations between them. The computational costs of these reasoning tasks depends on the constructs that are allowed in the fragment of the language. Reasoning with OWL languages further uses the *open-world assumption* to cope with incomplete knowledge. This means that reasoner cannot conclude that a statement is false if it cannot be proven to be true as this could be the case due to the lack of knowledge. This is in contrast to the *closed-world assumption* where everything that is unknown is believed to be untrue.

There are a variety of different syntaxes that can be used to exchange OWL ontologies. One of the most prominent ones is based on a mapping into the RDF language which is commonly used for the exchange of graph data, and for which different concrete syntaxes exist such as the *Terse RDF Triple Language* (Turtle) syntax. Another commonly used syntax is the *Manchester Syntax*. The mapping between DL and Manchester Syntax is shown in Table 2.1. It is designed as a compact and human-readable representation, and uses a *frame-based* syntax which is similar to object-oriented programming languages.. In the following, illustrative examples of OWL classes will be provided using the Manchester Syntax. An example of a `Cup` concept represented in this syntax is shown below:

```
Class: Cup
SubClassOf:
  DesignedContainer
  isDescribedBy some LiquidContainmentDesign
```

Above, `Cup` is defined as a sub-concept of `DesignedContainer`, and an anonymous class defined by a syntactic expression. The latter is the class of entities that are *isDescribedBy*-related to at least one instance of the concept `LiquidContainmentDesign` indicating that liquids can be stored in them.

2.1.4 Autonomous Robotics

Autonomy is a desirable quality for robots in many application domains, especially when the robot needs to act in real world environments together with other agents, and when the environment changes in unforeseeable ways. Robot autonomy is further critical when the robot is employed under certain legal and ethical constraints. Apart from the dictionary and subjective definitions, there exist several attempts to define the autonomy term. Beer et al. (2014) present a comprehensive analysis of existing definitions in several domains including robotics. The definition of *autonomy* provided by the authors is the following one:

> "The extent to which a robot can sense its environment, plan based on that environment, and act upon that environment with the intent of reaching some task-specific goal (either given to or created by the robot) without external control." (Beer et al., 2014, p. 77)

Autonomous systems can be based on different architectures with different levels of complexity ranging from simple reactive architectures, to deliberative architectures, to cognitive architectures. Reactive systems are based on simple *sense-act* loops, while deliberative systems employ more sophisticated *sense-decide-act* loops to endow the system with reasoning and decision making capabilities.

The architectures that implement a theory of how the human mind is structured are called cognitive architectures. Cognitive architectures are often classified into symbolic, connectionist or hybrid approaches, where connectionism attempts to capture cognitive phenomena using artificial neural networks that are often trained via Deep Learning. Deep Learning techniques had several breakthroughs in the last decade for application areas such as computer vision (Voulodimos et al., 2018) and natural language processing (Hinton et al., 2012). However, Deep Learning methods are data-hungry, and thus require that large training data sets can be acquired easily. This is not the case for robot task execution (Sünderhauf et al., 2018) which has also recently been recognized by the company OpenAI that discontinued their robotics department due to the lack of training data.

The quality of the training data in Deep Learning is further critical. If the data is biased, then it is likely that this bias is inherited onto the network trained

from the data. For example, an AI system that learns from data acquired from a racially biased society might reproduce this racial bias. This has happened several times already (Caton and Haas, 2020), and it shows the importance of defining a legal and ethical framework when AI systems interact with humans, and learn from them. I argue that these frameworks should not be learned, but implemented as fixed symbolic structures that can be manipulated and comprehended by humans easier compared to if they are encoded in huge neural networks. Fixed symbolic structures are further useful to explain decisions to humans, and to annotate robot experience data with causal relations. This cannot be achieved by current machine learning methods as they are limited to finding patterns, correlations and associations, while being incapable of yielding causal relations.

In general, it is still an open question what the computational capabilities are that enable human-level cognition. Vernon et al. (2007) provide a thorough discussion about this topic. The authors present a survey of different paradigms of cognition, and also efforts to combine the different approaches in hybrid systems. An extension of this survey is provided by Vernon (2014). In this last work, Vernon referred to the key architectural features that systems capable of autonomous development of mental capabilities should exhibit (Langley et al., 2009). Summarizing, Langley et al. postulate the following functional capabilities that are essential for the realization of a cognitive system:

- Recognition and categorization;
- Decision making and choice;
- Perception and situation assessment;
- Prediction and monitoring;
- Problem solving and planning;
- Reasoning and belief maintenance;
- Execution and action;
- Interaction and communication; and
- Remembering, reflection, and learning.

2.2 A Classification of Ontologies for Autonomous Robots

In this section, a classification of ontologies will be presented that will be utilized to structure and perform the review of the selected works in Section 2.3. The classification is split into three dimensions: ontology scope (Section 2.2.1), reasoning scope (Section 2.2.2), and application domain (Section 2.2.3).

2.2.1 Ontology Scope

Ontologies can be organized as networks of modules, each focusing on a specific topic. The scope of a module, is given by the range of categories that it covers. This section investigates a list of categories particularly relevant in autonomous robotics such as `Sensor`, `Capability`, and `Action`. The aim is to find how these categories have been used in the literature; and to provide an initial discussion about their meaning. References to more detailed discussions in the literature are also provided. The Oxford dictionary (Simpson et al., 1989) is used to provide informal definitions. In addition, it will be discussed how the terms are understood commonly in formal ontologies, and different usage in the robotics field will be highlighted. In Section 2.3.2, it will be analyzed whether or not each of the surveyed projects defines these categories and how.

Note that many concepts relevant for autonomous robotics lack a universally agreed meaning. Moreover, terms commonly used in the robotics field are also often part of everybody's everyday speech. Hence, everybody has *some* subjective definition of these terms that is often heavily influenced by personal experience, and thus it may be substantially different from that of other people. In the scope of this section, only the most widely agreed meanings of concepts will be covered, and a brief comparison between the different viewpoints will be provided.

Object The Oxford dictionary defines the term `Object` as *a material thing that can be seen and touched* (Simpson et al., 1989). But often a more fine-grained definition is needed that is not limited to material objects, for example, holes, or spatial behavioral patterns. More generally, mental and social objects depend on material acts (like communication acts) but they may be neither material (made of matter) nor physical (located in a region of space). A significant number of founda-

tional ontologies make a distinction between `Endurants` and `Perdurants` (Masolo et al., 2003; Niles and Pease, 2001). Endurants (aka continuants or objects), on the one hand, are wholly present at any time, but may change over time. Perdurants (aka occurrents or events), on the other hand, are extended in time, and only partially present at any time. This dichotomy is crucial in systems that have to cope with time. Physical objects are often further classified into `Artifact` and `Non-Artifact`, where artifacts are intentionally created, often according to a design, to fulfill a certain function, etc. (Borgo et al., 2014). Objects may further be classified as `Agent` or `Non-Agent`, where agents are capable to generate intentional behavior.

We humans tend to categorize objects because of the variety of `Qualities` and `Properties` that they exhibit, which give us a way to cluster them into similarity classes. There has been long philosophical discussions about what qualities and properties are, and among them which are primary or not, and if they can be exhaustively listed (Allen, 2016). One important reason to focus on qualities and properties is the understanding of (qualitative) change. One branch of formal models considers individual qualities (roughly, the way an individual manifests characteristics like weight, size, shape etc.) as basic entities in the ontology. Each individual quality is existentially dependent on an unique endurant (or perdurant) and associated to a quale (plural: qualia). Qualia are used to compare entities, and thus to discuss similarity/dissimilarity (w.r.t. the associated quality) across objects and events (Masolo and Borgo, 2005). In this view, qualities form a third fundamental category along with endurants and perdurants, and the associated qualia change over time to explain the changes in their corresponding endurants (or perdurants). Qualia are further organized in `Spaces` (e.g., the space of weight, the space of colors etc.) and can be given a quantitative/qualitative value (e.g., numerical) once the space is enriched with a reference system and unit of measure. An alternative approach uses the notion of `Tropes` (also called *abstract particulars*) where qualitative change is expressed through the substitution of tropes (Neuhaus et al., 2004). Thus, when an object changes, this modeling view assumes that the existing trope ceases to exist and a new one is created. Continuous change (like the increasing of room temperature) is often considered problematic to model in this latter approach.

Environment Map The term `Environment` is considered in the Oxford dictionary as *the surroundings or conditions in which a person, animal, or plant lives or operates* while `Map` is described as *a diagrammatic representation of an area of land or sea* (Simpson et al., 1989). Although some general way to understand the meaning of environment in robotics has been proposed (Borgo et al., 2019), in this domain the focus is more often oriented to the representation of the environment (Chella et al., 2002).

The format and information content of a map is not only diagrammatic in robotics as it is influenced by how and for what the map is to be used by the robot. For example, collision maps encode 3D geometric information of the environment to support generating collision free motions in 3D space, while navigation maps usually only use a 2D geometrical representation to support finding collision free navigation paths. The term `Semantic Environment Map` is often used to refer to environment representations that make the semantics of the environment and objects in the environment explicit. One prominent example, called SOM (Semantic Object Maps), was introduced by Rusu et al. (2009). SOM's also encode spatial information about the environment but, in addition, enrich the information content with encyclopedic and commonsense knowledge about objects, and also include knowledge derived from observations.

Affordance The term `Affordance` was introduced by Gibson as *what the environment offers the animal, what it provides or furnishes, either for good or ill* (Gibson, 1979). More recently, the meaning has shifted towards *"(perceived) possibility for action"* (Norman, 2002), i.e., something the object offers that allows the agent to interact with it or, more generally, something that allows objects to participate in actions or processes. However, there is no common agreement in the ontology engineering community on how this concept should be modeled. One way to model affordances is as individual qualities of an object (Ortmann and Kuhn, 2010), or as relational qualities of an object-agent pair (Turvey, 1992a). Another approach is to model them as events, as proposed by Moralez (2016). A notion of affordance is relevant to talk about possibilities as it enables to answer questions such as *what can the robot do with an object*, and *is it possible for an object to take a particular role when some task is performed.*

Action and Task The Oxford dictionary defines an action as *the fact or process of doing something, typically to achieve an aim* (Simpson et al., 1989). There have been several other attempts to define action in different disciplines. A notable one is Donald Davidson's philosophy of action, where an action is defined as something *intentional under some description* (Davidson, 2001). Krüger and colleagues surveyed the meaning of action in the robotics field (Krüger et al., 2007), and suggest a hierarchical representation using three levels of granularity – action and motor primitives, actions and activities. Similarly, Bobick (1997) proposes an action hierarchy for action recognition with three layers where movements are the most atomic primitives.

A task can be understood *as a piece of work that has to be done* (Simpson et al., 1989). Hence, tasks denote pending work, independently from how an agent exactly accomplishes this work. In this view, an action would be a way to execute a task. Technically, one can approach this by defining tasks as types (of event) used to classify actions, which then allows to explicate that a task can be accomplished in different ways, and to talk about individual tasks independently from their possible executions. A standardization of a task ontology for the robotics domain is currently being developed by Balakirsky et al. (2017).

Tasks and actions may further be classified according to their complexity, temporal extension, inter-task (inter-action) relationships, etc. However, such classifications are often not clear, e.g., the distinction simple vs. complex task would be dependent on the adopted granularity or robot's capabilities.

Activity and Behavior The Oxford dictionary considers an `Activity` as *the condition in which things are happening or being done*, and a `Behavior` as *the way in which one acts or conducts oneself, especially towards others* (Simpson et al., 1989). Hence, both terms refer to situations in which an agent performs actions, but with different viewpoints. Activities rather having an intrinsic, and behaviors an extrinsic viewpoint, e.g., was it good or bad behavior, and how it affected other agents. Note that in the case of behavior, it can also apply to non-agents as it is common to talk of the behavior of devices or tools, for instance.

Rodney Brooks and his colleagues did fundamental work in the research field *behavior-based robotics* where the term behavior also refers to extrinsic character-

istics of task execution (Brooks, 1991). The field of behavior-based robotics is motivated by the observation that emergent behavior can be generated by simple control systems, and that intelligence lies in the eye of the observer (Brooks, 1991). Brooks has also postulated that the world is its own best model, and hence argues that simple *Sense-Act* loops can be used to directly interact with the world without relying much on symbolic representations.

Other authors have focused on the terms `Behavior` and `Function`, for instance claiming that the function of an object denotes its intrinsic aspects (i.e., how it works), and behavior the extrinsic aspects (i.e., what it does). An engineering discussion of this dichotomy is provided by Salustri for the context of computer-based design tools (Salustri, 1998) while an ontological assessment is provided by Mizoguchi et al. (2016).

Plan and Method A `Plan` is *a detailed proposal for doing or achieving something* (Simpson et al., 1989). Similarly, the DOLCE+*Description and Situation* (DnS) Plan Ontology (Gangemi et al., 2004) defines `Plan` as *a description that defines or uses at least one task and one agentive role or figure, and that has at least one goal as a part*. Hence, plans have explicit goals to be achieved when the plan is executed by appropriate sequences of actions that comply with the plan. An execution of the plan can succeed, fail, be postponed, aborted, etc.

The generation, detection and assessment of plans is a long-standing sub-area of AI. A prominent framework for planning is the PDDL formalism (McDermott et al., 1998). PDDL *tasks* denote initial and goal state, and how the state can be modified by applying actions or operators. General purpose solvers are then used to generate a plan given the domain definition. Several authors have further combined standard planning techniques, such as PDDL, with more expressive representations (Balakirsky et al., 2013; Kootbally et al., 2015).

A `Method` is usually understood as being more abstract than a plan. The Oxford dictionary defines it as *a procedure for accomplishing or approaching something, especially a systematic or established one* (Simpson et al., 1989). In a sense, *methods* are guidelines for agents to choose actions towards achieving a specific goal instead of specifying an organization of the action into sub-actions that would cause the goal to be achieved.

Capability and Skill A `Capability` can be seen as *the power or ability to do something* (Simpson et al., 1989). Hence, a distinction is made between capabilities that are enabled by physical qualities and those that are enabled by social role(s) within a certain community. The term `Skill`, according to the Oxford dictionary, is more restrictive, namely, *the ability to do something well*. Thus, it only includes what the agent can do because of its physical qualities and it implies that the achievement is positively qualified (in terms of manners and results) (Fazel-Zarandi and Fox, 2013). One widespread use of the term in robotics is *skill learning* where it is used to refer to the ability of the robot to achieve something via a behavior learned through observation, communication, experimentation or simulation. However, both terms are also often used as synonym of each other, for example by Perzylo and colleagues in their work on the description and orchestration of manufacturing skills (Perzylo et al., 2019a).

Having capabilities represented in a formal model, the robot can reason about whether the necessary capability is present to perform a certain task in a given situational context and, if not, how the task could be accomplished otherwise. This is usually approached by defining capabilities with respect to hardware and software components of the robot (Kunze et al., 2011; Buehler and Pagnucco, 2014). A navigation capability would be enabled by a mobile base which is controlled by a navigation software component that interfaces with the mobile base. Furthermore, Tiddi et al. (2017) used a notion of capability to provide a more intuitive, capability-based interface for robot programming.

Capabilities cannot be manifested in arbitrary situations. For example, wheeled robots are not able to navigate along stairs and thus might not be able to reach a target location on another floor. However, if an elevator can be used and the robot is able to operate it, the robot may still be able to reach its navigation goal. Hence, capabilities do not automatically enable the robot to perform a task. Their use depend on suitable conditions of the situational context in which the robot should operate – e.g., who can perform the task, what specific variant of the task can be performed, and where the task can be performed. The degree of how capable a robot is may change over time to the point that a capability cannot be manifested at all. For example, due to attrition of hardware, broken hardware or missing components (hardware or software).

Hardware Components Ontologies may represent what chains of robot links and joints form what `BodyParts`, and how body parts can contribute to perform, for instance, tasks and capabilities. One of the most widely used formats to represent hardware components of robotic agents is *Unified Robot Description Format* (URDF)[3]. URDF allows to represent kinematic chains made of links and joints, and also to define the limits of each joint. This information is used, e.g., by inverse kinematics solvers to find a valid joint configuration in which the end-effector of the robot reaches a dedicated goal pose[4]. URDF files include both *actuators* (e.g., servos of the joints and grippers) which act in the environment and *sensors* (e.g., cameras and sonars) used to perceive the environment.

A `Sensor` is a device which detects or measures a physical property and records, indicates, or otherwise responds to it (Simpson et al., 1989). Sensors can be used for different objectives such as measuring robot parameters for control loops, correcting for errors in the robot's models of itself and of the world, and detecting and avoiding failure situations. One widely adopted ontology is the *Semantic Sensor Network* (SSN) ontology which describes sensors and their measurements, the involved procedures, the studied features of interest, the samples used to do so, and the observed properties, as well as actuators (Compton et al., 2012). SSN includes a lightweight and self-contained core ontology called SOSA.

Software Components A notion of robot software components in ontologies is crucial when these shall be automatically introspected and integrated into task execution. One of the most widely employed ontologies for modeling software in ontologies is the *Ontology of Information Objects* (IO) (Gangemi et al., 2004) where a distinction is made between an abstract `DataStructure` and the `DigitalResource` that concretely realizes the data structure within some physical storage medium. The broad goals for software ontologies in robotics are to enable the robot automated software discovery and installation to dynamically compose its control system for a given task, to decide for a given control system whether some capability can be realized through available software components, and to support introspection in case some software failure occurred.

[3] `https://wiki.ros.org/urdf` (accessed 22 May 2022)

[4] An end-effector is a device located at the end of a kinematic chain, designed to interact with the environment in some way.

One of the most widely used middlewares in robotics nowadays is the *Robot Operating System* (ROS)[5]. ROS organizes robot software components in a communication graph, where each node is a piece of software either listening or publishing messages on named topics, or offering a service that can be called via the node. Messages are defined using an abstract syntax, and concrete realizations of the message type in different target languages such as C++ and Python are generated automatically by ROS.

Interaction and Communication `Interaction` *is a reciprocal action or influence* (Simpson et al., 1989) between two or more entities. This comprehensive definition includes those interactions in which there is no explicit exchange of information. For example, interactions happening at the atomic level, physical interactions, and speech acts. Research in the robotics domain tends to concentrate on information exchange. Indeed, in the literature, both the human-computer interaction (Dix, 2009) and the human-robot Interaction (Yanco and Drury, 2004, 2002) domains, tend to provide a less general formal definition of `Interaction` which may be closer to `Communication`.

The term `Communication` *is the imparting or exchanging of information by speaking, writing, or using some other medium* (Simpson et al., 1989). Gangemi and Mika (2003) proposed to formalize this term within the *Description & Situation Ontology* viewpoint distinguishing two cases: an ontology for communication situations and roles, and an ontology for peer-to-peer communication.

2.2.2 Reasoning Scope

The scope of reasoning is the second classification criterion for the comparison between ontology-based approaches considered here. In the remainder of this section, a categorization of ontology-based reasoning tasks that are in particular relevant for autonomous robotics will be provided. This categorization is based on the nine functional capabilities that were discussed in Section 2.1.4. In Section 2.3.2, it will be discussed how the surveyed projects use ontologies to support these nine capabilities.

[5] `https://www.ros.org` (accessed 22 May 2022)

Recognition and Categorization For the purpose of establishing a link between the perceived environment and the knowledge base of a robot, the robot must be able to recognize events or situations (static and dynamic) and categorize them as named instances of already known patterns. For instance, let us consider a collaborative industrial robotic arm which picks pieces from a conveyor belt and places them on a table which a human operator has access to. The robot must recognize and categorize the conveyor belt, and the different pieces to manipulate (static), as well as a the human collaborator's movements and actions (dynamic). Both, recognition and categorization, *operate on abstract mental structures* (Langley et al., 2009). Langley et al. (2009) emphasize that, in order to support recognition and categorization, a cognitive architecture shall be able to represent patterns and situations in memory.

Decision Making and Choice An autonomous robot requires the ability to choose among several alternatives, which usually is considered together with the recognition and categorization problem in a recognize-act cycle. Nonetheless, Langley et al. (2009) considered the capability of decision making independently. It is important not to mistake this capability with planning, whose focus is on the achievement of a goal which will be explained later in this section. For example, a collaborative robot would apply decision making to choose between moving or standing when the operator is close, while planning would be used to find the sequence of actions which are likely to lead to a successful outcome. A cognitive architecture should be able to represent the different choices in a format the robot understands. Indeed, such a representation can also be used to improve the decision making process through machine learning methods.

Perception and Situation Assessment The environment where the robot exists must be sensed, perceived and interpreted. First, the robot senses its surroundings through possibly muti-modal sensors. Then, using the gathered information and relying on recognition and categorization, discussed earlier, and on inferential mechanisms, which will be covered shortly, the robot is able to perceive the environmental entities (e.g., objects and events). Finally, the situation assessment takes place when the perceived objects and events are interpreted.

Following with the example used before, a collaborative industrial robot would look at the conveyor, the pieces, and at the operator's movements to sense the environment. The pieces, their poses and other information would be recognized and categorized in order to assess the environmental situation so that the robot could, for example, interpret that an approaching piece should be picked. Just as occurred with previous cognitive capabilities, the inherent knowledge of the whole process must be represented in a manner the robot understands. Note that the representation requires memory, a resource which is often limited. Hence, the notion of attention emerges, meaning that the robot may only focus on relevant regions of the environment.

Prediction and Monitoring Prediction is a cognitive capability that requires the representation of the environment, the actions that can take place, and their effects. Therefore, the robot could predict future events and situations which did not occur yet by means of a proper mechanism which utilizes the representation. Applied to the collaborative robotics example, it would be possible for the robot to predict the operator's actions, so that the robot could adapt better to what is expected. Note that prediction enables robots to also monitor processes. When the perceived situation differs from the expected one, it means that either our knowledge is not complete or something did not go as it was supposed to. In the former case, it would be possible to store the facts in memory for posterior learning, in the latter case, an alarm or error could be triggered. For example, the robot could detect a malfunctioning in the conveyor belt if the pieces stopped arriving to the robot's workspace.

Problem Solving and Planning In novel situations, it might be necessary for a robot to plan and solve problems. For the purpose of generating a plan, the robot needs to utilize a model of the environment to predict the effects of its actions. Furthermore, the cognitive architecture must be able to represent a plan as an (at least partially) ordered set of actions, their expected effects, and the manner in which these effects enable subsequent actions. Despite often being viewed closely related, planning is somewhat less general than problem solving. In particular, the former usually refers to cognitive activities within the robot's head,

whereas the latter can also occur in the world. Concretely, when a problem to be solved is complex and the available memory is limited, a robot may search for solutions through experimentation, rather than constructing a complete internal plan. As an illustration, a collaborative robot could solve a problem by mixing the execution of actions such as *asking for operator's help* (external behavior) and the generation of action sequences (internal planning).

Reasoning and Belief Maintenance Reasoning is a cognitive activity which allows a robot to expand its knowledge state, drawing conclusions from other beliefs or assumptions the robot already has. Thus, it requires the existence of a representation of beliefs and the relationships among them. A common formalism used to encode such knowledge is FOL. Ontologies are often written in languages based on less expressive formalisms than FOL (e.g., OWL-DL) in order to reduce the computational cost of inference.

These formalisms, allow the use of different sorts of reasoning such as: deductive or inductive. For the robot in the previous example, it would be possible to infer how operators will react to unexpected interactions by knowing general information about human-robot interaction (deductive). Or the opposite, from specific operator's behaviors, inferring the norms to follow during human-robot interaction (inductive). Note that reasoning is not only relevant to infer new beliefs but also to decide whether to keep existing ones. This is commonly referred to as *belief maintenance*. Belief maintenance is especially important for dynamic environments in which situations may change in unexpected ways that has implications for the robot's behavior.

Execution and Action Cognition takes place to support and drive an activity in the environment. To this end, a cognitive architecture must be able to represent and store motor skills that enable such activity. For instance, a collaborative robotic arm should have skills or policies for manipulating its surroundings and for collaborating with other agents (e.g., humans). A robot should also be able to execute those skills in the environment, which can happen in a reactive form. Nevertheless, a cognitive architecture should enable a robot to maintain a continuous loop of execution. Hence, the robot can interpret how the execution

of actions is affecting the state of the environment and could adapt its behavior. A proper representation of the ongoing actions occurring in the environment is essential for aspects related to robot action execution: robot adaptation, learning new skills, action-execution-related knowledge, etc.

Interaction and Communication In some cases, the most effective way for a robot to obtain knowledge is from another agent (e.g., humans, robots). This makes communication another important ability that a cognitive architecture should consider. For example, a collaborative robot could request further information about how to perform a task, or which are the preferences of a human operator about where to place the picked objects. Regardless of the modality or means of communication, there should be a way to represent the transferred knowledge such that it is accessible to, and understandable for the robot. This should be bi-directional, meaning the robot must be able to transform stored knowledge into the concrete medium through which it will be communicated.

Remembering, Reflection, and Learning There are some capabilities which cut across those described before: remembering, reflection and learning. Remembering is the ability to encode and store the results (facts) of cognitive tasks so that they can be retrieved later. Once again, based on the previous example, a collaborative robot could store the results of an entire day of work (e.g., successful experiences, human-robot interactions, etc.). On the other hand, reflection stands for the serious thought or consideration (Simpson et al., 1989) about something which usually is represented and stored in memory and can be retrieved. For example, a collaborative robot could take memory into account in order to explain which was the rationale behind its actions. Finally, learning has the goal to improve the performance of a robot based on examples. For example, the collaborative robot could use the memories about successful and failed actions to generalize and learn from them. The knowledge used to learn may come from different sources: the observation of another agent, the result of previous experiences, through kinesthetic teaching, etc. No matter what the source of experience is, all of them require the existence of a memory in which the experiences are represented, and from which it can be retrieved.

2.2.3 Application Domain Scope

The last classification criterion for the comparison between ontology-based approaches in this work is regarding to the application domain. Some of the different application domains of robotics will be briefly discussed in this section. Robotics is a multidisciplinary and versatile discipline whose application is present in wide range of domains: medicine, industry, service, entertainment, space, military. Following the classification of robotics devices published in the ISO 8373:2012[6] (ISO, 2012) two general domains are considered here: industrial and service robotics. These two domains include many of the application sub-domains of interest for robotics (e.g., medicine, rescue, social). The ISO 8337:2012 standard specifies a vocabulary used in relation to robots and robotic devices operating in both industrial and non-industrial environments (service). It also provides definitions and explanations of the most commonly used terms.

Industrial Robotics The *industrial robotics* domain includes all those robots which are automatically controlled, re-programmable, multipurpose manipulator, programmable in three or more axes, which can be either fixed in place or mobile for use in industrial automation applications. Typical applications of industrial robots include welding, painting, assembly, pick and place for printed circuit boards, packaging and labeling, palletizing, product inspection, testing, and material handling. Industrial robots perform with high endurance, speed, and precision in all of those tasks, however, usually only with a low degree of autonomy.

Service Robotics The *service robotics* domain includes robots that perform useful tasks for humans or equipment excluding industrial automation applications. Also note that while articulated robots used in production lines are industrial robots, similar articulated robots used for serving food are service robots. Typical applications of service robots include those tasks which are dirty, dull, distant or dangerous. Service robots can be classified into service robots for professional use and service robots for personal use. Service robots for professional use are commonly used in application areas such as reconnaissance, search and rescue, firefighting, healthcare, construction, logistics, smart factories, smart

[6] `https://www.iso.org/standard/55890.html` (accessed 22 May 2022)

farming, hazardous environment monitoring and space exploration to name a few. Applications of service robots for personal use include, but are not limited to, home automation, personal assistant for elderly and physically challenged in support for aging in place, entertainment and vacuum cleaners.

2.3 Ontologies to support Robot Autonomy

In this section, a discussion and comparison of frameworks that use ontologies to support robot autonomy will be provided. In order to select a potential list of candidates for discussion, a systematic examination of the state-of-the-art was conducted, and, in addition, the results were filtered by a set of inclusion criteria. For each framework that fulfills these criteria, a brief discussion will be provided in Section 2.3.1, followed by a comparison of the projects in Section 2.3.2.

For the purpose of finding literature focused on using ontologies to enhance robot autonomy, a scientific databases was employed. Concretely, the literature browser *Web of Science*[7], previously known as *Web of Knowledge*, has been used for the study. It is an online subscription-based scientific citation indexing service that provides a comprehensive citation search. It gives access to multiple databases that reference cross-disciplinary research, which allows for in-depth exploration of specialized sub-fields within an academic or scientific discipline.

Through searches using the keywords *knowledge representation industrial robotics* and *knowledge representation service robotics* 133 and 148 scientific papers were identified respectively. The two lists found during the previous step, were combined in a single list with 281 articles in total. In the interest of identifying projects or initiatives that use ontologies to enhance robot autonomy, the list of papers was reduced following a list of criteria:

- It is proposed to use knowledge representation techniques (ontologies) in robotics applications to enhance robot autonomy.
- The work is not limited to a single article.
- Case studies where the knowledge base is used by a robot exist.

[7] `https://www.webofknowledge.com/` (accessed 22 May 2022)

After applying the criteria, the list was reduced to 21 articles, which correspond to five different projects: KnowRob (Tenorth and Beetz, 2009), IEEE-ORA (Schlenoff et al., 2012), ROSETTA (Stenmark and Malec, 2013), CARESSES (Bruno et al., 2017), and RehabRobo-Onto (Dogmus et al., 2019). This set of works, was enlarged by other four ones extracted from a related survey (Thosar et al., 2018). In that work, Thosar et al., reviewed a list of nine works, which, as in this work, were chosen following a systematic search and inclusion criteria. Only five of those nine works fit the purpose here, but one of them is KnowRob which was already included before, thus, only four will be investigated: ORO (Lemaignan et al., 2010), RoboBrain (Saxena et al., 2014), OUR-K (Lim et al., 2011), and OMRKF (Suh et al., 2007).

It is worth mentioning that the project IEEE-ORA does not actually provide a complete framework which is available to be used, as it consists of just an ontology. Indeed, the ontology developed in the framework of that project contains general concepts of the domain, so that it is not really useful in specific application scenarios. However, it is relevant enough to be considered in this work, since it aims at standardizing the representation of knowledge in the robotics domain. Therefore, possible extensions of the original work are considered here. Following a similar approach as before (using the Web of Science's browser), papers were identified that cited the most relevant article related to the project (Schlenoff et al., 2012). In this case, from the 34 initial works which cite it, only two followed the whole inclusion criteria presented in this section: OROSU (Gonçalves and Torres, 2015) and PMK (Diab et al., 2019a).

In total, ten projects were discovered that passed the search procedure. However, in the following, the focus lies only on the most influential approaches among them for the discussion of individual approaches (Section 2.3.1) and a comparison of them (Section 2.3.2). Accordingly, a list of inclusion criteria is employed to refine the list of surveyed projects. Projects will only be considered in the scope of this survey if they satisfy all of the criteria, which is the case for six of the projects. The inclusion criteria considered are:

Ontology Scope The project uses an ontology that defines one of the terms that were identified as particularly relevant in Section 2.2.1.

Reasoning scope It uses ontologies to support at least one of the cognitive capabilities that were discussed in Section 2.2.2.

Transparency Material is openly available that describes the overall goal of the project, what capabilities are considered, and how ontologies are used.

Curation It is maintained. Meaning that recent developments or future plans are evident or at least possible.

Accessibility There exists software that is accessible, and that demonstrates how ontologies are used to support a cognitive capability.

2.3.1 Discussion of Frameworks and Projects

In this section, an overview of the six projects that have been subject of this study will be provided. For each of them, their underlying principles and foundations are discussed, as well as what application domain the system was designed for. It will also be described how the frameworks evolved over time, and what impact they had so far. The selection of the presented projects has been done based on the selection criteria presented earlier. To the best of my knowledge, all projects that satisfy these criteria are included in this section.

KnowRob KnowRob (Knowledge processing for Robots)[8] [9] is an open source knowledge processing system that is designed for autonomous service robots. It was first introduced by Tenorth and Beetz (2009). The authors argue that autonomous robot control demands a KR&R system that addresses several aspects that are commonly not sufficiently considered in AI KR&R systems. One of these aspects is that robots need a more fine-grained action representation. This was discussed, in more detail, in another work where Tenorth and Beetz argue that service robots should be able to cope with (often) shallow and symbolic instructions, and to fill in the gaps to generate detailed, grounded, and (often) real-valued information needed for execution (Tenorth and Beetz, 2017).

Recently, a second generation of the KnowRob system was introduced where the focus has shifted towards the integration of simulation and rendering techniques

[8] `http://knowrob.org` (accessed 22 May 2022)
[9] `https://github.com/knowrob/knowrob` (accessed 22 May 2022)

into a hybrid knowledge processing architecture (Beetz et al., 2018; Haidu et al., 2018), which is inspired by the simulation theory of cognition (Hesslow, 2012). The rational is to re-use components of the control program in virtual environments with physics and almost photorealistic rendering, and to acquire experience knowledge from these sources. Experience knowledge, called NEEM in KnowRob, is used to draw conclusions about what action parameterization is likely to succeed in the real world (e.g., through learning methods).

KnowRob has also been used in several research initiatives including the European projects RoboHow (Beetz et al., 2016), RoboEarth (Waibel et al., 2011), SAPHARI (Beetz et al., 2015b), and SHERPA (Marconi et al., 2012). RoboEarth, for example, is a pioneer project to consider exchanging knowledge between robots using the World Wide Web, OWL, and Linked Data principles. For instance, it was demonstrated how such an infrastructure can be used to execute tasks that were not explicitly planned at design time. More recently, KnowRob has been used by the openEASE web knowledge service which is designed for the acquisition, storage, curation, visualization, and analysis of robot experience knowledge (Beetz et al., 2015c). KnowRob plays further a central role in the ongoing collaborative research center *Everyday Activity Science & Engineering* (EASE)[10] funded by the German DFG. The EASE project has the goal to uncover principles underlying everyday activities by, first, acquiring experience knowledge with different modalities, and, second, building models that generalize over these modalities (Bateman et al., 2017).

The main programming language used in KnowRob is (SWI) Prolog which has its roots in FOL. SWI Prolog comes with a library to manage RDF triples, which is used by KnowRob to represent explicit knowledge in memory such as facts encoded in OWL ontologies. Initially, KnowRob was deriving its concept definitions from the OpenCyc ontology (Lenat, 1995). However, only rather shallow symbolic representations were used that were tailored to provide useful information for task execution without enforcing consistency. In the scope of this work, KnowRob has shifted towards the use of the *DOLCE+DnS Ultralite* (DUL) ontology (Masolo et al., 2003) and a more careful and principled modeling of foundational concepts for autonomous robotics. Another important principle

[10] `https://ease-crc.org` (accessed 22 May 2022)

underlying KnowRob is about how data that already exists in the robot control system can be made *knowledgeable* – that is how this data can be integrated into symbolic reasoning. KnowRob employs the notion of *virtual knowledge bases* that are computed using control-level data such as data structures used by the perception and planning component of the robot control system. The computation is carried out by, so called, *computable properties* which are computation methods attached to symbolic relations defined in an ontology.

KnowRob is one of the most influential KR&R systems for autonomous robots nowadays. This is evident as many research papers and projects have been using and extending KnowRob since it was initially released. However, there are a couple of limitations worth mentioning. First, KnowRob has been using only a very shallow symbolic representation following the principles of behavior-based robotics, and in particular the claim that the world itself is its own best model. But having a lot of information only encoded implicitly in data structures of the control program also creates some problems such as the computational cost of abstraction when symbolic inference is performed. Furthermore, despite its long history, KnowRob's ontologies were not yet widely adopted in the robotics field.

ROSETTA ROSETTA stands for *RObot control for Skilled ExecuTion of Tasks in natural interaction with humans; based on Autonomy, cumulative knowledge and learning*. Its origin can be traced to the European projects SIARAS (Haage et al., 2011) and RoSta[11]. During the development of those projects, a set of ontologies of robot skills was implemented with the goal to create an intelligent support system for reconfiguration and adaptation of robot-based manufacturing cells. Those ontologies evolved throughout the scope of two other European projects, ROSETTA and PRACE (Stenmark and Malec, 2013). The former gave its name to the current ontology. The ROSETTA ontology[12] has further been employed in the research projects SMERobotics (Perzylo et al., 2019b) and SARAFun (Riva and Riva, 2019). In these projects, the ontology has been used to enhance cognitive abilities of robots that are required to plan and execute assembly tasks. The core ontology has been reorganized after the initial release (Jacobsson et al., 2016),

[11] Robot standards and reference architectures Project

[12] `https://github.com/jacekmalec/Rosetta_ontology` (accessed 22 May 2022)

and new case studies on robot programming support (Topp and Malec, 2018), and skill reusability in industrial scenarios (Topp et al., 2018) have been developed.

Originally, the ROSETTA ontology did not rely on any upper ontology, however, for more general terms regarding the robotics domain, it currently uses CORA (Schlenoff et al., 2012). Since CORA relies on SUMO (Niles and Pease, 2001), one can assume that ROSETTA utilizes SUMO as its upper ontology. Even though SUMO is written in SUO-KIF[13], ROSETTA is distributed in OWL. The *Knowledge Integration Framework* (Persson et al., 2010) connects all heterogeneous parts of the ROSETTA system: user GUI, simulation, external knowledge sources, task demonstration and the robot. It is the core of the whole system and its goal is to represent, store, adapt, and distribute knowledge across engineering platforms. The data, available in the *AutomationML* data exchange format[14], is stored as RDF triples.

Several git repositories exists which are proof of the availability of the ontology and parts of the software framework. However, there is no unique repository containing all pieces of the system as a whole, which reduces the degree of accessibility. Indeed, some parts of the framework (e.g., graphical user interface to program the robot) cannot be found at all. Furthermore, the OWL file does not contain definitions in natural language, which hinders the understanding of the formalization. Regarding the scalability of the system, which is a crucial element of any industrial environment, it is not possible to say much due to the small size of the conducted experiments.

CORA Ontology The 1872–2015 IEEE *Standard Ontology for Robotics and Automation* (Schlenoff et al., 2012)[15], was developed in the context of the IEEE work group *Ontologies for Robotics and Automation* (ORA). The standard defines an ontology for the robotics and automation domain which includes key terms as well as their definitions, attributes, constraints, and relationships. Sub-parts of the standard include a linguistic framework, generic concepts (an upper ontology), a methodology to add new concepts, and sub-domain ontologies.

[13]Variant of KIF, a knowledge representation language.

[14]AutomationML is an ongoing standard initiative that aims at unifying data representation and APIs used by engineering tools, `http://www.automationml.org` (accessed 22 May 2022)

[15]`https://github.com/srfiorini/IEEE1872-owl` (accessed 22 May 2022)

The purpose of the standard is to provide an overall ontology and an associated methodology for knowledge representation and reasoning in robotics and automation, together with the representation of concepts in an initial set of application domains. However, by itself, the ontology is too general to be useful in complex applications. Nevertheless, the ontology has been adopted by several authors, and extended for the respective application domain. For example, Jorge et al. (2015) consider a scenario where robots need to cooperate with a human for performing the task of collecting and delivering items.

OROSU The *Ontology for Robotic Orthopedic Surgery* (OROSU)[16] (Gonçalves and Torres, 2015) considers surgical robotics, and was applied in a scenario where a hip resurfacing surgery is performed. In this scope, the main goal of the research related to ontologies was to build a knowledge-based framework for this surgical scenario, along with a formal definition of components and actions to be performed during the surgery. The work was developed under the HIPROB and ECHORD projects, funded by the Portuguese Science Foundation and the EU-FP7, respectively. The framework is among the first to integrate robotic ontologies in the domain of surgical robotics. The OROSU ontology relies on SNOMED CT (Wang et al., 2002), the CORA ontology (Schlenoff et al., 2012) and the KnowRob framework (Tenorth and Beetz, 2013), which were adopted as the upper and reference ontologies. The formal language used to write the ontology was OWL. Material about OROSU is accessible, but it is rather sparse. Indeed, the ontology lacks of natural language definitions, which makes it more difficult to understand the specific meaning of the terms. Moreover, the system does not seem to be used by other researchers apart from the developers.

PMK *Perception and Manipulation Knowledge* (PMK)[17] (Diab et al., 2019a) is a knowledge-based framework that considers task and motion planning capabilities. The reasoning scope of PMK is divided into four parts: reasoning about perception, object features, situations, and planning. The ontological modeling in PMK is divided into three gradual layers called *metaontology*, *ontology-schema*, and *ontology instance*. Moreover, the PMK ontology relies on the SUMO (Niles and

[16] `https://github.com/pbsgoncalves/OROSU` (accessed 22 May 2022)
[17] `https://github.com/MohammedDiab1/PMK` (accessed 22 May 2022)

Pease, 2001) and CORA (Schlenoff et al., 2012) ontologies. PMK attempts to facilitate the process of manipulation by providing the required components for task and motion planning such as geometric reasoning, dynamic interactions, manipulation and action constraints. However, since the system has been recently published, it has not yet been extended among other researchers. The PMK ontology was implemented using the OWL language. Additionally, a querying interface based on SWI Prolog and its Semantic Web library is provided. It serves for loading and accessing ontologies represented in OWL using Prolog predicates.

ORO *OpenRObots* (ORO)[18] is focused on a common representation framework for autonomous robots with emphasizes on human-robot interaction (Lemaignan et al., 2010). The framework was meant to enhance a robot's interaction with complex and human-inhabited environments, where robots are expected to exhibit advanced cognitive skills such as object recognition, natural language interaction, task planning, cooperation with other agents, etc. The authors argued that these capabilities need to share common knowledge of the environment where the robot operates. ORO's primary component is the *OpenRobots Common Sense Ontology*, which provides a set of concepts upon which the robot can form statements about the world (Lemaignan et al., 2010). The ontology is built upon the OpenCyc ontology (Lenat, 1995). The *minimalKB*[19] ontology is a lightweight version of ORO, which shares the same objective and functionality as its predecessor. The underlying RDF triple storage used by ORO is based on the Jena framework, which is used together with the Pellet reasoner (Sirin et al., 2007).

CARESSES CARESSES[20] is an international research project whose goal is to design the first robots that can assist elderly people and adapt to the culture of the individual they are taking care of (Bruno et al., 2017). The robots are expected to help the users in many ways including reminding them to take their medication, encouraging them to stay active, and helping them keep in touch with family and friends. Each action should be performed with attention to the older person's customs, cultural practices and individual preferences.

[18] `https://www.openrobots.org/wiki/oro-server` (accessed 22 May 2022)
[19] `https://github.com/severin-lemaignan/minimalkb` (accessed 22 May 2022)
[20] `http://caressesrobot.org/en` (accessed 22 May 2022)

The principle idea of CARESSES is to built upon four fundamental backbones: transcultural robotic nursing, cultural knowledge representation, culturally sensitive planning and execution, and culture-aware human-robot interaction. Cultural knowledge is mainly represented using ontologies. It is used to enhance the robotic nursing by an integration of the knowledge into several processes, e.g., task planning, task execution and human-robot interaction. Other methodologies such as fuzzy logic and Bayesian networks are also employed. Despite the short life of CARESSES, it has become a prominent application example of how ontologies can enhance the autonomy of robots. Nonetheless, the project still presents some drawbacks. First, not all of the implemented solutions are publicly available. Second, the ontology lacks informal concept definitions which makes it hard to understand the formal definitions.

2.3.2 Comparison of Frameworks and Projects

For the purpose of comparing the frameworks considered herein, in this section, it will be explored how each of them addresses the different aspects included in the classification of ontologies proposed along Section 2.2. Concretely, the ontological scope, reasoning scope and application domain of considered frameworks will be examined and contrasted.

Ontology Scope

In this section, it will explored which of the terms discussed in the Section 2.2.1 are defined in each of the selected projects. Table 2.2 summarizes the findings of this study. Note that two versions of the KnowRob ontology are considered, which are referred to as KnowRob1 and KnowRob2 for the scope of this investigation where KnowRob1 corresponds to an earlier version based on the OpenCyc ontology (Lenat, 1995), and KnowRob2 to a more recent version which is based on a simplified version of the DOLCE ontology (Masolo et al., 2003).

Object Most of the compared frameworks consider objects from an endurant perspective. Both, KnowRob1 and ORO, employ the notion of `Spatial Thing` from Cyc ontology: *the collection of all things that have a spatial extent or location relative to some other Spatial Thing or in some embedding space.* OROSU uses

Term	KnowRob 1/2	ROSETTA	ORO	CARESSES	OROSU	PMK
Objects	**Yes**/**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**
Environment map	**Yes**/**Yes**	No	No	No	**Yes**	**Yes**
Affordance	No/**Yes**	No	**Yes**	No	**Yes**	No
Action	**Yes**/**Yes**	No	**Yes**	**Yes**	**Yes**	**Yes**
Task	No/**Yes**	**Yes**	**Yes**	No	No	**Yes**
Activity	No/No	No	No	**Yes**	No	No
Behavior	No/No	No	No	No	No	No
Function	No/No	No	No	No	No	**Yes**
Plan	No/**Yes**	No	**Yes**	No	No	No
Method	No/**Yes**	No	No	No	No	No
Capability	**Yes**/**Yes**	**Yes**	No	No	No	**Yes**
Skill	No/No	**Yes**	No	No	No	No
Hardware	**Yes**/**Yes**	**Yes**	**Yes**	No	**Yes**	**Yes**
Software	**Yes**/**Yes**	**Yes**	No	No	**Yes**	**Yes**
Interaction	No/No	No	No	No	No	No
Communication	**Yes**/No	No	No	No	No	No

Table 2.2: List of relevant terms for the autonomous robotics domain, and their coverage in considered projects.

SUMO's definition: an object *corresponds roughly to the class of ordinary objects*. Examples include regular physical objects, geographical regions, and locations of processes. ROSETTA does not concern about spatial regions and only focuses on `Physical Objects`: *Every automated work cell consists of physical objects. Some objects, devices, are active and have skills, while other, work pieces, are passive and are manipulated by the devices.* The PMK ontology defines the concept `WSObjectClass`, which is split into `Artifact`, `Artifact Components` and `Collections`. For example, a cup (artifact) is an object that has a body and a handle (artifact components), and it could be served with saucer (collection). On the other hand, KnowRob2, based on the DUL Ontology, considers not only physical entities: *any physical, social, or mental object, or a substance. Objects are always participating in some event (at least their own life), and are spatially located.* In CARESSES, no natural language definition exists, but `Object` is defined as a subclass of `Topic`, which is *any theme a robot can talk about.* This general definition also includes social objects that may have no spatial form such as an idea or a plan that could be communicated to another agent.

Environment Map In both versions of KnowRob, it is possible to find the concept of `SemanticEnvironmentMap` as a sub-class of `Map`. However, there is no natural language definition. OROSU defines places and environments where the robot works (e.g., `CTRoom`, `EngineeringRoom`, `OperatingRoom`) which are sub-classes of `Room`, and connected to actions which are expected to take place in the represented places. PMK employs the notion of `Workspace` which has three gradual sub-classes, `Region` (i.e., free and occupied regions), `Physical Environment` (topology of the environment entities), and `Semantic Environment` (semantic information of the workspace).

Affordance The concept of `Affordance` is not exactly defined in any of the works that are subject of this study, still, it is possible to find some related definitions. ORO defines the property `canBeManipulated` which indicates that *an object can be manipulated*, and that *the agent knows a grasping point for the object*. Thus, if the object can be manipulated, it is movable as well. OROSU describes a similar property, called `CanGrab`, which indicates that *a device can grab an object*. A model of affordances in the KnowRob2 ontology is subject of this work, and will be described in Section 3.2.

Action KnowRob1 uses the definition provided by the Cyc ontology in which an `Action` is considered as an event. ORO provides a more concrete natural language definition of the term: *The collection of Events that are carried out by some "doer". Instances of Action include any event in which one or more agents effect some changes in the state of the world.* KnowRob2 takes the term from the DUL ontology: *an event with at least one agent that is participant in it, and that executes a task that typically is defined in a plan, work flow, project, etc.* PMK defines the notion of `ActionClass` with three specifications: `Task`, `Sub-task` and `AtomicFunction`. For instance, *picking* would be an action class whose task can be a reachability-test, and it can have a sub-task that provides a list of potential grasping poses. Similarly, in OROSU the notion of sub-task has also been defined to form complex actions. Some of the analyzed frameworks do not provide any natural language definition for `Action`, even though they include the term in their ontology: CARESSES and OROSU.

Task In the ROSETTA ontology, the `Task` concept is formalized as disjoint with the concepts `Operation`, `Skill`, `Physical object`, and `Property`, but no further information is provided. Another example of formalization is found in PMK, where `Task` is a sub-class of `Action`. ORO views it as *an action considered in the specific context of robotics*. KnowRob2 uses DUL's definition: *an event type that classifies an action to be executed*.

Activity Only CARESSES covers the term of `Activity`, which is modeled as a sub-class of `Entity`. It has several sub-classes such as `Cooking`, `Reading` and `Sleeping`. Natural language definitions are not provided.

Behavior None of the frameworks defines `Behavior`.

Function Just the PMK ontology includes a term related to `Function`. Specifically, the notion of `AtomicFunction` which is a sub-class of `Action`. It refers to atomic entities in the action hierarchy, e.g., a computation method.

Plan KnowRob2 takes DUL's definition of `Plan`: *a description having an explicit goal, to be achieved by executing the plan*. Each plan defines a task that can be executed by following the plan. The execution of a plan is a situation that satisfies the plan, that is, where a sequence of actions was performed that satisfies the activity organization defined by the plan. ORO does not provide a definition in natural language, but a `Plan` is defined as being equivalent to a thing with a temporal extent, which is either a `Situation` or a `Time Interval`. The ORO viewpoint seems to be problematic as, for example, a time interval cannot be considered as a plan in the common sense.

Method KnowRob2 is the only framework that considers the notion of `Method`. The definition provided by DUL is adopted: *a method is a description that defines or uses concepts in order to guide carrying out actions aimed at a solution with respect to a problem*. This notion is similar to the notion of `Plan`, but more general in that variances of following the same method could satisfy different plans.

Capability The ROSETTA ontology defines `Capability` as *a property of a skill*, and, in PMK, the property `has capability` is defined as *a property of a robot*. In KnowRob1, as part of the module *Semantic Robot Description Language* (SRDL), capabilities *are considered to exist when the robot has a component which enable them*. In KnowRob2, `Capability` is formalized as sub-class of `Quality` meaning that agents inher individual qualities that may change over time, for example, due to attrition or because new software components are available.

Skill The term of `Skill` is the core of the ROSETTA ontology where *a skill represents an action, that might be performed (by a device) in the context of a production process*. Similarly to the task-action dichotomy, skills are used to classify particular actions that occurred.

Hardware Component The specific term of `Hardware component` is not tackled in any of the studied works. However, most of the works address one or more concepts related to its notion. KnowRob includes the SRDL ontology, which considers representations for robot hardware, among others. OROSU, from SUMO ontology, makes use of the term `Device` which *is an artifact whose purpose is to serve as an instrument in a specific subclass of a process*, where `Artifact` *refers to any object that is the product of a making*. An alike definition is found in ORO, where an `Artifact` *is a specialization of inanimate object, and each instance of artifact is an at least partially tangible thing which was intentionally created by an agent partially tangible (or a group of them working together) to serve some purpose or perform some function*. ROSETTA also includes the term of `Device`: *an active physical object which has some skills*. These notions are used to define `Sensors`. In PMK it is possible to find the terms `Actor Class` (e.g., robot components), `Sensor Class` (e.g., device components), and also the term `Artifact`, but none of those terms is defined using natural language. As shown, definitions of hardware components are closely related to the processes and events in which they play a role.

Software Component KnowRob includes SRDL, which extends KnowRob with representations for robot software, among others. Terms related to the

notion of `Computer-based Algorithm` can be found in both OROSU and PMK. ROSETTA ontology defines `Software` as *an abstract which has some skills*.

Interaction None of the frameworks defines `Interaction`.

Communication The ROSETTA ontology defines `Communication Property` as *the description of communication parameters*. It also includes the term of `Communication`, but as a subclass of device, which seems counter-intuitive. Therefore, it cannot be stated that ROSETTA considers `Communication`. Apart from that, the only complete and coherent definition related to `Communication` is found in KnowRob1, where the term of `Communicating` is taken from the Cyc ontology. It *is a specialization of purposeful action* and characterized by one or more information transfer sub-events. Each instance of `Communicating` is an event in which the transfer of information between agents is a focal action, that is, communicating is the main purpose and/or goal of the event.

Reasoning Scope

In this section, the selected frameworks will be compared regarding their reasoning scope. Specifically, it will be analyzed whether or not the different frameworks are used to support the cognitive capabilities presented in Section 2.2.2.

Recognition and Categorization Ros et al. (2010) present a use case where ORO is used to support object detection by disambiguating incomplete information extracted from human-robot interaction. The work proposes a scenario in which a human provides vague instructions such as *look at that object*, where the object can correspond to several entities in the environment. The ontology is used to represent facts about the user's visual spectrum, and the description of objects so that the system is able to infer and recognize which is the most likely object.

Within the framework of CARESSES, Menicatti et al. (2017) introduce an approach for human activity recognition where cultural information (represented using ontologies) drives the learning while improving the performance of the classification. Three human activities are considered: lying on the floor, sleeping on a futon and sleeping on a bed. Specially, lying on the floor and sleeping on a

futon are extremely similar classes, thus, cultural knowledge (e.g., user is from Japan) was used to improve the performance of the recognition algorithm.

KnowRob is concerned with acquiring experience knowledge through experimentation, and from observations (Beetz et al., 2018). One of the considered modalities is the virtual reality where force interactions can be monitored trivially. However, the intention and the task that the human executes might be unknown. KnowRob uses ontologies to represent tasks as patterns of force interactions, and state changes to be able to recognize high level activities given force event and state observations (Haidu et al., 2018; Haidu and Beetz, 2019).

Decision Making and Choice In the context of CARESSES, the robot builds a model of a person, which is represented using the ontology. The robot adapts its behavior to the facts of the knowledge base. For the adaptation, a Bayesian Network is employed in combination with the ontology (Bruno et al., 2019). Diab et al. (2017, 2019a) describe a robot system which adapts the execution of plans with the support of the PMK ontology. Based on the beliefs about the workspace (reachability of objects, feasible actions to execute, etc.), the system makes decisions about the distribution of actions among different robotic arms, and also about action parameters. Ontologies have also been used in KnowRob for object selection based on dispositional qualities (Beßler et al., 2020b). This will be described in Section 3.2.

Perception and Situation Assessment In the context of ORO, Sisbot et al. (2011) present a situation assessment reasoner which generates relations between objects in the environment and capabilities of agents. Being fully integrated to a complete architecture, this reasoner sends the generated symbolic knowledge to a fact base which is built on the basis of an ontology, and which is accessible to the entire system. The authors discussed how, based on spatial reasoning and perspective taking, the robot is able to reason from the human's perspective to reach a better understanding of human-robot interaction.

An example of a knowledge-based perception system is RoboSherlock (Beetz et al., 2015a). RoboSherlock uses an ontology to define different perception operators: what input they expect, what output they generate, etc. This information,

together with background knowledge KnowRob provides, is used for contextualized composition of perception pipelines. Furthermore, the PMK framework considers simplified perception using RFID tags where the ID is associated to entities in the knowledge base (Diab et al., 2019a). For situation assessment, relations between agent and objects in the environment are generated such as that a particular object can be pushed by a robot arm if it can be reached by it.

Prediction and Monitoring One of the crucial aspects for robots that have to perform manipulation tasks is to predict the effects of actions. To this end, KnowRob introduced the notion of pre- and post-actors of actions (Tenorth and Beetz, 2012). Pre-actors are the entities that must be known before the robot can enter the execution of the action, and post-actors describe what is expected when the action is successfully executed. For example, the task of *cracking an egg* would have a pre-actor of type `Egg` that takes the role of being the *destroyed* entity in the action, while the yolk and the shell would be considered as *created* entities. Hence, the robot can predict what action it needs to execute in order to obtain some egg yolk, e.g., in case it is required in a cooking activity. In a newer version of KnowRob, the pre- and post-actor relations are replaced by corresponding concepts describing roles that need to be taken by some entity when an action is performed.

Problem Solving and Planning A planning method in the KnowRob framework that uses ontologies was described by Beßler et al. (2018c). This mechanism is also subject of this work (Chapter 4). The rational is that the goal state, a fully assembled product, is described in an ontology, and that the robot compares its belief state with the goal state in order to infer what steps are required, and what objects are missing to build the product from parts that are available. An earlier version of the KnowRob ontology included action definitions axiomatized by roles that are separated into input and output of the action, and, in addition, defined a partial ordering on steps of a task (Tenorth and Beetz, 2012). This information was used to generate possible sequences of steps that would execute a task. The KnowRob ontology has further been used to represent motion constraints that were used by a constraint-based motion planner to generate

appropriate motions for the task ahead, and the objects involved (Tenorth et al., 2014). Another example is PMK which considers representation of pre- and post-conditions of actions for planning (Diab et al., 2019a).

Reasoning and Belief Maintenance As this study only considers frameworks that use ontologies, one can also expect that some form of reasoning is supported. Be it via a standard reasoner, or by rules that infer new facts from given ones. For example, PMK and KnowRob both use predicate logic rules to work with knowledge encoded in ontologies. Belief maintenance is not covered in both systems, however, KnowRob partly evades this problem by dynamically constructing parts of the knowledge base from perceptions of the robot. In the context of ORO, Warnier et al. (2012) propose a novel algorithm for belief maintenance, which relies on the use of the ORO ontology to represent facts about the environment. The robot builds an individual symbolic belief state for each agent taking part in the task. Within the CARESSES project, Bruno et al. (2019) present an algorithm for belief maintenance of person-specific knowledge, which uses culture-specific knowledge to drive the search. Diab et al. (2019a) use the PMK ontology to represent semantic maps of the robot's workspace. A reasoning process over those symbolic beliefs allows, for instance, to identify qualitative spatial relations. The KnowRob ontology was also used to represent a belief state of a robot (Beßler et al., 2018c). The environment representation includes spatial information and encyclopedic information about objects (Tenorth et al., 2010a).

Execution and Action In order to enhance autonomy when executing actions, ROSETTA proposes a system which translates high-level task-oriented language (ontology-based) into either the robot native code, or calls at the level of a common API, e.g., ROS (Stenmark et al., 2015b). This system is capable of handling complex, sensor-based actions, as well as the usual movement primitives.

In the context of CARESSES, Sgorbissa et al. (2018) discuss how guidelines describing culturally competent assistive behaviors can be encoded in a robot to effectively tune its actions, gestures and words. In the same context, an online constraint-based planner is used together with the *cultural knowledge base* to adapt the execution of the robot actions (Khaliq et al., 2018). When launched,

the planner requests operators and actions from the knowledge base. During execution it listens for new goals, updates on the execution status of actions, and messages about the state of the environment and people in the environment.

Gonçalves and Torres (2015) discus how the use of the OROSU ontology is beneficial to track the execution of actions of robotics systems in medical (surgical) scenarios. In this work, the main purpose is to adapt the robot pose to possible unexpected motions while performing drilling tasks during surgery. The robot pose adaptation is performed following the approach presented by Torres et al. (2015). The overall process is modeled with the OROSU ontology, which controls the robot's actions and sub-actions and allows the user to follow the sequence of those actions.

The *Cognitive Robot Abstract Machine* (CRAM) (Beetz et al., 2010) is a plan executive that uses the KnowRob ontology to rerpresent objects in the belief state, where they are located, how they can be operated, etc. Knowledge base queries are explicit steps in the plan contextualization procedure of CRAM (as indicated in Figure **??**). KnowRob has further been used to ontologically describe motion constraints that are used by a constrained-based motion controller to generate motions that execute a specific task (Tenorth et al., 2014), and to transform vague task descriptions in natural language to an ontological representation while using WordNet to disambiguate word senses (Tenorth et al., 2010b).

Interaction and Communication Lemaignan et al. (2011) present a simple natural language processor which employs ORO for dialogs with humans. The robot parses English sentences and, by means of the knowledge base, infers the sense of the sentences and answers the human's questions (both in English and with RDF statements). In addition, in a scenario where a robot disambiguates the information provided by the user, the ontology triggers the robot-human interaction – e.g., asking the user for further information (Ros et al., 2010).

In the scope of CARESSES, Bruno et al. (2018, 2019) describe two scenarios where human-robot speech-based interaction is adaptable by means of cultural knowledge-based assumptions. The system stores knowledge about the cultural information of the users, which is used by the robot's finite state machine to control the interaction.

The KnowRob ontology was used and extended in a research project that was concerned with mixed human-robot rescue tasks (Yazdani et al., 2018). The scenario is that a team of different robots have to locate an avalanche victim in hilly terrain where, first, a flying robot scans the area, and then, after the victim was found, the robot communicates the particular location, and an image captured by its camera to the human operator, and to the other robots. The KnowRob ontology was used to represent the communication acts. Note that the communication was not natural but instead based on a custom protocol.

Remembering, Reflection, and Learning Different approaches combining ontologies and robot learning are proposed in the context of ROSETTA. First, ontologies are used to support the kinesthetic teaching so that the learned primitives are sematically represented as skills (Stenmark et al., 2018). Second, Topp et al. (2018) discuss how the representation of already learned robot's skills enhances the transfer of knowledge between robots.

KnowRob considers experience knowledge as a part of the ontological characterization (Beetz et al., 2018). When a robot performs a task, a detailed story about the activity is stored. The story includes a *narrative* represented as an ontology that describes what events occurred, when they occurred, and what objects play what roles for the events. The narrative is coupled with control-level data such that learning mechanisms can correlate parts of the narrative to the control level data that was monitored during execution. Earlier, the knowledge web service openEASE was introduced (Beetz et al., 2015c). openEASE is used as a central storage for experience knowledge, and it has been adopted for KnowRob as a storage platform. A more detailed discussion about acquisition and use of experience knowledge in KnowRob will be provided in Chapter 5.

Application Domain Scope

In this section, it will be discussed in which domain the considered frameworks have been applied in previous works. Recall that, following the classification proposed in Section 2.2.3, two main domains are considered: industrial and service robotics. Along this section, the more specific sub-domains where the frameworks were used will be discussed.

In principle, it is noticed that most of the frameworks were conceived to be used in service scenarios. Indeed, the only framework which is intended to be used in industrial scenarios is ROSETTA, whose case studies consider industrial problems such as intuitive robot programming and safe human-robot interaction. Moving to the frameworks focused on service robotics, ORO is used in case studies where the robot is meant to perform everyday activities which usually take place in houses or similar environments such as human activity recognition and human-robot speech interaction. Closely related to it is KnowRob, which is mainly used for household scenarios, but also in scenarios where the robot is expected to perform some professional service tasks (e.g., cooking, in-store logistic processes). PMK presents case studies where the principle aim is to enhance robot manipulation, which is a general purpose robot ability which could potentially be used in a wide range of scenarios. Nevertheless, PMK has not been used nor thought to be used in industrial scenarios, thus, it can be considered under the umbrella of the service robotics domain. CARESSES is entirely developed towards the assistance of elderly people by means of robots with cultural-related knowledge. Finally, OROSU is mainly applied to the medical domain, particularly, the surgical robotics sub-domain. A summary of the survey is provided in Table 2.4.

2.4 Discussion

In this section, the most relevant findings of the review of projects which use ontologies to support robot autonomy will be summarized. In general, the six considered projects provided enough information to allow to reuse and reproduce their works. It is worth mentioning that KnowRob was comparatively well documented, including code, ontologies, as well as wiki pages explaining how to install and to use the different tools. It is also true that KnowRob is one of the first frameworks which made use of ontologies for autonomous robots. Meanwhile, other projects are much more recent. One important fact is that ORO seems to no longer be maintained. The reason is that it was a project partially developed by the same researchers who started developing KnowRob. That said, ORO is also well documented. Therefore, even though it is not currently maintained, it is, nevertheless, possible to reuse some parts of it.

Cognitive Capability	KnowRob	ROSETTA	ORO	CARESSES	OROSU	PMK
Recognition and categorization	Haidu et al. (2018)	-	Ros er al. (2010)	Menicatti et al. (2017)	-	-
Decision making and choice	Beßler et al. (2020b)	-	-	Bruno et al. (2019)	-	Diab et al. (2017; 2019a)
Perception and situation assessment	Beetz et al. (2015a)	-	Ros et al. (2010), Sisbot et al. (2011)	-	-	Diab et al. (2019a)
Prediction and monitoring	Tenorth et al. (2012)	-	-	-	-	-
Problem solving and planning	Beßler et al. (2018c), Tenorth et al. (2012)	-	-	-	-	-
Reasoning and belief maintenance	Beßler et al. (2018c), Tenorth et al. (2010a)	-	Warnier et al. (2012)	Bruno et al. (2019)	-	Diab et al. (2019a)
Execution and action	Beetz et al. (2010), Tenorth et al. (2014; 2010b)	Stenmark et al. (2015b)	-	Sgorbissa et al. (2018)	Gonçalves et al. (2015)	-
Interaction and communication	Yazdani et al. (2018)	-	Ros et al. (2010), Lemaignan et al. (2011)	Bruno et al. (2018; 2019)	-	-
Remembering, reflection and learning	Beßler et al. (2018), Tenorth et al. (2015c)	Stenmark et al. (2018), Topp et al. (2018)	-	-	-	-

Table 2.3: List of cognitive capabilities for the autonomous robotics domain and their coverage in different projects. It is possible to find the reference to the articles in which the different reasoning capabilities are addressed using ontologies.

Framework	KnowRob	ROSETTA	ORO	CARESSES	OROSU	PMK
Application Domain	Household	Manufacturing	Household	Elderly Care	Medical Care	Manipulation

Table 2.4: Application domain of related systems.

The first classification criterion that was employed in this work is concerned about the ontological scope of the considered projects, namely which of the relevant terms for the robotics domain are covered by the ontologies used in the projects (Section 2.2.1). Every project that fulfilled this criterion defines its own domain ontologies. These are, in all cases, derived from well known foundational ontologies (i.e., SUMO, DUL, and Cyc), and encoded in a common ontology languages (i.e., SUO-KIF, OWL, CycL). Hence, all of the projects have very general categories such as `Object`, `Event` and `Environment` defined in their ontologies by importing them from a common foundational ontology.

Looking at Table 2.2, it can be seen that most of the considered projects include the terms `Action` and `Task` to capture the notion of robots acting in their environment towards the achievement of a goal. In the same table, it can be observed that `Behavior`, `Function` and `Method` are only rarely covered. This is rather surprising given the fact that they are extremely related to applications where agents execute actions. It is likely that this is the case due to their polysemous nature. Therefore, more work is needed in order to come up with a standard definition for them. Some other terms such as `Plan`, `Capability`, `Hardware` and `Software` are defined by at least half of the surveyed projects, which indicates that they are relevant for the domain but it is still necessary to continue working on them. Furthermore, the terms `Interaction` and `Communication` are not defined in any of the projects, even though some projects propose scenarios where robots interact and communicate with other agents (e.g., humans). Hence, no ontological reasoning is done in this regard. Surprisingly, some other terms with strong connotation in robotics such as `Activity` and `Skill` are also only rarely considered in formal ontological models.

The second classification criterion is related to the reasoning scope of the ontologies of each of the projects (Section 2.2.2). In other words, which cognitive capabilities of autonomous robots have already been supported by those ontologies. The review has shown that a wide range of cognitive capabilities are already covered, at least in a prototypical way, by knowledge-based approaches that employ ontologies (Table 2.3). Out of the nine cognitive capabilities considered in this work, two are commonly tackled within the studied frameworks: reasoning and belief maintenance, and execution and action. Reasoning is well supported

by many ontology formalisms as standard reasoners exist that can perform this task automatically and efficiently for many practical problems. On the other hand, robots are essentially developed to automatize the execution of actions; hence, it is not a surprise to also find several works tackling this cognitive task. Surprisingly, important cognitive capabilities such as decision making, problem solving and planning, are only tackled in a few projects. Probably, the existence of other widely used formalisms to do planning (e.g., PDDL) is the principal reason for this fact. Finally, it can also be seen in Table 2.3 that ontologies have been used for learning tasks. It seems plausible that considering the current trend of non-explainable machine learning approaches, the formal and logical nature of ontologies could be beneficial in robot learning applications.

Lastly, the application domain of the selected projects was also investigated. Most of the projects were conceived with the purpose of being used in service robotics applications. Indeed, only ROSETTA was specifically designed for industrial robotics applications. Nevertheless, the frameworks considered in this work that are designed for service robotics scenarios are somewhat general such that it can be expected that they can also be employed in industrial settings. For the case of the KnowRob knowledge base, an industrial assembly scenario will be considered in the scope of this work (Chapter 4).

Considering the work presented in this section, it is possible to state that ontologies have proved to be valuable for the robotics domain in order to support robot autonomy. It is true, however, that the great effort done by all the frameworks discussed in this chapter should be continued and extended with new applications. Furthermore, it is still pending to promote the reuse of existing ontologies, which seeks for homogeneity and interchangeability among different frameworks. This will only be possible if researchers share and properly document their contributions. Indeed, the spread of standard conceptualizations would also help to achieve the previous purpose. In this regard, the survey presented herein has been summarized on a website[21] where users can access the major findings of this work. Specifically, the page allows to search/select by projects and by each of the criteria employed along this study.

[21] `https://ease-crc.org/ontology-survey-2019` (accessed 22 May 2022)

2.5 Conclusion

This chapter was concerned with research located in the intersection of the fields autonomous robotics and formal ontologies. First of all, it was investigated what the basic notions in the robotics domain are, and it was contrasted how these notions are commonly defined in formal ontologies, and how they are used in the robotics domain. Furthermore, the term autonomy was broken down into several cognitive capabilities that a fully autonomous robot must exhibit, and projects were systematically searched that employ ontologies for the realization of one or more of these capabilities. Each project that was discovered builds on top of a well known foundational ontology that is used to conceptualize parts of the robotics domain. Finally, the projects were compared with each other according to what terms are defined in their ontologies, how these terms are used to support different cognitive capabilities of a robot, and in which application domain they are used. The study has shown that a wide range of cognitive capabilities is considered by existing frameworks, and that each of the frameworks covers a set of relevant terms in their ontologies to support such capabilities. However, several relevant terms are only rarely covered.

As a final remark, there is a huge potential that ontologies can become essential in the robotics domain for applications that require the robot to act within the boundaries of a certain legal framework, or where the behavior of the robot must be assessed by a human operator through abstract queries. This includes topics such as explainability, ethics-awareness, and human-robot interaction. One of the remaining problems the research community has to tackle is that of reusability of ontologies in the robotics domain. To this end, additional efforts regarding the standardization of robot ontologies are required.

3 An Ontological Framework for Robot Knowledge

In this chapter, foundations of SOMA and one of the extensions of the core ontology will be presented. The first contribution of this chapter is the SOMA ontology that represents both the physical as well as the social context of everyday activities. Such tasks seem to be trivial for humans, however, they pose severe problems for artificial agents. For starters, a natural language command requesting something will leave much of the information necessary for performing the task unspecified. Rather than enumerating fine-grained physical contexts, SOMA sets out to include socially constructed knowledge about the functions of actions to achieve a variety of goals or the roles objects can play in a given situation. This is facilitated by the link between the physical and social context in SOMA where relationships are established between occurrences and generalizations of them. The extension of SOMA that will be presented is concerned with inferring possible actions that can be accomplished with a given object at hand. This cognitive task is commonly referred to as inferring the affordances of objects. Accordingly, the second contribution of this chapter is a novel conceptualization of affordances and its realization as a DL ontology. The key idea of the framework is that it proposes candidate affordances through inference, and that these can be validated through physics-based simulation. The proposed formal models are implemented as an OWL ontology based on the DUL foundational ontology. Related prior publications are Beßler et al. (2021), and Beßler et al. (2020b).

3.1 The Socio-physical Model of Activities (SOMA)

Despite being undoubtedly ubiquitous, the domain of *everyday activities* poses considerable challenges. Many people perform activities such as cooking almost every day. This includes to select and manipulate ingredients, use tools and devices, arrange the dishes, and clean up afterwards – and to do it all quickly and robustly without recourse to an advanced computational theory. Further, the amount of information provided in a description of a task – such as a natural language command requesting its completion – is much less than the amount of information needed to perform the task. This raises the question how humans are able to decide so quickly what to do next, despite ambiguity and underspecification.

Lenat and Feigenbaum (1991) observe that *more knowledge implies less search*. Knowledge of many possible plans, as well as knowledge of the world in general, seems to be the secret of human performance. There is no algorithmic reason why tomatoes and oregano go well together, or why a raw egg must be handled with care. The cook simply has to know these things. Such knowledge of the world is taught, observed, and then ingrained by practice. As Anderson observes, *an agent has a great deal of knowledge [of everyday activities], which comes as a result of the activity being common* (Anderson, 1995). As human beings, we acquire such knowledge naturally over the course of our lives. A lot of what we learn, we learn by doing, or by watching others. This suggests that a robot must have mechanisms to organize and interpret observations, either of its own behavior or of other agents, into structures that are then amenable for other computational tasks. To this end, the concept of NEEM was introduced (Beetz et al., 2018, 2020). NEEMs are comprehensive logs of raw sensor data, actuator control histories and perception events, all semantically annotated with information about what the robot is doing and why using the terminology provided by SOMA.

The computational tasks that must be solved when acting in the physical world are often very complex and beyond what is thought to be tractable. This, however, is only the case when these problems are regarded in their full generality, and not for restricted versions of these problems. However, the knowledge representing such pragmatic solutions goes beyond modeling physical events and requires models of the social context by means of which the physical events can be realized

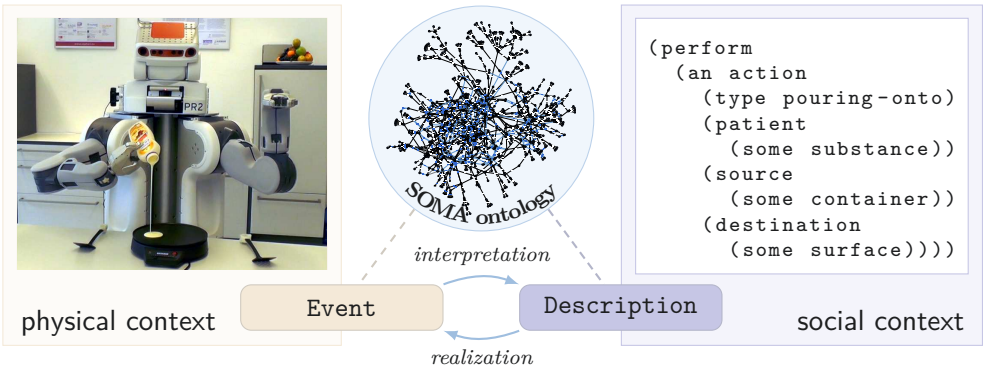

Figure 3.1: SOMA represents physical and social context, and supports robotic agents in interpreting observed events, and realizing abstract descriptions.

and interpreted. For this, an existing upper-level ontology will be employed in this work, and augmented with general design patterns and specific modules that are pertinent for robot knowledge modeling. In this section, an overview of this approach will be provided where all of the extensions to the given foundational framework rely on the differentiation between the observable physical domain and the conceptualized social interpretations thereof.

The overall goal of this research is to enable robotic agents to perform everyday activities with similar robustness and flexibility as human agents do. Given this aim, the robot must, in some sense, know what humans know about the world, at least as it pertains to everyday activities. This presents several challenges, beyond the scope of what needs to be known to represent such intricate and extensive domain. There lies the question of how to represent and structure this knowledge in order to realize a similar robustness, flexibility and efficiency in performance. In addition, there are challenges concerning the acquisition and learnability of the corresponding structures. In this section, it will be investigated how this knowledge can be represented. To this end, an employment of an existing upper level ontology, and the development of several ontology modules aimed to address this general ontology design challenge will be described. The resulting ontology is openly available[1], and additional documentation is available online[2]. As depicted in Figure 3.1, the focus lies on representing both the physical context of realized everyday activities, as well as interpretations thereof as the social context.

[1] `https://github.com/ease-crc/soma` (accessed 22 May 2022)
[2] `https://ease-crc.github.io/soma` (accessed 22 May 2022)

3.1.1 Related Work

Ontology-based knowledge representation and reasoning in autonomous robot control is a fairly extensive field of research with developments in both service and industrial robotics. In the following, most relevant works will be briefly discussed. A more detailed discussion about how ontologies are used to support robot autonomy is provided in Chapter 2.

One example in the industrial robotics domain is the ROSETTA project (Patel et al., 2012; Stenmark et al., 2015a). Its initial scope was reconfiguration and adaptation of robot-based manufacturing cells, however, the authors have, since then, further developed their activity modeling for coping with a wider range of industrial tasks. Other authors have focused on modeling industrial task structure, part geometry features, or task teaching from examples (Balakirsky, 2015; Polydoros et al., 2016; Kootbally et al., 2015; Perzylo et al., 2016). Compared to the everyday activity domain, industrial tasks considered in above works are more structured, and less demanding in terms of flexibility.

An approach to activity modeling in the service robotics domain is presented by Tenorth and Beetz (2017). Foundationally, their modeling is based on a subset of the discontinued OpenCyc ontology (Lenat, 1995) with much weaker axiomatization compared to SOMA's foundational layer, and less inferential power and guidance during modeling. The scope of their work is similar to the scope of SOMA as the authors also consider how activity knowledge can be used to fill knowledge gaps in abstract instructions given to a robotic agent performing everyday activities. However, the scope of this work is wider, as the presented ontology also considers how activity knowledge can be used for the interpretation of observations. The activity modeling is further more detailed in terms of activity structure as the processes and states that occur during an activity are also considered. Another difference is that, in their modeling, there is no distinction between physical and social context, but this dichotomy is central in SOMA.

A more general approach to activity modeling for robotic agents is presented by the IEEE-RAS WG ORA (Schlenoff et al., 2012). The group has the goal of defining a standard ontology for various sub-domains of robotics, including a model for object manipulation tasks. It has defined a core ORA ontology (Prestes et al., 2013), as well as additional modules for industrial tasks such as kitting (Fiorini

et al., 2015). In terms of scope, SOMA rather considers the service robotics domain, and employs a more fine-grained action model designed to formalize how actions are organized into motion phases separated by force dynamical events. In terms of methodology, SOMA differs in foundational assumptions asserted, which has important consequences on the structure of the ontology, modeling workflow, and inferential power. In the case of ORA, the SUMO upper-level ontology is used as foundational layer. However, the foundational layer of SUMO is rather weakly axiomatized compared to other models. In particular central in SOMA is the distinction between ground and descriptive concepts to represent physical and social activity context, and that this distinction is tightly coupled with the foundational layer.

3.1.2 Overview

In this section, the scope of SOMA, its underlying foundational commitments, and how it is organized will be discussed.

Scope

The broad scope of this work is everyday object manipulation tasks in autonomous robot control, and in particular the motion and force characteristics of objects. The research question driving us is whether a single general control program can be written that can generate adequate behavior in many different contexts: for different tasks, objects, and environments. The employment of a general plan thus requires an abstract task and object model, and a mechanism to apply this abstract knowledge in situational context.

A more fine-grained scope is defined through a set of competency questions that are documented in the NEEM-Handbook (Beetz et al., 2020). Some examples related to the modeling of affordances are *what can an object be used for*, and *what can an object be used with* (referring to the fact that affordances arise through the meeting of compatible dispositions), as well as *what cannot be used to manifest an affordance*. Thus, the ontology offers ways to indicate what objects – given semantic knowledge about them – provably can or provably cannot be used for some purpose, with undecided cases being passed on to other mechanisms, e.g., simulation-based testing.

Foundational Commitments

SOMA is based on the DUL foundational framework (Masolo et al., 2003). This decision is greatly motivated by their underlying ontological commitments. Firstly, DUL is not a revisionary model, but seeks to express stands that shape human cognition. It assumes a multiplicative approach. This work, however, seeks to apply a reductionist approach where possible – rather than capturing, for example, the flexibility of our usage of objects via multiple inheritance in a multiplicative manner, SOMA commits to a reduced *ground* classification and use a *descriptive* approach for handling this flexibility, as provided by the addition of the *Descriptions and Situations* extension of DUL (Gangemi and Mika, 2003). For this a primary branch of the ontology represents the ground **physical model**, e.g., objects and actions, while a secondary branch represents the **social model**, e.g., roles and tasks. All entities in the social branch are mind-dependent entities, i.e., they constitute social objects that represent concepts about, or descriptions of ground elements.

Every axiomatization in the physical branch of SOMA can, therefore, be regarded as expressing some physical context whereas axiomatizations in the descriptive social branch are used to express social contexts. Already some dedicated relations and design patterns are provided that connect both branches[3]. For example, as detailed in Section 3.1.3, the DUL relation *classifies* connects ground objects, e.g., a hammer, with the roles they can play. While this does not represent classification in the logical sense, the distinction between what an object is and what it can be used is apparently very suitable for the proposed SOMA ontology. Thus, it can be stated that a hammer can, in different contexts, be conceptualized as a murder weapon, a paper weight or a door stopper. Nevertheless, neither its ground ontological classification as a tool will change nor will hammers be subsumed as kinds of door stoppers, paper weights or weapons via multiple inheritance. Following a quick overview of the central modules of SOMA, detailed examples of where and how these commitments apply will be provided in Section 3.1.3 and Section 3.1.4.

[3] `http://ontologydesignpatterns.org` (accessed 22 May 2022)

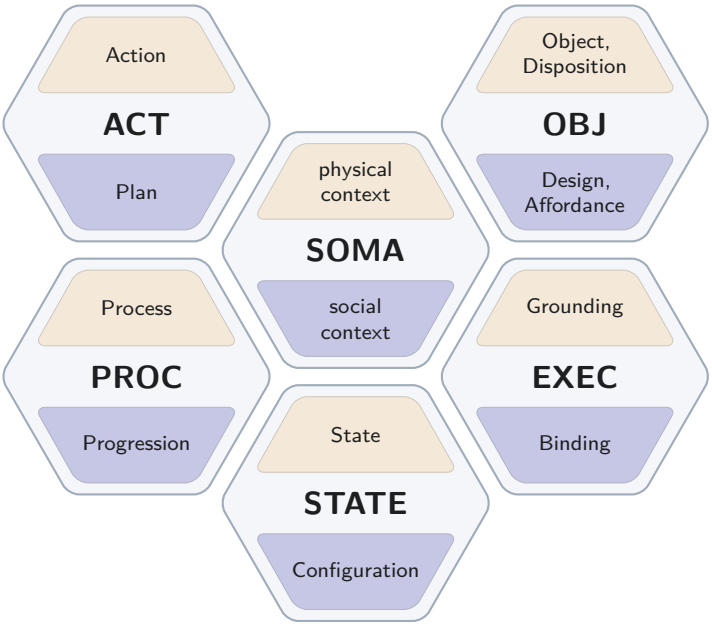

Figure 3.2: The modular organization of SOMA. Each module defines concepts and relationships used to represent physical (orange) and social (purple) activity context.

Module Overview

SOMA is organized in several modules that conceptualize different aspects of physical and social activity context (Figure 3.2). The different modules correspond to different event types (*ACT*, *PROC*, *STATE*), objects that participate in the activity (*OBJ*), and execution context (*EXEC*).

The scope of the *OBJ* module is the representation of physical objects, and their qualities. The module includes two taxonomies used to classify objects: an object taxonomy in the grounded branch, and a role taxonomy in the descriptive branch. It further includes a taxonomy of dispositions to represent the potential of using an object, and a taxonomy used to categorize objects based on their design. This will be described in more detail in Section 3.1.3.

The scope of the *ACT*, *PROC* and *STATE* modules is the contextualization of actions, processes and states. Actions are defined as events performed by an agent (physical context), and structured by a plan that is executed by the agent (social context). Plans may further impose constrains on steps in the plan, and objects that may play a role (*EXEC* module of SOMA). An action may cause processes to be started or stopped, and states to be changed. Processes, such as motions,

```
Class: LiquidContainmentDesign
SubClassOf:
  FunctionalDesign
  isDesignOf only (hasDisposition
    some (Containment
    and (affordsTrigger some
        (classifies only Liquid)))
```

```
Class: Cup
SubClassOf:
  DesignedContainer
  isDescribedBy some
    LiquidContainmentDesign
```

Figure 3.3: An example of how object classes are represented in *Manchester OWL Syntax*.

are defined as events considered in their evolution. The difference between a state and a process is that, when considering time slices of the event, for states, these time slices always have the same type as the state (states are homeomeric), but for processes this is not the case. SOMA defines a taxonomy of event types in the descriptive branch used to classify actions, and to further decompose them into motion phases, state changes, and physical interactions caused by them. This will be described in more detail in Section 3.1.4.

3.1.3 Object Representation

One of the reasons that everyday activity is a hard problem is the immense amount of variations an unrestricted environment may have, and the resulting potentials of interaction for an agent. Each type of object needs to be handled differently depending on its properties. However, object manipulation tasks are often defined independent of the type of object that is manipulated. It is thus crucial to employ an abstract object model, and a mechanism for applying abstract object knowledge to novel situations.

The main link between objects and actions in SOMA is that objects participate in events in the physical branch, and that the social branch represents the interpretation of their participation. Objects are further organized along their design. However, the agent might further need to find suitable candidate objects to perform a task by reasoning about which objects have the potential to be used in a certain way. SOMA employs an object disposition model for that purpose. An example object class is illustrated in Figure 3.3. In the example, a `Cup` is defined as a type of `DesignedContainer`, and thus inherits qualities through the class membership such as that it has a shape, and that it can be used as a container.

However, a cup is specifically designed to contain liquid substances which can be captured by the notion of `FunctionalDesign` in SOMA. The full axiomatization of the concept may contain several similar statements to specify other aspects of object qualities such as that cups afford containment for other object classes too, that they have a specific structural design, etc. Such a definition can be exploited to formulate reasoning queries such as which known objects could be used for a particular purpose, e.g., for storing water, or what the potential uses of an object are. An example of such a reasoning query is provided in Section 3.1.5.

Object Types

For the classification of objects, SOMA employs the `Role` pattern provided by the foundational layer. Roles are `Concepts` and, as such, reside in the `SocialObject` branch of DUL. For human agents the ascription of roles to entities comes very natural. *He is a student* does not imply an *isa* or *instanceof* relation between some male individual and a student class. It is rather meant that at this point of his or her life the individual plays the role of a student, which, however, can and will change over time.

This role pattern is of paramount importance, especially in the modeling of affordances discussed in Section 3.1.3 and Section 3.2. In the model presented herein, the roles are imported that have been established in the field of frame semantics (Baker et al., 1998). The selectional restrictions imposed by the *classifies* relation are used in a number of reasoning processes ranging from natural language understanding to tool selection. As certain roles can only classify physical agents or specific types of designed artifacts these axiomatizations provide substantial information about context dependent *meaning* of objects.

Object Designs

The organization of objects along a taxonomy is difficult as objects can be categorized in many ways. A notion of design is useful to capture object categories corresponding to structural, functional, or aesthetic patterns. Designs are in particular useful to conceptualize refunctionalized entities, and to support an agent to hypothesize unknown functions served by an entity. For example, a wooden pallet can be reused for the construction of furniture such as a sofa, or a

bed. The categorization of objects along their design can be employed in order to allow the use of more general plans, where, instead of object types, the plan refers to structure, aesthetics, or function.

Within the scope of SOMA, the `Design` concept belongs to the social branch. A design describes classes of objects that host a common design-relevant quality. These qualities are dispositional, geometrical, and aesthetic aspects of the object. This corresponds to SOMA's design categorization into functional, structural, and aesthetic design. Each `Design` concept defines restrictions on the corresponding quality type that needs to be fulfilled by any object described by the design. These restrictions can also represent sufficient conditions under which an object is thought to be described by the design which allows the classification of entities given their design pattern can be detected.

In the scope of this work, only functional aspects of objects are considered. These are represented through dispositional qualities which is discussed next.

Object Dispositions

Objects are important to an agent because they allow it to perform, or prevent it from performing, actions to achieve its goals. The notion of *affordance* was put forth by Gibson as *what it [the environment] offers the animal, what it provides or furnishes, either for good or ill* (Gibson, 1979, p. 127). However, though evidently useful as a way to organize actionable knowledge about the world (Yamanobe et al., 2018), affordances proved very difficult to model ontologically. Several approaches have been proposed, such as regarding affordances as qualities (Ortmann and Kuhn, 2010) or as events (Moralez, 2016). Nonetheless, these approaches are not entirely satisfactory. Affordances are relational, characterizing a potential interaction of several objects, and therefore should not be treated as either a quality belonging to an object, nor as an event. It is evident that some qualitative aspects of objects contribute to affordances, which is why the model proposed in this work is constructed around the interplay of Turvey's notion of *disposition* (Turvey, 1992b) and Gibson's notion of *affordance*.

In the SOMA ontology, the `Disposition` concept is defined as an object quality that allows an object to participate in events that *realize* an affordance. The `Affordance` concept itself, however, is defined as the relational context holding

between several objects that play different roles such as being the *bearer*, *trigger*, or *background* of an affordance. This modeling allows, via a mixture of DL and other reasoning mechanisms such as simulation, to answer several interesting questions such as what affordances might an object provide in some combination with others, what objects might, or probably would not, be able to provide a given affordance, what combinations of objects would work towards providing an affordance etc. More details about the disposition and affordance model in SOMA will be provided in Section 3.2.

3.1.4 Event Representation

The information gap between an instruction given to an embodied agent and the way it has to move its body to *successfully* execute the instruction is often immense. Consider, for example, a recipe for cooking noodles that contains an instruction to *boil water in a pot*. It is simple to decompose this instruction into several steps with individual sub-goals such as finding pot and tap, placing the pot underneath the tap, and filling the pot with water. However, the more difficult problem is how the agent has to move its body in each step such that the goal is achieved, and unwanted side-effects are avoided. Little variations in motion behavior may have drastic consequences in tasks that require delicate interaction. It is thus essential for agents performing actions in the physical world to reason about *how they should move* to achieve their goals in an appropriate, flexible and robust manner which is an unsolved problem for the general case.

SOMA attempts to support an agent facing this problem by equipping it with knowledge about relationships between abstract descriptions and their realization. The support is twofold. First, the agent may employ more general plans where informational gaps are filled by reasoning over knowledge represented with SOMA. Second, the agent may employ SOMA for understanding and generalizing observations. This means that agents can safer interact in environments with incomplete information, and that they can learn general patterns from specific situations.

An illustrative example of the representation of a pouring plan in OWL Manchester Syntax is provided in Figure 3.4. The plan is represented as an ABox ontology, i.e., as a collection of facts about the plan: what task it defines, and what steps it describes. Steps are conceptualizations of the events that realize them.

```
Individual: PouringPlan_0
  Types: Plan, Description
  Facts: defines Pouring_0,
         hasPhase Approaching_0,
         hasPhase Tilting_0
Individual: Pouring_0
  Types: Task, Concept
  Facts: usesRole Patient_0,
         usesRole Source_0,
         usesRole Destination_0,
         startedBy Approaching_0
Individual: Approaching_0
  Types: Motion Type, Concept
  Facts: usesRole Destination_1,
         overlapsWith Tilting_0
Individual: Tilting_0
  Types: Motion Type, Concept
  Facts: usesRole Patient_1
Individual: Binding_1
  Types: Role Binding,
         Description
  Facts: hasBinding Source_0,
         hasBinding Patient_1
```

Figure 3.4: An example of how plans are represented in *Manchester OWL Syntax*.

They specify the roles objects need to play during the event, and may further specify ordering constraints using Allen's relations (Allen, 1983) such as that realizations of the step *Pouring0* are *started by* realizations of the *Approaching0* step. Finally, resources may need to be shared among different steps within a plan, for example the source from which is poured (with role *Source0*) is the same entity as the patient during the tilting motion (with role *Patient0*) which is captured through a role binding in the plan definition.

The reason that plans are represented as ABox ontologies in SOMA is that identity constraints cannot be expressed as OWL DL axioms, i.e., distinct steps of a plan with the same type cannot be defined by different axioms. However, such sequencing information can be encoded in the ABox. This has the drawback that an OWL reasoner cannot recognize the plan that was executed by an agent. In general, the machinery necessary to perform, or recognize the execution of a task is outside the scope of OWL DL. Nonetheless, SOMA commits to encoding as many constraints on tasks as possible via OWL DL axioms.

In the following, the hierarchical organization of tasks, processes and states in SOMA will be discussed first. Second, it will be discussed how such events are decomposed into phases with explicit goals and individual knowledge preconditions. Finally, the modeling of force dynamical characteristics in the SOMA ontology will be discussed.

Event Types

One of the most important demands on a cognitive system is to reason about actions; colloquially speaking, an agent constantly asks itself what to do, and how to do it. This opens up another question, namely what exactly is the entity that the agent represents – an actual event, or an interpretation of one.

As an example, consider this scenario: a robot moves toward a table carrying a plate. Midway, its gripper releases, dropping the plate, which shatters against the floor. Perhaps the robot had to transport the plate to the table, and it failed to do so; or perhaps it was required to drop the plate as part of some material test, and the table was just there for some other reason. Just by observation of the action, without other interpretive context which includes knowledge of what the robot was told to do, there is no reliable way to tell. The failed transport interpretation does seem more likely a priori, but only because we have more often seen people tell robots to transport plates rather than break them; we still make use of an expected interpretive context.

As a result, SOMA does not define a taxonomy of action events, but rather of tasks that are used to conceptualize actions. For example, the `Grasping` concept is defined as task in SOMA, and it is used for the *classification* of events that are interpreted as an intentional grasping activity. This classification pattern between events and their conceptualization is provided by the foundational layer of SOMA. However, within the foundational layer, this pattern is only instantiated for actions and their conceptualization. In SOMA, motions of an agent and other processes, as well as state events are used to structure an activity. Thus, SOMA also needs to represent processes and states in the ground and the descriptive branch of the ontology. The same pattern applies: the concepts `Process` and `State` are defined in the ground ontology, and their conceptualization in the descriptive ontology, and a relationship between both branches is established through the aforementioned classification pattern.

Event Phases

Actions in SOMA are composed of distinct *phases*. Each phase has its individual goal, and requires a different movement strategy to be executed successfully. The

phases correspond to different stages of an object manipulation task, usually separated through *contact events*. Flanagan et al. (2006) have pointed out the importance of contact events in object manipulation tasks. The authors have shown that contact events cause a distinct pattern in sensory events, and that they can be used as *sensorimotor control points* for aligning and comparing predictions with actual sensory events. Another justification is that humans have shown to direct their gaze to contact points when they perform object manipulation tasks, or when they observe another agent performing a task.

The structure of activities in SOMA is governed by a set of design patterns. At its core, SOMA activity modeling builds on top of the *basic plan* ontology design pattern that represents plans and their execution. The pattern defines that an execution is a situation that *satisfies* the description of the plan. However, the pattern is defined too specific for the scope of this work, as the descriptive context for states and processes is also considered. Hence, a generalized version of this pattern is used in SOMA such that it can be instantiated for actions, states and processes. An `Action` is described in a `Plan` which is a description having an explicit goal. A plan satisfies situations that include action sequences that match the structure of the plan, such situations are called `Plan Executions`; a `State` is described in a `Configuration` which includes constraints on regions of entities and relationships between them. A configuration satisfies situations in which all constraints of the configuration are satisfied; and a `Process` is described in a `Process Flow` which is a description of the progression of the process. A process flow satisfies situations that include a process that progresses in the described way. Another aspect of activity structure can be captured by SOMA in, so called, *execution contexts*. These are representations of how different phases of an activity constrain each other depending on conditions encountered in the activity execution. In particular, the `Binding` concept can be defined as identity constrain representing that a parameter or role grounding is the same in different phases, however potentially being classified differently.

Ordering constraints are expressible in SOMA through a sequence pattern based on Allen's interval calculus (Allen, 1983). Allen's calculus defines thirteen relations between time intervals including *before*, *after*, *overlaps*, and *meets*. This is useful, on the one hand, to represent precedence of one phase strictly following

the other, and, on the other hand, it allows to cope with concurrency in the sequence. This algebra can be applied to event types that are defined within the descriptive context of a plan or process flow. However, reasoning about sequences is not well supported in OWL. Instead, interval relations can, e.g., be translated into a point graph to perform point-based reasoning (Zaidi and Wagenhals, 2006). Point graphs are directed acyclic graphs where nodes are the endpoints of intervals, and an edge is added for each axiom $a < b$ where a, b are interval endpoints. A non empty path from an endpoint a to another endpoint b further implies that $a < b$ through the transitivity of the relation. Event relations can be inferred through relations between their endpoints. For example, an interval i_1 *precedes* another interval i_2 if and only if $e_1 < s_2$ where s_2 is the starting point of i_2 and e_1 is the ending point of i_1. However, this only covers the *pointisable* subclass of the algebra which means that, e.g., disjunction axioms are not expressible.

Knowledge about the structure of activities can be employed by an agent in both directions: for planning an activity, and for interpreting observed events. Planning can be seen as a mapping from the descriptive to the grounded branch of SOMA, while interpretation maps the other way. For embodied agents, planning goes beyond mere decomposition of an activity into steps, the agent may further need to decide what objects it should use, how it should move, with what speed, and how much force it should apply when getting into contact with some object. SOMA can be employed, on the one hand, to find potential sequences of steps and motions to execute a task, to support finding potential objects playing some role during the activity, and to constrain the values of parameters of a task. Interpretation of observed events, on the other hand, is often possible through detection of contact events, types of motions, and states. These can be used as tokens for an activity parser that uses SOMA as a grammar, this will be described more in Section 3.1.5.

Knowledge Pre-Conditions. In order to execute a motion, an agent has to invoke one of its control routines with a set of arguments. Higher-level routines may have a notion of object, but at a lower-level all boils down to numbers such as with what effort the robot moves, how fast, etc. SOMA allows to define constraints for both cases: for the types of objects that can play a role during the action, and

for the value of parameters. This is done by using restrictions on what types of objects or regions can be classified by some role or parameter. This information is used to reduce the search space for performing an appropriate object or parameter selection (Section 3.1.3).

Goals. A goal is a description of a desired situation, and it is achieved only if the situational context, after the execution has been finished, satisfies this description. SOMA is more specific about what it means to execute an action successfully as it decomposes it into processes and states where the goal of the task is that the progression of processes evolves, and that state changes occur as described. Particularly important are the contact states in object manipulation tasks, as they represent control points for the agent when generating or observing behavior.

Event Force Characteristics

A contact state is an indicator for whether objects are touching each other or not. Patterns of such states are useful for distinguishing between categories of activities. However, different activities may cause the same pattern while their goal is different, or even the opposite of each other. This is, for example, the case for pulling and holding. Both cause the same pattern of an endeffector getting into contact with another object. But the force characteristics are different: the goal of a pulling task is to overcome the inertial force of the object to set it into motion, and the goal of a holding task is to neutralize any external force that would set the object into motion. Another aspect is that an agent performing such a task needs to decide how much force to apply. In order to make this decision it is valuable to know what the intended force-related consequences are.

SOMA supports the representation of force characteristics using Talmy's notion of force dynamics (Talmy, 2000). Talmy distinguishes between two entities that participate in force dynamical processes: the `Agonist`, and the `Antagonist`. An agonist is the subject of a force dynamical expression, while the antagonist is the opposing force in the expression. Each expression has an intrinsic force tendency either to set the agonist into motion, or to keep it resting. Whether the tendency can be realized or not depends on which of the two entities is the *stronger* entity.

3.1.5 Evaluation

SOMA was developed to provide robots with the capability to answer a set of competency questions about everyday activities. Thus, SOMA be validated by showing that these questions can be answered. Due to space limits, only selected examples will be elaborated here. The full range of competency questions is documented in the NEEM-Handbook (Beetz et al., 2020). Furthermore, the relevance of these competency questions can be demonstrated through applications of SOMA and their evaluation. A practical employment of SOMA is demonstrated in the *EASE Robot Household marathon* (Kazhoyan et al., 2021). Here, an overview of SOMA applications in prior work will be provided to verify its use in the application domain of autonomous robotics.

In version 1.1.0, SOMA contains 1330 logical axioms, 416 classes, 203 object properties, and 38 data properties. Its expressivity is $\mathcal{SROIQ}^{(D)}$. More metrics are listed on the SOMA webpage[4]. They are automatically computed when SOMA is deployed through a web service based on OntoMetrics[5].

Reasoning with SOMA

Being written in DL, SOMA can be processed with standard DL reasoners such as HermiT (Shearer et al., 2008). Because reasoning with the ontology is important during its use, the process of updating the ontology includes a reasoning step as well, also performed with HermiT, to verify that updates do not insert unsatisfiable concepts or empty properties. In more detail, every commit to the SOMA repository triggers subsumption and classification queries, and the discovery of concepts or properties equivalent to `Nothing` triggers a warning. This eases maintenance and scaling up of SOMA while keeping it consistent.

Next, it will be exemplified how the ontology can be reasoned with "at runtime", during some activity of a robot. Knowledge in SOMA covers, among others, aspects such as dispositions and affordances of objects. A question a robot might have is, what object in its environment could be used for a particular purpose, e.g., to contain some liquid. To this end, the robot will query the ontology by first defining a new "query" concept, formulated in Listing 3.1 and then ask

[4] `https://ease-crc.github.io/soma` (accessed 22 May 2022)
[5] `https://ontometrics.informatik.uni-rostock.de` (accessed 22 May 2022)

which of the objects it knows about can be proven to belong to this query concept via a subsumption query – objects that are individuals of subconcepts of the query concept can be used.

Sometimes, no known objects might be provably appropriate for a purpose. In such cases, one might try some other methods, such as testing in simulation, but such methods are themselves costly and so a filtering of candidates via reasoning is useful. In this example, the robot might ask, *what cannot be used to contain a liquid*. This is also achieved with the help of the "query" concept illustrated in Listing 3.1, but in a different manner. For a named concept `C` present in SOMA, that is a subconcept of `DesignedArtifact`, do a satisfiability query for the intersection of `C` with the query concept. If this intersection is provably empty, objects that are instances of `C` need not be tested for the affordance.

Listing 3.1: A query concept to find objects which can contain liquids

```
Class: WithAffordance_Containment_Liquid
EquivalentTo:
  DesignedArtifact and (hasDisposition some
    (Containment and (affordsTrigger some
    (classifies only Liquid))))
```

OWL DL was designed for the representation of encyclopedic knowledge, and has limited scope for domains such as dynamical characteristics. This concerns, for example, reasoning about the temporal ordering of steps that execute a task which can be handled through *point-based* reasoning where SOMA is enriched with definitions of temporal relations. Another example is *simulation-based* reasoning for affordance testing (Bateman et al., 2019; Pomarlan and Bateman, 2020). Thus, SOMA is used as a common model in a hybrid reasoning framework.

Applications of SOMA

The applicability of SOMA in the domain of autonomous robotics has been demonstrated in several scientific publications which will be briefly discussed in this section.

Grounding task parameters often requires predictive models which can be trained over instances of successful performance. Such experiential knowledge is in particular useful to learn context-dependent *plan specializations*. That is, how the parameters of the plan can be constrained within the scope of some

context to reduce the search space of parameter selection during plan execution. The learning problem is then defined with respect to a contextual pattern, and experiential samples are only considered when their contextualization matches this pattern. This capability was demonstrated in another work where a robot learns to execute a general fetch and place plan based on experience acquired through the execution of more constrained tasks (Koralewski et al., 2019).

Learning mechanisms often require large amounts of training data. One modality for acquisition is observation of other agents. It has been shown that an activity parser can be used to find possible interpretations for observed patterns of occurrences such as that objects get into contact with each other, or that the state of an object changed (Haidu and Beetz, 2019). The grammar used by the parser can be generated from a library of plans represented using SOMA. In prior work, more details about how the social context in SOMA can be grounded in data structures of a game engine has been provided (Haidu et al., 2018). The game engine implements an immersive virtual reality environment with photorealistic rendering and state of the art physics engine. Users perform object manipulation tasks while interactions, states, and motions are monitored, and used as tokens by the activity parser.

The employed modeling of tasks also helps to disambiguate vague natural language commands a robot may receive. SOMA allows to model how tasks relate to and depend on one another, and thus define execution contexts containing not just information about a task's parametrization, but also information about what other tasks it should enable. Such execution contexts are used to set up simulation scenarios in which to test task executions and thus select among several interpretations of a vague natural language command (Bateman et al., 2019).

3.1.6 Summary

In this section, SOMA was introduced, a novel activity ontology for robotic agents that combines several established ontology design patterns with models of human cognition. SOMA provides an ontological characterization of experience knowledge, and further supports robot decision making through formalized models of human cognition. One of these models will be described in the next chapter.

3.2 The Descriptive Affordance Ontology

Everyday activities, such as preparing meals, setting tables and cleaning up, take place in non-standardized environments and can vary greatly in their procedural execution. Enabling artificial robotic agents to perform such tasks under realistic conditions goes far beyond programming them to perform a specific action sequence in a given environment. Some consider the technological leap necessary to go from achieving a task to mastering an activity to be a pivotal challenge in cognitive robotics today (Beetz et al., 2018).

Anyone waking up in a household that has not been visited before and entering the unfamiliar kitchen in the morning with the intention to make some coffee, would theoretically have to solve a problem that features an infinitely large search space. Nevertheless, no one would divide the kitchen into a grid and start searching for the coffee grounds, let us say, at square *A1*. Human agents would start looking for suitable containers within cupboards that are in reach of the coffee maker first. Additionally, in the case that the coffee filters cannot be found, we would consider the use of suitable alternatives such as sieves or paper towels.

In an effort to describe the interaction between living beings and their environment Gibson coined the term *affordance* stating that:

> "The affordances of the environment are what it offers the animal, what it provides or furnishes, either for good or ill." (Gibson, 1979, p. 127)

Gibson readily acknowledged the problematic ontological character of this concepts by continuing:

> "... an affordance is neither an objective property nor a subjective property; or it is both if you like. An affordance cuts across the dichotomy of subjective-objective. [...] It is both physical and psychical, yet neither." (p. 129)

The central notion – the capability of living beings to interact with and change their environment are tightly linked to an ability to infer affordances – is undisputed. Additionally, this inference must be based on a coupling of our perceptions of the situation at hand to our knowledge of the world.

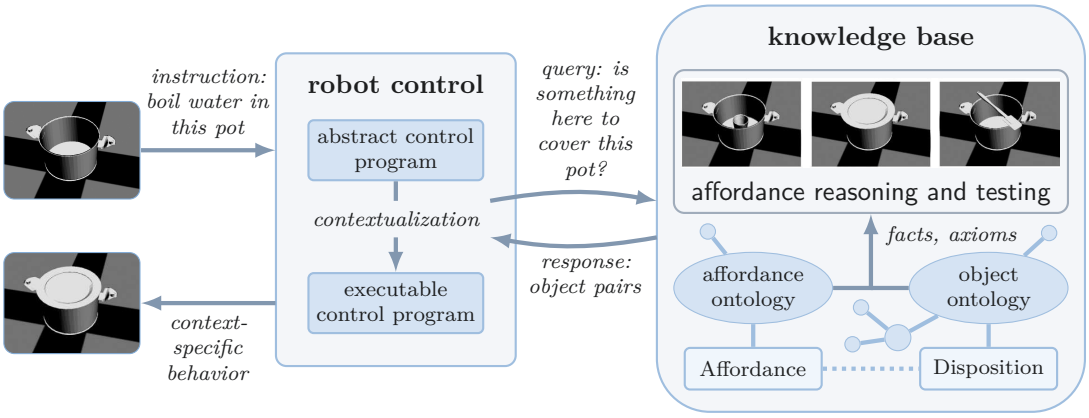

Figure 3.5: The schematic view of an affordance reasoning and testing system where the selection of object pairs is constrained through definitions in an affordance and object ontology.

The relevance of affordances for robotics has been recognized by several authors. An overview about this topic is provided by Yamanobe et al. (2018). The reason is that, once outside rigidly structured environments, an agent – biological or artificial – needs to adapt, and to some degree, improvise, or take advantage of action possibilities that are beneficial to its goals. This needs a theory of affordances, and a reformulation of current object-centric reasoning and planning around these concepts. Without affordance-based reasoning, a robot is, essentially, stuck into thinking about objects as in, *is there a lid around here?* With affordances, object uses and roles become more important, thus opening up a new avenue for more abstract thinking as in, *is there something here that can cover a container*, or *is there something here that can store liquids?* Such an example is displayed in Figure 3.5.

Affordances are also important when taking the perspectives of other agents. Household service robots, for example, are often required to arrange an environment – e.g., by setting a table – such that the future actions of human users are enabled. Whether an arrangement is good or not depends on whether it provides the needed affordances for the future action.

In the following, an overview of prior modeling approaches will be given to motivate the congruencies and divergences of this work. Thereafter, the *Descriptive Affordance Ontology* will be presented, and how it contributes to reasoning about finding and combining appropriate objects and discovering possible dispositions at hand. Lastly, some practical applications of the ontology will be presented.

3.2.1 Related Work

Modeling affordances has been a challenging problem for numerous years. As the ontology proposed here seeks to build on the DUL foundational framework (Masolo et al., 2003), prior approaches will be highlighted as they pertain to the DOLCE modeling paradigm. Other work concerning, for example, the learning of relational affordance models for robots (Moldovan et al., 2012), is independent from the respective modeling approach taken. Any of the following approaches could be populated with the learned affordances.

One attempt to model affordances within the DOLCE framework is as `Qualities` of a `Physical Artifact` as proposed by Ortmann and Kuhn (2010). This modeling approach assumes an affordance to be inherent in a given object (the afforder) and to exist independent of the other entities (the affordees) involved. However, if one models affordances as qualities, questions such as *what can something be used with so as to manifest an affordance* are out of scope; the answer requires functional relations between objects.

An alternative approach is to model affordances as `Events`, as proposed by Moralez (2016), based on the idea that an affordance happens when the right objects participate in it. The problems that arise with this approach are, besides going against the fundamental construal of events, it also confuses affordances with the act of perceiving them. Agents do not just perceive, but also conceptualize affordances by exploiting similarities underlying classes of situations regardless of whether these situations have been perceived or are imagined.

Another approach is to model functional affordances (Awaad et al., 2013a,b, 2014), which describe the typical affordances that an object provides, often because it is designed with particular uses in mind. The cited research highlights situations where substitutions of objects or actions are necessary, and describes how the substitution is guided by knowledge of functional affordances, conceptual similarity of objects or actions from a taxonomy, and a preference order on substitutions. The difference to this work is that a functional affordance is, essentially, a property of an object; a cup is *for holding water*, and *for drinking from*. In this work, an affordance is rather seen as a descriptive context between several entities.

The model proposed in the following section seeks to remedy the shortcomings of the aforementioned approaches and is based on the dispositional theory of

Turvey (Turvey, 1992b), in which a disposition of an object (the bearer) can be realized when it meets another suitably disposed object (the trigger) in the right conditions (the background). Toyoshima and Barton (2018) consequently consider affordances as dispositional qualities inherited in objects.

However, as Turvey's approach bases on particular kinds of dispositions inherited by the environment, it is important to point out that notions of affordances as a function of environmental properties have been criticized several times (Stoffregen, 2003; Chemero, 2003) as the agentive aspect is ignored – i.e., what is available to the agent. Chemero (2003), for example, defines affordances as relations between abilities of organisms and features of the environment. This definition has also been extended with a notion of interaction based on the view proposed by Norman, who puts the visibility of an object's affordance at the center (Norman, 2002). While this view can be regarded as appropriate for the perspective of human-computer interaction and design, the dispute with Gibson's view that an affordance is there whether an agent can perceive it or not has not been resolved (Norman, 1999). In the model proposed herein, an environmentally facilitated affordance exists when it can be conceptualized by an agent as such.

3.2.2 Formalization and Implementation

The dispositional theory of Turvey states that a disposition is *the property of a thing that is a potential* (Turvey, 1992b). This work is only concerned with dispositions that enable robots to perform tasks such that it is possible to be more specific regarding the *potential* of dispositions.

Definition 1. *A disposition is a property of an object that can enable an agent to perform a certain task.*

It is important to note that a task is seen as a conceptualization of an event, abstracting away from particularities of its occurrence by only referring to roles objects need to take during the task. For example, the actual event of an object leaving a hand and landing on the floor can be construed as accidental dropping or intentional throwing of that object. Fundamentally, dispositions are absolute properties, and therefore not contingent on a given context. Consequently, a disposition is seen as a quality of the environment that is implied by the existence

of the object that carries it. Regardless of the exploitation of an object disposition in a given task, the disposition is there as a quality of that object.

Here, affordances are viewed as descriptions of what objects in the environment offer to the agent, and that conceptualize tasks afforded by them. Hence affordances exist independently of the ability to manifest them, they are inherited from what the environment offers. However, an affordance is not a *property* of the environment but rather describes how an agent may make use of some property of the environment by executing the task that is defined by the affordance.

Definition 2. *An affordance is the description of a disposition.*[6]

Each disposition is described by an affordance that defines the task afforded by the disposition. An apple, for example, has a disposition that affords us to eat it. The task defined by this affordance includes, for example, the role *edible*, and another role *consumer*. A further distinction can be made between the role of the carrier of the disposition (the bearer), and the role of the object that is afforded by the disposition (the trigger), assuming that any disposition is described by exactly one affordance that defines both roles.

The manifestation of an affordance is a situation that satisfies the affordance, meaning that an action was performed that executes the afforded task with appropriate objects taking roles during that action. However, the afforded task may need to be decomposed into several sub-tasks, or executed in different ways depending on, for example, the ability of the agent, or availability of objects. Hence, the way an afforded task is to be executed by an agent is not implied by the affordance, so the task execution can be described by several alternative plans that define the afforded task in different ways.

Dispositions can be seen as the objective analog to capabilities for agentive entities. Therefore, objects can be the bearer of dispositions while agents can have capabilities. Consequently, the `Capability` concept can also be viewed as a type of quality inherited by the agent that carries it. However, the `Capability` concept is not considered in the scope of this work. The proposed model extends previous

[6]Please note that this definition is open to manifestations of affordances that take place without agentive entities involved, e.g, due to a serendipitous event involving inanimate objects, but as this bears no relevance to the considered domain of everyday activities, the focus in this work lies on those affordances that enable agents to perform tasks.

ones that also take Turvey as a starting point (Stoffregen, 2003; Chemero, 2003; Toyoshima and Barton, 2018) by situating dispositions in the so-called *ground ontology* and connecting them to affordances that are part of the *descriptive* branch of the ontology (Masolo et al., 2003). As will be shown below, this decoupling gives the model a level of flexibility analogous to the flexibility gained by the decoupling of actions from tasks.

Affordance Concept

Let us first formalize the `Disposition` concept. Fundamentally, it is a property of the object that is the bearer of the disposition. This relationship between bearer and disposition is denoted by the relation symbol *hasDisposition* (HAS_D). For the sake of saving space, the affordance concept is referred to by the letter `A`, and the disposition concept by the letter `D`.

$$\texttt{D}(x) \rightarrow \texttt{Quality}(x) \tag{3.1}$$

$$\texttt{D}(x) \rightarrow \forall y(HAS_D(y,x) \rightarrow \texttt{Object}(y)) \tag{3.2}$$

$$\texttt{D}(x) \rightarrow \exists! y(HAS_D(y,x)) \tag{3.3}$$

$$\texttt{D}(x) \rightarrow \exists! y(DESCR(y,x) \wedge \texttt{A}(y)) \tag{3.4}$$

$$DESCR(x,y) \rightarrow \texttt{Description}(x) \tag{3.5}$$

Dispositions are modeled as qualities (3.1) carried only by exactly one object (3.2,3.3), and described by exactly one affordance (3.4). The *describes* relation ($DESCR$) holds between a description (e.g., an affordance) and entities that are conceptualized by the description (3.5).

Affordances define task and roles afforded by the disposition they describe. Roles afforded to the bearer of the disposition can be distinguished from the ones afforded to the trigger. Consequently, the relations *definesBearer* (DEF_B) and *definesTrigger* (DEF_T) are used to link an affordance to the respective role.

$$DEF(x,y) \rightarrow \texttt{Description}(x) \wedge \texttt{Concept}(y) \tag{3.6}$$

$$DEF_B(x,y) \rightarrow DEF(x,y) \wedge \texttt{A}(x) \wedge \texttt{Role}(y) \tag{3.7}$$

$$DEF_T(x,y) \rightarrow DEF(x,y) \wedge \texttt{A}(x) \wedge \texttt{Role}(y) \tag{3.8}$$

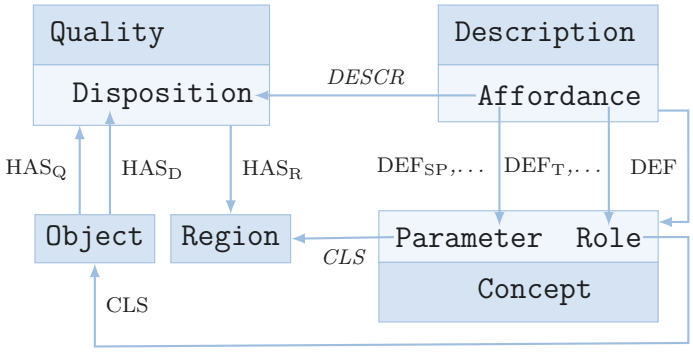

Figure 3.6: Relationships between affordances and dispositions in the proposed theory.

The *defines* relation (DEF) holds between a description and a conceptualization (3.6), both *definesBearer* and *definesTrigger* are subproperties of this relation (3.7 and 3.8). The *definesTask* relation (DEF_{Tsk}) is formalized analogously.

Finally, the `Affordance` concept can be formalized by axiomatizing its relationship to dispositions described, and concepts defined by it.

$$\mathtt{A}(x) \rightarrow \mathtt{Description}(x) \tag{3.9}$$

$$\mathtt{A}(x) \rightarrow \forall y(DESCR(x,y) \rightarrow \mathtt{D}(y)) \tag{3.10}$$

$$\mathtt{A}(x) \rightarrow \exists!y(DEF_{Tsk}(x,y) \wedge \mathtt{Task}(y)) \tag{3.11}$$

$$\mathtt{A}(x) \rightarrow \exists!y(DEF_{B}(x,y) \wedge \mathtt{Role}(y)) \tag{3.12}$$

$$\mathtt{A}(x) \rightarrow \exists!y(DEF_{T}(x,y) \wedge \mathtt{Role}(y)) \tag{3.13}$$

$$\mathtt{A}(x) \rightarrow \forall c,y(DEF_{B}(x,c) \wedge CLS(c,y) \rightarrow \exists z(DESCR(x,z) \wedge HAS_{D}(y,z))) \tag{3.14}$$

Affordances are seen as descriptions (3.9) that only describe dispositions (3.10). An affordance defines exactly one task (3.11), and two roles for bearer and trigger of the disposition (3.12,3.13). Axiom 3.14 is an identity constraint that restricts the bearer role to the objects that carry a disposition that is described by the affordance, meaning that the role may only *classify* (CLS) these objects. These relationships are depicted in Figure 3.6.

A disposition further *affords* the task and roles defined by the affordance that describes the disposition. Different specifications of this relation are useful when particular disposition types are axiomatized. Consequently, a relation *affords*

(AFF) can be introduced, and sub-relations *affordsBearer* (AFF_B), *affordsTrigger* (AFF_T), and *affordsTask* (AFF_{Tsk}).

$$AFF(x,y) \rightarrow D(x) \wedge \texttt{Concept}(y) \tag{3.15}$$
$$AFF(x,y) \rightarrow \exists a(DESCR(a,x) \wedge DEF(a,y)) \tag{3.16}$$
$$AFF_B(x,y) \rightarrow AFF(x,y) \wedge \texttt{Role}(y) \tag{3.17}$$
$$AFF_B(x,y) \rightarrow \exists a(DESCR(a,x) \wedge DEF_B(a,y)) \tag{3.18}$$

The *affords* relation links dispositions and concepts (3.15). Concepts afforded by a disposition are defined in an affordance that describes the disposition (3.16). Specifications of this relation further constrain the type of the afforded concept (3.17), and how the concept is related to the affordance that defines it (3.18). The formalization of *affordsTrigger* and *affordsTask* is done analogously.

The manifestation of an affordance ($MAFF$) can be seen as a situation (SIT) that *satisfies* (SAT) the affordance describing the dispositions that are included in the situation (3.19). More concretely, it is a situation where an agent executes the task defined by the dispositions by following a plan (which is a description) involving objects playing certain roles and regions setting specific parameters for that execution. Hence, situations in which affordances are manifested also satisfy the plan that the agent executes (3.20).

$$MAFF(x) \rightarrow \exists! y(SAT(x,y) \rightarrow A(x)) \tag{3.19}$$
$$MAFF(x) \rightarrow \exists! y(SAT(x,y) \rightarrow P(x)) \tag{3.20}$$

The formal plan is referred to as P above. A formalization of plans that describe tasks afforded by dispositions is out of scope of this section, as well as a full formalization of affordance manifestations. Above axioms only serve the purpose to provide a starting point for investigating this concept at a later point.

Disposition Hierarchy

Dispositions can be classified according to different roles for bearer and trigger. Axiom 3.12 and 3.13 ensure that there is exactly one role for them such that disposition types may use an universal quantification axiom to constrain the roles.

One example is the `Blockage` disposition (BLK). It can be seen as the disposition to prevent others from accessing, leaving, or seeing a restricted space or group. It further affords the bearer role `Barrier` and the trigger role `Blocked`.

$$\texttt{BLK}(x) \rightarrow \texttt{D}(x) \tag{3.21}$$

$$\texttt{BLK}(x) \rightarrow \forall y(AFF_B(x,y) \rightarrow \texttt{Barrier}(y)) \tag{3.22}$$

$$\texttt{BLK}(x) \rightarrow \forall y(AFF_T(x,y) \rightarrow \texttt{Blocked}(y)) \tag{3.23}$$

Hence, *blockage* is a disposition (3.21) that only affords the *barrier* role for the bearer of the disposition (3.22), and only the *blocked* role for the trigger (3.23).

Another class of dispositions refers to the potential to modify aspects of others. Such objects, that afford us to change others, inherit the `Alteration` (`ALT`) disposition. Alteration is primarily inherited by designed tools and devices such as a dishwashers that afford to clean cutlery, or freezers that afford to regulate the temperature of objects. However, a counterpart of the `Alteration` disposition may be inherited by objects that tend to undergo certain modifications. This is, for example, the dirty cutlery after dinner that affords us to clean it, or the melting ice cream that affords us to put it back into the freezer. The main difference is that these counterpart dispositions afford different roles for the bearer and the trigger of the disposition, namely that the bearer of one disposition needs to take the role of the trigger in the other.

Tasks defined by the affordance describing an disposition include an additional parameter, the setpoint (`SP`), that qualifies the targeted change, for example, that dirty cutlery affords to change its `Cleanliness` quality such that the cutlery is not qualified as *dirty* anymore. The actual quantification of the targeted change may differ from one property type to another such that it cannot be axiomatized on the general level. However, sub-classes of the `Alteration` disposition can be classified according to the type of quality that is afforded to be altered, and axiomatized with a more concrete notion of what values the property may take.

$$\texttt{SP}(x) \rightarrow \texttt{Parameter}(x) \tag{3.24}$$

$$\texttt{SP}(x) \rightarrow \forall y(CLS(x,y) \rightarrow \texttt{Region}(y)) \tag{3.25}$$

$$\texttt{SP}(x) \rightarrow \exists y(DEF(y,x) \rightarrow \texttt{A}(y)) \tag{3.26}$$

Hence, setpoints are used to classify regions that quantify the targeted change (3.25), and there exists an affordance that defines each setpoint (3.26).

Based on the *setpoint* notion, the relations DEF_{SP} and AFF_{SP} can be defined analogously to the other variants of the *DEF* and *AFF* relations, and assert that alterations also afford a setpoint parameter:

$$\texttt{ALT}(x) \quad \rightarrow \quad \exists! y(AFF_{SP}(x, y)) \tag{3.27}$$

The other two types of dispositions considered in the proposed model are `Connectivity` and `Containment`. The `Connectivity` disposition of an object affords to connect others with it, such as a hook that affords to hang up objects. `Linkage` is a more specific type of connectivity that implies a stronger connection between bearer and trigger such that they resist spatial separation to some extend. Another variant of connectivity is the `Support` disposition that affords, for example, stabilizing a posture, or controlling an object. The `Containment` disposition affords the `Container` role for the bearer of the disposition, however, without implying a portal that can be used to insert items into the container. `Insertion` is a variant of containment that, in addition, affords a `Portal` role.

As the goal of this work is to introduce a modeling approach and corresponding ontology pattern for affordances, some concrete examples from the domain are employed. Given the multitude and diversity of tasks executed by human and artificial agents, a fully fleshed out model can be expected to include a small set of high-level dispositions that feature increasingly specific sub-dispositions. However, as with the discussion on the number and nature of image schema (Johnson, 1987), that are influenced by our perception and interactions with the physical world, an exhaustive model is likely to be attainable, and moreover learnable (Koppula et al., 2013), given a suitable target representation as the one proposed here.

Inherited Dispositions

The dispositions of an object are usually inherited from the class the object belongs to. The apple that affords us to eat it, for example, does that because the individual apple belongs to a category of objects that all inherit this disposition, for example, the category of comestible objects. Note that, e.g., the `Perishable`

quality of objects that afford us to eat them could be restricted to values that indicate that the food is not spoiled given this quality can be detected.

For example, let us consider a dirty piece of cutlery in the kitchen (denoted DC). With dirty it is meant that the object has a quality Cleanliness that takes some value from a region Dirty whose members only quantify dirty objects.

$$\texttt{DC}(x) \iff \exists q(HAS_Q(x,q) \wedge \texttt{Cleanliness}(q)) \wedge \exists r(HAS_R(q,r) \wedge \texttt{Dirty}(r)) \tag{3.28}$$

$$\texttt{DC}(x) \rightarrow \exists! y(HAS_D(x,y) \wedge \texttt{C}_0(y)) \tag{3.29}$$

Axiom 3.29 states that dirty cutlery affords to clean it. HAS_Q refers to the *hasQuality* relation between objects and their qualities, and HAS_R (*hasRegion*) is a relation between qualities and their value.

Next, it can be formalized that *demand to be cleaned* means to alter the cleanliness quality of the demanding object to some value from a region Clean whose members only quantify clean objects, and that the disposition of the dirty cutlery to be cleaned (denoted as $\texttt{C}_0$) affords this task where the bearer of the disposition takes the role of the Cleaned object.

$$\texttt{C}_0(x) \rightarrow \forall y(AFF_{Tsk}(x,y) \rightarrow \texttt{CT}(y)) \tag{3.30}$$

$$\texttt{C}_0(x) \rightarrow \forall y(AFF_T(x,y) \rightarrow \texttt{Cleaner}(y)) \tag{3.31}$$

$$\texttt{C}_0(x) \rightarrow \forall y(AFF_B(x,y) \rightarrow \texttt{Cleaned}(y)) \tag{3.32}$$

$$\texttt{C}_0(x) \rightarrow \forall y,z(AFF_B(x,z) \wedge CLS(z,y) \rightarrow \texttt{DC}(y)) \tag{3.33}$$

$$\texttt{C}_0(x) \rightarrow \forall y,z(AFF_{SP}(x,z) \wedge CLS(z,y) \rightarrow \texttt{Clean}(y)) \tag{3.34}$$

The task afforded by the disposition is referred to as CT (short for *cleaning task*). Analogously, it can be stated that a dishwasher (denoted DW) has the disposition to clean dirty cutlery.

$$\texttt{DW}(x) \rightarrow \exists! y(HAS_D(x,y) \wedge \texttt{C}_1(y)) \tag{3.35}$$

The disposition of the dishwasher to clean cutlery (denoted $\texttt{C}_1$) can be seen as the counterpart of the disposition $\texttt{C}_0$ of the cutlery to be cleaned. Meaning that both

dispositions afford the same task, and that the bearer and the trigger roles are reversed. However, the disposition of the dishwasher is more restrictive as it also constrains the trigger object of the disposition to be *dirty cutlery*. The disposition of the cutlery, on the other hand, does not constrain the class of the trigger object used to clean it.

$$\mathtt{C}_1(x) \rightarrow \forall y(AFF_B(x,y) \rightarrow \mathtt{CT}(y)) \tag{3.36}$$

$$\mathtt{C}_1(x) \rightarrow \forall y(AFF_B(x,y) \rightarrow \mathtt{Cleaner}(y)) \tag{3.37}$$

$$\mathtt{C}_1(x) \rightarrow \forall y(AFF_T(x,y) \rightarrow \mathtt{Cleaned}(y)) \tag{3.38}$$

$$\mathtt{C}_1(x) \rightarrow \forall y,z(AFF_B(x,z) \wedge CLS(z,y) \rightarrow \mathtt{DC}(y)) \tag{3.39}$$

$$\mathtt{C}_1(x) \rightarrow \forall y,z(AFF_T(x,z) \wedge CLS(z,y) \rightarrow \mathtt{DW}(y)) \tag{3.40}$$

$$\mathtt{C}_1(x) \rightarrow \forall y,z(AFF_{SP}(x,z) \wedge CLS(z,y) \rightarrow \mathtt{Clean}(y)) \tag{3.41}$$

Implementation

For practical use of the proposed theory, a reference implementation is provided as OWL2 DL ontology. In the process of creating this ontology, the axioms listed in this section were converted to DL. More specifically, to the $\mathcal{SROIQ}^{(D)}$ fragment of DL. Each of the axioms is represented in the OWL ontology except of the identity constraint (Axiom 3.14) which is not expressible in OWL2 because co-reference of an entity with different roles cannot be expressed. This means that the OWL ontology does not enforce that only the bearer of a disposition may take the bearer role defined in the affordance.

Let us consider again the example of a dishwasher affording the task to clean dirty cutlery. One aspect of it is that the dishwasher affords the `Alteration` of the cleanliness quality of cutlery. Another aspect is that the cutlery can be inserted into it which is a disposition with type `Insertion`. This can be written in *Manchester OWL Syntax* as shown in Figure 3.7.

This is not a complete list of all dispositions a dishwasher has, however, it serves the purpose to give an intuition about how the dispositions of objects are formalized in OWL using the proposed affordance theory.

The reference implementation of the *Descriptive Affordance Ontology* is publicly accessible as part of the SOMA framework (Section 3.1). It can be obtained

```
Class: Dishwasher
SubClassOf:
  hasDisposition exactly 1 Insertion
  hasDisposition exactly 1 (Insertion
    and (affordsTrigger only
        (classifies only DirtyCutlery)))
  hasDisposition exactly 1 Alteration
  hasDisposition exactly 1 (Alteration
    and (affordsTrigger only
        (classifies only DirtyCutlery))
    and (affordsSetpoint only (Clean
        and (isRegionFor only Cleanliness))))
```

Figure 3.7: An example of how dispositions are represented in *Manchester OWL Syntax*.

through the website of the framework[7]. The overarching goal of the SOMA framework is to enhance robot decision making capabilities, and therefore to make the plans robots execute more general and re-usable. This is achieved by providing a *tell* and *ask* interface to interact with the knowledge content of the ontologies which is realized by the KnowRob knowledge base (Tenorth and Beetz, 2009).

3.2.3 Evaluation

In the following section, the ensuing flexibility that the proposed model facilitates in the domain of cognitive robotics for everyday activities (Beetz et al., 2018) will be showcased. Goal of this larger research undertaking is to depart from single task robotics, where a robot has to be trained to cover an object provided a designated pot and lid are at hand, to a flexible mastery of a basic activity, such as covering. For this, first of all, a flexible means for dispositional matching is needed for a given dispositionally qualified object, such as a container. We seek for dispositional counterparts and respective objects that can play the corresponding roles, for example, to serve as a cover. This type of flexible commonsense reasoning will provide potential triggers for specific bearers and *vice versa*, but will not guarantee that a specific instance of that object type will actually work in creating the needed affordance. Therefore, in a second step, it will be tested via simulation how the potential afforders and affordees work together.

[7] `https://ease-crc.github.io/soma` (accessed 22 May 2022)

Dispositional Matching

One of the most essential aspects covered by the proposed model is that dispositions have a counterpart, another disposition whose carrier is *compatible* with the disposition offered by some other object. Objects are seen as having a *dispositional match* in such a case. The match is not dependent on agentive aspects, it is only derived from properties of the objects carrying the dispositions. Hence, the existence of a match is not equivalent to say that some agent can actually use both objects with each other – this further depends on the agents' capability to execute the task afforded by the dispositions.

Both dispositions contributing in a match afford their own individual task (i.e., an individual that is an instance of the `Task` concept). One aspect of a dispositional match is that both individual tasks are instances of the same task concept. The other aspect is that any object contributing in a match must be compatible to the trigger role afforded by the disposition of the other object. A match exists in case these axioms can be added to the knowledge base without making it inconsistent. This can be written as displayed in Algorithm 3.1.

The match between tasks is axiomatized as $tsk_1 \doteq tsk_2$, meaning that tsk_1 and tsk_2 are the same individual. This is inconsistent, for example, in case tsk_1 and tsk_2 are individuals of *disjoint* classes. Hence, it is assumed that a task taxonomy that includes disjointness axioms between tasks sharing a direct superclass. However, the axiom $tsk_1 \doteq tsk_2$ still holds in case tsk_1 is an instance of a class that is a direct relative of the class instantiated by tsk_2.

The other two axioms are used to test whether the object carrying the counterpart disposition is a valid assignment for the *CLS* role that links the trigger role to the objects that are classified by this role. This would be inconsistent, for example, in case the bearer of the disposition, the disposition itself, or the affordance describing the disposition constrains the type of object that may take the trigger role to one disjoint from the type of the potential trigger object.

It is further assumed that the bearer role is a valid role to take for the bearer of the disposition. The identity axiom (3.14) would contradict an invalid assertion. However, the OWL implementation does not contain this axiom such that this relation must be enforced elsewhere when this implementation is used.

Algorithm 3.1: Dispositional Matching

Input : Two disposition individuals d_1 and d_2, and a consistent ontology $\mathcal{O} = (\mathcal{A}, \mathcal{T})$ where $\mathcal{A}$ is an ABox, and $\mathcal{T}$ is a TBox.

Output: *True* in case there is a dispositional match between d_1 and d_2, otherwise *false* is returned.

/* Retrieve functional properties of d_1 and d_2 from $\mathcal{A}$. */

$o_1 \leftarrow isDispositionOf(d_1)$ /* $isDispositionOf \equiv hasDisposition^-$ */

$o_2 \leftarrow isDispositionOf(d_2)$

$tsk_1 \leftarrow affordsTask(d_1)$

$tsk_2 \leftarrow affordsTask(d_2)$

$tri_1 \leftarrow affordsTrigger(d_1)$

$tri_2 \leftarrow affordsTrigger(d_2)$

/* The axiomatization of a dispositional match. */

$\mathcal{O}_1 \leftarrow (\mathcal{A} \cap \{tsk_1 \doteq tsk_2, CLS(tri_1, o_2), CLS(tri_2, o_1)\}, \mathcal{T})$

/* Is $\mathcal{O}_1$ inconsistent? */

if $\mathcal{O}_1 \models \top \sqsubseteq \bot$ **then**

  return *false*

else

  return *true*

end

Competency Questions

The goal of a reasoning system is to infer useful answers to questions an agent faces. In the case of the propose affordance and dispositions model, these are questions related to item usage and action potentialities.

The general procedure to answer such questions is to create an ontology importing the theory of dispositions and affordances, and an ontology of objects, and add axioms to define concepts relevant for a particular query. These axioms can be generated via Generic Ontology Design Patterns (Krieg-Brückner et al., 2019) written in generic DOL (Mossakowski, 2016). For example, the query concept in Listing 3.3 is instantiated from the pattern in Listing 3.2.

Listing 3.2: A pattern in Generic DOL to create a query concept for affordance testing

```
pattern AffordanceQuery[Class: D; Class: T] given EASE =
Class: WithAffordance[D,T]
EquivalentTo:
  DesignedArtifact and hasDisposition some (D and
    (affordsTrigger some (classifies only T)))
```

What can be used for a particular purpose? The main question is, given an affordance, what is the combination of objects needed for it to manifest? This is important for an autonomous robot for several reasons. First, the robot might need to formulate intermediate goals and plan sub-tasks to achieve them, and to have flexibility it must not be limited by blind, hard-coded object choice; rather, it should be able to select appropriate objects for the goals it sets for itself. Second, commands given by a human, or instruction steps meant to teach how to do something, are often ambiguous about tools to use – because such commands or instructions were typically meant for other humans, which have the cognitive machinery to go from a requested action to the items needed to manifest it.

To answer such queries, one would add a concept for classes of objects that can be, for example, the bearer of a disposition to manifest an affordance. A DL reasoner can then be queried for subsumption.

Let us consider the following example: we are looking for an object that is the bearer of a `Tempering` disposition – that is, it can subject some `Substance` to a set temperature.

Listing 3.3: OWL expansion of the Generic DOL instantiation AffordanceQuery[Tempering;Substance], yielding a named concept to query for object classes that can be the bearer of a disposition to manifest an affordance

```
Class: WithAffordance_Tempering_Substance
EquivalentTo:
  DesignedArtifact and hasDisposition some (Tempering and
    (affordsTrigger some (classifies only Substance)))
```

Running a DL subsumption query on the objects and affordances ontology, augmented with the definition of the concept listed above, produces that `Refrigerator` and `HotPlate` are sub-concepts, i.e., they can be used to temper substances.

What can this be used for? A second competency question an agent such as an autonomous robot must answer is, given an item, what can it be used for. This is an important step for a robot to understand an environment in terms of the affordances it actually provides, which in turn is useful when it has to answer how to manifest a particular affordance, or to verify that the environment has been set up in such a way that affordances are available to another agent. Specifically, this question addresses the issue of identifying dispositions of items in the surroundings of the robot that could be exploitable by the robot in the

course of an activity.

Semantic maps of environments are often populated with objects and annotations about them, including dispositions. If this is available, then querying for the known dispositions of an object is simply a lookup in a semantic map. However, the information in the semantic map may be incomplete, such as when seeing a new item, in which case one must reason based on TBox axioms.

Let us consider the following example: the robot is looking at an item it recognizes as a dishwasher, and asks itself what dispositions it might have. As before, the general approach is to define a concept to add to the ontology for a reasoning query, but there is a complication. The fact that a disposition is not mentioned in the definition of an object concept does not mean some individual object will not have the disposition. Instead, the query concept is about what disposition an object must have to be a member of its class.

Listing 3.4: An example named concept to query for disposition classes that are borne by an object

```
Class: WithoutDisposition_Dishwasher_Insertion
EquivalentTo:
  Dishwasher and (hasDisposition only (not Insertion))
```

Running a DL subsumption query on the dispositions and objects ontology, augmented with the definition of the concept listed above, produces that this concept is subsumed by `Nothing`, therefore every individual dishwasher must allow some kind of items to be inserted in it. A similar query with a concept `WithoutDisposition(Dishwasher,Movable)` does not return that the query concept is empty, meaning there may be individual dishwashers that cannot or should not be moved.

What can this be used with? This competency question tackles the fact that, typically, the manifestation of an affordance needs appropriate combinations of objects – that is, objects with sufficiently matched dispositions. In some sense, a cookie cutter is an object allowing the shaping of (some) other objects – but it cannot be used to shape a block of wood. This is the other step towards building an affordance-aware understanding of the robot's environment.

Consider the following example: we want to test whether a particular item can be tempered by the `Refrigerator`.

Listing 3.5: An example named concept to query whether an object class can be the trigger for a disposition borne by another object class

```
Class: IsTrigger_PancakeMix_Tempering_Refrigerator
EquivalentTo:
  PancakeMix and (isClassifiedBy some
    (isTriggerAffordedBy some
      (Tempering and (isDispositionOf some Refrigerator))))
```

Running a DL subsumption query on the dispositions and objects ontology, augmented with the definition of the concept listed above, proves this concept is not subsumed by `Nothing`, therefore the `PancakeMix` can be tempered by the `Refrigerator`. A similar test run with `Stove` as the item to try and temper produces that `IsTrigger(Stove,Tempering,Refrigerator)` is a subconcept of `Nothing`; this is because the `Stove` is not a `Substance` or `FoodItem`, which are the only classes in the onbtology, a `Refrigerator` can temper.

What cannot be used to manifest a particular affordance? This, and likewise negative versions of the other previous competency questions, are important to consider because the knowledge of the world is likely incomplete; not being able to prove, via inference, that a disposition is available or an affordance manifestable does not mean this is actually so. It simply means the agent does not know enough to prove, and perhaps might find it illuminating to test, whether in simulation or in the real world.

Sometimes however, a robot does know enough to prove that an object does not have a particular disposition, or that a combination of objects cannot be used to manifest an affordance. These negative results can be used to avoid unnecessary tests via other techniques to detect what dispositions and affordances are available.

For example, let us consider that we need to cover a pot, and it is not clear which items might provide the necessary disposition to achieve this. However, we know that whatever is a potential cover must afford moving by a human agent using their hands only. We can then define a query concept similar to the first competency question (Listing 3.3).

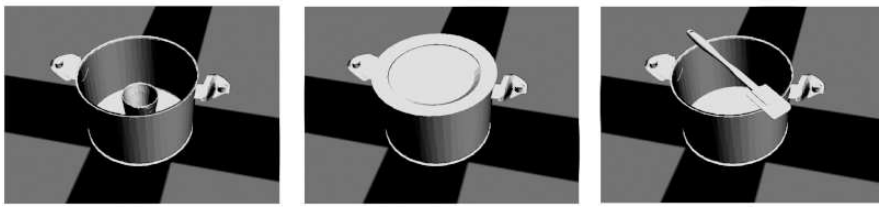

Figure 3.8: Pairs of items in the coverage affordance test: the coverage of a pot is attempted by a cup, plate, and spatula respectively. The images show final states of the world computed by letting the simulated world run through several simulation steps.

Listing 3.6: An example named concept to query for object classes that can be the bearer of a `Movable` disposition triggerable by a humanoid hand

```
Class: WithTriggerable_Movable_HumanoidHand
EquivalentTo:
  DesignedArtifact and hasDisposition some (Movable and
    (affordsTrigger some (classifies only HumanoidHand)))
```

To query for object classes that do *not* obey the requirement, it is not enough to add this concept to the ontology; instead, named concepts for intersections of the query concept in Listing 3.6 with object classes need to be included, e.g.:

Listing 3.7: An example named concept to query whether a particular object class has a disposition triggerable by another

```
Class: TriggerableBearer_Movable_HumanoidHand_Refrigerator
EquivalentTo:
  Refrigerator and WithTriggerable_Movable_HumanoidHand
```

Then, a DL reasoner can be queried for subsumption, and object classes that are now inferred empty correspond to items that cannot be used to manifest the desired affordance.

Simulation-based Affordance Testing

In addition to ontological reasoning, to test whether a particular affordance is made available by a set of objects, the simulation approach described by Bateman et al. (2019) is used. A few examples of this test are shown in Figure 3.8. A *scene*, i.e., the world of a physics engine, is populated with objects placed such that various pre-determined initial relations hold between them; the simulation runs for a short period of time, and the resulting timeline is then analyzed to check whether several conditions hold.

Scene specification(container=X, cover=Y)
(an object (type Y) (at (a location (on (an object (recognizable-as container) (type X))))))
Plan to run
(perform (a motion (trajector (an object (type pellet))) (source (a location (outside (an object (recognizable-as container))))) (destination (a location (inside (an object (recognizable-as container)))))))
To check on simulation timeline
(not (holds ?tl (and (world ?w) (object ?w (an object (type pellet) (at (a location (inside (an object (recognizable-as container))))))) (at :final)))

Table 3.1: The execution context for testing a *cover* affordance.

A description of the initial scene, the simulation behavior, and timeline conditions is called an *execution context*. Table 3.1 describes an execution context for testing a *coverage* affordance, where coverage is understood as blocking all paths to the interior of a container. The notation is that of CRAM *designators* (Beetz et al., 2010), that are key-value pairs describing objects, locations, and actions. Designators are *resolved*, that is, converted into quantitative descriptions, via a mix of reasoning processes such as generation and validation of position samples.

Testing in simulation is done because, ultimately, the behaviors that use an affordance are the behaviors of physically embodied agents, and thus physical and geometric considerations become important. However, simulation is computationally intense, and without some knowledge in the system, it is not clear a-priori which object combinations are worth testing. A theory of affordances can narrow down the list to a small number of plausible candidates, leaving the simulation as an arbiter for cases that are ambiguous or yet unknown in the affordance theory.

3.2.4 Summary

In this section, a theory inspired by Turvey's notion of dispositions was introduced that attempts to capture the descriptive nature of affordances. It was shown, by using an OWL implementation of the proposed theory, that models of the theory can be used to answer a set of relevant questions in the robotics domain.

3.3 Conclusion

This chapter was concerned with an ontological characterization of robot activities. A novel domain ontology, called SOMA, was proposed. The ontology has been designed to cover a set of competency questions that support robot decision making during action execution and observation. This has been achieved through the representation of physical and social context of an activity, and by establishing relationships between both contextualizations. These representations are used by robotic agents to fill knowledge gaps in general plans applicable to many situations, and to generate context-specific behavior. The SOMA ontology is further used for the representation of memorized events, and for reasoning about how they are to be interpreted. A robot can memorize instances of concepts defined in SOMA when it performs, simulates, or observes an activity, and thereby it can gain experience. Such memorized experiences can be leveraged for learning, i.e., by finding correlations between the physical or social activity context and how the robot has interacted with the world through its sensorimotor system. Since the concepts are organized in a hierarchy, correlations can be identified on different levels of abstraction which is useful for training models of different generality.

SOMA builds upon a well established foundational ontology, and extends the core ontology with a more fine-grained action model, and several models that implement different theories of human cognition. One of the theories of human cognition that has been implemented as a part of the SOMA ontology is the *Descriptive Affordance Ontology*. Its axiomatization and implementation in OWL has been discussed in this chapter, and several relevant competency questions were identified that can be implemented through a standard OWL reasoner. Such a theory of affordances can be a vehicle to reduce the programming effort needed to implement robots that perform many tasks in many different environments in a flexible manner. Affordances may also allow and constrain the execution of motions in certain ways which is a pertinent subject for future work. Ultimately, there is a need to establish a set of well-founded modular theories for the cognitive building blocks for flexible, adaptive and robust behavior, such as image schema (Lakoff, 1987), force dynamics (Talmy, 2000) and x-schema (Bergen and Chang, 2003).

4 Ontology-enabled Planning in Robot Manufacturing

Assembly cells run by intelligent robotic agents promise highly flexible product customization without the cost implication product individualization has nowadays. One of the main questions an assembly robot has to answer is which sequence of manipulation actions it should perform to create an assembled product from scattered pieces available. In this chapter, a novel approach to assembly planning will be presented that employs an ontology to describe what an assembled product should look like, and to plan the next action according to faulty and missing assertions in the robot's beliefs about an ongoing assembly task. It will be shown that assembly recipes can elegantly be represented in DL ontologies. With such a recipe, the robot can figure out the next assembly step through logical inference. The inference can further be performed efficiently using a standard OWL reasoner. However, before performing an assembly action, the robot needs to ensure various spatial constraints are met, such as that the parts to be put together are reachable, and non occluded. Such problems would be very complicated to support in logic theories, but specialized algorithms exist that efficiently can solve them for realistic scenarios. The second contribution of this chapter is concerned with combining a logic-based planner for assembly tasks with geometric reasoning capabilities to enable robots to perform their tasks under spatial constraints. The geometric reasoner is integrated into a logic-based knowledge base through decision procedures attached to symbols in the ontology. Related prior publications are Beßler et al. (2018c), and Beßler et al. (2018b).

4.1 Flexible Assembly Planning

Some industrial actors have expressed interest in developing assembly cells using newly available impedance controlled robot arms. Such cells promise to be versatile and reconfigurable, a desired quality when customization is important. However, several challenges remain before such promises can be delivered. This section will focus on issues of knowledge representation and planning.

Every product that an assembly cell can produce requires a different assembly plan, and every time the product changes, e.g., because new standards require some parts to be replaced, the plan needs to be adapted or recreated from scratch. Further, plans are sensitive to the kinds of resources available (the parts the cell can access) and to the capabilities of the robot. Finally, when creating small batches of customized products it is often the case that the customizations are variations of each other: using one part instead of another, or varying in terms of which accessories are attached to a common skeleton. In principle, assembly re-planning can tackle all of these issues, but is computationally expensive. Instead, here, a knowledge-enabled approach is pursued where the robotic agent is informed about its own capabilities and available resources, has knowledge about the products it needs to assemble, and the representation is such that it allows quick adaptation. OWL and rule-based reasoning are used to represent and process this knowledge.

Classical planning methods, such as PDDL (Ghallab et al., 1998), usually make the closed world assumption, and thus may not be able to find a solution if some facts about the world are not known. This is fairly limiting: robots cannot begin an assembly until all parts are known, and cannot react on unexpected changes of the world state or have to re-plan in such a case. Instead, the proposed planning method uses open world semantics such that the robot can begin an assembly activity with incomplete knowledge, identify the missing knowledge pieces, and reason about how the missing information can be obtained. Another difference is that, instead of using a specialized solver, the proposed planning method employs general DL reasoning in combination with domain-specific heuristics.

The proposed approach is to describe, in an ontology, concepts of finished assemblages by their parts, sub-assemblies, and how the parts are connected with each other via their dispositions, and to compare these descriptions with

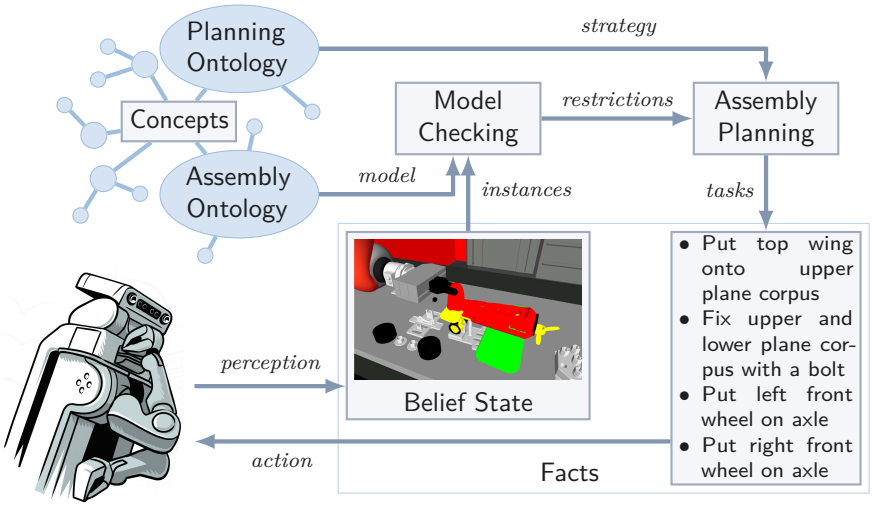

Figure 4.1: An architecture for flexible assembly planning where tasks are inferred by a comparison of the robot's belief state with an ontological model of the assembled product.

what the robot believes about an ongoing assembly task to infer what to do next. Concept definitions include restrictions the finished assemblages must satisfy. The restrictions would initially not be materialized by an individual, and the robot will act so as to obtain the missing information. In more detail, based on the difference between an initial state (available scattered parts) and a desired state (a particular assemblage), an ordered list of tasks is created that, when performed, will transform the assemblage such that it is materialized in the knowledge base. This mechanism runs within the perception-action loop of the robot: the robot's beliefs are updated according to objects perceived and actions performed, and assembly actions are selected and parameterized by reasoning about the robot's beliefs and how parts can be assembled. This architecture is depicted in Figure 4.1.

Knowledge modeling for assembly processes is also non-trivial, and has not been thoroughly developed for robots. For this work, the SOMA ontology (Chapter 3) will be extended with general assembly concepts such as mechanical part, atomic part, assemblage, assembly disposition etc. Furthermore, reasoning methods will be added to the KnowRob knowledge base (Tenorth and Beetz, 2013) that allow to reason about assemblages, their parts, connections, and configurations, and how configurations can be established. To this end, KnowRob's handling of the robot's belief state will be extended such that it includes information about (partial) assemblages present, and the connections between their parts.

Summarizing, the contributions of this section are as follows:

- A novel ontology for assembly tasks that describes assemblages, their parts, and how parts should be grasped, held and connected with each other.
- A set of extensions to the KnowRob knowledge base that enable to reason about (partial) assemblages, and to identify characteristics that are implied by terminological definitions but not materialized in the knowledge base.
- A novel method for knowledge-enabled assembly planning that controls the process of materializing missing facts about an assemblage, and that is designed to run within the perception-action loop of a robotic agent.

4.1.1 Related Work

There are several efforts to provide ontologies for robotics. By far the largest is that of the IEEE-RAS working group ORA (Schlenoff et al., 2012), which aims at standardizing knowledge representation for robotics. The ORA core ontology (Prestes et al., 2013) has been augmented for specific industrial tasks (Fiorini et al., 2015). These extensions cover tasks such as kitting, where a robot places a set of parts on a tray or similar receptacle to be carried towards an assembly cell. Other robotic ontologies are the *Affordance Ontology* (Varadarajan and Vincze, 2012) and the SOMA ontology presented in this work, the latter of which will be used and extended in this section. None of the mentioned ontologies model assembly concepts as of yet.

As mentioned, the knowledge-enabled approach has been successfully employed for some industrial processes such as kitting (Balakirsky et al., 2013; Balakirsky, 2015; Polydoros et al., 2016). Knowledge-enabled assembly has been investigated by the EU ROSETTA project (Patel et al., 2012; Malec et al., 2013; Stenmark et al., 2015a); their knowledge representation includes concepts for tasks and sequences of tasks and basic robot skills. Other approaches to knowledge-enabled assembly convert OWL descriptions into PDDL specifications (Balakirsky et al., 2013; Kootbally et al., 2015). Knowledge-enabled programming has also been employed as a means to ease teaching a robot cell to assemble new products (Perzylo et al., 2016); similar to the knowledge modeling approach in this work, they concentrate

on annotating geometric data about mechanical parts with information about semantically meaningful features which can then be used to construct constraints and sub-goals for an assembly task description.

In the cited works, the represented knowledge is to be used and exchanged between various system components such as planning, perception, and executives (so as to allow reasoning and re-planning) or training interfaces (in which case they provide the "vocabulary" to describe a sequence of tasks in). Generation of action sequences based on the semantic descriptions themselves is not done, unlike in this work. Also, to the extent that knowledge modeling intended for assembly appears, it is either very generally about sequences of tasks (Malec et al., 2013), or focuses on geometric features of atomic parts (Perzylo et al., 2016) as opposed to dispositions and intermediary sub-assemblages.

4.1.2 Modeling Assemblages

Assemblages can be described by their parts, sub-assemblies, and configurations thereof. Robots further need to know how to grasp and hold parts to connect them with each other. In the scope of this section, such type of information will be represented in a novel robot ontology for assembly tasks.

Ontology Hierarchy

Assembly Upper Ontology The most general assembly-related concepts are defined in this ontology. These can be divided into concepts subsumed by `DesignedComponent`, i.e., objects that were designed as a component of a bigger whole; `Linkage`, i.e., the disposition of objects being in a rigid connection; and `Affordance`, i.e., the descriptive context of a dispositional match, which are defined in the SOMA ontology (Chapter 3). In this ontology, general concepts are represented such as `MechanicalPart` $\sqsubseteq$ `DesignedComponent`, `Assemblage` $\sqsubseteq$ `MechanicalPart`, and `AssemblyConfiguration` $\sqsubseteq$ `Affordance`, but also some commonly used classes of configurations, parts and assembly dispositions (e.g., screwing, sliding and snapping configurations). The ontology counts 236 logical axioms and 56 classes, and has the DL expressivity $\mathcal{SROIQ}^{(D)}$.

Parts Ontology Part classes, dispositions they provide, and types of assembly configurations they can enter in are defined in this ontology. Each user of the system would define their own part ontology/ies, as these are project-specific. Parts, however, can be reused for different projects, hence they get a special level in the ontology hierarchy.

Assemblages Ontology Concepts for a top-level assemblage, and the sub-assemblages that it contains are defined in this ontology. These concepts are highly project specific. They describe the assembled product in terms of parts and configurations thereof, and all intermediate steps that determine how it can be constructed (potentially in different variants).

Simulation Ontology The previous ontologies define classes, rather than individuals. In a real world use case, individuals such as parts may be asserted by perception, and any dependent individuals (such as the dispositions the parts are expected to provide) are asserted through reasoning on part definitions. It is convenient however, especially for simulation experiments (Section 4.1.6), to have an OWL file describing the initial state of the world: what parts are available, where they are located, whether sub-assemblages already exist etc.

Defining Configurations and Assemblages

The pattern for modeling an assemblage is that it uses a `AssemblyConfiguration` that may need or block `AssemblyDispositions` offered by `MechanicalParts`.

MechanicalPart The concept for objects that get manipulated during an assembly operation. It is subdivided into `AtomicPart`, which refers to parts without separable components, and `Assemblage`. `AtomicPart` is further subdivided into `FixedPart`, useful for representing holders, and `MobilePart`. A `MechanicalPart` must offer at least one `AssemblyDisposition`, which allows the part to be used for certain types of assembly configurations. `MobileParts` should also offer at least one `GraspingDisposition`, which allows them to be grasped in certain ways. Note that both `AtomicParts` and `Assemblages` can offer `AssemblyDispositions`. This is because assembly-relevant features may appear only when more parts are

put together, for example a tunnel formed by two longitudinal halves. When a `MechanicalPart` is asserted to the belief state (e.g., because it was perceived or a new assemblage was put together) its dispositions are automatically inferred from existential restrictions that constrain the *hasDisposition* relation.

AssemblyConfiguration An `AssemblyConfiguration` is seen as a description that *needs* two `AssemblyDispositions`, and may block several dispositions. Typically an `AssemblyDisposition` that is needed is also blocked, and this is represented by a $consumesDisposition \sqsubseteq needsDisposition$ object property. But other dispositions, e.g., `GraspingDispositions`, may be blocked by a configuration: some grasps become impossible once a part is in a connection. Configurations are defined in terms of dispositions they need (rather than the `MechanicalParts` they connect) so as to also capture which configurations *prevent* others from being formed, and therefore have information in the ontology to reason about operation sequencing. Parts involved are linked to the configuration via the property chain:

$$hasAtomicPart \equiv needsDisposition \circ hasDisposition^{-} \tag{4.1}$$

Configurations are also associated with a transformation that defines how parts are placed relative to each other, and use one of the parts as a *reference* – if the `MechanicalPart` in question is an `Assemblage`, then its reference part is used. The current implementation presents a limitation, in that the parts in a configuration cannot be of the same type, or else it would be impossible to disambiguate which should be the reference without further data. This rare issue should be addressed in a future revision of the proposed planning method.

Assemblages The aggregates of mechanical parts. An `Assemblage` uses exactly one `AssemblyConfiguration`, and may place further restrictions on the types of involved parts. This is because configurations can be fairly general, such as slide-in configurations, whereas an assemblage may require a particular wheel to be slid on a particular axle. Another important property when defining `Assemblage` subclasses is the *linksAssemblage* property chain that can be defined as:

$$linksAssemblage \equiv hasAtomicPart \circ hasAtomicPart^{-} \circ usesConfiguration^{-} \tag{4.2}$$

Using restrictions involving the *linksAssemblage* property, sequencing information about assembly operations can be encoded, and it can be inferred what parts are needed for an assembly task. However, properties defined by means of property chains cannot appear in OWL DL axioms as they are not *simple*, and hence the chain must be spelled out in form of a class description.

Assembly Belief State KnowRob has been designed for service robots acting in the kitchen (Beetz et al., 2011). The belief state, in this case, contains information about the position of objects in the world and which objects are being grasped at the moment. Another way to look at this is that KnowRob can answer questions such as *what objects are present*, *what objects are being grasped*, and *what is the transformation between tool frame and grasped object?*

Assembly robots also need information about how objects are connected into (sub)assemblages. For this work, KnowRob's belief state was extended to include this information. In other words, the extension makes KnowRob able to answer queries such as *what are the parts that are contained in an assemblage*, *what parts are (possibly indirectly) connected to a given part*, and *what is the transformation between two parts that have a connection between them?* These are implemented as part of an open-source add-on for the knowledge base system KnowRob. The implementation of some of the example queries is discussed in Section 4.1.6.

In terms of implementation, KnowRob's handling of belief states has been adjusted such that an object pose (for all objects that are not robot links) is reported in a frame in which the object is fixed. For a free-floating object, this is the *world* frame. For an object that is part of an assembly with none of its component parts grasped, either that object is the reference object for the assembly (in which case its pose is given in world coordinates), or the object pose is given relative to the reference object. A grasped object, or an object that is part of an assembly which has one component grasped, has its pose given in the tool frame of the gripper. The reason for this approach is that there could be many objects available to a robot doing assembly, so to reduce the load on the transformation message channel and its subscribers that keep track of where everything is, the object poses are published rarely: whenever the reference frame changes, or at some large interval (typically a couple of seconds).

4.1.3 Reasoning Methods

The problem of finding a set of components that fit together is a configuration problem. Configuration is a topic with a long history in the area of artificial intelligence (Hotz et al., 2014). It was pointed out early that DL-based approaches are well suited for configuration applications (McGuinness, 2003), and it was further proposed to use semantic web technologies for product configuration software (Yang et al., 2008). In the following, a similar method will be proposed that uses DL-level reasoning to compute a set of tasks supposed to be executed by a robot to materialize a configuration of an assembled product.

Specializability of Individuals

The assembly ontology restricts assemblages to link certain parts that have certain properties. Initially, some of the restrictions may be not materialized such as that an assemblage must link a particular part type. Given the type concept, potential candidates that are instances of it can be deduced. Individuals that are not instances of the type, on the other hand, could be *specializable* such that they turn into an instance of the range by asserting new facts about them.

Definition 3. *An individual x is specializable to a class C if new (consistent) facts can be asserted such that the individual turns into an instance of the class.*

In the following, $x \gg C$ is written if individual x can be specialized to the concept C. This is the case if all restrictions imposed by C are satisfiable by x, i.e., if there are no conflicts between the definition of C and role and type assertions of x. The role and type assertions of x correspond to the concept $\{x\}$ that only has x as an instance. Accordingly, x is specializable to C if and only if the intersection of $\{x\}$ with C is satisfiable, i.e., if there can be instances of this concept. Thus, the specializability of individuals can be defined as:

$$x \gg C \iff \neg(\{x\} \sqcap C \sqsubseteq \bot) \tag{4.3}$$

For example, an individual is specializable to a universal restriction $\forall P.C$ if all the existing values of P are specializable to C, and if no axiom restricts the range of P to a concept D with $C \sqcap D \sqsubseteq \bot$.

Computation of Materialization Sets

In this work, planning is driven by identifying and materializing missing facts in the ABox of a knowledge base. An individual x is assumed to be materialized with respect to an ontology $\mathcal{O} = (\mathcal{A}, \mathcal{T})$ if the ABox $\mathcal{A}$ contains all the facts about x that are implied to exist through subsumption axioms in the TBox $\mathcal{T}$. Usually, the ontology does not imply specific role assertions, and rather defines conditions under which objects can have a certain role. The robot must, however, pick particular objects for its actions. To this end, it must figure out how an individual can materialize a concept by adding and removing facts that describe the individual. A unmaterialized fact implied by the subsumption hierarchy corresponds to an arbitrary DL concept C, and it can further be deduced up to which axiom an individual x materializes the definition of C. In the following, the set of such missing facts will be called *materialization set*.

A materialization set consists of binary predicates using the functors + and - to indicate assert and retract operations respectively. In the following, such *operation terms* will be written in infix notation. Each operation term is about the assertion of a role or type, or the retraction thereof. Type assertion terms have the form $x{+}C$ where $x \in N_O$ is an individual, and $C \in N_C$ is a concept name. The meaning is that x should be classified as an instance of C, i.e., a fact $x : C$ should be added to $\mathcal{A}$. Role assertion terms may either have the form $(x, y){+}P$ where $x, y \in N_O$ and $P \in N_R$ is a role name, or $(x, C){+}P$ where C is a concept. The former case represents a particular role assertion, while the latter specifies that an additional assertion is needed using a value of the type defined by concept C. Operation terms for retractions are represented analogously, i.e., $x{-}C$ means to retract $x : C$, and $(x, y){-}P$ to retract $(x, y) : P$ from $\mathcal{A}$.

There may be multiple options how an individual materializes a concept. For example, a concept $\forall P.C$ is materialized by an individual x if all existing P-values of x have the type C. If this is not the case, role assertions can either be retracted, or existing values can be classified as instances of C. Such different options are included in the materialization set. It is thus required to check whether the operation associated to an item in the set is still required before executing it. The execution of an operation is not required anymore after some alternative operation has been executed that materializes a restriction that has caused both items.

The cause of an item is always that a restriction is not completely materialized by an individual, and before processing the item it can be checked if this is still the case. Each cause is represented as a tuple (x, C) where $x \in N_O$ is an individual name, and C a concept description. There may be multiple causes of a single item. The reason is that items can be generated for sub-expressions in concept descriptions. Thus, each operation term is associated with a set Z of tuples that represent what caused the item.

Let $\mathcal{M}$ be a matrialization set. Each $m \in \mathcal{M}$ is a tuple $m = (t, Z)$ where t is an operation term and Z the set of unmaterialized restrictions that caused the item. The materialization set is computed via the function $\mu(x, C)$ given an individual $x \in N_O$ and a concept C. The function is recursively defined, and the set of causes in the next step is iteratively computed as $Z' = Z \cup \{(x, C)\}$ given the set of causes Z of the current step. Note that $Z = Z'$ indicates that further evaluation of μ would cause an infinite recursion. Hence, an empty set is returned in such cases. Accordingly, a function $\mu(x, C, Z)$ can be introduced that also keeps track of causes. It can be defined along different cases of concept C:

$$\bigcup_{(C \sqsubseteq D) \in \mathcal{T}} \mu(x, D, Z') \cup \mu_N(x, C, Z') \qquad \text{if } C \in N_C \quad (4.4a)$$

$$\bigcup_{1 \leq i \leq n} \mu(x, C_i, Z') \qquad \text{if } C = C_1 \sqcap \cdots \sqcap C_n \quad (4.4b)$$

$$\bigcup_{1 \leq i \leq n} \mu(x, C_i, Z') \qquad \text{if } C = C_1 \sqcup \cdots \sqcup C_n \wedge \varphi_{\sqcup} \quad (4.4c)$$

$$\bigcup_{(Q,y) \in \mathcal{D}_-} \mu(y, D, Z') \cup \{((x, y)\text{-}Q, Z')\} \qquad \text{if } C = (\forall P.D) \quad (4.4d)$$

$$\bigcup_{(Q,y) \in \mathcal{D}_+} \mu(y, \neg D, Z') \cup \{((x, y)\text{-}Q, Z')\} \qquad \text{if } C = (\leq n P.D) \wedge \kappa > n \quad (4.4e)$$

$$\bigcup_{(Q,y) \in \mathcal{D}_-} \mu(y, D, Z') \cup \{((x, D)\text{+}P, Z')\} \qquad \text{if } C = (\geq n P.D) \wedge \kappa < n \quad (4.4f)$$

$$\{((x, y)\text{+}P, Z')\} \qquad \text{if } C = (P{:}y) \wedge (Q, y) \notin \mathcal{D}_P \quad (4.4g)$$

$$\mu(x, (\geq 1 P.D), Z) \qquad \text{if } C = (\exists P.D) \quad (4.4h)$$

$$\mu_{\neg}(x, D, Z') \qquad \text{if } C = \neg D \quad (4.4i)$$

Where the set $\mathcal{D}_P = \{(Q, y) \mid ((x, y){:}Q) \in \mathcal{A} \wedge Q \sqsubseteq P\}$ contains a tuple for each role assertion $((x, y){:}Q) \in \mathcal{A}$ if the asserted role Q is a sub-role of P, and

$\mathcal{D}_+ = \{(Q, y) \in \mathcal{D}_P | D(y)\}$ and $\mathcal{D}_- = \{(Q, y) \in \mathcal{D}_P | \neg D(y)\}$ divide $\mathcal{D}_P$ based on whether the individuals are instances of concept D. Finally, $\kappa = |\mathcal{D}_+|$ is the number of instances of D that are P-related to x. Note that $\mu(x, C) = \mu(x, C, \emptyset)$, i.e., the computation is started with $Z = \emptyset$. In the following, the different cases of Equation 4.4 will be explained in more detail. For all other cases, the function μ yields an empty set indicating that x already materializes concept C.

First, it is assumed that a named concept C is materialized by an individual x only if there exists an assertion $(x : C) \in \mathcal{A}$ that classifies x as an instance of C, and if x materializes all explicitly asserted super-concepts of C (4.4a). Accordingly, the function μ_N can be defined as:

$$\mu_N(x, C, Z) = \begin{cases} \emptyset & \text{if } (x : C) \in \mathcal{A} \\ \{(x{+}C, Z)\} & \text{else} \end{cases} \tag{4.5}$$

Furthermore, intersection and union concepts are materialized by an individual if all or at least one of their members are materialized by the individual respectively (4.4b, 4.4c). Accordingly, the condition that a union concept is not completely materialized is defined as $\varphi_\sqcup = (\neg\exists D \in \{C_1, \ldots, C_n\} : \mu(x, D) = \emptyset)$, and items are only added to the materialization set if this condition holds true, i.e., if there is no C_i that is completely materialized by x. Note that, in the case of unions, different alternatives how the concept can be materialized are represented in the materialization set, while, in the case of intersections, all items in the materialization set are required to materialize the concept.

Universal restrictions of the form $\forall P.C$ can be materialized by an individual by removing P-related values that are not instance of C, or by classifying those values (4.4d). Cardinality constraints are materialized by an individual if the number of P-related values does not exceed, or is higher than a certain value. If the individual has too many P-related values, either role or type assertions can be removed (4.4e). In case there are not enough P-related values of a given type, additional values can be added, or existing values can be classified (4.4f). Value constraints of the form $P{:}y$ are materialized by an individual x only if y is P-related to x (4.4g). Finally, existential restrictions of the form $\exists P.D$ are translated into a cardinality constraint of the form $\geq 1 P.D$ (4.4h).

Furthermore, the function $\mu_\neg(x, C, Z)$ is used for concepts of the form $\neg C$ (4.4i). It can be defined along different cases of the concept C:

$$\mu(x, D, Z) \qquad \text{if } C = \neg D \tag{4.6a}$$
$$\mu(x, (\exists P.(\neg D)), Z) \qquad \text{if } C = (\forall P.D) \tag{4.6b}$$
$$\mu(x, (\leq 0 P.D), Z) \qquad \text{if } C = (\exists P.D) \tag{4.6c}$$
$$\mu(x, (\leq (n-1) P.D), Z) \qquad \text{if } C = (\geq n P.D) \tag{4.6d}$$
$$\mu(x, (\geq (n+1) P.D), Z) \qquad \text{if } C = (\leq n P.D) \tag{4.6e}$$
$$\mu(x, (\neg C_1 \sqcup \cdots \sqcup \neg C_n), Z) \qquad \text{if } C = C_1 \sqcap \cdots \sqcap C_n \tag{4.6f}$$
$$\mu(x, (\neg C_1 \sqcap \cdots \sqcap \neg C_n), Z) \qquad \text{if } C = C_1 \sqcup \cdots \sqcup C_n \tag{4.6g}$$
$$\bigcup_{((x,y):Q) \in \mathcal{A} \wedge Q \sqsubseteq P} \{(x, y)\text{-}Q\} \qquad \text{if } C = (P{:}y) \tag{4.6h}$$
$$\bigcup_{D \in \mathcal{C}_1} \{(x\text{-}D, Z)\} \cup \bigcup_{D \in \mathcal{C}_2} \mu(x, \neg D, Z) \qquad \text{if } C \in N_C \tag{4.6i}$$

Where the set $\mathcal{C}_1 = \{D \mid (x{:}D) \in \mathcal{A} \wedge D \sqsubseteq C\}$ contains sub-concepts of C that are asserted as types of x, and $\mathcal{C}_2 = \{D \mid C \equiv D \wedge C \neq D\}$ is the set of concepts that are equivalent to concept C.

The complement of a value restriction on role P is materialized by an individual x if the value given in the restriction is not P-related to x (4.6h). Furthermore, the complement of a named concept C is materialized by an individual if there is no type asserted for the individual that is a sub-concept of C, and if the complement of each concept that is equivalent to C is also materialized (4.6i). The other cases are simple mappings to the μ function. For example, the complement of a universal restriction $\forall P.D$ is materialized if the existential restriction $\exists P.(\neg D)$ is materialized (4.6b). The function $\mu_\neg$ yields an empty set for all other cases.

Note that enumeration concepts of the form $\{y_1, \ldots, y_n\}$ where each $y_i \in N_O$ is an individual name are ignored in the definition of μ. Also, in the case of value restrictions, it is ignored that individuals can be equivalent to each other, i.e., an equivalence assertion could be added between an existing value and the one specified in the restriction. Further note that the execution of operation terms could render the ontology inconsistent. It is thus required to test whether an inconsistency would be caused before executing an item of the materialization set.

4.1.4 Materialization of Restrictions

The process of transforming an incomplete and underspecified assemblage to one that links all required parts, and which is fully specified, needs to be controlled. This work proposes a knowledge-enabled approach that allows to switch between different tasks, order actions using domain specific knowledge, explain why actions were planned, and reason about how an action can be performed. In this section, the knowledge pieces and method used to implement this process will be defined.

Materialization Agenda

The goal of the planner is the specification of all necessary facts about an individual assemblage. Such missing facts are deduced from types of the assemblage, and added to a priority queue that is called *agenda*. Elements of this queue are called *agenda items*. Agenda items describe a hypothetical fact about an individual. They are generated due to the following reasons:

`ClassifyItem` An individual must be an instance of a named concept, and thus must be classified accordingly.

`IntegrateItem` A required property of an individual is not specified, and an existing individual must be selected to specify the property value.

`DecomposeItem` A required property of an individual is not specified, and new symbols must be computed to be used as a value of the property.

`DetachItem` A property value of an individual must be retracted.

Note that these concepts do not cover the case of removing type assertions. Hence, it is assumed that the type of objects cannot be removed once asserted, and only the removal of role assertions is considered.

Agenda items are sequentially processed in preference order, i.e., by materializing facts described by them. They can, however, not simply be processed in a disembodied fashion. The reason is that symbols referred to by the items may need to be grounded in the sensorimotor system of the robot. For such cases, the robot might need to interact with the environment, and change relationships between physical objects according to requirements represented by the item.

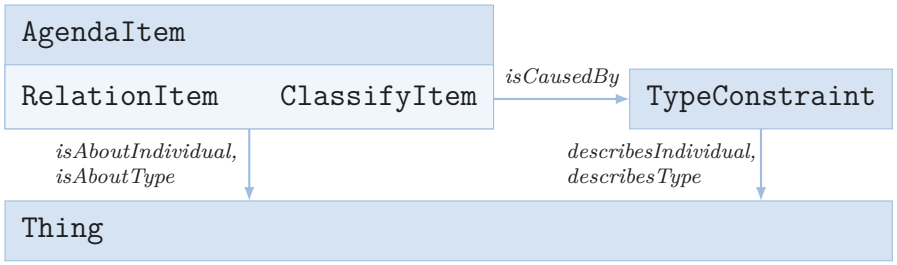

Figure 4.2: Concepts and relationships used to represent agenda items.

The proposed modeling of agenda items is based on the `AgendaItem` concept, and the *isAbout* relation (Figure 4.2). On the top-level, a distinction is made between the disjoint sub-concepts `RelationItem` and `ClassifyItem`. The difference between them is that the former is concerned with a relation, while the latter with a type of an individual. Other previously mentioned item types are all disjoint and sub-concepts of `RelationItem`. The hypothetical fact associated to an item is represented through the *isAbout* relation. The relation is divided into two sub-properties *isAboutIndividual* and *isAboutType*.

An agenda item is always about a particular individual that needs to be modified. This individual is denoted by the *isAboutIndividual* property. The *isAboutType* property is used by `ClassifyItems` where items have an additional type of the form $\forall isAboutType.C$, meaning that the individual must be an instance of the concept C. Finally, the *isAboutType* property is used by `RelationItems` in type assertions of the form $\forall isAboutType.(\exists P.C)$ where the item is about the specification of property P by a value of type C.

The planner requires a mechanism for filtering the agenda based on item patterns. These can be represented as sub-concepts of `AgendaItem` such that it can be inferred whether the pattern is a valid type for a given item. The treatment of the *isAboutType* property requires additional axioms for `RelationItems`:

$$\texttt{RelationItem} \sqsubseteq = 1\ isAboutType.\texttt{Thing} \tag{4.7}$$

$$\texttt{RelationItem} \sqsubseteq \forall isAboutType.(\leq 1\ isRelatedTo.\texttt{Thing}) \tag{4.8}$$

Thus, `RelationItems` must specify the *isAboutRelation* property once (4.7), and concepts defining the relation can only specify one relation symbol (4.8). The latter assumes that each considered property is a sub-property of *isRelatedTo*.

Items are caused by unmaterialized restrictions. The restricted individual is not necessarily the same as the one denoted by the *isAboutIndividual* property, i.e., in case missing facts are derived from nested restrictions. The causes of an item are represented by the `TypeConstraint` concept. It *describes* an individual and type through axioms of the form $\forall describesType.C$ where C is a concept. Let $A = \{(y_1, R_1), \ldots, (y_n, R_n)\}$ be the set of type constraints of an agenda item x where each $y_i \in N_O$ is an individual, and R_i is the concept defining the constraint that shall be satisfied by y_i. The item x is said to be *completed* if there exists a type constraint that caused the item which has been materialized already:

$$\texttt{completed}(x) \iff (\exists (y, R) \in A : \mu(y, R) = \emptyset) \tag{4.9}$$

Items are generated by iterating over type assertions of an individual x, and finding implied facts that are not materialized. Let $\varphi = (x : C)$ be a type assertion where $x \in N_O$ is an individual and C a concept. The set of unmaterialized facts of φ can be identified with $\mu(x, C) = \{t_1, \ldots, t_n\}$ where each t_i is a term representing an operation that would materialize one of the constraints. Next, the `items` function is defined as a constructor of agenda items given a type assertion φ:

$$\texttt{items}(\varphi) = (\mathcal{A}_1 \cup \cdots \cup \mathcal{A}_n) \tag{4.10}$$

Where each $\mathcal{A}_i = \texttt{item}(t_i)$ is the set of assertions defining an agenda item. The set $\mathcal{A}_i$ can be computed trivially given that $t_i = (s, Z)$ encodes what the item is about as operation term s, and the causes of the item as the set Z.

As an illustration, the following listing shows an example agenda item:

Listing 4.1: An example named individual of an agenda item

```
Individual: Item42
Types:
  ClassifyItem and (isAboutType only TopWingInBody)
Facts:
  isAboutIndividual Configuration1
  isCausedBy Contraint42.1
```

The item represents that the existing individual `Configuration1` needs to be classified as an instance of `TopWingInBody`, and that this is caused by a type constraint `Contraint42.1` that refers to the configuration.

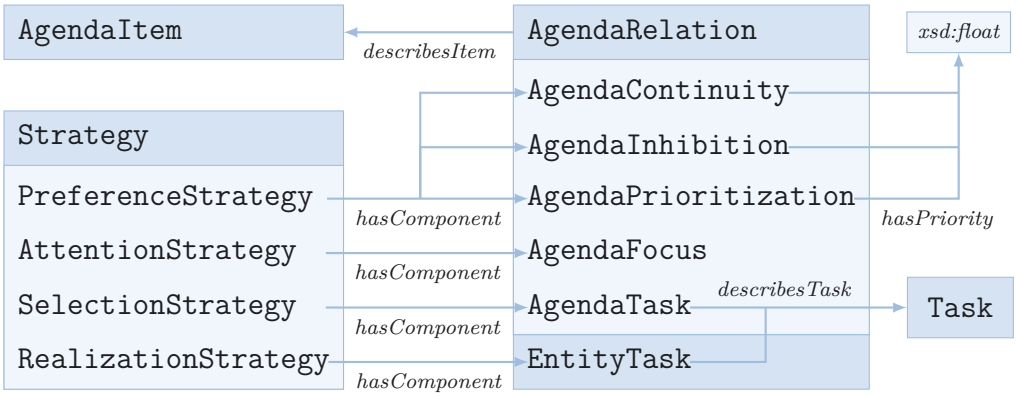

Figure 4.3: Concepts and relationships used to represent planning strategies.

Materialization Strategy

The process of transforming an incomplete assemblage into one that is fully materialized is carried out by sequentially processing agenda items. Several ways to control this process are considered in this work. First of all, a preference order between items will be defined. Furthermore, certain items are irrelevant for an activity, and thus do not need to be considered. Finally, processing an item involves the selection of a value, and the assertion of a fact in the knowledge base. Those facts may correspond to physical objects, and need to be anchored in the physical world through actions of the robot.

The strategy to control this process is defined as $\mathcal{S} = (\mathtt{S}_F, \mathtt{S}_C, \mathtt{S}_S, \mathtt{S}_R, <_{\mathcal{P}})$, where $\mathtt{S}_F, \mathtt{S}_C, \mathtt{S}_S, \mathtt{S}_R$ are concepts that are a component of the strategy, and $<_{\mathcal{P}}$ is a preference order between items. The concepts and relations used to represent strategies are displayed in Figure 4.3. They will be described in the following.

Each `Strategy` has exactly one component $\mathtt{S}_F \sqsubseteq$ `AttentionStrategy`. An `AttentionStrategy` consists of different `AgendaFocus` descriptions that each define a class of agenda items that is considered to be relevant in the scope of the strategy. The *describesItem* relation is used to denote the agenda items referred to in components of the strategy. For example, the following axioms define an `AgendaFocus` concept for items that are about `Assemblages`:

$$\mathtt{Focus}_{\mathtt{Assemblage}} \sqsubseteq \mathtt{AgendaFocus} \tag{4.11}$$

$$\mathtt{Focus}_{\mathtt{Assemblage}} \equiv \forall \mathit{describesItem}.(\exists \mathit{isAboutIndividual}.\mathtt{Assembage}) \tag{4.12}$$

1. Prefer items that were processed less often (Inhibition)
2. Prefer to stick to the same individual (Continuity)
3. Prefer *classify* tasks (Pattern)
4. Prefer *usesConfiguration* relations of *Assemblage* instances (Pattern)

Figure 4.4: A sequence of prioritized selection criteria that are used to sort agenda items.

Axiom 4.12 is used to define the class of agenda items that is considered relevant by restricting the range of *describesItem* to the concept defining the class of agenda items, in this case $\exists \mathit{isAboutIndividual}.\texttt{Assembage}$. Further note that it is an equivalence axiom. The idea is that a reasoner can infer sub-concepts provided that they specify a value for the *describesItem* relation, i.e., a particular agenda item. To this end, the sub-concepts must include a completeness axiom stating that the agenda item in question is the only one described by the concept.

The concept $\mathtt{S}_F$ of a strategy is defined as the union over all its `AgendaFocus` components: $\mathtt{S}_F \equiv \mathtt{F}_1 \sqcup \cdots \sqcup \mathtt{F}_n$ where each $\mathtt{F}_i \sqsubseteq \texttt{AgendaFocus}$ is a component of the strategy. Let $a = \{a_1, \ldots, a_n\}$ be a set of agenda items. The set can be filtered by testing which of the items can be described by $\mathtt{S}_F$:

$$\texttt{filter}(\mathtt{S}_F, a) = \begin{cases} \emptyset & \text{if } n = 0 \\ \texttt{filter}(\mathtt{S}_F, \{a_2, \ldots, a_n\}) & \text{if } \mathtt{F}_a \sqsubseteq \bot \\ \{a_1\} \cup \texttt{filter}(\mathtt{S}_F, \{a_2, \ldots, a_n\}) & \text{otherwise} \end{cases} \tag{4.13}$$

The concept $\mathtt{F}_a = (\mathtt{S}_F \sqcap \forall \mathit{describesItem}.\{a_1\} \sqcap \mathit{describesItem} : a_1))$ is a sub-concept of $\bot$ only if a_1 cannot be described by $\mathtt{F}_a$. The item can be ignored in such a case.

Each strategy also consists of a `PreferenceStrategy` used to define a preference order $<_{\mathcal{P}}$ between items. An example is illustrated in Figure 4.4. Such strategies are divided into components where each specifies a priority value used to determine the order in which it is tested whether an item is described by the preference, and thus preferred in the scope of the strategy. The components are defined via axioms using the *describesItem* property analogously to `AgendaFocus`

descriptions. The `AgendaContinuity` concept $\mathtt{S}_C$ is used to prefer items about the same individual as for the item processed before which can expressed by a value restriction on that property. Finally, `AgendaInhibition` is used to prefer items that were attempted less often. Let $\mathcal{P} = \mathtt{P}_1, \ldots \mathtt{P}_n$ be the non-empty sequence of preference relations ordered by their priority. Then, $<_{\mathcal{P}}$ can be defined as:

$$a_1 <_{\mathtt{P}_1,\ldots\mathtt{P}_n} a_2 \iff (v_1 \neq v_2 \wedge v_1 <_{\mathcal{N}} v_2) \vee (v_1 = v_2 \wedge n > 1 \wedge a_1 <_{\mathtt{P}_2,\ldots\mathtt{P}_n} a_2) \quad (4.14)$$

Where $v_1 = \mathtt{priority}(a_1, \mathtt{P}_1)$, $v_2 = \mathtt{priority}(a_2, \mathtt{P}_1)$, and $<_{\mathcal{N}}$ is the natural order.

The priority of an item a with respect to a preference relation `P` is determined based on whether the item can be described by the relation. In the case of `AgendaPrioritizations`, the value -1 is assigned to an item if it is not described by the relation, and 0 if it is described by it. `AgendaContinuity` is a sub-concept of `AgendaPrioritization`, and does not need special treatment. Finally, the priority value for `AgendaInhibition` relations is defined as the additive inverse of the number of times the item has been selected. Thus, we can write:

$$\mathtt{priority}(a, \mathtt{P}) = \begin{cases} -1 & \text{if } \mathtt{P} \sqsubseteq \mathtt{AgendaPrioritization} \wedge \mathtt{P}_a \sqsubseteq \bot \\ -v & \text{if } \mathtt{P} \sqsubseteq \mathtt{AgendaInhibition} \wedge v = \mathit{attempts}(a) \\ 0 & \text{otherwhise} \end{cases} \quad (4.15)$$

Where $\mathtt{P}_a = (\mathtt{P} \sqcap \forall \mathit{describesItem}.\{a\} \sqcap \mathit{describesItem} : a)$ is the preference relation for a, and *attempts* is a functional data property maintained by the planner.

Finally, a strategy consists of a component $\mathtt{S}_S \sqsubseteq \mathtt{SelectionStrategy}$ and $\mathtt{S}_R \sqsubseteq \mathtt{RealizationStrategy}$. These concepts are used to define *selection* and *realization* tasks via the notion of `AgendaTask` and `EntityTask`. An `AgendaTask` relates a class of items to a selection task whose execution specifies a role assertion described by a particular item. `EntityTasks`, on the other hand, correspond to physical tasks that the robot needs to perform for the realization of an assemblage in the physical world. They can be executed once an assemblage is materialized.

Selection methods are needed for `IntegrateItems` as the necessary type of individuals is defined by `ClassifyItems` and `DecomposeItems` through axioms involving the *isAboutType* property. Let a be a `ClassifyItem` which is about

the type C and individual x. The fact described by a is simply $x : C$. For the case of `IntegrateItems`, a selection method is needed to determine an existing individual to specify a role assertion. Let us assume that a is an `IntegrateItem` about individual x. Thus, the item is about $C = \forall P.C'$, i.e., a selection method must find an instance y of C' to specify the assertion $(x, y) : P$. To this end, the `specify` function that maps an item to a set of facts ca be defined:

$$\texttt{specify}(\texttt{S}_T, a) = \begin{cases} \{x : C\} & \text{if } \texttt{ClassifyItem}(a) \\ \{(x, y_u) : P, y_u : C'\} & \text{if } \texttt{DecomposeItem}(a) \\ \{(x, y_t) : P\} & \text{otherwise} \end{cases} \tag{4.16}$$

Where $y_u \in N_O$ is the unique name of a new individual, and $y_t = \texttt{select}(\texttt{S}_S, C')$ is an individual of type C' that has been selected with respect to the concept $\texttt{S}_S \equiv T_1 \sqcup \cdots \sqcup T_n$ where each $T_i \sqsubseteq \texttt{AgendaTask}$. The computation of the `AgendaTask` relation is done analogously to how the other agenda relations are computed, i.e., by testing if a relation cannot describe a given item. In the following, it is assumed that individuals are randomly selected, and it is only noted that selection could be done based on, e.g., distance to the robot, and that the robot may need to search for the object first.

`EntityTasks` are used for the realization of assemblage descriptions created by the planner. These are materialized but not manifested in the physical world. For this, the robot has to execute physical tasks. To this end, `EntityTasks` are used that *describe* a physical task, and a class of objects that can be realized by the execution of the task. The class of objects is defined through axioms of the form $C \sqsubseteq \forall \mathit{describesIndividual}.D$ where $C \sqsubseteq \texttt{EntityTask}$, and D is the concept that defines the class of objects. Tasks can be executed for each materialized individual $x \in N_O$ that is an instance of D, i.e., if $D(x)$ and $\forall (x : C') \in \mathcal{A} : \mu(x, C') = \emptyset$. Given a completely specified assemblage, action commands for the robot can be generated. This will be discussed in Section 4.1.5. This mechanism is implemented through the `realizeEntity` function that receives as input the concept $\texttt{S}_R = E_1 \sqcup \cdots \sqcup E_n$ where each $E_i \sqsubseteq \texttt{EntityTask}$. For example, an `EntityTask` can be defined that is used for the realization of assemblages through an action where the robot connects two parts with each other.

Materialization Method

Next, a method for the materialization of assemblage restrictions will be defined, i.e., a method that yields facts about an assemblage that were not stored explicitly before. The method is displayed in Algorithm 4.1. It receives as input an assemblage individual x, and a strategy $\mathcal{S} = (\mathtt{S}_F, \mathtt{S}_C, \mathtt{S}_S, \mathtt{S}_R, <_\mathcal{P})$. The individual x is a fresh instance of an assemblage concept, or a partial materialization thereof. The output of the method is an ABox that contains facts materializing different restrictions for x, i.e., in the case of assemblages, role assertions defining the configurations used, and the parts and sub-assemblages they link.

Algorithm 4.1: Materialization of Restrictions

Input : An individual x, a strategy $\mathcal{S} = (\mathtt{S}_F, \mathtt{S}_C, \mathtt{S}_S, \mathtt{S}_R, <_\mathcal{P})$, and a consistent ontology $\mathcal{O} = (\mathcal{A}, \mathcal{T})$.
Output: An ABox where the type restrictions of x have been materialized.

```
/* Initialize agenda items based on existing type assertions. */
for (x : C) ∈ A do A ← A ∪ items(x : C)
T ← T ∪ {S_C ≡ ⊥} /* Initially, continuity relation cannot be fulfilled. */
while true do
    [a_1, ..., a_n] ← sort(<_P, filter(S_F, {y ∈ N_O | AgendaItem(y)}))
    if n = 0 then break
    a_1.attempts ← a_1.attempts + 1 /* Increment data property value. */
    /* Update continuity axiom. */
    T ← (T \ {S_C ≡ ∗}) ∪ {S_C ≡ (AgendaContinuity ⊓
    ∀describesItem.(isAboutIndividual : isAboutIndividual(a_1)))}
    if completed(a_1) then delete(a_1)
    else
        A_1 ← specify(S_S, a_1)
        /* Remove role assertions from ABox in case of DetachItem. */
        if DetachItem(a_1) then A ← A \ A_1
        /* Modify ABox only if assertions would not cause an inconsistency. */
        else if ∀(y : C) ∈ A_1 : y ≫ C ∧ ∀((y, z) : P) ∈ A_1 : y ≫ P : z then
            A ← A ∪ A_1
            for (x : C) ∈ A_1 do A ← A ∪ subitems(a_1, items(x : C))
        end
        realizeEntity(S_R) /* Execute physical tasks. */
    end
end
return A
```

Initially, agenda items are generated based on type restrictions that are asserted for individual x. These are used to express, for example, that a particular type of configuration must be used. Furthermore, the continuity relation $\mathtt{S}_C$ needs special treatment. It is initially defined to be equivalent to $\bot$ such that no item can be described by it, i.e., every item has initially the priority -1 with respect to $\mathtt{S}_C$.

In each step, agenda items are filtered and sorted according to strategy $\mathcal{S}$, and the item a_1 with highest priority is selected. Each item stores how often it has been selected so far. This counter must be incremented after a_1 has been selected. Furthermore, $\mathtt{S}_C$ is updated given the entity the item is about. In case a_1 is completed already, it can be deleted from the ABox. Next, the set of facts $\mathcal{A}_1$ of the item is computed, and removed from the ABox if a_1 is an instance of `DetachItem`. If this is not the case, the facts are added to the ABox, and additional items are generated for type assertions in $\mathcal{A}_1$ if no inconsistency is caused by adding the facts. The function `subitems` is used to add each `TypeConstraint` of a_1 to items generated by the `items` function. Finally, the `realizeEntity` function is used to generate action descriptions to be executed by the robot. Note that new facts, such as that a part is connected to another one, could be created through task execution. Furthermore, the robot may figure out additional constraints such as that a particular part cannot be used, e.g., because it is out of reach. The handling of these aspects would be an interesting direction for future work.

4.1.5 Assembly Actions

In this section, two actions are considered: grasping of parts and assemblages, and connecting two parts to form an assemblage. These actions may be performed in many different ways, and with different part and configuration types – e.g., screwing and slide-in configurations need to be operated with different motions.

General action descriptions, called *action designators*, can be derived from knowledge about the structure of an assemblage, and how parts can be put together. It is up to the plan executive how action designators are instantiated such that they are executable on a particular robot. Additional action-relevant information, such as the location of the object, where to grasp it, or its physical properties, can be queried by the plan executive during action designator instantiation to enhance the reusability of the plan schemata.

```
action-type : AssembleTwoParts
part
  part-type : PlaneBottomWing
  part-name : PlaneBottomWing_1
with-part
  part-type : PlaneChassis
  part-name : PlaneChassis_1
with-configuration
  configuration-type : BottomWingSlideInChassis
  configuration-name : BottomWingSlideInChassis_1
```

Figure 4.5: An example action designator generated during assembly planning. It is used by the robot for the parameterization of action routines in the control program.

Connecting Things Action designators are generated in a bottom-up fashion starting from fully specified assemblages without sub-assemblages, and following the inverse of the sub-assemblage relation until an underspecified assemblage was reached. They include information about parts and their intended configuration with each other, and thus can only be generated after this information was specified. An example action designator is depicted in Figure 4.5. Mapping of partially specified assemblages to action designators executable by the robot is done in the `realizeEntity` function: each time it is invoked it is checked whether an action can be derived from facts about assemblages.

The robot has to reason about which of the parts needs to be moved into the other part, and which part should remain fixed during action execution. The fixed part must be held such that the assembly disposition that is intended to connect it to the other part is exposed. Single armed robots potentially need additional tools, such as holders or clamps, to fix one of the parts. Dual arm robots may choose to hold the fixed part with one gripper in such a way that the other gripper can move the other part into it. The best strategy for a robot thus depends on its capabilities. However, the assembly ontology does not represent this type of information (it only models how parts are connected in the final product), and thus it cannot be directly inferred which of the objects should remain fixed.

For conducting experiments, a heuristic for the determination of mobile and fixed part in an assembly action will be employed. The heuristic uses prioritized criteria to decide which of the parts is better suited to be moved into the other

one. The prioritized list is as follows: (1) prefer parts with an unblocked grasping disposition, or which are connected to a part with an unblocked grasping disposition – i.e., try not to destroy existing assemblages to be able to grasp some part; (2) prefer parts whose assembly disposition is blocked by a fixture (they must be moved anyway to expose the disposition); (3) prefer parts that are not attached to a fixture; and (4) prefer parts connected to a small number of other parts – i.e., move small assemblages into big ones.

Grasping Things Each part may have multiple grasping dispositions that imply some way to grasp the part. Assemblages may block the grasping disposition in case it is occluded by some other part. Parts are only graspable if they have at least one grasping disposition that is not blocked by some assemblage. However, the robot may indirectly grasp a part if it is rigidly connected to some other part with an unblocked grasping disposition.

The proposed ontology employs a simple grasping subsumption hierarchy with different types of power and precision grasps. Assembly parts can further describe the contact point, pre-grasp pose, grasping force, and how wide the robot should open its gripper when approaching the object. A connection is established between the end effector of a robot and the part it holds. Grasping configurations may block assembly dispositions. This allows robots to reason about whether they hold the part in the right way to perform an assembly action.

4.1.6 Evaluation

The performance of the assembly planning will be characterized along following three dimensions: steps required to model a new assembly task, what types of queries can be answered, and how fast the next action can be determined.

For the evaluation, a toy plane assembly targeted at 4 year old children is used. The toy plane is made of 21 plastic parts that can be connected with each other using loose slide in configurations, and fixed with bolts afterwards. The parts are comparably huge such that grasping them is easier. A single armed robot with a KUKA LWR arm is used, and the assembly planning is tested in a Gazebo (Koenig and Howard, 2004) simulation environment. The plane is part of the YCB Object and Model Set (Çalli et al., 2015).

Modeling First, a parts ontology for the toy plane needs to be created and populated with concepts corresponding to the parts (in this case, there are 12 part types). Each part concept defines the assembly and grasp dispositions that it offers. The dispositions were added to the definitions by hand, but there are ways to automate this process, either based on semantic annotations provided with the *Computer Aided Design* (CAD) files as in Perzylo et al. (2016), or possibly through geometric reasoning on part features (Tenorth et al., 2013b).

The next step is to create an assemblages ontology and populate it with sub-assemblages, up to the complete toy plane (in this case, the product consisted of 22 sub-assemblages). Each sub-assemblage also defines what sub-assemblages it should contain, which places some ordering constraints on how they are created. In this way, users can describe not only sequencing knowledge, but also knowledge about variations: sub-assemblages can be specified in terms of more general classes from the parts ontology (e.g., `Wing`) as long as any subclass is appropriate (e.g., `StraightWing`, `BacksweptWing`).

The parts and assemblages modeling was done using a general ontology modeling tool. This process took about four hours. However, a dedicated tool (e.g., with a form to specify a list of dispositions) would considerably speed up the process. Additionally, CAD models of the parts, and an instruction sheet on how to assemble the plane were available.

Querying In the assembly domain, querying is concerned with the structure of assemblages, as well as how to grasp and put together parts. In the following, it will be shown how such queries can be implemented using a DL reasoner.

The knowledge base initially consists of entities that correspond to objects in the environment, and previously build assemblages that may be incomplete. Initially, one of the incomplete assemblages must be selected by the robot. Here, incomplete means that certain facts about the assemblage must exist according to the subsumption hierarchy, but are not explicitly asserted in the knowledge base such as that a particular type of configuration must be used by the assemblage.

For each configuration used in an assemblage, parts must be selected that provide the necessary dispositions required by the configuration. Dispositions that can be used are constrained by the definition of the configuration concept.

For example, the dispositions that can be used for connecting a wing of the plane with its body can be defined by following axioms:

$$\begin{aligned}
\texttt{TopWingSlideInBody} &\sqsubseteq \forall \mathit{needsDisposition}.\texttt{PlaneTopWingSlideIn} \\
\texttt{PlaneTopWing} &\sqsubseteq =1\mathit{hasDisposition}.\texttt{PlaneTopWingSlideInM} \\
\texttt{PlaneTopWingSlideInM} &\sqsubseteq \texttt{PlaneTopWingSlideIn} \\
\texttt{PlaneTopWing} &\sqsubseteq \forall \mathit{hasDisposition}.(\texttt{PlaneTopWingSlideInM} \sqcup \ldots)
\end{aligned}$$

The last axiom is a completeness axiom that lists all disposition types of the part. It is needed when reasoning is performed under the open-world assumption to infer that a part cannot have a disposition of a certain type.

Based on these axioms, a DL concept can be defined for the purpose of querying a DL reasoner about whether a part type can be used for a configuration type. For the previous example, such a concept can be written as:

Listing 4.2: An example named concept to query if a part type can be used for a configuration

```
Class: IsUsable_PlaneTopWing_TopWingSlideInBody
EquivalentTo:
  TopWingSlideInBody and (needsDisposition some
            (isDispositionOf some PlaneTopWing))
```

Running a subsumption query on the assembly ontology, augmented with the definition of the concept listed above, produces that this concept is not subsumed by `Nothing`, i.e., instances of `PlaneTopWing` can be used in configurations of type `TopWingSlideInBody`. Note that this requires disjointness of disposition concepts.

Once a part has been selected, the robot needs to decide how the part can be grasped such that the disposition that is intended to be used is not blocked. Grasping is defined as the configuration between a disposition of the robot's gripper and a disposition of the grasped object. For example, a `PlaneTopWingSlideInM` disposition can be used for grasping the object that hosts the disposition. This can be formalized through a named concept `GraspTopWingSlideInM`:

$$\begin{aligned}
\texttt{GraspTopWingSlideInM} &\sqsubseteq \texttt{GraspingConfiguration} \\
\texttt{GraspTopWingSlideInM} &\sqsubseteq =1\mathit{consumesDisposition}.\texttt{PlaneTopWingSlideInM} \\
\texttt{GraspTopWingSlideInM} &\sqsubseteq \forall \mathit{needsDisposition}.(\texttt{PlaneTopWingSlideInM} \sqcup \ldots)
\end{aligned}$$

The task is then to identify `GraspingConfigurations` that may consume a type of disposition that is hosted by instances of some part type while not blocking the disposition type needed for the intended assembly configuration. For example, different ways to grasp `PlaneTopWings` while not blocking `PlaneTopWingSlideIn` dispositions can be determined through a query concept such as:

Listing 4.3: An example named concept to query ways to grasp that enable an assembly action

```
Class: UnblockedGrasp_PlaneTopWing_PlaneTopWingSlideIn
EquivalentTo:
  GraspTopWingSlideInM and
  (needsDisposition some (isDispositionOf some PlaneTopWing)) and
  (not (blocksDisposition some PlaneTopWingSlideIn))
```

Running a DL subsumption query produces that the listed concept is a subconcept of `Nothing`, i.e., `GraspTopWingSlideInM` cannot be used for the intended configuration. The reason is that the configuration *consumes*, and thus also *blocks* the required disposition.

Runtime Performance The runtime performance of the planning method depends on factors such as the size of the knowledge base, and the structural complexity of the assembled product. Varying the complexity of the assembly ontology itself is difficult. Instead, the structural complexity can be varied by testing the approach with different assemblages that the plane ontology describes. Further, the planning performance can be measured for growing size of statements in the ABox by varying the number of parts available for assembly.

In the first test, n random toy plane parts were asserted in addition to the twenty-one parts that are required. The parts are asserted together with their dispositions, which yields, depending on the part type, from eight up to twenty-nine triples for each of the asserted parts. The number n is varied from 0 (i.e., the minimum number of parts are available) to 3.8×10^4 (i.e., the robot can choose out of 38.021 different parts it knows about). This test is repeated thirty times to average out the random selection of part types. With only twenty-one parts, the planner is able to find a solution in roughly $1.1s$ within thirty-two planning steps. On average, each step takes $34ms$. In the experiments, the runtime performance dropped maximally to $2.103s$ for the test with 3.6×10^4 additional parts. However, the performance does not drop continuously. This is caused by hash table behavior

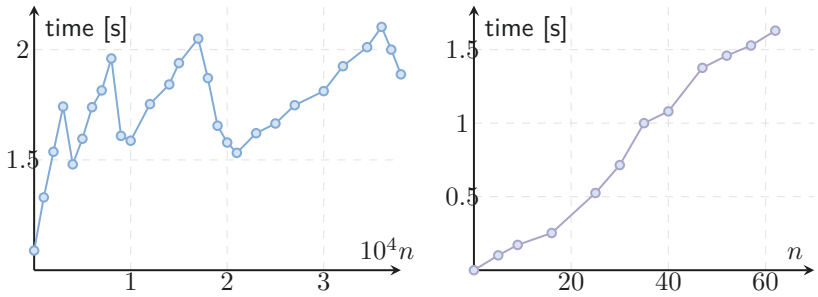

Figure 4.6: Runtime performance for toy plane assembly planning with varying number of available assembly parts (left), and with varying number of constraints the assemblage must satisfy (right).

of the triple storage employed in SWI Prolog (Wielemaker et al., 2003). Anyway, the test shows that the current implementation can deal with huge batches of parts: even with 5×10^5 additional parts a solution was found in $1.75s$

In the second test, the complexity of the assemblage task is varied in terms of constraints the final assemblage must satisfy. The toy plane ontology includes twenty-two assemblage descriptions with varying complexity from which eleven are chosen for this test. The complexity is measured by counting the number of assertions and retractions that were required to make an assemblage consistent. The most simple assemblage requires five transactions and can be planned in $0.101s$ (averaged over twenty trials), while the most complex assemblage with sixty-two transactions requires $1.629s$ of planning time. However, the average time between transactions remains nearly constant at $0.025s$ such that online planning is not affected much by the structural complexity of the assembled product. In more detail, the runtime test results are shown in Figure 4.6.

4.1.7 Summary

In this section, an ontology-based approach to assembly planning was proposed. The proposed method employs formal descriptions of fully assembled products in order to determine the next action of a robotic agent according to faulty and missing assertions in its belief state. It was shown that the presented approach scales well with growing number of parts available, and also with varying structural complexity of the assembled product, such that online planning is feasible. However, additional reasoning about action feasibility is needed by robots, which is the subject of the next section.

4.2 Assembly Planning with Geometric Relations

Robot tasks are usually described at a high level of abstraction. Such representations are compact, natural for humans for describing the goals of a task, and at least in principle applicable to variations of the task. An abstract *pick part* action is more generally useful than a more concrete *pick part from position* x, as long as the robot performing the action can locate the target part and reach it.

Robotics manipulation problems, however, may involve many task constraints related to the geometry of the environment and the robot. Constraints which are difficult to represent at a higher level of abstraction. Such constraints are, for example, that there is either no direct collision-free motion path or feasible configuration to grasp an object because of the placement of some other, occluding object. Recently, a lot of research has been centered on solving manipulation problems using geometric reasoning, but there is still a lack of incorporating the geometric information into higher abstraction levels.

In this section, robotic assembly planning will be investigated. It will be approached at the higher abstraction level, in a knowledge-enabled way. An assembly planner will be used that is based on formal specifications of the products to be created, the parts and sub-assemblages they are to be created from, and the mechanical connections to be formed between them. At this level, it is represented that a part must provide dispositions in order for it to be able to enter a particular connection, or to be grasped in a certain way, as well as that certain grasps and connections block certain dispositions.

The planning itself proceeds by comparing individuals in the knowledge base with their terminological model, finding missing facts, and producing action items to materialize these. For example, if the asserted type of an entity is `Car`, the robot can infer that, to be a `Car`, this entity must have some wheels attached, and if this is not the case, the planner will create action items to add them.

In Section 4.1, the various geometrically motivated constraints pertaining, for example, what grasps are available on a part depending on what mechanical connections it has to other parts, were modeled symbolically. To this end, axioms were added to the knowledge base that assert that a connection of a given type will block certain dispositions, thus preventing the connected part to enter certain

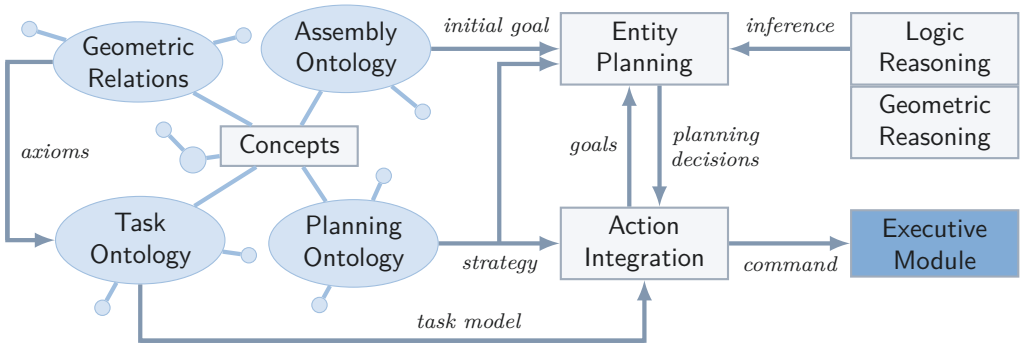

Figure 4.7: An architecture that combines different ontologies in a heterogeneous reasoning system to control robot decision making during assembly activities in cluttered workspaces.

other connections, and to be grasped in certain ways. It was further assumed that the workspace of the robot is sufficiently uncluttered such that abstract actions like *pick part* will succeed. The work presented in this section goes beyond these limitations by grounding geometrically-meaningful symbolic relations (that can hold between individuals in the knowledge base) through geometric reasoning that can perform collision and reachability checking, and sampling of good placements. The resulting architecture is depicted in Figure 4.7. Its main components are used for the configuration of entities, and to translate hypothesized objects into action commands that materialize them in the robot's world. To this end, the components are supplied with knowledge from different ontologies, and logic-based as well as geometric reasoning capabilities.

The contributions of this section are the following ones:

- An extension of the planning ontology presented in Section 4.1. The extension is concerned with the representation of spatial relations that are inferred on demand through geometric computation methods commonly available in integrated robot control systems, and with the representation of tasks and roles objects play during task execution.

- An extension of the planning method presented in Section 4.1. The extension is concerned with, first, switching between different planning strategies with different goal configurations, and, second, the integration of action preconditions and planning strategies for assembly tasks in cluttered scenes into the planning framework.

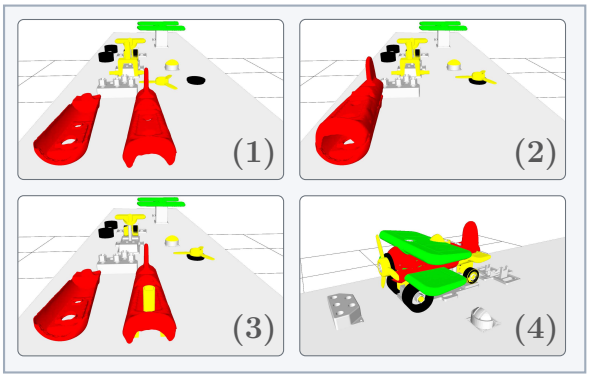

Figure 4.8: Initial workspaces of a toy plane assembly (1-3), and the goal state thereof (4).

4.2.1 Related Work

This section improves on work presented in Section 4.1. First, reasoning about action feasibility will be investigated by means of methods for geometric computation, and how such methods can be integrated into the planner. Second, a model of the tasks and roles objects play during task execution will be introduced, and flexible selection of planning strategies will be investigated. Compared to the previous work, the domain used by the planner has been extended for additional planning tasks beyond the ontology characterizing the assembled product. To this end, it will be demonstrated that the planner can also be used for task planning, i.e., for the purpose of object selection and role assignment.

Spatial relations are often considered in robotics research in the scope query-answering with spatial references. Deeken et al. (2018), for example, have used spatial database technology to store geometric information of the robot's environment. Qualitative spatial relations are then grounded through the evaluation of specialized operators of the database system. These relations are used to implement a query-answering system that can identify objects in the environment through spatial references. In this work, rather the reachability of objects is considered which requires additional computation beyond what spatial databases are designed for. The planner could nevertheless make use of spatial database technology, e.g., for planning arrangements of objects. The reachability of objects is often considered in integrated task and motion planning approaches. Akbari et al. (2015), for example, present an approach where ontological symbols are grounded in a combined task and motion planning framework.

4.2.2 Overview

Assembly tasks often have a recipe that, if followed correctly, would transform available parts into an assembled product. But inferring the sequence of assembly actions is not sufficient for robots because actions may not be performable in the current situation. For example, this is the case when the robot cannot reach an object because it is occluded. A situation that might even occur in rather organized spaces such as the ones displayed in Figure 4.8 if the robot has only limited mobility. A notion of space is further very complicated in a logic formalism, but specialized methods exist that efficiently can compute qualitative spatial relations such as if objects are occluding each other.

The proposed solution is depicted in Figure 4.7. It builds upon an existing planner and extend it with a notion of task, and geometric reasoning capabilities. Robot tasks are represented in terms of the task ontology which also defines action pre-conditions. The pre-conditions are ensured by running the planner for the task entity. This is used, for example, to ensure that the robot can reach an object, or else it would try to put away occluding objects. To this end, a geometric reasoner is integrated with the knowledge base. The geometric reasoner uses its own data structures and interfaces. These are hooked into the logic-based reasoning through procedural attachments in the knowledge base.

The planner (Section 4.1) is implemented as an open-source extension of the KnowRob knowledge processing system[1] (Tenorth and Beetz, 2013). KnowRob is a knowledge base for robots with OWL support. One useful aspect of KnowRob is that symbols can be grounded through the evaluation of computational procedures such as geometric reasoner.

The geometric reasoner used in this work is a module of the Kautham Project[2] (Rosell et al., 2014). It is a C++ based open-source tool for motion planning that enables to plan under geometric and kinodynamic constraints. It uses the *Open Motion Planning Library* (OMPL) (Sucan et al., 2012) as a core set of sampling-based planning algorithms. In this work, the RRT-Connect motion planner (Kuffner and LaValle, 2000) is used. For the computation of inverse kinematics, the approach developed by Zaplana et al. (2017) is used.

[1] `http://knowrob.org` (accessed 22 May 2022)
[2] `https://sir.upc.edu/projects/kautham` (accessed 22 May 2022)

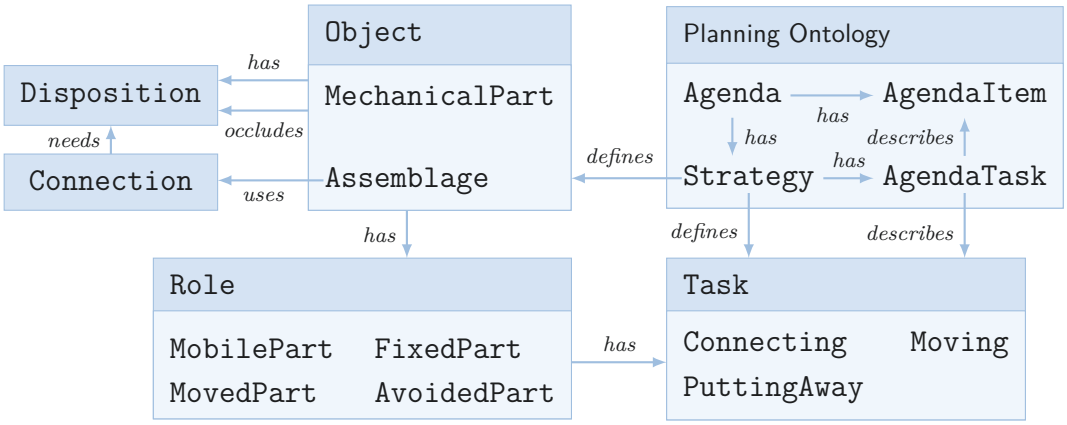

Figure 4.9: Concepts and relationships in the assembly domain.

4.2.3 Modeling Assembly Tasks

In the presented approach, the planner runs within the *perception-action* loop of a robot control system. The knowledge base maintains a belief state, and entities in it may be referred to in planning tasks. In Section 4.1, ontologies were defined to describe assemblages, their connections and which parts they use, and meta knowledge to control assembly activities. In the following sections, further additions to these ontologies that were implemented for this work will be presented. The interplay between the different ontologies used in the planning framework is depicted in Figure 4.9.

Assembly Ontology

The upper level of the assembly ontology defines general concepts such as `MechanicalPart` and `AssemblyConfiguration`. Underneath this level there are part ontologies that describe properties of parts such as what connections they may have, and what ways they can be grasped. Finally, assemblage ontologies describe what parts and connections may form an assemblage. This layered organization allows part ontologies to be reused for different assemblies. Also important is the `Disposition` concept. Mechanical parts provide dispositions, which are required (and possibly blocked) by grasps and connections. Apart from these very abstract concepts, some common types of dispositions and connections are also defined (e.g., screwing, sliding, and snapping connections).

To these, a new relation *occludesDisposition* is added with domain `AtomicPart` and range `Disposition`. A triple p *occludesDisposition* a means the atomic part p is placed in such a way that it prevents the robot from moving one of its end effectors to the disposition a (belonging to some other part q). Parts can be said to be graspable if they have at least one non-occluded grasping disposition. The motivation for the addition of this property is that it helps representing spatial constraints in the workspace, a consideration that was not addressed in Section 4.1.

Also, in the previous section, the belief state of the robot was stored entirely in the knowledge base. It includes object poses, if and how an object is grasped, mechanical connections between objects, etc. Consistency is easier to maintain for a centralized belief state, but components of the robot control system need to be tightly integrated with the knowledge base for this to work. In the previous section, this could be enforced as both perception and executive components of the system were developed in the same research group. For this secction, however, an external motion planner must be integrated that stores its own representation of the robot workspace, and uses its own naming convention for the objects. To this end, a data property `planningSceneIndex` is used to help relate object identifiers with Kautham planning scene objects.

Task Ontology

At some point during the planning process, the robot has to move its body to perform an action. In Section 4.1, action data structures were used which were passed to the plan executive. The plan executive had to take care that pre-conditions were met, which sub-actions to perform, etc. In this work, explicit task representations are used to ensure that some of the pre-conditions are met before performing an action. The task ontology includes relations to describe objects and locations involved via their roles, sub-tasks organized in plans, etc. Here, the focus lies on the representation of pre-conditions.

The modeling of action pre-conditions is based on the roles that objects play during an action. The idea is that knowing an appropriate filler for each required role in a task is a pre-condition for performing the action. Roles are characterized as concepts that classify objects during the execution of a task. The SOMA

ontology (Section 3.1) includes a taxonomy of general roles including, for example, `Patient`, i.e., the role of objects primarily affected by the event, and `Tool`, i.e., the role of objects used to modify or actuate other objects. These can be referred to in task definitions to assert axioms about involved entities. This allows, for example, to state that the robot needs a tool for fixing a part with a screw, and that this tool needs to be a `ScrewDriver`. This example can be represented using a constraint such as $\forall HAS_R^{-}.\texttt{ScrewDriver}$, which can be asserted for the `Tool` role of a `FixingAScrew` task where HAS_R (*hasRole*) denotes the role of an object.

ConnectingParts The most essential action in the assembly domain is to connect parts with each other. At least one of the parts must be held by the robot and moved in a way that establishes the connection. Performing the action would not be directly possible in case the part to be moved cannot be grasped. This is the case when a part is positioned in a way that blocks a required disposition, for example, due to being in the wrong holder, blocked by another part, etc.

First of all, the role `AssembledPart` (AP) of parts in `ConnetingPart` (CP) actions can be defined as $\texttt{AP} \sqsubseteq \texttt{Patient}$. Furthermore, mobile parts (MP) and fixed parts (FP) can be distinguished through the roles $\texttt{MP} \sqsubseteq \texttt{AP}$ and $\texttt{FP} \sqsubseteq \texttt{AP}$. These roles must be assigned to various objects that participate in a connection. Following the affordance modeling presented in Section 3.2, it can be stated that `AssemblyConfigurations` (AC) *define* the task afforded by the configuration, which is denoted by DEF_{Tsk}. Each role is further associated to the task via the HAS_{Tsk} relation. Hence, the task can be defined using the following axioms:

$$\texttt{CP} \sqsubseteq = 1 DEF_{Tsk}^{-}.\texttt{AC} \tag{4.17}$$

$$\texttt{CP} \sqsubseteq \geq 2 HAS_{Tsk}^{-}.\texttt{AP} \tag{4.18}$$

$$\texttt{CP} \sqsubseteq \geq 1 HAS_{Tsk}^{-}.\texttt{MP} \tag{4.19}$$

$$\texttt{CP} \sqsubseteq \leq 2 HAS_{Tsk}^{-}.\texttt{MP} \tag{4.20}$$

These axioms define that `ConnetingPart` tasks are defined by exactly one assembly configuration (4.17); at least two parts are involved in a connection (4.18); at least one of these parts must be mobile (4.19); and at max two parts can be mobile (4.20) – which is a robot specific statement used for evaluation experiments.

Through these definitions, agenda items will be created for the relation symbols appearing in the cardinality constraints of the axioms, i.e., for the roles of the task, and the configuration that defines it. However, a remaining problem is that the axioms do not ensure that the dispositions used in the connection belong to the same objects that are classified by the roles of the task. This can be fixed by adding the following property chain axiom:

$$HAS^{-}_{Tsk} \equiv DEF^{-}_{Tsk} \circ HAS_{AP} \circ HAS_{R} \tag{4.21}$$

The relation symbol HAS_{AP} (*hasAtomicPart*) denotes the parts linked in an assembly connection. The axiom ensures that HAS^{-}_{Tsk} only denotes roles of parts that are required by the connection. However, this definition cannot be used in axioms defining tasks as a property is not *simple* if defined by a property chain. Thus, this identity constraint must be enforced outside of DL reasoning.

Another pre-condition is that the robot needs to be able to grasp any of the involved mobile parts. Parts may have `GraspingDispositions` (GD) that indicate how the robot should position its gripper, how much force to apply, etc. to grasp the part. The following axioms can be asserted that express each mobile part must offer at least one unblocked `GD`:

$$\texttt{FD} \equiv \texttt{D} \sqcap (= 0 BLOCKS^{-}_{D}) \tag{4.22}$$

$$\texttt{MP} \sqsubseteq \forall HAS^{-}_{R}.(\exists HAS_{D}.(\texttt{GD} \sqcap \texttt{FD})) \tag{4.23}$$

The concept `FD` (read: free disposition) is the class of dispositions that are not blocked (4.22); and a mobile part has a disposition which is both a grasping disposition and a free disposition (4.23).

Furthermore, the *hasLinkWith* (HAS_{LW}) property is used to relate two parts with each other in case there is a connection between them. Objects are viewed as being directly linked if they are both an atomic part of the same assemblage. This can be expressed via the property chain:

$$HAS_{LW} \equiv HAS^{-}_{AP} \circ HAS_{AP} \tag{4.24}$$

Indirectly linked parts are connected via the transitive closure of this relation.

Next, the HAS_{LW} relation is used to characterize fixed parts as parts that are connected to a fixture. This can be written as:

$$\texttt{FP} \sqsubseteq \forall HAS_R^-.(\exists HAS_D.(\exists NEEDS_D^-.(\exists NEEDS_D.(\forall HAS_D^-.\texttt{Fixture})))) \tag{4.25}$$

The axioms defining mobile and fixed parts can then be used to implement the selection heuristic described in Section 4.1.5.

Also, it should be ensured that the part in question is in the correct fixture. It could be the case that a fixture blocks a required disposition. The part should be moved into another fixture that exposes the required disposition in such a case. To ensure this, it can be asserted that all dispositions used in a connection must be unblocked:

$$\texttt{CP} \sqsubseteq \forall DEF_{Tsk}^-.(\forall USES_D.\texttt{FD}) \tag{4.26}$$

Finally, it should be ensured that each part involved must be non-occluded, i.e., each part must be reachable by the robot. This can be written as:

$$\texttt{CP} \sqsubseteq \forall DEF_{Tsk}^-.(=0\, USES_D.(\exists OCCLUDES_D^-)) \tag{4.27}$$

MovingPart and PutAwayPart The above statements refer to entities in the belief state and may require certain actions to be performed to destroy or create relations between them. In this work, the focus lies on ensuring valid spatial arrangement in the scene.

First, the robot should break non-permanent connections in case one of the required dispositions is blocked by some other connection. This type of action is conceptualized as `MovingPart` tasks. The only patient of such a task is the part itself. It has the role $\texttt{MovedPart} \sqsubseteq \texttt{Patient}$ in the scope of a task execution. Furthermore, it must be ensured that the part has least one unblocked grasping disposition. This can be written analogously to Axiom (4.23).

Further, the robot should put away parts that occlude required parts. Such a `PutAwayPart` task needs exactly one `MovedPart`, and, additionally, refers to the parts that should be avoided – i.e., the ones having the `AvoidedPart` role.

Describing possible target positions in detail would be extremely difficult in a logical formalism, and is not considered in the scope of this work.

Planning Ontology

The planner presented here is controlled by meta knowledge called *planning strategy*. The planning strategy determines which parts of the ontology are of interest in the current phase, how steps are ordered, and how they are performed in terms of how the knowledge base is to be manipulated. Possible planning decisions are represented in form of an agenda. Planning agendas are ordered sequences of steps that each, when performed, modify the belief state of the robot in some way. The planner succeeds if the belief state is a complete materialization of the goal description, i.e., if all necessary facts are explicitly represented.

Different tasks require different strategies that focus on different parts of the ontology, and that have specialized rules for performing the agenda items. The strategy for planning an assemblage, for example, focuses on relations defined in the assembly ontology. Planning to put away some part such that other parts are accessible, in contrast, is only concerned with spatial relations and may ignore assembly relations entirely. In previous work, the strategy selection was done externally. Here, strategies are associated to entities that can be planned with them through axioms involving the *defines* (*DEF*) property. Strategies assert a universal restriction on this relation in order to define what type of entities can be planned with them. For the assemblage planning strategy (APS), for example, this type restriction can be written as:

$$\texttt{APS} \sqsubseteq \forall DEF.(\texttt{Assemblage} \sqcup \texttt{AssemblyConfiguration}) \tag{4.28}$$

In previous work, the notion of `AgendaTask` was used to relate a class of agenda items to a task used for the selection of individuals. Planning strategies may consist of multiple of such relations, linked to the strategy via the partonomy relation *hasComponent*. Hence, for a given agenda item and strategy, different tasks can be obtained by testing which of the `AgendaTask` relations holds true for the agenda item in question. Furthermore, assemblages are constructed by the execution of physical tasks which was represented via the notion of `EntityTask`. Here, `AgendaTasks` are further used to relate classes of agenda items to physical tasks that must be executed when the item is selected, e.g., a `PutAwayPart` task must be executed if the item is about removing an *occludesDisposition* assertion.

Tasks are represented independently from how they can be executed. To this end, they can be associated to `Plans` and `Methods`. Here, computational methods are considered, and conceptualized within planning strategies as `AgendaMethods`. An `AgendaMethod` defines a task by means of evaluation: when the method has been evaluated successfully, it means that the corresponding task has been executed. `AgendaMethods` are linked to the strategy via the relation *hasComponent*. They are activated based on whether the task in question belongs to the class of tasks defined by the method, and evaluated with the item as a parameter.

4.2.4 Reasoning Methods

In the presented approach, different reasoning resources and representations are fused into one coherent picture that covers different aspects of the world. In this section, the two different reasoning methods used by this system will be described: knowledge-based reasoning and geometric reasoning.

Knowledge-based Reasoning

In this work, knowledge-based reasoning refers primarily to checking whether an individual obeys the restrictions imposed on the classes to which it is claimed to belong, identifying an individual based on its relations to others, and identifying a set of individuals linked by certain properties – as done when identifying which parts have been linked, directly or indirectly, via connections. This is done by querying the knowledge base to check whether appropriate triples have been asserted to it or can be inferred, e.g., a property holds for an object because a property chain defining that property holds for the object.

KnowRob, however, allows more underlying mechanisms for its reasoning. In particular, decision procedures, which can be arbitrary programs, can be linked to properties. In that case, querying whether an object property holds between two individuals is not a matter of simply testing whether the appropriate triples have been asserted to the database. Rather, the decision procedure is called, and its result indicates whether the property holds or not. Such properties, with decision procedures attached to them, are referred to as computables, and they offer a way to bring together different reasoning mechanisms into a unified framework of knowledge representation and reasoning.

For this work, the geometric reasoner of the Kautham Project (Rosell et al., 2014) is integrated with a knowledge base. The reasoner is called to find occluding parts for a given disposition, and to compute the `occludesDisposition` property.

Geometric Reasoning

The main role of geometric reasoning is to evaluate geometric conditions of symbolic actions. Two main geometric reasoning processes are provided.

A robot can transit to a pose if it has a valid goal configuration. This is inferred by calling an *Inverse Kinematic* (IK) module for a candidate target pose and evaluating whether the IK solution is collision-free. The first found collision-free IK solution is returned, and, if any, the associated pose. Failure may occur if either no IK solution exists or if no collision-free IK solution exists.

Furthermore, placements of objects within a given region must be determined. For an object, a pose is sampled that lies in the surface region, and it is checked for collisions with other objects, and whether there is enough space to place the object. If the sampled pose is not feasible, another sample will be tried. If all attempted samples are infeasible, the reasoner reports a failure, which can be due to a collision with the objects, or because there is not enough space for the object.

4.2.5 Planner Extensions

The planner needs to be extended for dynamically computed relations, and also for being able to generate sub-plans in case some pre-conditions of actions the robot needs to perform are not met. The changes made for this section will be explained in the following.

Selection of Planning Strategies

The planner is driven by finding differences between a designated goal state and the belief state. The goal is the classification of an entity as a particular assemblage concept. The initial goal state is part of the meta knowledge supplied to the planner (i.e., knowledge that controls the planning process). Strategies further declare meta knowledge about prioritization of actions, and allow ignoring certain relations entirely during a particular planning phase.

Strategies are useful because it is often hard to formalize a complete planning domain in a coherent way. One way to approach such problems is decomposition: planning problems may be separated into different phases that have different planning goals, and that have a low degree of interrelations.

The overall goal of the planner is the materialization of an entity, i.e., the explicit representation of facts that must exist according to type assertions about the entity. Here, it is considered that each of the planned entities may use its own planning strategy. The strategy for a planning task is selected based on universal restrictions involving the *defines* relation. The selection procedure iterates over all known strategies and checks for each whether the planned entity is a consistent value for the *defines* property. Only the first matching strategy is selected.

Activating a strategy while another is already active pauses the former until the sub-plan was run successfully. In case the sub-plan fails, the parent plan also fails if no other way to achieve the sub-plan goal is known. It is expected that the meta-knowledge that controls the planner ensures to some extent that the planner does not end up in a bad state where it loops between sequences of decisions that revert each other. In case this happens, the planner will detect the loop and fail. The agenda of re-started plans is updated to account for sub-plans changing planning relevant information unexpectedly.

Integration with Task Executive

Assembly actions can be performed whenever an assemblage is entirely specified. This is the case if the assemblage materializes all asserted restrictions including the connection it must use, and the sub-assemblies it must link. Further action commands must be generated if a part of interest cannot be grasped because another part is occluding it. To this end, the planning loop is extended such that it relates agenda items to physical tasks, and methods to execute them.

In each step of the planning loop, the agenda item $x \in N_O$ with highest priority is selected from the agenda. First, the item x is either deleted if it was completed already, or skipped if the materialization of the item would cause an inconsistency. Each uncompleted item has an associated constraint that is not materialized in the belief state of the robot. Next, the planner specifies the set of facts $\mathcal{A}_x$ that materialize this constraint by using the `specify` function. That is,

for example, which part should be used in a configuration. This step is followed by the projection into the knowledge base where facts are asserted or retracted. Finally, new items are added to the agenda based on the assertions that have been added to the knowledge base.

For this work, an additional function is used to execute a physical task based on the agenda item selected. The function, named `realizeItem`, is called just after the `specify` function, and receives as input the strategy, the item x, and the corresponding set of facts $\mathcal{A}_x$. For the identification of possible tasks, the notion of `AgendaTask` is used as described in Section 4.1.4, i.e., the task described by an `AgendaTask` relation can be used for an item if the item can be described by the relation. A strategy further can have `AgendaMethod` components. Let $\mathcal{S}_M \equiv M_1 \sqcup \cdots \sqcup M_n$ be the concept defining these components for a strategy. Each M_i is defined by a subsumption axiom of the form $M_i \sqsubseteq \forall DEF.C_i$ where C_i is a concept defining the task associated to the method. Accordingly, a task defined by concept T can be executed by a method M_i if $\neg(M_i \sqcap \exists DEF.T \sqsubseteq \bot)$.

The `realizeItem` function involves that a fresh task individual is created. The task individual is an instance of the concept described by the `AgendaTask` relation that holds for the selected item. It is usually not fully materialized at this point. To this end, a strategy is selected for the task, and the planning procedure is recursively called to materialize the task before it is executed. The strategy is defined such that the agenda for the task is restricted to the materialization of roles, and the selection of individuals for the roles. Finally, an execution method is selected and evaluated for the task. In the case that the execution fails, the agenda item is postponed. Note that loops may occur in case all remaining agenda items fail, and that these should be detected in the planning loop.

Planning with Computable Relations

In the reference implementation of the proposed architecture, the *occludesDisposition* property is computed within the `realizeEntity` function. Within the function, materialized but yet unrealized assemblages are identified, i.e., the ones that specify all linked parts, and that have not been build yet. The property is updated for each of the dispositions that are used by these assemblages before the task to realize one of them is planned and executed.

4.2.6 Evaluation

The proposed extensions of the planning framework will be evaluated along the following dimensions: the variations of spatial configurations that can be handled, and the types of queries that can be answered. The planning domain for evaluation is the same as used in Section 4.1, i.e., a part of the YCB Object and Model Set (Çalli et al., 2015). It uses slide in connections for the parts, and bolts for fixing the parts afterwards. However, a different robot is used (a YuMi robot). It is simulated in a kinematics simulator and visualized in RViz.

Simulation The planning system was tested with different initial spatial configurations, depicted in Figure 4.8. The first scene has no occlusions. In the second, the upper part of the plane's body is occluding the lower part, and the propeller is occluding the motor grill. Finally, in the third, the chassis is not connected to the holder, and occluded by the upper part of the plane body. Collision checking between the airplane parts was disabled to avoid spurious collisions being found at the goal configurations (the connections fit snugly). Geometric reasoning about occlusions allows the robot to know when it needs to move parts out of the way and change the initial action sequence provided by the OWL planner.

Querying The proposed extensions include geometric relations, pre-conditions of actions, and the selection of tasks that correspond to planning decisions. The additional querying capabilities will be demonstrated in the following.

The geometric reasoner computes the *occludesDisposition* relation whenever an assemblage is to be build, and facts in the knowledge base are updated accordingly. It is further needed to define completeness axioms representing that no occlusions exists other than the ones detected by the geometric computation. For example, let us consider an instance of the `PlaneTopWingSlideInF` disposition named `SlideInF1` which is occluded by some part named `Part1`:

Listing 4.4: An individual disposition occluded by an object

```
Individual: SlideInF1
Types:
  isOccludedBy exactly 1 Thing
Facts:
  isOccludedBy Part1
```

Let us further assume that `SlideInF1` is a disposition of the object `PlaneBody1`, and that this object has been selected by the planner to specify a configuration of type `TopWingSlideInBody`. Configurations of this type consume exactly one `PlaneTopWingSlideInF` disposition. However, the robot can only realize such a configuration if the disposition is non-occluded. This can be ensured by defining the following DL concept:

Listing 4.5: A named concept to query whether an object is non-occluded for an intended connection

```
Class: NonOccludedFor_PlaneBody1_TopWingSlideInBody
EquivalentTo:
  TopWingSlideInBody and
  (needsDisposition only (not (isOccludedBy some Thing))) and
  (needsDisposition some (isDispositionOf value PlaneBody1))
```

Running a DL subsumption query on the assembly ontology, augmented with the definition of the concept listed above, produces that this concept is subsumed by `Nothing`, i.e., `PlaneBody1` has a disposition that is blocked and also required in an intended `TopWingSlideInBody` configuration.

The *isOccludedBy* relation is further used to express that objects playing certain roles in tasks must be non-occluded. Planning is initiated by asserting an instance of a `Task` sub-concept. Initially, not all facts that must exist according to the subsumption hierarchy are explicitly asserted. In particular, the objects and their roles during the task must be determined. The planner treats instances of tasks exactly the same as instances of assemblages, and determines the set of operations that must be performed before executing the task by an evaluation of the μ function for each type explicitly asserted for the task individual. For example, assume that the disposition `SlideInF1` was selected to play a role in a task, and that this task requires the disposition to be non-occluded. The set computed by the μ function consists in this case of an item (`Part1`-$\exists$*occludesDisposition*.`SlideIn1`), meaning that the part *Part1* should be moved such that it does not occlude the disposition *SlideIn1*.

Such agenda items of the planner are associated to tasks that the robot can execute to establish the planning decision represented by the item. For example, agenda items about the *occludesDisposition* relation can be established by actions of type `PutAwayPart`. This relation between agenda item and task is represented

as part of the planning strategy. For the example of the `SlideInF1` disposition being occluded by `Part1`, the agenda item is represented as:

Listing 4.6: An example named individual of an agenda item

```
Individual: Item21
Types:
  DetachItem
  isAboutType only (occludesDisposition value SlideIn1)
Facts:
  isAboutIndividual Part1
```

The relation between agenda items and tasks is represented through the `AgendaTask` concept. For the previous example, it can be represented as:

Listing 4.7: A named concept of an agenda task

```
Class: AgendaTask_occludes_PutAway
SubClassOf:
  describesItem only (isAboutRelation only (
                      occludesDisposition some Thing))
  describesTask only PutAwayPart
```

Finally, a concept used to query whether an agenda item can be established by the execution of a task can be defined. For example, it can be written as:

Listing 4.8: A named concept used to query a relation between agenda items and tasks

```
Class: IsTaskFor_PutAway_Item21
EquivalentTo:
  AgendaTask_occludes_PutAway and (describesItem value Item21)
```

Running a DL subsumption query produces that this concept is not subsumed by `Nothing`, i.e., `Item21` can be established by actions of type `PutAway`.

4.2.7 Summary

In this section, it was investigated how geometric reasoning can be incorporated into logic-based assembly planning through decision procedures that are attached to relation symbols in an ontology. Such relations are referred to in task definitions to make assertions about what should hold true for parts involved in the execution of the task. It was demonstrated that this planning framework enables the robot to handle workspace configurations with occlusions between parts, to reason about them, and to plan sub-activities required to achieve its goals.

4.3 Conclusion

In this chapter, the planning problem for assembly tasks has been investigated through ontological modeling and reasoning. The goal state of the plan and the belief state of the robot are both conceptualized through an ontology. The ontological characterization of assemblages has been based on the SOMA ontology (Section 3.1), and its model of affordances and dispositions (Section 3.2).

The method used for planning compares the belief and goal state, and identifies differences between them. Each difference represents a missing fact, or one that is in conflict with the goal state. The set of differences is organized in a priority queue that is sorted and processed based on control knowledge. The method succeeds when the belief state was transformed such that it is in accordance with the goal state, i.e., when all necessary facts have been materialized in the knowledge base. Certain sub-problems such as reasoning about action feasibility under geometric constraints are not feasible with DL-level reasoning. But, e.g., geometric relation symbols can be grounded using common procedures which has also been demonstrated. However, some relevant issues remain unconsidered in the scope of this chapter. First of all, only a simulated robot has been used for the experiments, and the system should be employed on a real robot in the future. Furthermore, some of the manual steps could be automated. For example, some parts of the terminological model can be auto-generated from existing documents such as technical drawings or CAD models.

Unlike general planning, most of the reasoning involved in the proposed planning method is within the reach of efficient DL reasoning. Product definitions can be altered by editing the ontology, and it is possible to use DL reasoning to check that the new definitions are logically consistent. This allows, for example, to develop a web-application in which users can configure the product within the boundaries of the ontology. In an ideal case, these definitions would be handed directly to the assembly robot without any human interaction. The approach presented here thus displays a step towards more versatile and reconfigurable assembly cells compared to the status quo in industrial assembly. Such flexibility in robot manufacturing could allow customers in the future to individualize products without the cost implication customization has nowadays.

5 Ontology-enabled Inspection of Robot Behavior

A lot of the knowledge we humans employ to master activities in our everyday lives is learned from previous experiences that are stored in our episodic memory. Episodic memory allows us to remember detailed experiences including sensations and a story of what happened, and it serves as a kind of repository from which more abstract knowledge can be learned. One of the problems that needs to be tackled to create a memory system which is similar to the episodic memory of humans is how experiences should be represented to realize a similar capability. In this chapter, it is considered that such experiences consist of experience data, and an ontological characterization thereof. The hypothesis is that a linked ontological characterization of robot experiences is an integrative framework for query answering to support flexible execution and monitoring of tasks, and querying and inspection of experiences using suitable abstractions.

The querying of experiences that belong to a common category is also important in the context of machine learning. Several authors have employed machine learning techniques in the robotics domain, e.g., to learn flight behavior through experimentation (Kim et al., 2004), and to predict grasping points on novel objects given a set of objects with known grasping points (Saxena et al., 2008). However, the data sets are, in such cases, usually designed with a particular learning task in mind, and thus can only be used for a rather limited range of learning problems. In this work, a more comprehensive representation is considered that is less restricted in what learning tasks can be considered, but it requires more sophisticated querying to compute data sets suitable for particular learning tasks.

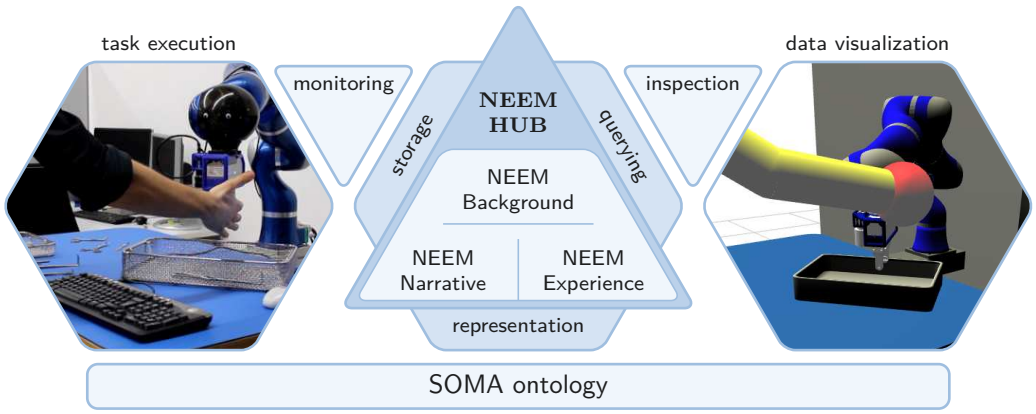

Figure 5.1: An architecture for robot behavior inspection that employs the SOMA ontology to encode symbolic activity data which can be selectively queried and visualized for inspection purposes.

The ontological characterization of experiences considered in this work has four levels of granularity: situations, actions, motion phases, and force dynamical events. A situation includes several relevant entities that coincide in an experience or part of it, e.g., several actions performed towards a common goal. An action, on the other hand, is executed towards a more immediate goal that is thought to be achieved when the action was executed successfully. Here, it is considered that actions are executed through temporally related phases of body motions that cause distinct patterns of interactions with the environment. This modeling decision is inspired by the observation that humans use such interactions as control points during object manipulation tasks (Flanagan et al., 2006).

It is further important how particular experiences can be created, and accessed from memory through querying. In this chapter, as displayed in Figure 5.1, the investigation is focused on the use of a linked ontological characterization as an integrative framework supporting these tasks. Generally speaking, robot experiences are created by monitoring how a robot executes a given task. This is a rather trivial problem if the organization of the robot control program corresponds sufficiently with the modeling of actions such that the activation and completion of procedures in the control program can be directly translated into an instance of the action model. In the following, such instance data is represented as an OWL DL ABox ontology based on the SOMA ontology (Chapter 3). The monitored instance data is stored on a dedicated server that also provides a querying interface to retrieve experiences that belong to the category defined by the query.

The suitability of this representation will be demonstrated by means of a case study where a robot performs service tasks in a realistic environment with humans in the scene while its performance is monitored and stored. To this end, it will be demonstrated that a set of relevant competency questions can be implemented based on the ontological representation, and that statistics can be computed to help a human operator performing an inspection of the robot behavior. The case study further validates the SOMA ontology for the representation of qualitative experience data. Please note that the case study was first published by Bartels et al. (2019). The authors provide a more complete technical description of the robot control system used in the case study. Here, the focus is on the use of ontologies for task execution, monitoring and querying.

The remainder of this chapter is organized along the different components depicted in Figure 5.1. First, the representation of robot experiences will be introduced, and the modeling of the particular domain investigated in the case study (Section 5.2). Second, the task execution in the case study will be discussed, and how the experience data is acquired and stored (Section 5.3). Third, a query language will be proposed that is used to retrieve particular experiences given abstract descriptions of situations (Section 5.4). Finally, a more detailed assessment of the robot behavior in the case study will be provided (Section 5.5).

5.1 Related Work

A lot of the research that considers memory of robot experiences rather uses ad-hoc representations, e.g., for reinforcement learning of navigation behavior (Kim et al., 2004). However, such restricted representations are too specific to be considered as useful for a more general memory of experiences. This has also been noted, e.g., by Fourie et al. (2017) who suggest the use of a graph-database with a spatio-temporal search index, and with links into a key-value store where larger amounts of experience data are stored. However, no ontological modeling of experiences was considered by the authors. Ontological modeling of robot experiences has been considered by Tenorth et al. (2013a) in the RoboEarth project, and in the follow-up project openEASE (Beetz et al., 2015c). In both cases, experiences are modeled ontologically based on the Cyc foundational ontology (Tenorth and Beetz,

2013), as opposed to the DOLCE ontology (Masolo et al., 2003) used here, which has a stronger cognitive basis. The consequences of the resulting foundational commitments are discussed in more detail in Section 3.1. Another related work that makes use of formal ontologies is presented by Breux et al. (2018). The authors present a framework for the representation of robot experiences that has three layers: sensor data, instance data, and formalized concepts. Instances are classified by concepts, and linked to segments of the sensor data. In contrast to this work, their knowledge modeling concentrates on environmental characteristics, and is based upon a segment of the WordNet lexical resource (Miller, 1995).

Furthermore, this work is concerned with a knowledge base for safety-aware robots that encodes the activities in *Physical Human-Robot Interaction* (pHRI) scenarios. This knowledge base allows the robot and its human developers to analyze the interactions in terms of safety events such as intrusions. A related robot knowledge base for HRI is presented by Lemaignan et al. (2010). However, that system focuses on dialog grounding, and neither considers physical interactions nor safety aspects. More recently, Umbrico et al. (2020) introduced the SOHO ontology which considers safety aspects, but no physical interactions.

5.2 Representation of Experience Knowledge

We humans have the cognitive capability to recall memorized situations, and to re-experience them to some extend. We can, for example, reconstruct a scene we once saw in our *mind's eye*. The memory mechanism that allows us to recall these detailed pieces of information from abstract descriptions is our episodic memory. Analogously, it can be understood as a video that the agent makes of an ongoing activity coupled with a detailed story about the actions, motions, their purposes, effects, the behavior they generate, images that are captured, etc. It can further serve as a repository from which we learn general knowledge by finding correlations among different instances of performing the same activity under varying circumstances. In the context of robot programming, the episodic memory can be integrated deeply into the knowledge acquisition, representation, and processing system: whenever a robotic agent performs, observes, prospects, and reads about an activity, it can memorize this experience. These memories

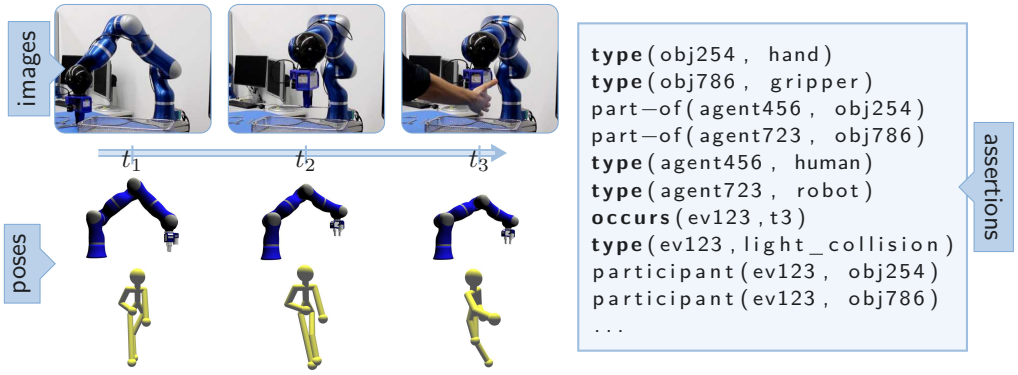

Figure 5.2: Experience knowledge about a safety aware robot with images captured and poses recorded over time (left), and symbolic assertions that describe the activity (right).

can be accessed by a human operator to inspect the behavior of the robot, or be leveraged for machine learning applications.

In this work, experience knowledge is represented in form of *Narrative-Enabled Episodic Memories* (NEEMs) (Beetz et al., 2018). A NEEM consists of *NEEM experience*, *NEEM narrative*, and *NEEM background*. The NEEM experience is a detailed, low-level, time-indexed recording of a certain episode. It contains records of poses, percepts, control signals, etc. NEEM experiences are linked to NEEM narratives, which can be seen as stories that provide more abstract descriptions of what is happening in an episode. The link is established in part through temporalized relations in the knowledge base that are defined for particular points in time through queries accessing the experience data. NEEM narratives contain information regarding the tasks, the context, intended goals, observed effects, etc., and they are conceptualized within the ontological framework that was discussed in Chapter 3. They are further enriched through static environment knowledge by the NEEM background. More technical details about representation and use of NEEMs can be found in the NEEM-handbook (Beetz et al., 2020).

An example of the information contained in a NEEM is illustrated in Figure 5.2. In this episode, a robot was performing pick and place tasks while a human was being monitored who physically interacted with the robot. The depicted timeline has marks for some time instants at which the robot experienced an event. Images that were captured at these time instants are shown above, and the poses of the robot and human are shown below of the timeline. The robot picks up a surgical

instrument at t_1, drops it into a basket at t_2, and it is in physical contact with the human at t_3. Some of the corresponding assertions in the knowledge base are shown in the right part of Figure 5.2. These assertions represent, for example, that at t_3 an event ev_{123} occurred, that this event was classified as a light collision, that the objects obj_{254} and obj_{789} are participants in that event, and that these objects are classified as hand of the human and gripper of the robot respectively.

5.2.1 Ontological Characterization of Experience Data

Experience data represents the extrinsic and intrinsic sensory experience of an activity. It includes sensations of the robot's environment, and control signals representing its internal state during the activity. Sensory experiences are stored in a separate database, and linked to entities in the knowledge base either explicitly via a field in the database record, or implicitly via the time of measurement which is also part of the record. Each sensory measurement represents a particular value of an object quality. The qualities of an object can roughly be seen as the way the object manifests its characteristics. As humans we exploit such qualitative characteristics to categorize objects into similarity classes.

There are different ways how object qualities can be characterized ontologically. One way is to represent qualities as data properties. For example, let $x \in N_O$ be a joint of the robot with a sensor measuring the external torque exerted on the joint. Further, let $v_t \in \mathbb{R}$ be the value measured by the sensor at time t, and $hasValue \in N_R$ a data property for representing the sensor value in the knowledge base. Hence, a fact $hasValue_t(x, v_t)$ indicates that v_t was measured as the torque exerted on x at time t (note that *hasValue* is considered as a temporalized relation in the experience data). However, this approach is not suitable in cases where the quality itself is within the domain of discourse. Accordingly, qualities are often considered as a fundamental category in the ontological modeling. Here, the view of Masolo and Borgo (2005) is taken where qualities are inherited in other entities, and associated to a, so called, *quale* which can be given a value (quantitative or qualitative) within a certain space.

The value of a quale is a (not necessary atomic) region within the corresponding quality space. Quality spaces can further be clustered into regions that are used to categorize entities. A common example is that of the HSL color space where

regions of points in the space represent certain named colors. In the following, the space of external torque measurements is considered. For simplicity, it is assumed that the force is measured as a single value between zero indicating no torque, and one indicating maximum torque. Accordingly, the space can be called *normalized scalar space* (*NSS*), and modeled ontologically as a (maximal) `Region` of the `ExternalTorque` (*ET*) quality of joints. The quantitative value measured by the sensor of a joint is then represented as a temporalized data property of the individual region of the joint's external torque quality.

$$NSS \sqsubseteq \forall hasValue_t.\texttt{double}[\geq 0, \leq 1] \tag{5.1}$$

$$ET \sqsubseteq \exists hasRegion.(NSS \sqcap \exists hasValue_t) \tag{5.2}$$

Consequently, it can be stated that the *NSS* region never has a value larger than one or smaller than zero (5.1), and the external torque quality always has a value within the *NSS* region (5.1). Note that t appears unquantified in the formula. It is assumed to be quantified to the current point in time during task execution, and, when querying past experiences, to a time point quantified in the query.

Finally, the space can further be segmented into meaningful regions. In this case, the measurements are classified into different types of collision. For example, the region R_{LC} of light collisions of a joint is defined as:

$$R_{LC} \equiv \exists hasRegion^{-}.ET \sqcap \exists hasValue_t.\texttt{double}[\geq 0.1, \leq 0.15] \tag{5.3}$$

Thus, a measurement between 10% and 15% of the maximum exerted force is interpreted as a light collision, and the region of the external torque quality is classified accordingly for time intervals where the measurement is within this range. In the presented case study, thresholds are defined for four collision types: *severe collision*, *strong collision*, *light collision*, and *contact*. These collisions occur whenever the measured external torque of any joint exceeds 5%, 10%, 15%, and 30% of the maximum torque, respectively. Note that this is a case where the classification of regions is within the range of an OWL DL reasoner, however, in other cases the classification might need to be carried out through other means.

Furthermore, physical objects have a location and orientation within a frame of reference. The position of an object is stored using the parent in the kinematic

chain as a frame of reference, and, for the root of the chain, a global reference frame w is used. In the following, the location is seen as a vector $p \in \mathbb{R}^3$, and the orientation as a quaternion vector $q \in \mathbb{R}^4$. Thus, the pose of an object x at a time point t within a reference frame y is seen as a tuple of the form (p, q) which is computed as a function of time $T_x^y(t)$. In the following, it is assumed that poses can be retrieved from the experience memory with arbitrary frames of references and points in time using a spatio-temporal search index. This requires that the frames appear in the data, that they are connected via a chain of poses, and that the position data is logged at a sufficient frequency such that the two records closest to the given time point can be interpolated without loosing much accuracy.

Each body part of the robot further has an associated safety distance, and it is considered as insecure if the robot continues operation when a human body part is located within that region. In the following, the region defined by the safety distance is called the *safety space* of the robot (RSS), it is defined as $RSS \equiv RSS_1 \sqcup \cdots \sqcup RSS_n$ where each RSS_i is the safety region of a particular body part of the robot. Let $h \in N_O$ be the name of an individual that represents the head of the robot, and $d_h \in \mathbb{R}$ the safety distance associated to it. Further, let $hasDistance^h \in N_R$ be a temporalized relation that assigns to objects their distance to h. Then, the corresponding safety region RSS_h can be defined as:

$$RSS_h \equiv \exists hasDistance_t^h.\texttt{double}[\leq d_h] \tag{5.4}$$

The distance relation for an object x is simply evaluated as the length of the position vector in the reference frame of the head. Thus, $hasDistance_t^h(x) = |p|$ where $(p, q) = T_x^h(t)$.

5.2.2 Ontological Representation of Actions

The model proposed by Flanagan et al. (2006) forms the basis of the NEEM narrative representation. The model structures actions into motion phases, where the phases have subgoals, which are force dynamic events that generate distinctive sensory feedback. The role of force interactions between entities is prominently put forward in cognitive linguistics by Talmy (2000) who proposes to characterize situations and actions through concepts such as the exertion of force, resistance

to such exertion and the overcoming of such resistance, blockage of a force and the removal of such blockage, and so forth. Force dynamics analyzes *causing* into finer primitives and sets it naturally within a framework. Thus, force dynamic states and events can build a strong foundation of a naive physics understanding of the world. Such force dynamic states can, e.g., be monitored in physics engines of virtual worlds (Haidu et al., 2018), are visually observable (Fern et al., 2002; Siskind, 2003), or can be monitored via ambient intelligence using sensoric materials and smart objects (Cook et al., 2009).

Here, actions are further considered as part of a conceptual structure that represents the relational context between several actions, and other entities. One of the reasons that the relational context of a set of actions is important is that it allows us to group experiences into similarity classes, e.g., situations where a similar task was executed, a similar plan was used, or where similar types of objects participated in events. For example, we humans often follow a recipe for cooking a dish by performing several actions towards the accomplishment of the common goal to prepare a meal. In this case, the recipe represents the descriptive context of the experience, while the coincidence of the several required entities the situational context. Thus, a `Situation` can be seen as a view on a set of entities that satisfies a description. This is a common view which, e.g., also appears in the *Description & Situation Ontology* (Gangemi and Mika, 2003).

Overall, the modeling of actions in this work includes four levels of granularity: situations, action events, motion phases, and contact events. An example of this hierarchy is shown in Figure 5.3. On the highest level, there is the situation of the robot preparing instruments for a surgery, following the instructions (plan) given to it. This activity involves several picking and placing actions with their immediate goals of changing the location of a specific instrument. Some instruments might be delicate, fragile, small, and thus need to be handled with care. The particular way a task is executed by the robot is defined by a motion plan that organizes the execution into different phases, such as an approaching motion that brings the robot into grasping position, and a grasping motion that brings the gripper of the robot into contact with the instrument. The contact event can further be characterized from a force dynamical perspective, e.g., through a distinction of the roles `Agonist` and `Antagonist` for the participants of the contact event.

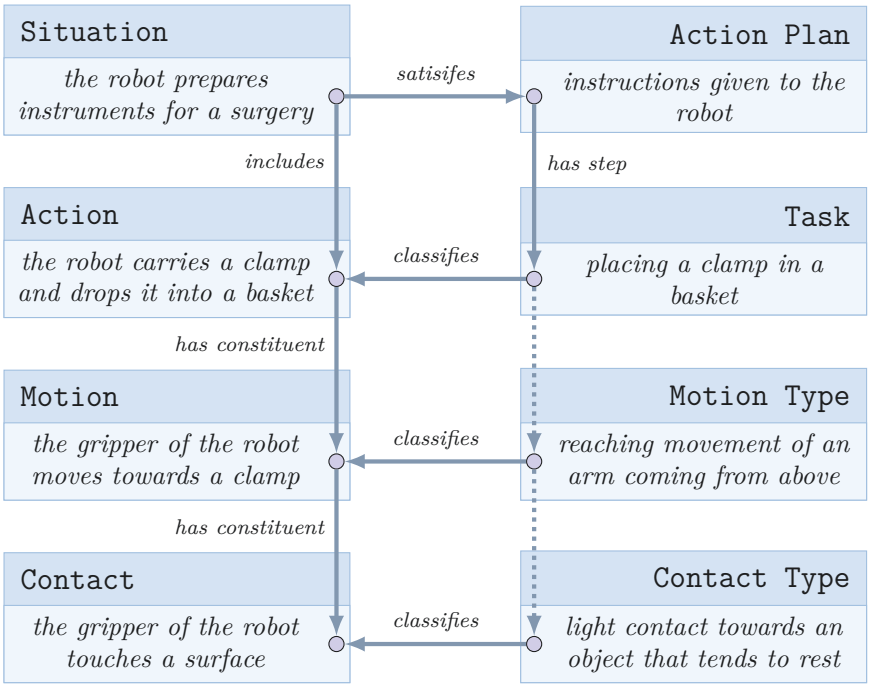

Figure 5.3: Hierarchical representation of actions with four levels of granularity.

Figure 5.3 further highlights that each level in the action hierarchy is represented in both the physical (left side) and social branch (right side of Figure 5.3) of the SOMA ontology. The physical (or ground) branch of the ontology consists of event entities, physical objects, and situations where they coincide; and the social (or descriptive) branch consists of abstract conceptualizations that classify, and descriptions that define entities in the physical branch. The former is used to represent concrete experiences, and the latter is used to categorize them into conceptual groups. Accordingly, only a rather shallow taxonomy of events is defined in the physical branch, and a more fine-grained event classification is considered in the social branch, e.g., via a taxonomy of tasks that are seen as concepts classifying action events. More details about the ontological modeling of actions in the SOMA framework are provided in Section 3.1 of this work.

5.2.3 Ontological Representation of the pHRI Domain

A key cognitive capability of safety-aware robots is the ability to remember actions and conceptualize them in terms of safety-relevant events. Using such a memory

system, robots can answer questions about their actions at the level of safety concepts. Example queries are *was the robot close to a vulnerable body part of a human co-worker*, *did the human interrupt the robot*, and *was the blade of the scalpel pointing away from all co-workers*? This type of question answering capability is a crucial resource for both run-time decision making and offline safety analysis. Here, the focus lies on the representation of pHRI scenarios. This involves concepts such as monitoring humans, human intrusions, and physical contact with humans.

The basis of safe pHRI is that robots are aware of the humans in their vicinity. First of all, it is of concern whether humans are present in the room where the robot is located. Each human is represented as an individual agent with several components such as *left shoulder*, *right forearm*, and *head*. The components are organized in a kinematic structure which is linked to the components of a human using the SRDL ontology (Kunze et al., 2011). In the case study, the identity of humans that were perceived is further tracked, and corresponding representations in the knowledge base are created accordingly. More concretely, a template SRDL ontology is instantiated for each individual human where the unique id of the human provided by the tracker is used as a prefix for the names of individuals in the knowledge base.

The tracking of humans also causes event instances to be created in the knowledge base. Each event is classified as an instance of `HumanPresence` (*HP*). The perceived human is a participant of the event, and takes the role of being the `DetectedPerson`.

$$\texttt{DetectedPerson} \sqsubseteq (\texttt{DetectedObject} \sqcap \forall \mathit{isRoleOf}.\texttt{Human}) \tag{5.5}$$

$$\mathit{HP} \sqsubseteq \exists \mathit{hasParticipant}.\texttt{Human} \tag{5.6}$$

Furthermore, a human within the safety region *RSS* of the robot may get injured if the robot continues its operation. The detection of such events is performed based on whether the distance of monitored humans to components of the robot is below a certain threshold, e.g., 40 centimeter for the head, and 30 centimeter for the gripper of the robot. The region *RSS* contains all these points. Hence, the presence of a human in the workspace of a robot can be classified as a

`HumanIntrusion` for time durations where the location of a tracked body part of the human lies within the *RSS* region:

$$\begin{aligned}\texttt{HumanIntrusion} \equiv HP \sqcap \exists hasParticipant.(\texttt{Human} \sqcap \exists hasComponent.(\\ \exists hasQuality.(\texttt{Location} \sqcap \exists hasRegion.RSS)))\end{aligned} \tag{5.7}$$

Similarly, it is considered as an interruption if one of the components of the monitored human is in physical contact with a component of the robot. Let $\{x_1, \ldots, x_n\}$ be the set of components of the robot. The presence of a human in the room of a robot can then be classified as `HumanInterruption` if there is a human body part that currently touches one of the components:

$$\begin{aligned}\texttt{HumanInterruption} \equiv HP \sqcap \exists hasParticipant.(\texttt{Human} \sqcap \\ \exists hasComponent.(\exists touches_t.\{x_1, \ldots, x_n\}))\end{aligned} \tag{5.8}$$

The temporalized relation *touches* is computed based on feedback from the external torque sensors which is used to constrain the existence of the relation to moments where sensors report external force. Such moments are represented as `ContactEvents`, and classified based on the external torque measured. For example, a contact event is classified as a `LightContact` if the external torque quality of one of the participants of the event has a value within the region R_{LC}:

$$\begin{aligned}\texttt{LightContact} \equiv \texttt{ContactType} \sqcap \exists classifies.(\texttt{Contact} \sqcap \\ \exists hasParticipant.(\exists hasQuality.(ET \sqcap \exists hasRegion.R_{LC})))\end{aligned} \tag{5.9}$$

5.3 Monitoring Experiences

Experience data in the robotics domain is rather expensive to acquire by means of robot experiments and simulations. Furthermore, in industrial settings, such data is usually not shared with the scientific community. It is thus needed that the scientific community establishes infrastructure for storage, curation and use of experience knowledge, and representational standards that make it easier to share experience data between different research institutes. In the scope of this work, the openEASE platform was used (Beetz et al., 2015c). openEASE is a storage, analysis, and visualization platform for experience data acquired from

different modalities such as robots performing everyday activities. The system is accompanied by a set of tools designed to record and semantically annotate activity episodes. Such episodes of agents performing certain tasks are centrally stored, and can be combined for analysis and visualization purposes. The underlying database is used to store large amounts of data including trajectories carried out by agents, and additional sensor data that is available through the experimental setup. Such a cloud robotics system for robot experience data can be leveraged for Deep Learning applications, and, as Pratt (2015) has argued, could cause a *virtuous cycle of explosive growth* in robot capabilities.

5.3.1 Storage Infrastructure

Data storage technology has made huge progress over the last decades in terms of being scalable to huge data sets, and to handle distributed data. The *Map-Reduce* programming paradigm is often used by such systems for efficient data processing (Dean and Ghemawat, 2004). In the *Map-Reduce* framework, computational problems are formulated through the phases *map* and *reduce* where the former phase can operate on different data records in parallel. This principle is implemented in several database framework such as *Hadoop*, and it was employed, e.g., to perform RDFS reasoning (Urbani et al., 2009). openEASE further couples the *Hadoop* infrastructure with tools for data versioning, and a user interface to upload and maintain data sets. The resulting storage platform for robot experience data is called the NEEM-Hub[1]. Its user interface is realized with *GitLab* where data sets are treated as projects, and thus documentation, usage examples, additional links, etc. can be provided for them. Additional information about the NEEM-Hub can be found in the NEEM-handbook (Beetz et al., 2020).

5.3.2 Acquisition Infrastructure

Experiences can be acquired through different means that also require different infrastructure. In the case of robot behavior, e.g., when a robot control program is used, experiences can be acquired by monitoring and memorizing the state of the robot control program over time, and what inputs the sensorimotor system of the

[1] `https://neemgit.informatik.uni-bremen.de` (accessed 22 May 2022)

agent received, and what outputs it generated. Such events can be conceptualized ontologically, and entities can be classified by these concepts to construct a description of a particular occurrence. Such an infrastructure can be used when robot agents perform tasks in the real world, or if a simulated agent performs tasks in a simulated environment. In the case of a simulated world, additional information can be acquired directly by monitoring the state of the virtual world which also includes explicit events for collisions between objects in the scene, and physical qualities such as velocity and inertia.

Robot control programs are usually based on some middleware infrastruture, e.g., to realize communication between different components of the system. The *Robot Operating System* (ROS) is one of such middle wares, and commonly used in robotics research. ROS is highly modular, and allows, among other things, software components in many languages to communicate with each other through a *publish-subscribe* scheme. Messages send via this interface can be recorded using a generic software tool listening to topics while recording messages in the desired format. One example of such a logging tool is the *Mongo DB Logging Tool* (Niemueller et al., 2012). It organizes message topics in different database collections, and encodes messages as JSON objects.

Different infrastructure is needed for behavior generated by humans in a real-world or simulated scenario because there is no control program that can be monitored. Instead, only the interactions of the human with the world, and the motions performed to carry out the interactions can be monitored directly. In this case, tasks performed by the human must be manually annotated or inferred through additional domain constraints, or by a classification method that can recognize tasks given a sequence of observed events.

The conceptualization of activities is tightly coupled with the high-level plan of the agent that is used to select and parameterize actions the agent intends to perform. In the case of a robotic agent, such representation can be created automatically as the plan is known, and also when the different steps are executed. The logging infrastructure for robot experiences used for the experiments conducted in the scope of this work was introduced by Winkler et al. (2014). Another framework for the acquisition of human NEEMs in virtual environments was presented by Haidu et al. (2018).

5.3.3 Task Execution

The application domain considered here is a service robotics usecase in the medical domain where a robot has the task to sort surgical instruments needed for a medical operation. Throughout its work, the robot has to ensure the safety of humans that could physically interact with the tools and the robot. Figure 5.1 depicts this experimental setup. During the experiments, the robot keeps track of all surgical instruments, and recovers from human interruptions using online reasoning on facts that are stored in its belief state. Symbols in the knowledge base are asserted on the fly, and are also used as symbolic belief state during plan execution. The plan executive uses the belief state to infer what to do next, and also to infer plan parameters to make the plans more invariant to changes in the environment.

In the presented study, DL-level reasoning is only considered during task execution. The reasoning is performed based on the belief state of the robot which represents the experienced situation only for the current point in time. It contains descriptions of detected objects and persons, and how they relate to actions performed by the robot, and other events that occurred. Accordingly, temporalized relations that appear in axioms are computed for this time point when reasoning is performed. Inferred facts are then stored as relational data (potentially in a temporalized form) as part of the experience of the robot.

In the considered scenario, the task of the robot is to put a specific set of items into a basket at pre-defined locations. This can be seen as a kind of recipe that decomposes the basket into different slots that have a dispositional match with objects that can be inserted into the respective slot. This recipe can be represented in terms of the assembly ontology presented earlier in this work (Chapter 4), and also the related mechanism for selecting the next action based on a description of the basket with objects can be used. The states of the slots are then updated when actions are performed, e.g., a slot is classified as occupied if a *putting down* action where the slot was the target location was successfully executed.

For the execution of the different motion phases, so called *action designators* are generated that consist of a detailed parametrization of the motion controller. These designators are not stored as factual knowledge, but rather as lists of key value pairs. To enable retrieval, they are linked to motion symbols via a dedicated relation. Figure 5.4 depicts an example designator. It specifies the control scheme, stiffness

- **command-type** : *CARTESIAN-IMPEDANCE*
- **joint**$_0$
  - **stiffness** : 500.0
  - **damping** : 0.699999988079071
  - **vel**$_{max}$: 0.5
  - **acc**$_{max}$: 1.0
- [**joint**$_1$, ..., **joint**$_6$]
- [**trans**$_x$, **trans**$_y$, **trans**$_z$, **rot**$_x$, **rot**$_y$, **rot**$_z$]
- **nullspace-stiffness** : 50.0
- **nullspace-damping** : 0.699999988079071
- **tool-mass** : 1.5800000429153442
- **contact-strategies**
  - **contact** : *IGNORE*
  - **light-collision** : *SOFT-STOP*
  - **strong-collision** : *SOFT-STOP*
  - **severe-collision** : *HARD-STOP*

Figure 5.4: A motion designator retrieved from experience memory. It consists of different parameters that were passed to the safety-aware motion controller.

and damping parameters, velocity and acceleration thresholds, tool configuration parameters, as well as the specified safety reactions in case of collisions. The designator further contains constants, e.g., *CARTESIAN-IMPEDANCE* and *HARD-STOP*. These are pre-defined constants of the communication protocol between the task executive and motion controller. Ontologically, the values of a motion designator are viewed as parameters of motion events where the name of the parameter corresponds with the key in the designator data.

5.4 Querying Experiences

Particular experienced situations can be retrieved from the database through queries that exploit the structure of the ontology, and abstract categories it defines for the classification of entities on different levels of the action hierarchy. The experience data further consists of all necessary assets to reconstruct the 3D environment where the experienced events are situated. Figure 5.5 depicts such a reconstruction for the experiment displayed in Figure 5.2. The scenes are generated for time instants where relevant interactions occurred. These are (a)

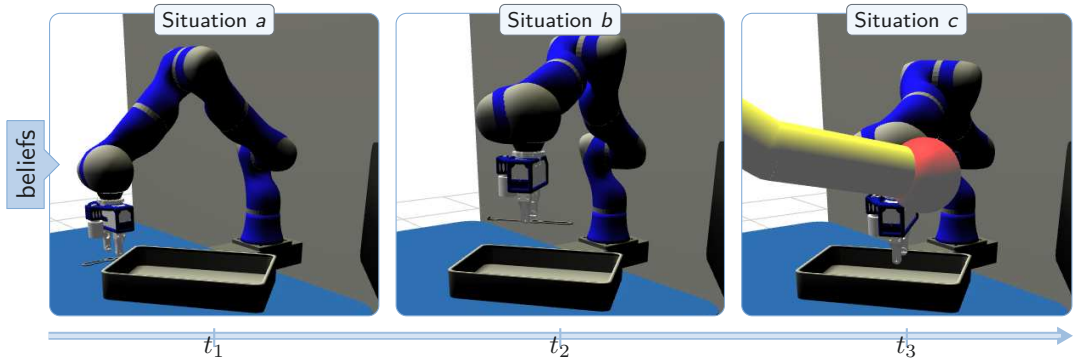

Figure 5.5: A visualization of the belief state for different time instants.

the collision between gripper and table when the robot intends to grasp an object from the table, (b) the disappearing collision between gripper and object when the object is dropped, and (c) the contact between human and robot when the human intervenes the robot performance for some reason.

The classification of safety-relevant events already happens at the time of task execution, and the results are stored as relational data in the experience database. Hence, much of the relevant information can be retrieved through efficient database queries. The querying language used in the study presented herein is based on the Prolog language which belongs to the family of *logic programming languages* (Lloyd, 1987). Prolog has its roots in FOL, but differs in syntax and semantics. In the following, the Prolog language will be extended with predicates that interact with the database where the experiences are stored.

Prolog Basics A Prolog program consists of a finite set of *facts* and *rules*. A rule *if b is true, then h is also true* is written as $h \leftarrow b$ where h is called *head*, and b *body* of the rule. The head of the rule is a n-ary predicate denoting a user-defined relation, and its body consists of a sequence of *goals* separated with a comma. Each predicate symbol either corresponds to a user-defined relation (i.e., a relation defined by a rule, or a set of facts), or has built-in semantics like the comma symbol which is a binary built-in predicate that corresponds to conjunction of goals. Goals are expressions formed over user-defined relations, and built-in predicates. The full syntax of the language is defined along an ISO standard (ISO, 1995).

Prolog programs are defined by an *operational* semantics, i.e., how they are processed by SLD-resolution (Vieille, 1987). This is in contrast to model theoretic semantics of many logical languages. SLD resolution is a technique based on *proof by contradiction* which is used for deciding the satisfiability of a propositional formula, and it uses syntactic unification for on demand instantiation of variables.

The evaluation of Prolog programs is performed in a goal-directed fashion given a query by the user. The query consists of a sequence of comma-separated sub-goals that may contain variables, and the knowledge base searches for different instantiations of variables that render each of the sub-goals true.

5.4.1 Querying Language for Experience Data

NEEMs can be queried about which actions were performed, when, how, and why they were performed, if they were successful, what the robot saw, and what the robot believed when the action was performed. Many of such queries have a similar structure. First, they retrieve actions using a symbolic query that refers to concept and relation names that are defined in an ontology. Then, they use the name of action individuals to retrieve the time instants that correspond to their begin and end. These time instants are used to retrieve non-symbolic data such as the pose of objects and agents at the respective time instants, or trajectories during the respective time intervals.

Let us consider the following example query to get a better intuition of how these queries are used to retrieve experiences from memory:

Listing 5.1: A query that retrieves experiences from episodic memory.

```
entity(CE, [[type, 'Contact'],
    [hasParticipant, EE, [[type, 'EndEffector']]],
    [hasParticipant, OB, [[type, 'PhysicalObject']]]],
    [hasBeginTime, Begin], [hasEndTime, End]]),
entity(PE, [[type, 'Placing'],
    [placedObject, OB], [during, CE]]),
holds(hasLocationValue(EE, w, Pos), Begin).
```

This query navigates through each experienced event *CE* that satisfies the description provided, retrieves the time interval during which the event occurred, and the location of the end effector at the begin of the event. Entity descriptions are nested expressions that include a sequence of symbolic constraints that must

be satisfied by matching entities. In this case, each matching entity *CE* must be classified as a contact event between some object *OB*, and an end effector *EE* during which the object was placed.

The `entity` predicate uses an object-oriented syntax for the representation of entities. An object S is described by a sequence of triples of the form (P, O, D) where P is the name of a predicate, O the name of an entity, and D a description of the entity O. Such triples are translated to goals of the form $P(S, O)$ assuming that the binary relation P is defined in the knowledge base. The description D is used to further constrain instantiations of O. Additionally, entity descriptions may consist of tuples of the form (P, O) or (P, D), i.e., for cases where O is not further constrained, or where the instantiation of O is not relevant respectively. The rule that defines the `entity` predicate in the knowledge base constructs, accordingly, callable goals from entity descriptions in a recursive manner:

$$\texttt{entity}(S, []).$$

$$\texttt{entity}(S, [[P, D]|R]) \leftarrow \texttt{entity}(S, [[P, \_, D]|R]).$$

$$\begin{aligned}\texttt{entity}(S, [[P, O, D]|R]) \leftarrow\ & G =.. [P, S, O], \\ & (\texttt{is\_list}(D) \rightarrow (\texttt{call}(G)\,, \texttt{entity}(O, D))\,; (O = D\,, \texttt{call}(G)))\,, \\ & \texttt{entity}(S, R).\end{aligned}$$

The first clause is the terminal condition for recursion, i.e., that the remaining entity description is empty. The second clause translates each tuple appearing in an entity description into a triple using a new variable (indicated by the underscore character). Note that $[[P, D]|R]$ means that $[P, D]$ is the next element in the sequence, and that R is the remainder. The last clause processes the next element of the sequence, and performs a recursive call for the remainder thereof. To this end, the goal $P(S, O)$ is first constructed (using the *univ* operator $=..$), and then processed conditionally depending on whether D is an instantiated list. Furthermore, note that each P and D must be grounded in entity expressions, while S and each O may be variables. The predicate P cannot be a variable as functors of predicates are not allowed to be variables in the Prolog language, and, if D is a variable, it would be unified with O such that both have the same instantiation, and hence no description would be generated.

The sequence of goals constructed by the `entity` predicate can also be seen as a FOL query. For example, for the first sub-goal of Query 5.1, it generates:

$$\begin{aligned}&\texttt{Contact}(CE)\wedge\\&\mathit{hasParticipant}(CE, EE)\wedge\texttt{EndEffector}(EE)\wedge\\&\mathit{hasParticipant}(CE, OB)\wedge\texttt{PhysicalObject}(OB)\wedge\\&\mathit{hasBeginTime}(CE, \mathit{Begin})\wedge\mathit{hasEndTime}(CE, \mathit{End})\end{aligned}\tag{5.10}$$

Each of the predicates corresponds to a relation defined via the SOMA ontology, and is stored explicitly in the relational data of experiences. Accordingly, such predicates are defined through database queries. For the `hasParticipant` predicate, for example, the following rule is defined:

$$\texttt{hasParticipant}(E, OB)\leftarrow\texttt{lookup}(\texttt{hasParticipant}, [\texttt{ev}(E), \texttt{obj}(OB)]).$$

Where `lookup` is a 2-ary predicate whose first argument is the name of a database table, and the second argument encodes a query in form of key-value pairs. The keys correspond to column names in the database table, i.e., in this case, the table named *hasParticipant* is assumed to have columns with the names *ev* and *obj*.

Note that predicates in Prolog, unlike FOL, may have a procedural reading. This is, e.g., the case if predicates are defined by rules that use negation. This has the disadvantage that some optimization techniques cannot be used to full extend. This problem is often approached via special evaluation techniques, e.g., through *stratification* where a logic program is organized in several *strata* that impose limited ordering constraints on the evaluation of goals (Apt et al., 1988). In general, above definition can be considered as inefficient when using the regular evaluation strategy of Prolog. The reason is that Prolog fetches one tuple at a time for each of the calls where each successive call is performed potentially several times for each of the possible groundings found by the previous call. It is far more efficient to generate large conjunctive queries from logic programs that can be processed by the underlying database system through sophisticated indexing mechanisms as, e.g., done by Beßler et al. (2021). However, efficient database querying is not the subject of this work, and it is only noted here that entity descriptions should not be naively evaluated using Prolog's strategy.

The second sub-goal of Query 5.1 highlights that roles can be used as relations in entity descriptions even though they appear as concepts in the experience database. This is the case for the *placedObject* relation in the example indicating that the object *OB* has the `PlacedObject` role during the event *PE*. This can be captured by the following definition in the knowledge base:

$$\texttt{placedObject}(\mathit{Evt}, \mathit{Obj}) \leftarrow \texttt{entity}(\mathit{Evt},$$
$$[[\texttt{classified}^{-}, [[\texttt{isTaskOf}, [[\texttt{type}, \texttt{PlacedObject}], [\texttt{classifies}, \mathit{Obj}]]]]]]).$$

Hence, predicates in entity descriptions can be defined by rules, and do not need to be materialized in the database. From a database perspective, rules correspond to database views that are computed as the result set of a query. The temporal predicate *during* is another example of such a predicate. It is simply computed through a comparison of event interval data:

$$\begin{aligned}\texttt{during}(E_1, E_2) \leftarrow\ & \texttt{hasBeginTime}(E_1, t_{01}), \texttt{hasEndTime}(E_1, t_{11}),\\ & \texttt{hasBeginTime}(E_2, t_{02}), \texttt{hasEndTime}(E_2, t_{12}),\\ & t_{01} \geq t_{02}, t_{11} \leq t_{12}.\end{aligned}$$

The last sub-goal of Query 5.1 is the higher-order predicate `holds` which receives another predicate G in its first argument, and a time point or interval in its second argument. The `holds` predicate relates ground instantiations of the predicate G to time intervals or time points where the instantiation holds true. Hence, the predicate G must be defined in a temporalized form. To this end, an operator $\leftarrow_Q$ is used for the definition of temporalized predicates where Q is a list of additional constraints for the lookup of data records, and additional variables that should be instantiated from the experience memory. For example, for the `hasLocationValue` predicate, the following definition is used:

$$\texttt{hasLocationValue}(O, \mathit{RF}, V) \leftarrow_Q \texttt{lookup}(\texttt{tf}, [\texttt{obj}(O), \texttt{pos}(V), \texttt{parent}(\mathit{RF})|Q]).$$

Where `parent` is not the name of a column but a special key used for the lookup of spatial data indicating that the data is requested in a particular frame of reference, in this case with the name *RF*.

Finally, the `holds` predicate is defined by, first, constructing temporal constraints for the evaluation of the predicate, and, second, calling the temporalized rule that defines it using a variant of the `call` predicate that also considers temporalized predicates:

$$\texttt{holds}(G, T) \leftarrow \texttt{var}(T), T = [T_0, T_1], \texttt{call}_{[\texttt{begin}(T_0),\texttt{end}(T_1)]}(G).$$
$$\texttt{holds}(G, T) \leftarrow \texttt{number}(T), \texttt{call}_{[\texttt{begin}(\leq T),\texttt{end}(\geq T)]}(G).$$
$$\texttt{holds}(G, [T_0, T_1]) \leftarrow \texttt{ground}([T_0, T_1]), \texttt{call}_{[\texttt{begin}(\leq T_0),\texttt{end}(\geq T_1)]}(G).$$

Note that this definition is only concerned with the retrieval of temporalized instance data. The formalization of temporal characteristics is usually rather considered in a specialized logic such as *linear temporal logic* (Pnueli, 1977). However, this is not considered here as it is expected that the experience data consist of all the necessary time data for behavior inspection.

5.4.2 Construction of Data Sets for Machine Learning

NEEMs are data sets that consist of experience data and narrative. In the context of supervised learning, the input of training data samples is represented by a segment of the experience data, and the desired output by the corresponding narrative part of the NEEM which is represented as an ABox ontology. The ontology provides additional knowledge about entities that are referred to in the narrative. Such knowledge can be leveraged by machine learning techniques for different purposes. To this end, knowledge base queries can be formulated that retrieve data segments corresponding to individuals that satisfy the query, e.g., for the purpose of computing a subset of the data used for training.

Data sets used for machine learning are usually designed with a particular learning task in mind, and thus can only be used for a limited range of applications. NEEMs are rather designed as comprehensive data sets without prior assumptions about which tasks are trained based on the data. To this end, ontologies are used for the segmentation and classification of the data into different abstract categories. Data of different motion phases, for example, can be identified with named individuals in the ontology, and data can accordingly be selected or omitted based on whether a data selection query is satisfied by an individual.

For example, let us consider a case where a robot needs to figure out where it should be located to pick up a surgical instrument from a table. To this end, it can gather situations that correspond to previous similar experiences, i.e., where the same task was performed, and where the same type of objects were involved:

Listing 5.2: A query that computes a data set with ground truth data about where the robot was located relative to a scalpel when picking it up.

```
findall(RelPose,
    ( entity(Act, [[type, 'Picking'],
          [performedBy, Agent],
          [patient, Obj, [[type, 'Scalpel']]],
          [sourceLocation, [[type, 'Table']]],
          [hasEndTime, End]]),
      holds(hasLocationValue(Agent, Obj, RelPos), End)
    ),
    Features).
```

In this case, features are computed as the pose of the robot relative to the object that was picked up. Note that the builtin predicate `findall` is used to gather all instantiations of the pattern *RelPos* in a list data structure *Features*.

Given this data set, a model can be learned that predicts regions where the robot should stand such that the picking is likely to be successful provided that samples in the set represent successful actions. Further constraints that training examples must satisfy can be formulated in data selection queries through the different relation and concept names defined in the ontology such as that a particular robot platform was used, or that it operated in a particular environment.

Another aspect is that learning problems are usually formulated within a carefully defined mathematical framework, and that reasoning is limited to the scope of that framework. This is, e.g., often the case for motion skill learning based on the optimization of an objective function. It has been shown that such frameworks are capable of generating dexterous motions, however, usually it is not possible within these frameworks to reason about the consequences that motions might have. But such common sense understanding is important if the agent needs to adapt its behavior in the face of a novel situation. A gap that can be filled through the use of ontologies for the characterization of experience knowledge. A more thorough discussion about the use of comprehensive experience knowledge for robot learning applications is, e.g., provided by Bozcuoglu (2019).

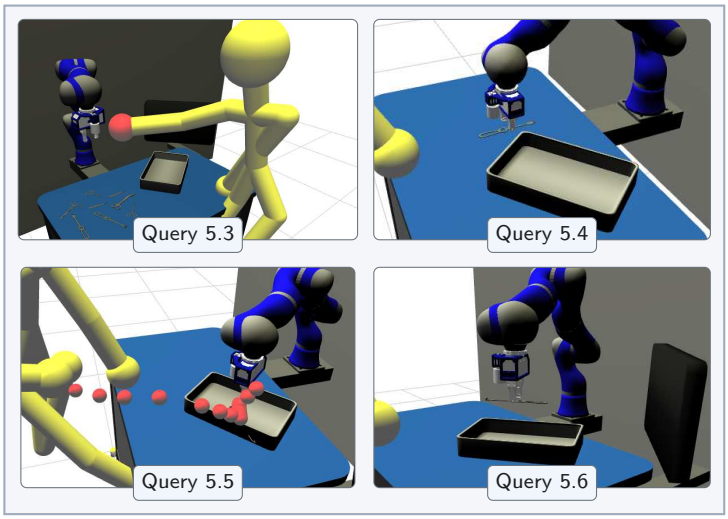

Figure 5.6: Example scenes generated using user queries: co-occurring contact and intrusion events (top left), a contact event without human intrusion (top right), a trajectory during an intrusion event (bottom left), and instruments perceived by the robot (bottom right).

5.5 Evaluation

The experiment was performed 15 times with varying interactions with humans, and with an overall length of 47 minutes. During the experiments, the robot performed 197 *pick-and-place* actions while safely handling 196 intrusion events and 226 contact events. After the experiments, a safety analysis of the robot behavior was conducted. The results show that it is possible to reconstruct the geometric environment of the robot, its course of action, and the motions it performed from abstract descriptions of safety-relevant situations.

5.5.1 Inspection of Safety-Relevant Situations

In the considered domain, the most crucial aspect is how the robot acted in safety-relevant situations that are characterized through co-occuring events where objects of a particular type participate. To this end, such situations must be retrieved from the experience memory through queries. Figure 5.6 depicts four different types of such situations. The corresponding queries used to generate these visualization via the openEASE platform will be discussed in the following.

One of the conditions to classify a situation as safety-relevant is the presence of the human in the safety space of the robot. The event of the human being present

in the room where the robot operates is accordingly classified as a `HumanIntrusion` for time intervals where this is the case. In the following, let *robot1* be an individual of type `Robot` that represents the robot agent in the scene. Particularly relevant are the situations where a component of the detected human touches on of the robot's components. Such situations can be retrieved by the following query:

Listing 5.3: A query that retrieves contact events with co-occurring intrusion events.

```
entity(Intr, [[type, 'HumanPresence'],
    [detectdPerson, Human]]),
holds(type(Intr, 'HumanIntrusion'), TI),
entity(Cont, [[type, 'ContactEvent'],
    [during, TI],
    [hasParticipant, [[isComponentOf, robot1]]]]),
    [hasParticipant, [[isComponentOf, Human]]]]).
```

As explained earlier, the *during* relation is not stored explicitly, but computed on demand given the temporal data associated to events and time intervals. That is, all contact events are searched that start after and end before an intrusion of a human into the safety space of the robot named *robot1*.

Furthermore, situations can be identified where events of a particular type did not occur during the occurrence of some other type of event. It is, for example, relevant to inspect the contact events that occurred when no human was close to the robot. Such situations can be retrieved through the following query:

Listing 5.4: A query that retrieves contact events without co-occurring intrusion events.

```
entity(Cont, [[type, 'ContactEvent']],
    [hasParticipant, [[isComponentOf, robot1]]]]).
not(entity(Intr, [[type, 'HumanPresence']]),
    holds(type(Intr, 'HumanIntrusion'), TI),
    during(TI, Cont)).
```

These type of contact events typically occurred during picking actions when the gripper got into contact with the table. Note that the built-in `not` predicate has closed-world semantics, i.e., it checks whether the database contains records that satisfy the negated goal, and it succeeds exactly if this is not the case.

The previous queries highlight how particular types of situations can be retrieved that are defined by a query provided by the user. Given such a situation, experience data that quantifies the situation can furthermore be retrieved by selecting the corresponding temporal segment of the data. It is, for instance, possible to retrieve whole trajectories of body parts during the events that are

included in the inspected situation. The following query demonstrates this capability for the trajectory of a human body part during intrusion events:

Listing 5.5: A query that retrieves the trajectory of intruding body parts during intrusion events.

```
entity(Intr, [[type, 'HumanPresence'],
     [detectedPerson, [[hasComponent, BodyPart, [[hasQuality, [
        [type, 'Location'], [hasRegion, R]]]]]]]]),
holds(type(Intr, 'HumanIntrusion'), TI1),
holds(type(R, 'RobotSafetySpace'), TI2),
during(TI2, TI1),
findall(Pos,
    holds(hasLocationValue(BodyPart, w, Pos), TI),
    Trajectory).
```

The trajectory is represented as a list constructed over different possible groundings of the *Pos* variable, i.e., the list contains each $p \in \mathbb{R}^3$ with $(p, q) = T^{w}_{BodyPart}(t)$ where t is the timestamp of a database record which lies within the time interval *TI*. Note that, depending on the data frequency, it might be more practical to sample the position at fixed time steps instead.

The last example query is concerned with situations where the robot looses contact to the object it carries during a placing task. The position of the object relative to the basket where it is placed is one of the factors that determine whether the intended goal will be achieved. Let *basket1* be the individual that represents the basket in which an object needs to be placed by the robot. For example, the following query can be used to retrieve the position of the object relative to the basket when it is dropped by the robot:

Listing 5.6: A query that retrieves the distance where objects were dropped.

```
entity(Act, [[type, 'Placing'],
    [patient, Obj]),
entity(Cont, [[type, 'ContactEvent'],
    [hasParticipant, Obj],
    [hasParticipant, [[isComponentOf, robot1]]],
    [during, Act], [hasEndTime, End]]),
holds(hasDistanceValue(Obj, basket1, Distance), End).
```

Where the distance is computed as the length of p with $(p, q) = T^{basket1}_{Obj}(End)$ being the pose of *Obj* in the reference frame of *basket1* at time point *End*.

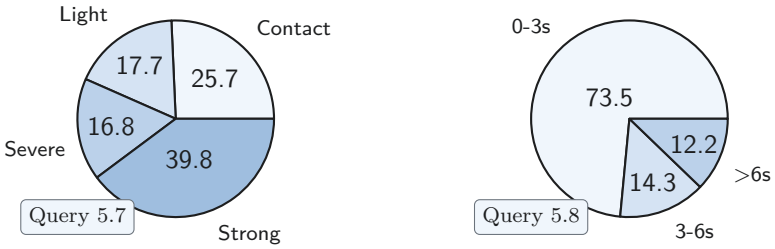

Figure 5.7: The distribution of contact events (left), and the duration of human intrusions (right) over 15 runs of the experiment. 196 intrusions and 226 contacts occurred in total.

5.5.2 *Classification of Safety-Relevant Situations*

Queries can construct groups of situations. This functionality is crucial to perform a statistical analysis of the robot behavior as a whole instead of inspecting details of individual situations as demonstrated earlier. The results of two such queries is shown in Figure 5.7. The corresponding queries will be discussed in the following.

First of all, categories considered during task execution can be inspected. For instance, contact events can be grouped by their classification:

Listing 5.7: A query that constructs a list of contact types of occurring events.

```
findall(EvT,
    entity(Ev, [[type, 'ContactEvent'], [isClassifiedBy, EvT]]),
    EventTypes).
```

The right side of Figure 5.7 depicts the result of a more complex query. Here, all human intrusion events are retrieved, and they are grouped by duration into three user-defined ranges. Namely, intrusions that last at most three seconds, intrusions with a duration of three to six seconds, and intrusions that last at least six seconds. This categorization can be done using the following query:

Listing 5.8: A query that classifies intrusion events along their duration.

```
Ranges = [[0, 3, '0–3s'], [3, 6, '3–6s'], [6, inf, '>6s']],
findall(Label,
    ( entity(Evt, [[type, 'HumanPresence']]),
      holds(type(Evt, 'HumanIntrusion'), [Begin,End]),
      Duration is End - Begin,
      member([Min,Max,Label], Ranges),
      Min =< Duration, Max >= Duration
    ),
    RangeLabels).
```

5.6 Conclusion

This chapter was concerned with the representation and use of experience knowledge that is in part modeled using a common ontology, and stored in a long-term memory. The ontology characterizes actions in terms of their motion phases, and force interactions that are used to segment the different phases of task execution. The phases also correspond to different routines in the robot control program whose activation during task execution can be monitored, and stored in terms of the ontology. Such a long-term storage of experiences is crucial to enhance the performance of a robot over time if it is deployed for a longer duration, and it is also useful for inspecting the behavior of a robot through the evaluation of abstract queries that yield specific situations satisfying the query.

To this end, a query language for robot experiences was proposed. The language is defined within the framework of logic programming, and has its roots in FOL. It interacts with the experience memory through instance data that was acquired during task execution. The instances are classified by abstract concepts defined in the ontology, and linked to spatio-temporally indexed experience data via the notion of object qualities, and their spaces. The temporal data is further used to define temporalized relations in the knowledge base via rules using a dedicated operator. Such rules are used to query for ad-hoc relations that are not stored explicitly in the data.

The feasibility of automatically acquiring such experience knowledge from a robot performing a service task has been demonstrated through a case study. To this end, an ontological classification of safety-relevant situations in the case study domain has been proposed. It has also been demonstrated that the behavior of a robot can be inspected through abstract queries that generate answers for particular situations that satisfy the query. The set of situations that satisfy a query can further be casted as training data set used by methods that learn associations between situational context specified in the query, and the corresponding sensorimotor data that was recorded during the situation. It is likely that the coupling of such an infrastructure for storage and use of experience knowledge with learning methods can create a rapid growth in robot capabilities if the necessary data will be shared among the scientific community.

6 Discussion and Conclusion

6.1 Summary

In this work, the problem of flexible robot task execution was investigated, i.e., how robot control programs can be formulated that can generate competent behavior depending on characteristics of the robot, the task it executes, and the environment where it operates. This problem was approached by means of a specialization of the knowledge-enabled robot programming paradigm (Beetz et al., 2012), i.e., by the separation of re-usable knowledge from the robot control program, and the determination of context-specific information at runtime through reasoning over this knowledge. To this end, it was investigated how such knowledge can be represented and reasoned about using a formal ontology.

It was shown that certain aspects of flexible robot behavior can be realized through an ontological conceptualization of the robot's activity context where the robot can improvise, to a certain extend, given the action possibilities it detects through reasoning over the ontology. The reasoning is performed in case that the particular action sequence the robot executes, the way it moves, and the types of objects it uses are not determined before the task is executed. It was further investigated what the benefits are that the proposed robot ontology yields for the representation of knowledge gained through experimentation and simulation where ontological symbols are grounded in the sensorimotor system of the robot. It was shown that such experience knowledge can be acquired automatically, and that its ontological representation enables the realization of interfaces for robot

behavior inspection based on abstract queries. However, it is important to note that certain characteristics required for flexible task execution, such as activity dynamics, cannot be defined well in DL languages, and thus need to be accounted for by other means.

The problem of planning action sequences in a flexible manner was addressed for assembly tasks through an ontological model that was used to reason about what steps are required to form an assembled product from components that are available. To this end, a planning mechanism was proposed that first infers these steps efficiently using DL-level reasoning, and second organizes them within a priority queue ordered by domain-specific heuristics. The queue is maintained through the course of task execution, and polled whenever the robot needs to determine the next step towards an assembled product.

The ontological model from which the steps are inferred characterizes the goal assemblage by connections between objects that have a suitable type of connector. This type of abstract knowledge about dispositional qualities of objects, such as whether an object is suitably disposed to enter a connection, is more generally useful. To this end, the *Descriptive Affordance Ontology* was proposed, and its implementation in an ontology language was investigated. The theory is inspired by Turvey's notion of dispositions (Turvey, 1992b). It characterizes an affordance by the meeting of two appropriately disposed objects, e.g., two components with fitting connectors. It was shown that current object-centric reasoning in robot control programs can be reformulated around object uses and their roles by using the proposed theory, and thereby robots can take advantage of action possibilities that their environment provides. Another use of the theory is the anticipation of object arrangements that enable future actions.

One of the big advantages of ontologies is that they can be combined, and integrated into a larger framework of modules dedicated to different aspects of a domain. This can be achieved best if the modules agree on the same ontological commitments. To this end, the SOMA ontology was proposed. SOMA is a formal domain ontology for the robotics domain that conceptualizes the context of robot activities, and establishes ontological commitments in the domain. Many of the commitments are cognitively justified, and derived from a well-established foundational ontology that models a range of cognitive phenomena. Accordingly,

the core ontology can be further extended in future work with microtheories covering different cognitive building blocks that enable flexible, adaptive and robust behavior. One of the aspects considered in this work is the organization of actions into distinct phases that are determined through force dynamical states. Representations at this level of granularity enable the robot to reason about how tasks can be accomplished in different ways through motions and interactions with the environment instead of treating them as atomic entities. It was further highlighted that this modeling of actions corresponds to routines in the robot control program whose activation and completion can be monitored to acquire an ontology of the corresponding activity. It was shown by means of a case study that such experience knowledge can be acquired automatically, and used for behavior inspection via the query-answering interface of the robot's knowledge base.

6.2 Discussion

The research presented in this work aimed to investigate the use of formal ontologies in the domain of autonomous robotics for the purpose of flexible task execution, i.e., the execution of tasks in different situational contexts. The autonomy of a robot strongly depends on its capabilities to perceive and conceptualize its environment, and to plan and act towards the achievement of a dedicated goal. The overarching research question that motivated this work is: *what modeling strategies are suitable for the realization of such cognitive capabilities that enable robots to execute tasks in a flexible manner?* More concretely, this work investigated how an OWL DL ontology of the autonomous robotics domain should be designed, i.e., what foundational commitments it makes, and how fundamental terms such as action and object are defined; and how it is employed for reasoning and querying to support the flexibility of robot task execution.

The modeling effort conducted in the scope of this work resulted in a novel ontology for the robotics domain called the SOMA ontology (Section 3.1). It is based on the DUL foundational framework (Masolo et al., 2003), and its distinction between ground and descriptive concepts which is employed in SOMA for the representation of actions and objects, and conceptualizations thereof. Several comparable domain ontologies rather are based on the SUMO upper-level ontology (Niles and

Pease, 2001). This includes the IEEE CORA ontology (Schlenoff et al., 2012), and frameworks such as PMK (Diab et al., 2019a) and ROSETTA (Stenmark and Malec, 2013) that import the CORA ontology.

The choice of the foundational framework has important consequences on the structure of the ontology, modeling workflow, and inferential power. In this work, DUL was preferred over SUMO as a foundational layer for three reasons:

1. DUL attempts to model different cognitive phenomena underlying human cognition. This fits well with the scope of this work as it seeks to support different cognitive capabilities required for flexible robot task execution.

2. DUL has a stronger axiomatization of the more general terms. Therefore, a reasoner can find out more about general concepts, and requires less axioms for the specific branches of the ontology. A strong axiomatization of general terms also supports modeling as the definition of more specific concepts is more constrained, and, thus, less error-prone. On the other hand, more effort is required to align definitions with the foundational layer.

3. DUL, in contrast to SUMO, makes a distinction between ground and descriptive concepts which is crucial if the robot is supposed to conceptualize activities before they are executed, or when another agent is observed.

Nevertheless, some of the definitions in related ontologies, such as the CORA ontology, can be aligned to foundational commitments in DUL, and then imported into the SOMA ontology. Note that the recently published IEEE standard *1872.2-2021* (Olszewska et al., 2017) for the domain of autonomous robotics does not commit to a foundational layer, and rather includes information how the definitions can be aligned with different ontological commitments.

SOMA considers actions to be composed of phases separated through force dynamical events. This viewpoint is motivated by the role of contact events in object manipulation tasks as sensorimotor control points for aligning and comparing predictions with actual sensory events (Flanagan et al., 2006). The rational is that an ontology for the autonomous robotics domain should include knowledge about how goals are achieved through changes in the environment as this knowledge constrains the motions required to execute an abstract task, and

how observations can be interpreted. Several authors have proposed a hierarchical organization of actions for the purpose of action recognition which usually includes motions or motor primitives as most atomic entities in the hierarchy (Bobick, 1997; Krüger et al., 2007). While being motivated by similar observations, the work presented herein, in contrast, investigates the concrete representation of actions using the OWL DL language. The CORA ontology does not consider action hierarchies as of now, however, it is extended in that regard in an ongoing effort (Balakirsky et al., 2017). Nevertheless, frameworks that import the CORA ontology usually define a hierarchy in their own ontologies. For example, PMK considers functions corresponding to robot skills as most atomic components of an action (Diab et al., 2019a). However, in related ontological frameworks that were reviewed in Section 2.3.2, the force dynamical changes caused by the execution of such skills are not considered. Furthermore, frameworks that consider robot experience knowledge usually use ad-hoc representations that are too restricted to be considered useful for a more general memory of robot experiences. More general frameworks exist (Fourie et al., 2017; Breux et al., 2018), however, in contrast to this work, these approaches are either not based on ontological models, or concentrate only on environmental characteristics.

One extension of the SOMA core ontology for the representation of affordances was proposed in Section 3.2. The ontology is used for reasoning about finding and combining appropriate objects, and discovering action possibilities at hand. It is based on the dispositional theory proposed by Turvey (1992b) which assumes that a disposition of an object can be realized when it meets another suitably disposed object in the right conditions. The relevance of knowledge about affordances for flexible task execution has been long recognized in robotics research (Yamanobe et al., 2018). The reason is that, in case a robot operates in unstructured or unknown environments, it needs to adapt, and to some degree improvise, or take advantage of action possibilities that are beneficial to its goals. It was shown that this can be achieved by using the proposed ontology when robot control programs are reformulated around object uses, and dispositions required to play certain roles during an activity. It was further shown that the proposed ontology can be employed to anticipate object arrangements that enable future actions by ensuring that the objects are suitably disposed when being placed by the robot.

The problematic ontological nature of affordances was already recognized by Gibson (1979) who first introduced the term. This problem is presumably the reason why, as of writing this thesis, affordances are not considered in any of the related ontological frameworks that were reviewed in Section 2.2. In several other works, affordances are considered as object qualities that exist independently of other entities, e.g., because the object was designed with particular uses in mind (Ortmann and Kuhn, 2010; Awaad et al., 2014). Turvey's theory of dispositions has also inspired Toyoshima and Barton (2018) who suggest to consider affordances as dispositions that inhere in non-agentive objects. However, qualities cannot capture well the relational nature of affordances, and that they may only exist if appropriately disposed objects meet each other. The consideration of affordances as functional properties of the environment has also been criticized several times in literature as the agentive aspect of affordances is ignored (Stoffregen, 2003; Chemero, 2003). Moralez (2016) suggests that affordances *happen* when the agent perceives the opportunity for action, and, accordingly, characterizes affordances ontologically as events. However, this viewpoint is rejected in this work as it confuses affordances with the act of perceiving them. Finally, Chemero (2003) suggests to consider affordances as relations between abilities of agents, and properties of the environment which is aligned with the perspective of the proposed ontology if abilities of agents are considered as dispositional qualities of the agent. In contrast, in the proposed ontology, an affordance is rather seen as a descriptive context between several entities, and it is assumed to exist when it can be conceptualized by an agent as such.

The use of the proposed ontologies for discovering action possibilities was investigated for the assembly domain in Section 4.1 and Section 4.2. To this end, the representation of dispositional object qualities was exploited to infer possible combinations of objects that can form partial constructions towards the creation of an assembled product. Furthermore, a planning method was proposed that makes use of this inference in combination with domain-specific heuristics for the selection of the next task and participating objects from a list of candidates. Using this method, assembly tasks can be partially defined using the OWL DL language, and, accordingly, a robot can plan its actions using DL reasoning such that all formal constraints are satisfied by the created product. The suitability of

DL languages for creating a powerful abstraction of planning knowledge was, e.g., also recognized by Gil (2005). The rational is that a lot of the planning knowledge required can be determined using subsumption reasoning in an expressive DL formalism, and that this reasoning can combine knowledge of different taxonomies for objects, actions, plans and goals. The proposed mechanism can further be employed in similarly structured application domains where part of the task of the robot can be formulated as a configuration problem.

The de-facto standard formalism for classical planning is PDDL (McDermott et al., 1998). It has been extended several times to meet requirements in the planning community, e.g., to account for temporal domains (Fox and Long, 2003). One fundamental difference between OWL DL and PDDL is the underlying assumption about whether everything about the world is known a priori. Usually, systems based on PDDL make the closed world assumption, and thus require complete knowledge of the domain. In contrast, the open world assumption is crucial in the proposed method to identify facts about entities that are required to exist, but that are not explicitly asserted in the knowledge base. This has the advantage that robots can begin an assembly activity with incomplete knowledge, and that they can cope better with unexpected changes. PDDL domain definitions are evaluated by solvers designed for the language. In contrast, most of the reasoning in the proposed method is performed by a general DL reasoner. General planning is out of scope for a DL reasoner. The proposed method rather highlights how DL-level inference can be incorporated into a planning framework to infer possible combinations of objects for a task at hand.

Ontological modeling of assembly tasks has been the subject of several works in robotics research (Fiorini et al., 2015; Balakirsky, 2015; Polydoros et al., 2016). Some of the authors suggest to translate DL ontologies into PDDL domain specifications (Balakirsky et al., 2013; Kootbally et al., 2015). Hence, unlike the proposed method, no DL reasoning is used during the planning process. Knowledge-driven assembly was further investigated in the scope of the EU ROSETTA project (Patel et al., 2012; Malec et al., 2013; Stenmark et al., 2015a). However, the authors either focused on sequences of tasks (Malec et al., 2013), or on geometric features of atomic parts (Perzylo et al., 2016) as opposed to dispositions and intermediary sub-assemblages.

Overall, the results of this work suggest that relevant parts of the knowledge needed for flexible task execution can be accounted for via the OWL DL language. Using the proposed ontologies, robots can organize their tasks hierarchically, detect action possibilities provided by objects in their environment, and anticipate appropriate placements that enable future actions by exposing the dispositions needed for participation. Related ontological frameworks in the robotics domain that were discussed in Section 2.2, such as PMK (Diab et al., 2019a) and ROSETTA (Stenmark and Malec, 2013), differ in several regards including foundational commitments they make, and granularity and scope of modeling. First of all, in contrast to the related frameworks, the SOMA ontology builds upon the DOLCE ontology (Masolo et al., 2003), and, hence, has a stronger cognitive basis. Another difference is that the modeling of actions in related frameworks is less fine-grained as none of them considers force dynamical changes caused by motions in their ontological modeling. Finally, none of the related frameworks, including PMK and ROSETTA, considers affordances or dispositional object qualities. Accordingly, several of the relevant queries considered in this work cannot be answered within these frameworks.

It is important to note that the problem of flexible robot task execution cannot be fully accounted for by reasoning over the SOMA ontology. One of the reasons is that ontology languages have a limited expressiveness. In this work, OWL DL was used as ontology language. The logical foundation of the language is a decidable fragment of FOL for which reasoner exist that can answer relevant queries efficiently. However, certain characteristics such as temporal and spatial relationships, uncertainty, and activity dynamics cannot be formalized well in a DL theory. Thus, other representations, formalisms and methods must be employed to reason about these characteristics. Furthermore, robot control programs are often parameterized through quantitative data, e.g., through the desired velocity of the robot's arm, or the force it wants to exert on the object it touches. The determination of such parameters is out of scope for an OWL reasoner. However, quantitative parameters and measurements can be linked to the ontology through a categorization of values into regions. Finally, the SOMA ontology has only limited scope and coverage. To this end, the ontology should further be extended to cover more aspects that enable flexible robot task execution.

To conclude, there is a huge potential that ontologies can become essential in the robotics domain for topics such as explainability, ethics-awareness, and human-robot interaction. This requires that the knowledge encoded in ontologies has a consistent meaning among different robots and humans. To this end, additional efforts regarding the standardization of robot ontologies are required. Nevertheless, ontological modeling has already attracted attention in the robotics industry. The company *Ubica*[1] is one example where ontologies are used to characterize the structure of retail stores to support a mapping process. With efforts in research to establish standard ontologies for robotics, it can be expected that more robotics companies will employ knowledge-based approaches in the future.

6.3 Conclusion

One of the insights from this work is the importance of an ontological distinction between the observable physical domain, and the mentally-formed conceptualized social interpretations thereof. For example, an activity is mentally conceptualized as a plan before it is executed by a robot. This dichotomy is crucial for the representation of knowledge that links mental and physical processes. The results of this work further support that dispositional object qualities can be well formalized in a decidable ontology language, and that such a formalization can be employed to identify objects that can participate in an action, and thus to detect action potentials in an environment. Another insight of this work is that an ontological characterization of robot actions should take into account how actions are organized into distinct motion phases enclosed by force dynamical events. Such events, on the one hand, serve as control points for aligning observations and expectations, and, on the other hand, correspond to events caused by the execution of routines in the robot control program whose activation and completion can be monitored to create an ontology of an experienced activity.

The contributions of this work open the doors for significant future research. First of all, the ontological modeling presented herein concentrates on object-centric aspects, i.e., properties of objects and what they afford. It was stated that capabilities of agents can also be seen as qualities, but a theory of capabilities was

[1] `https://www.ubica-robotics.eu/` (accessed 22 May 2022)

neither investigated, nor how bodily characteristics of the robot should be represented. Concerning the former, initial work was presented that aligns well with the proposed approach to formalize affordances (Krieg-Brückner and Codescu, 2019). As for the latter, one possible direction is to align the SRDL ontology (Kunze et al., 2011), or the modeling of robots in the CORA ontology (Schlenoff et al., 2012) to the ontological commitments of the SOMA ontology. Furthermore, SOMA should be integrated with the recently published IEEE standard *1872.2-2021* (Olszewska et al., 2017). The coverage of the disposition ontology should further be enhanced, and mechanisms for automated population should be investigated. To this end, existing commonsense knowledge bases such as *wikidata*, and other existing documents such as technical drawings or CAD models could be used to gather knowledge about object uses as, e.g., done by Tenorth et al. (2013b). Affordances may further allow and constrain the execution of motions in certain ways which also is a pertinent subject for future work. Ultimately, there is a need to establish a set of well-founded microtheories for the cognitive building blocks that enable flexible, adaptive and robust behavior such as image schema (Lakoff, 1987), force dynamics (Talmy, 2000) and x-schema (Bergen and Chang, 2003).

As a closing remark, I would like to emphasize that a linked ontological specification of the robotics domain, such as the one presented herein, can greatly improve the flexibility of robot task execution while retaining means to easily predict, inspect and understand the robot's behavior. In my opinion, this is in particular important for cases where robots are deployed under ethical or legal constraints, and interact with humans. For example, it has been a long dream to create autonomous robots that follow the famous *Three Laws of Robotics* postulated by Asimov (2004). The warranty that such constraints will always be fulfilled can only be issued if the constraints are represented as fixed symbolic structures that can be manipulated and comprehended easier compared to if they were modeled, e.g., as an artificial neural network. I think that these aspects are in particular relevant for future applications in the service robotics domain, and that ontology languages will play a vital role for deploying robots in environments with humans where normed and explainable behavior is crucial for acceptance.

References

E. Ackerman. Atlas shows most impressive parkour skills we've ever seen, 2021. URL `https://spectrum.ieee.org/boston-dynamics-atlas-parkour`. Accessed: 2022-05-22.

A. Akbari, Muhayyuddin, and J. Rosell. Task and motion planning using physics-based reasoning. In *ETFA*, pages 1–7. IEEE, 2015.

J. F. Allen. Maintaining knowledge about temporal intervals. *Commun. ACM*, 26 (11):832–843, 1983.

S. R. Allen. *A Critical Introduction to Properties*. London, UK: Bloomsbury, 2016.

J. E. Anderson. *Constraint-Directed Improvisation for Everyday Activities*. PhD thesis, The University of Manitoba (Canada), 1995. AAINN99082.

G. Antoniou, P. Groth, F. van Harmelen, and R. Hoekstra. *A Semantic Web Primer, 3rd Edition*. MIT Press, 2012.

K. R. Apt, H. A. Blair, and A. Walker. *Towards a Theory of Declarative Knowledge*, page 89–148. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988. ISBN 0934613400.

R. Arp, B. Smith, and A. D. Spear. *Building Ontologies with Basic Formal Ontology*. MIT Press, 2015.

I. Asimov. *I, Robot*. A Bantam book. Random House Publishing Group, 2004. ISBN 9780553294385. URL `http://books.google.de/books?id=2vnbMzYXBQsC`.

I. Awaad, G. Kraetzschmar, and J. Hertzberg. Socializing robots: The role of functional affordances. In *International Workshop on Developmental Social Robotics (DevSoR)*, 11 2013a.

I. Awaad, G. Kraetzschmar, and J. Hertzberg. Affordance-based reasoning in robot task planning. In *Planning and Robotics (PlanRob) Workshop at the 23rd International Conference on Automated Planning and Scheduling (ICAPS)*, 2013b.

I. Awaad, G. Kraetzschmar, and J. Hertzberg. Challenges in finding ways to get the job done. In *2nd Planning and Robotics (PlanRob) Workshop at 24th International Conference on Automated Planning and Scheduling (ICAPS)*, 06 2014.

C. F. Baker, C. J. Fillmore, and J. B. Lowe. The berkeley framenet project. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics - Volume 1*, ACL '98/COLING '98, page 86–90, USA, 1998. Association for Computational Linguistics. doi: 10.3115/980845.980860.

S. Balakirsky. Ontology based action planning and verification for agile manufacturing. *Robotics and Computer-Integrated Manufacturing*, 33(Supplement C):21 – 28, 2015. Special Issue on Knowledge Driven Robotics and Manufacturing.

S. Balakirsky, Z. Kootbally, T. Kramer, A. Pietromartire, C. Schlenoff, and S. Gupta. Knowledge driven robotics for kitting applications. *Robot. Auton. Syst.*, 61(11):1205–1214, Nov. 2013.

S. Balakirsky, C. Schlenoff, S. R. Fiorini, S. Redfield, M. Barreto, H. Nakawala, J. L. Carbonera, L. Soldatova, J. Bermejo-Alonso, F. Maikore, et al. Towards a robot task ontology standard. In *ASME 2017 12th International Manufacturing Science and Engineering Conference collocated with the JSME/ASME 2017 6th International Conference on Materials and Processing*, pages V003T04A049–V003T04A049. American Society of Mechanical Engineers, 2017.

F. Balint-Benczedi, T. Wiedemeyer, M. Tenorth, D. Beßler, and M. Beetz. A knowledge-based approach to robotic perception using unstructured information management. In *AAMAS*, pages 1941–1942. ACM, 2015.

G. Bartels, M. Beetz, D. Beßler, M. Tenorth, and J. O. Winkler. How to use openEASE: An online knowledge processing system for robots and robotics researchers (demonstration). In *AAMAS*, pages 1925–1926. ACM, 2015.

G. Bartels, D. Beßler, and M. Beetz. Episodic memories for safety-aware robots - knowledge representation and reasoning for robots that safely interact with human co-workers. *Künstliche Intell.*, 33(2):123–130, 2019.

J. Bateman, M. Pomarlan, and G. Kazhoyan. Embodied contextualization: Towards a multistratal ontological treatment. *Applied Ontology*, 14(4):379–413, 2019.

J. A. Bateman, M. Beetz, D. Beßler, A. K. Bozcuoglu, and M. Pomarlan. Heterogeneous ontologies and hybrid reasoning for service robotics: The EASE framework. In *ROBOT (1)*, volume 693 of *Advances in Intelligent Systems and Computing*, pages 417–428. Springer, 2017.

S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. McGuinness, P. Patel-Schneijder, and L. A. Stein. OWL Web Ontology Language Reference. Recommendation, World Wide Web Consortium (W3C), 02 2004. See `http://www.w3.org/TR/owl-ref/`.

J. M. Beer, A. D. Fisk, and W. A. Rogers. Toward a framework for levels of robot autonomy in human-robot interaction. *Journal of human-robot interaction*, 3 (2):74–99, 2014.

M. Beetz, L. Mösenlechner, and M. Tenorth. CRAM – a cognitive robot abstract machine for everyday manipulation in human environments. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1012–1017. IEEE, 2010.

M. Beetz, U. Klank, I. Kresse, A. Maldonado, L. Mösenlechner, D. Pangercic, T. Rühr, and M. Tenorth. Robotic roommates making pancakes. In *Humanoids*, pages 529–536. IEEE, 2011.

M. Beetz, D. Jain, L. Mösenlechner, M. Tenorth, L. Kunze, N. Blodow, and D. Pangercic. Cognition-enabled autonomous robot control for the realization of home chore task intelligence. *Proceedings of the IEEE*, 100(8):2454–2471, 2012.

M. Beetz, F. Balint-Benczedi, N. Blodow, D. Nyga, T. Wiedemeyer, and Z. Marton. Robosherlock: Unstructured information processing for robot perception. In *ICRA*, pages 1549–1556. IEEE, 2015a.

M. Beetz, G. Bartels, A. Albu-Schäffer, F. Balint-Benczedi, R. Belder, D. Beßler, S. Haddadin, A. Maldonado, N. Mansfeld, T. Wiedemeyer, R. Weitschat, and J. Worch. Robotic agents capable of natural and safe physical interaction with human co-workers. In *IROS*, pages 6528–6535. IEEE, 2015b.

M. Beetz, M. Tenorth, and J. O. Winkler. Open-ease. In *ICRA*, pages 1983–1990. IEEE, 2015c.

M. Beetz, D. Beßler, J. O. Winkler, J. Worch, F. Balint-Benczedi, G. Bartels, A. Billard, A. K. Bozcuoglu, Z. Fang, N. Figueroa, A. Haidu, H. Langer, A. Maldonado, A. L. P. Ureche, M. Tenorth, and T. Wiedemeyer. Open robotics research using web-based knowledge services. In *ICRA*, pages 5380–5387. IEEE, 2016.

M. Beetz, D. Beßler, A. Haidu, M. Pomarlan, A. K. Bozcuoglu, and G. Bartels. Knowrob 2.0 – A 2nd generation knowledge processing framework for cognition-enabled robotic agents. In *2018 IEEE International Conference on Robotics and Automation, ICRA 2018, Brisbane, Australia, May 21-25, 2018*, pages 512–519, 2018. doi: 10.1109/ICRA.2018.8460964.

M. Beetz, D. Beßler, S. Koralewski, P. Mihai, A. Vyas, A. Hawkin, and K. Dhanabalachandran. NEEM Handbook, 2020. URL `https://ease-crc.github.io/soma/owl/current/NEEM-Handbook.pdf`. Accessed: 2022-05-22.

M. Beetz, S. Stelter, D. Beßler, K. Dhanabalachandran, M. Neumann, P. Mania, and A. Haidu. Robots collecting data: Modelling stores. In L. Villani, C. Natale,

M. Beetz, and B. Siciliano, editors, *Robotics for Intralogistics in Supermarkets and Retail Stores*. Springer International Publishing, 2022. Accepted for publication.

B. Bergen and N. Chang. Embodied construction grammar in simulation-based language understanding. *Construction grammars: Cognitive grounding and theoretical extensions*, 07 2003. doi: 10.1075/cal.3.08ber.

D. Beßler, S. Koralewski, and M. Beetz. Knowledge representation for cognition- and learning-enabled robot manipulation. In *CogRob@KR*, volume 2325 of *CEUR Workshop Proceedings*, pages 11–19. CEUR-WS.org, 2018a.

D. Beßler, M. Pomarlan, A. Akbari, Muhayyuddin, M. Diab, J. Rosell, J. A. Bateman, and M. Beetz. Assembly planning in cluttered environments through heterogeneous reasoning. In *KI*, volume 11117 of *Lecture Notes in Computer Science*, pages 201–214. Springer, 2018b.

D. Beßler, M. Pomarlan, and M. Beetz. OWL-enabled assembly planning for robotic agents. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '18, pages 1684–1692, Richland, SC, 2018c. International Foundation for Autonomous Agents and Multiagent Systems. Finalist for the Best Robotics Paper Award.

D. Beßler, A. K. Bozcuoglu, and M. Beetz. Information system for storage, management, and usage for embodied intelligent systems. In C. S. Große and R. Drechsler, editors, *Information Storage: A Multidisciplinary Perspective*, pages 135–159. Springer International Publishing, 2020a. doi: 10.1007/978-3-030-19262-4_5.

D. Beßler, R. Porzel, M. Pomarlan, M. Beetz, R. Malaka, and J. A. Bateman. A formal model of affordances for flexible robotic task execution. In *ECAI*, volume 325 of *Frontiers in Artificial Intelligence and Applications*, pages 2425–2432. IOS Press, 2020b. doi: 10.3233/FAIA200374.

D. Beßler, R. Porzel, M. Pomarlan, A. Vyas, S. Höffner, M. Beetz, R. Malaka, and J. Bateman. Foundations of the Socio-physical Model of Activities (SOMA)

for autonomous robotic agents. In *FOIS*, volume 344 of *Frontiers in Artificial Intelligence and Applications*, pages 159–174. IOS Press, 2021. doi: 10.3233/FAIA210379.

D. Beßler, R. Porzel, P. Mihai, and M. Beetz. Foundational models for manipulation activity parsing. In T. Jung and M. C. tom Dieck, editors, *Extended Reality - XR in Times of Crisis*. Springer, 2022. Accepted for Publication.

D. Beßler, S. Jongebloed, and M. Beetz. Prolog as a querying language for MongoDB, 2021. URL `https://arxiv.org/abs/2110.01284`.

A. F. Bobick. Movement, activity and action: the role of knowledge in the perception of motion. *Philosophical transactions of the Royal Society of London. Series B, Biological sciences*, 352 1358:1257–65, 1997.

S. Borgo, M. Franssen, P. Garbacz, Y. Kitamura, R. Mizoguchi, and P. E. Vermaas. Technical artifacts: An integrated perspective. *Applied Ontology*, 9(3-4):217–235, 2014.

S. Borgo, A. Cesta, A. Orlandini, and A. Umbrico. Knowledge-based adaptive agents for manufacturing domains. *Eng. with Comput.*, 35(3):755–779, July 2019.

P. Borst, H. Akkermans, and J. Top. Engineering ontologies. *International Journal of Human-Computer Studies*, 46:365–406, 1997.

A. K. Bozcuoglu. *Fast Robot Learning using Prospection and Experimental Knowledge: A Cognitive Approach with Narrative-Enabled Episodic Memories and Symbolic Knowledge*. PhD thesis, Universität Bremen, 2019.

A. K. Bozcuoglu, D. Beßler, and M. Beetz. Rendering semantically-annotated experiment videos out of robot memories. In *Humanoids*, pages 541–546. IEEE, 2015a.

A. K. Bozcuoglu, F. Yazdani, D. Beßler, B. Togorean, and M. Beetz. Reasoning on communication between agents in a human-robot rescue team. In *Towards Intelligent Social Robots – Current Advances in Cognitive Robotics*, Seoul, Korea, 2015b.

Y. Breux, S. Druon, and R. Zapata. From perception to semantics: An environment representation model based on human-robot interactions. *2018 27th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, pages 672–677, 2018.

R. Brooks. Intelligence without representation. *Artificial Intelligence*, 47:139–159, 1991.

B. Bruno, N. Y. Chong, H. Kamide, S. Kanoria, J. Lee, Y. Lim, A. K. Pandey, C. Papadopoulos, I. Papadopoulos, F. Pecora, et al. The caresses eu-japan project: making assistive robots culturally competent. In *Italian Forum of Ambient Assisted Living*, pages 151–169. Springer, 2017.

B. Bruno, R. Menicatti, C. T. Recchiuto, E. Lagrue, A. K. Pandey, and A. Sgorbissa. Culturally-competent human-robot verbal interaction. In *2018 15th International Conference on Ubiquitous Robots (UR)*, pages 388–395. IEEE, 2018.

B. Bruno, C. T. Recchiuto, I. Papadopoulos, A. Saffiotti, C. Koulouglioti, R. Menicatti, F. Mastrogiovanni, R. Zaccaria, and A. Sgorbissa. Knowledge representation for culturally competent personal robots: Requirements, design principles, implementation, and assessment. *International Journal of Social Robotics*, pages 1–24, 2019.

J. E. Buehler and M. Pagnucco. A framework for task planning in heterogeneous multi robot systems based on robot capabilities. In *AAAI*, pages 2527–2533. AAAI Press, 2014.

B. Çalli, A. Walsman, A. Singh, S. Srinivasa, P. Abbeel, and A. M. Dollar. Benchmarking in manipulation research: The YCB object and model set and benchmarking protocols. *CoRR*, abs/1502.03143, 2015.

W. A. Carnielli. Systematization of finite many-valued logics through the method of tableaux. *J. Symb. Log.*, 52(2):473–493, 1987.

S. Caton and C. Haas. Fairness in machine learning: A survey. *CoRR*, abs/2010.04053, 2020.

B. Chandrasekaran, J. R. Josephson, and V. R. Benjamins. Ontology of tasks and methods. In *11th Workshop on Knowledge Acquisition, Modeling and Management (KAW'98)*, Banff, Canada, 1998.

A. Chella, M. Cossentino, R. Pirrone, and A. Ruisi. Modeling ontologies for robotic environments. In *Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering*, SEKE '02, pages 77–80, New York, NY, USA, 2002. ACM.

A. Chemero. An outline of a theory of affordances. *Ecological Psychology*, 15(2): 181–195, 2003.

M. Compton, P. Barnaghi, L. Bermudez, R. GarcíA-Castro, O. Corcho, S. Cox, J. Graybeal, M. Hauswirth, C. Henson, A. Herzog, et al. The ssn ontology of the w3c semantic sensor network incubator group. *Web semantics: science, services and agents on the World Wide Web*, 17:25–32, 2012.

D. J. Cook, J. C. Augusto, and V. R. Jakkula. Ambient intelligence: Technologies, applications, and opportunities. *Pervasive Mob. Comput.*, 5(4):277–298, 2009.

J. Crespo, R. Barber, Ó. M. Mozos, D. Beßler, and M. Beetz. Reasoning systems for semantic navigation in mobile robots. In *IROS*, pages 5654–5659. IEEE, 2018.

D. Davidson. *Essays on actions and events: Philosophical essays*, volume 1. Oxford University Press on Demand, 2001.

E. Davis. Naive physics perplex. *AI Mag.*, 19(4):51–79, 1998.

J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI'04: Sixth Symposium on Operating System Design and Implementation*, pages 137–150, San Francisco, CA, 2004.

H. Deeken, T. Wiemann, and J. Hertzberg. Grounding semantic maps in spatial databases. *Robotics Auton. Syst.*, 105:146–165, 2018. doi: 10.1016/j.robot.2018.03.011.

M. Diab, Muhayyuddin, A. Akbari, and J. Rosell. An ontology framework for physics-based manipulation planning. In *ROBOT (1)*, volume 693 of *Advances in Intelligent Systems and Computing*, pages 452–464. Springer, 2017.

M. Diab, A. Akbari, M. U. Din, and J. Rosell. PMK - A knowledge processing framework for autonomous robotics perception and manipulation. *Sensors*, 19 (5):1166, 2019a.

M. Diab, M. Pomarlan, D. Beßler, A. Akbari, J. Rosell, J. A. Bateman, and M. Beetz. An ontology for failure interpretation in automated planning and execution. In *ROBOT (1)*, volume 1092 of *Advances in Intelligent Systems and Computing*, pages 381–390. Springer, 2019b.

M. Diab, M. Pomarlan, D. Beßler, A. Akbari, J. Rosell, J. A. Bateman, and M. Beetz. Skillman - A skill-based robotic manipulation framework based on perception and reasoning. *Robotics Auton. Syst.*, 134:103653, 2020a.

M. Diab, M. Pomarlan, D. Beßler, S. Borgo, and J. Rosell. "knowing from" - an outlook on ontology enabled knowledge transfer for robotic systems. In *JOWO*, volume 2708 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2020b.

M. Diab, M. Pomarlan, S. Borgo, D. Beßler, J. Rossel, J. A. Bateman, and M. Beetz. Failrecont - an ontology-based framework for failure interpretation and recovery in planning and execution. In *JOWO*, volume 2969 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2021.

A. Dix. *Human-computer interaction*. Springer, 2009.

Z. Dogmus, E. Erdem, and V. Patoglu. Rehabrobo-query: Answering natural language queries about rehabilitation robotics ontology on the cloud. *Semantic Web*, 10(3):605–629, 2019.

M. Fazel-Zarandi and M. S. Fox. Inferring and validating skills and competencies over time. *Appl. Ontol.*, 8(3):131–177, July 2013.

A. Fern, J. M. Siskind, and R. Givan. Learning temporal, relational, force-dynamic event definitions from video. In *AAAI/IAAI*, pages 159–166. AAAI Press / The MIT Press, 2002.

S. R. Fiorini, J. L. Carbonera, P. Gonçalves, V. A. Jorge, V. F. Rey, T. Haidegger, M. Abel, S. A. Redfield, S. Balakirsky, V. Ragavan, H. Li, C. Schlenoff, and E. Prestes. Extensions to the core ontology for robotics and automation. *Robot. Comput.-Integr. Manuf.*, 33(C):3–11, June 2015.

J. R. Flanagan, M. C. Bowman, and R. S. Johansson. Control strategies in object manipulation tasks. *Current Opinion in Neurobiology*, 2006.

D. Fourie, S. Claassens, S. Pillai, R. Mata, and J. J. Leonard. Slamindb: Centralized graph databases for mobile robotics. *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6331–6337, 2017.

M. Fox and D. Long. PDDL2.1: An extension of PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20:61–124, 2003.

A. Gangemi and P. Mika. Understanding the semantic web through descriptions and situations. In *OTM*, volume 2888 of *Lecture Notes in Computer Science*, pages 689–706. Springer, 2003.

A. Gangemi, S. Borgo, C. Catenacci, and J. Lehmann. Task taxonomies for knowledge content d07. Technical report, Metokis Project, 2004.

M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL–the planning domain definition language. *AIPS-98 planning committee*, 1998.

J. J. Gibson. *The Ecological Approach to Visual Perception*. Psychology Press Classic Editions, 1979.

Y. Gil. Description logics and planning. *AI Magazine*, 26(2):73–84, 2005.

A. Gómez-Pérez, M. Fernández-López, and O. Corcho. *Ontological Engineering with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web*. Advanced Information and Knowledge Processing. Springer, 1st edition, 2004.

P. J. Gonçalves and P. M. Torres. Knowledge representation applied to robotic orthopedic surgery. *Robotics and Computer-Integrated Manufacturing*, 33:90–99, 2015.

T. Gruber. A translation approach to portable ontologies. *Knowledge Acquisition*, 5(2):199–220, 1993.

M. Grüninger. Ontology of the process specification language. In *Handbook on ontologies*, pages 575–592. Springer, 2004.

N. Guarino. Formal ontology in information systems. In *Proceedings of FOIS'98*, pages 3–15, Trento, Italy, 06 1998. IOS Press, Amsterdam.

N. Guarino and P. Giaretta. Ontologies and knowledge bases: Towards a terminological clarification. In M. N, editor, *Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing (KBKS'95)*, pages 25–32, University of Twente, Enschede, The Netherlands, 1995. IOS Press, Amsterdam, The Netherlands.

N. Guarino, D. Oberle, and S. Staab. What is an ontology? In *Handbook on ontologies*, pages 1–17. Springer, 2009.

M. Haage, J. Malec, A. Nilsson, K. Nilsson, and S. Nowaczyk. Declarative-knowledge-based reconfiguration of automation systems using a blackboard architecture. In *Eleventh Scandinavian Conference on Artificial Intelligence*, volume 227, pages 163–172. IOS Press, 2011.

A. Haidu and M. Beetz. Automated models of human everyday activity based on game and virtual reality technology. *2019 International Conference on Robotics and Automation (ICRA)*, pages 2606–2612, 2019.

A. Haidu, D. Beßler, A. K. Bozcuoglu, and M. Beetz. KnowRobSIM - game engine-enabled knowledge processing towards cognition-enabled robot control. In *IROS*, pages 4491–4498. IEEE, 2018.

K. Hammar, O. Kutz, A. Dimou, T. Hahmann, R. Hoehndorf, C. Masolo, R. Vita, S. B. Abbès, R. Hantach, P. Calvez, T. P. Sales, D. Porello, D. Beßler, S. Borgo, M. Diab, A. Gangemi, A. O. Alarcos, M. Pomarlan, R. Porzel, M. G. Skjæveland, D. P. Lupp, I. Horrocks, J. W. Klüwer, C. Kindermann, L. Bozzato, T. Mossakowski, and L. Serafini, editors. *Proceedings of the Joint Ontology Workshops co-located with the Bolzano Summer of Knowledge (BOSK 2020),*

Virtual & Bozen-Bolzano, Italy, August 31st to October 7th, 2020, volume 2708 of *CEUR Workshop Proceedings*, 2020. CEUR-WS.org.

S. Harnad. The symbol grounding problem. *Physica D: Nonlinear Phenomena*, 42(1-3):335–346, 1990.

P. J. Hayes. The naive physics manifesto. In D. Michie, editor, *Expert Systems in the Electronic Age*, pages 242–270. Edinburgh University Press, Edinburgh, Scotland, 1979.

M. M. Hedblom, O. Kutz, R. Peñaloza, and G. Guizzardi. Image schema combinations and complex events. *Künstliche Intell.*, 33(3):279–291, 2019.

M. Hegarty. Mechanical reasoning by mental simulation. *Trends in cognitive sciences*, 8(6):280–285, 2004.

G. Hesslow. The current status of the simulation theory of cognition. *Brain Research*, 1428:71–79, 2012.

G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012. doi: 10.1109/MSP.2012.2205597.

I. Horrocks and P. F. Patel-Schneider. Reducing OWL entailment to description logic satisfiability. *J. Web Semant.*, 1(4):345–357, 2004.

I. Horrocks, O. Kutz, and U. Sattler. The even more irresistible SROIQ. In *KR*, pages 57–67. AAAI Press, 2006.

L. Hotz, A. Felfernig, A. Günter, and J. Tiihonen. *A Short History of Configuration Technologies*, pages 9–19. 04 2014. ISBN 978-0-12-415817-7. doi: 10.1016/B978-0-12-415817-7.00002-5.

ISO. Information technology – Programming languages – Prolog. Standard, ISO, Vernier, Geneva, Switzerland, 1995.

ISO. Robot and Robotics Devices – Vocabulary. Standard, ISO, Vernier, Geneva, Switzerland, 2012.

L. Jacobsson, J. Malec, and K. Nilsson. Modularization of skill ontologies for industrial robots. In *Proceedings of ISR 2016: 47st International Symposium on Robotics*, pages 1–6. VDE, 2016.

M. Johnson. *The Body in the Mind Metaphors*. University of Chicago Press, 1987.

A. Jordanous. Intelligence without representation: A historical perspective. *Syst.*, 8(3):31, 2020.

V. A. Jorge, V. F. Rey, R. Maffei, S. R. Fiorini, J. L. Carbonera, F. Branchi, J. P. Meireles, G. S. Franco, F. Farina, T. S. Da Silva, et al. Exploring the ieee ontology for robotics and automation for heterogeneous agent interaction. *Robotics and Computer-Integrated Manufacturing*, 33:12–20, 2015.

L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1–2):99–134, 1998. ISSN 0004-3702.

A. Kalyanpur. *Debugging and Repair of OWL Ontologies*. PhD thesis, University of Maryland, College Park, MD, USA, 2006.

G. Kazhoyan, S. Stelter, F. K. Kenfack, S. Koralewski, and M. Beetz. The robot household marathon experiment. In *ICRA*, pages 9382–9388. IEEE, 2021.

A. A. Khaliq, U. Köckemann, F. Pecora, A. Saffiotti, B. Bruno, C. T. Recchiuto, A. Sgorbissa, H.-D. Bui, and N. Y. Chong. Culturally aware planning and execution of robot actions. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 326–332. IEEE, 2018.

H. J. Kim, M. I. Jordan, S. Sastry, and A. Y. Ng. Autonomous helicopter flight via reinforcement learning. In *Advances in neural information processing systems*, pages 799–806, 2004.

N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *In IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2149–2154, 2004.

Z. Kootbally, C. Schlenoff, C. Lawler, T. Kramer, and S. Gupta. Towards robust assembly with knowledge representation for the planning domain definition language (pddl). *Robot. Comput.-Integr. Manuf.*, 33(C):42–55, June 2015.

H. S. Koppula, R. Gupta, and A. Saxena. Learning human activities and object affordances from RGB-D videos. *Int. J. Robotics Res.*, 32(8):951–970, 2013.

S. Koralewski, G. Kazhoyan, and M. Beetz. Self-specialization of general robot plans based on experience. *IEEE Robotics Autom. Lett.*, 4(4):3766–3773, 2019.

B. Krieg-Brückner and M. Codescu. Deducing qualitative capabilities with generic ontology design patterns. In *ROBOT (1)*, volume 1092 of *Advances in Intelligent Systems and Computing*, pages 391–403. Springer, 2019.

B. Krieg-Brückner, T. Mossakowski, and F. Neuhaus. Generic ontology design patterns at work. In A. Barton, S. Seppälä, and D. Porello, editors, *Proceedings of the Joint Ontology Workshops 2017. Joint Ontology Workshops (JOWO-2019), Episode V: The Styrian Autumn of Ontology„ September 23-25, Graz, Austria*. CEUR-WS.org, 9 2019.

V. Krüger, D. Kragic, A. Ude, and C. Geib. The meaning of action: a review on action recognition and mapping. *Advanced Robotics*, 21(13):1473–1501, 2007.

J. J. Kuffner and S. M. LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 2, pages 995–1001. IEEE, 2000.

L. Kunze, T. Roehm, and M. Beetz. Towards semantic robot description languages. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 5589–5595, Shanghai, China, 05 2011.

G. Lakoff. *Women, Fire, and Dangerous Things*. University of Chicago Press, Chicago, 1987.

P. Langley, J. E. Laird, and S. Rogers. Cognitive architectures: Research issues and challenges. *Cognitive Systems Research*, 10(2):141–160, 2009.

S. Lemaignan, R. Ros, L. Mösenlechner, R. Alami, and M. Beetz. Oro, a knowledge management platform for cognitive architectures in robotics. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3548–3553. IEEE, 2010.

S. Lemaignan, R. Ros, R. Alami, and M. Beetz. What are you talking about? grounding dialogue in a perspective-aware robotic architecture. In *2011 RO-MAN*, pages 107–112. IEEE, 2011.

D. B. Lenat. Cyc: A large-scale investment in knowledge infrastructure. *Communications of the ACM*, 38(11):33–38, November 1995.

D. B. Lenat and E. A. Feigenbaum. On the thresholds of knowledge. *Artificial Intelligence*, 47(1):185 – 250, 1991. ISSN 0004-3702. doi: 10.1016/0004-3702(91)90055-O.

D. B. Lenat and R. V. Guha. *Building large knowledge-based system: representation and inference in the Cyc project*. Addison-Wesley, New York, 1990.

G. H. Lim, I. H. Suh, and H. Suh. Ontology-based unified robot knowledge for service robots in indoor environments. *IEEE Trans. Syst. Man Cybern. Part A*, 41(3):492–509, 2011.

J. W. Lloyd. *Foundations of Logic Programming, 2nd Edition*. Springer, 1987.

J. Malec, K. Nilsson, and H. Bruyninckx. Describing assembly tasks in declarative way. In *IEEE/ICRA Workshop on Semantics*, 2013.

L. Marconi, C. Melchiorri, M. Beetz, D. Pangercic, R. Siegwart, S. Leutenegger, R. Carloni, S. Stramigioli, H. Bruyninckx, P. Doherty, A. Kleiner, V. Lippiello, A. Finzi, B. Siciliano, A. Sala, and N. Tomatis. The SHERPA project: Smart collaboration between humans and ground-aerial robots for improving rescuing activities in alpine environments. In *SSRR*, pages 1–4. IEEE, 2012.

C. Masolo and S. Borgo. Qualities in formal ontology. In *Foundational Aspects of Ontologies (FOnt 2005) Workshop at KI 2005*, pages 2–16, 2005.

C. Masolo, S. Borgo, A. Gangemi, N. Guarino, and A. Oltramari. WonderWeb deliverable D18 ontology library (final). Technical report, IST Project 2001-33052 WonderWeb: Ontology Infrastructure for the Semantic Web, 2003.

D. McDermott. A temporal logic for reasoning about processes and plans. *Cogn. Sci.*, 6(2):101–155, 1982.

D. McDermott. The formal semantics of processes in PDDL. In *Proceedings of the ICAPS Workshop on PDDL*, 2003.

D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL—The Planning Domain Definition Language, 1998.

D. L. McGuinness. *Configuration*, page 388–405. Cambridge University Press, USA, 2003. ISBN 0521781760.

R. Menicatti, B. Bruno, and A. Sgorbissa. Modelling the influence of cultural information on vision-based human home activity recognition. In *2017 14th international conference on ubiquitous robots and ambient intelligence (URAI)*, pages 32–38. IEEE, 2017.

G. A. Miller. Wordnet: A lexical database for english. *Communications of the ACM*, 38(1):39–41, 1995. doi: 10.1145/219717.219748.

R. Mizoguchi, Y. Kitamura, and S. Borgo. A unifying definition for artifact and biological functions. *Applied Ontology*, 11(2):129–154, 2016.

B. Moldovan, P. Moreno, M. Van Otterlo, J. Santos-Victor, and L. De Raedt. Learning relational affordance models for robots in multi-object manipulation tasks. In *2012 IEEE International Conference on Robotics and Automation*, pages 4373–4378. IEEE, 2012.

L. A. Moralez. Affordance ontology: towards a unified description of affordances as events. *Res. Cogitans*, 7(1):35–45, 2016.

L. Morgenstern. Mid-Sized Axiomatizations of Commonsense Problems: A Case Study in Egg Cracking. *Studia Logica*, 67(3):333–384, 2001.

L. Mösenlechner and M. Beetz. Parameterizing actions to have the appropriate effects. In *IROS*, pages 4141–4147. IEEE, 2011.

T. Mossakowski. The distributed ontology, model and specification language - DOL. In *WADT*, volume 10644 of *Lecture Notes in Computer Science*, pages 5–10. Springer, 2016.

B. Motik, P. Patel-Schneider, C. Bock, A. Fokoue, P. Haase, R. Hoekstra, I. Horrocks, A. Ruttenberg, U. Sattler, and M. Smith. Owl 2 web ontology language: Structural specification and functional-style. *Journal of Pragmatics - J PRAGMATICS*, 27, 01 2008.

F. Neuhaus, P. Grenon, and B. Smith. A formal theory of substances, qualities, and universals. In *Formal Ontology in Information Systems: Proceedings of the Third International Conference (FOIS-2004)*. IOS Press, 2004.

T. Niemueller, G. Lakemeyer, and S. S. Srinivasa. A generic robot database and its application in fault analysis and performance evaluation. In *IROS*, pages 364–369. IEEE, 2012.

I. Niles and A. Pease. Towards a standard upper ontology. In *Proceedings of the international conference on Formal Ontology in Information Systems-Volume*. ACM, 2001.

D. Norman. Affordance, conventions, and design. *Interactions*, 6:38–42, 05 1999. doi: 10.1145/301153.301168.

D. A. Norman. *The design of everyday things*. Basic Books, New York, 2002. ISBN 0465067107 9780465067107.

A. Olivares-Alarcos, D. Beßler, A. Khamis, P. Gonçalves, M. Habib, J. Bermejo, M. Barreto, M. Diab, J. Rosell, J. Quintas, J. Olszewska, H. Nakawala, E. Pignaton de Freitas, A. Gyrard, S. Borgo, G. Alenyà, M. Beetz, and H. Li. A review and comparison of ontology-based approaches to robot autonomy. *The Knowledge Engineering Review*, 34, 12 2019. doi: 10.1017/S0269888919000237.

J. I. Olszewska, M. E. Barreto, J. Bermejo-Alonso, J. L. Carbonera, A. Chibani, S. R. Fiorini, P. J. S. Gonçalves, M. K. Habib, A. M. Khamis, A. O. Alarcos,

E. P. de Freitas, E. Prestes, S. V. Ragavan, S. A. Redfield, R. Sanz, B. Spencer, and H. Li. Ontology for autonomous robotics. In *RO-MAN*, pages 189–194. IEEE, 2017.

J. Ortmann and W. Kuhn. Affordances as qualities. In *Proceedings of the 2010 Conference on Formal Ontology in Information Systems: Proceedings of the Sixth International Conference (FOIS 2010)*, pages 117–130, Amsterdam, The Netherlands, 2010. IOS Press. ISBN 978-1-60750-534-1.

Z. Pan, J. Polden, N. Larkin, S. van Duin, and J. Norrish. Recent progress on programming methods for industrial robots. In *ISR/ROBOTIK*, pages 1–8. VDE Verlag, 2010.

C. H. Papadimitriou. *Computational complexity*. John Wiley and Sons Ltd., 2003.

R. Patel, M. Hedelind, and P. Lozan-Villegas. Enabling robots in small-part assembly lines: The "rosetta approach" - an industrial perspective. In *ROBOTIK*. VDE-Verlag, 2012.

J. Persson, A. Gallois, A. Björkelund, L. Hafdell, M. Haage, J. Malec, K. Nilsson, and P. Nugues. A knowledge integration framework for robotics. In *ISR 2010 (41st International Symposium on Robotics) and ROBOTIK 2010 (6th German Conference on Robotics)*, pages 1–8. VDE, 2010.

A. Perzylo, N. Somani, S. Profanter, I. Kessler, M. Rickert, and A. Knoll. Intuitive instruction of industrial robots: Semantic process descriptions for small lot production. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2293–2300, 2016.

A. Perzylo, J. Grothoff, L. Lucio, M. Weser, S. Malakuti, P. Venet, V. Aravantinos, and T. Miny. Capability-based semantic interoperability of manufacturing resources: A basys 4.0 perspective. *International Federation of Automatic Control - PapersOnLine*, 52:1590–1596, 12 2019a. doi: 10.1016/j.ifacol.2019.11.427.

A. Perzylo, M. Rickert, B. Kahl, N. Somani, C. Lehmann, A. Kuss, S. Profanter, A. B. Beck, M. Haage, M. R. Hansen, M. T. Nibe, M. A. Roa, O. Sornmo, S. G. Robertz, U. Thomas, G. Veiga, E. A. Topp, I. Kessler, and M. Danzer.

Smerobotics: Smart robots for flexible manufacturing. *IEEE Robotics Autom. Mag.*, 26(1):78–90, 2019b.

A. Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 46–57, 1977. doi: 10.1109/SFCS.1977.32.

A. S. Polydoros, B. Großmann, F. Rovida, L. Nalpantidis, and V. Krüger. Accurate and versatile automation of industrial kitting operations with skiros. In *Towards Autonomous Robotic Systems - 17th Annual Conference (TAROS)*, pages 255–268, 2016.

M. Pomarlan and J. A. Bateman. Embodied functional relations: A formal account combining abstract logical theory with grounding in simulation. In *FOIS*, volume 330 of *Frontiers in Artificial Intelligence and Applications*, pages 155–168. IOS Press, 2020.

G. A. Pratt. Is a cambrian explosion coming for robotics? *Journal of Economic Perspectives*, 29(3):51–60, September 2015. doi: 10.1257/jep.29.3.51.

E. Prestes, J. L. Carbonera, S. R. Fiorini, V. A. M. Jorge, M. Abel, R. Madhavan, A. Locoro, P. J. S. Gonçalves, M. E. Barreto, M. K. Habib, A. Chibani, S. Gérard, Y. Amirat, and C. Schlenoff. Towards a core ontology for robotics and automation. *Robotics Auton. Syst.*, 61(11):1193–1204, 2013.

G. Riva and E. Riva. Sarafun: Interactive robots meet manufacturing industry. *Cyberpsychology, Behavior, and Social Networking*, 22(4):295–296, 2019.

R. Ros, S. Lemaignan, E. A. Sisbot, R. Alami, J. Steinwender, K. Hamann, and F. Warneken. Which one? grounding the referent based on efficient human-robot interaction. In *19th International Symposium in Robot and Human Interactive Communication*, pages 570–575. IEEE, 2010.

J. Rosell, A. Pérez, A. Aliakbar, Muhayyuddin, L. Palomo, and N. García. The Kautham Project: A teaching and research tool for robot motion planning. In *IEEE Int. Conf. on Emerging Technologies and Factory Automation*, 2014.

R. B. Rusu, Z. C. Marton, N. Blodow, A. Holzbach, and M. Beetz. Model-based and learned semantic object labeling in 3d point cloud maps of kitchen environments. In *IROS*, pages 3601–3608. IEEE, 2009.

F. A. Salustri. *Ontological Commitments in Knowledge-Based Design Software: A Progress Report*, volume 161 of *IFIP Conference Proceedings*, pages 41–72. Kluwer, 1998.

E. M. Sanfilippo, O. Kutz, N. Troquard, T. Hahmann, C. Masolo, R. Hoehndorf, R. Vita, M. M. Hedblom, G. Righetti, D. Sormaz, W. Terkaj, T. P. Sales, S. de Cesare, F. Gailly, G. Guizzardi, M. Lycett, C. Partridge, O. Pastor, D. Beßler, S. Borgo, M. Diab, A. Gangemi, A. O. Alarcos, M. Pomarlan, R. Porzel, L. Jansen, M. Brochhausen, D. Porello, P. Garbacz, S. Seppälä, M. Grüninger, A. Vizedom, D. M. Dooley, R. Warren, H. Küçük-McGinty, M. Lange, A. Algergawy, N. Karam, F. Klan, F. Michel, and I. Rosati, editors. *Proceedings of the Joint Ontology Workshops 2021 Episode VII: The Bolzano Summer of Knowledge co-located with the 12th International Conference on Formal Ontology in Information Systems (FOIS 2021), and the 12th International Conference on Biomedical Ontologies (ICBO 2021), Bolzano, Italy, September 11-18, 2021*, volume 2969 of *CEUR Workshop Proceedings*, 2021. CEUR-WS.org.

A. Saxena, J. Driemeyer, and A. Y. Ng. Robotic grasping of novel objects using vision. *The International Journal of Robotics Research*, 27(2):157–173, 2008.

A. Saxena, A. Jain, O. Sener, A. Jami, D. K. Misra, and H. S. Koppula. Robobrain: Large-scale knowledge engine for robots. *CoRR*, abs/1412.0691, 2014.

C. Schlenoff, E. Prestes, R. Madhavan, P. Goncalves, H. Li, S. Balakirsky, T. Kramer, and E. Miguelanez. An IEEE standard ontology for robotics and automation. In *IEEE Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 1337–1342, 2012.

A. Sgorbissa, I. Papadopoulos, B. Bruno, C. Koulouglioti, and C. Recchiuto. Encoding guidelines for a culturally competent robot for elderly care. In *2018*

IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 1988–1995. IEEE, 2018.

M. Shanahan. A logical formalisation of ernie davis's egg cracking problem. In *Problem. Fourth Symposium on Logical Formalizations of Commonsense Reasoning*, 1997.

R. Shearer, B. Motik, and I. Horrocks. HermiT: A Highly-Efficient OWL Reasoner. In A. Ruttenberg, U. Sattler, and C. Dolbear, editors, *Proc. of the 5th Int. Workshop on OWL: Experiences and Directions (OWLED 2008 EU)*, Karlsruhe, Germany, October 26–27 2008.

B. Siciliano and O. Khatib, editors. *Springer Handbook of Robotics*. Springer, Berlin, Heidelberg, 2008.

J. Simpson, E. Weiner, and O. U. Press. The Oxford English Dictionary. In *The Oxford English Dictionary*. Clarendon Press, 1989. ISBN 9780198611868. URL `https://books.google.de/books?id=4qcRAQAAMAAJ`. Accessed: 2022-05-22.

E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical owl-dl reasoner. *Web Semantics: science, services and agents on the World Wide Web*, 5(2):51–53, 2007.

E. A. Sisbot, R. Ros, and R. Alami. Situation assessment for human-robot interactive object manipulation. In *2011 RO-MAN*, pages 15–20. IEEE, 2011.

J. Siskind. Reconstructing force-dynamic models from video sequences. *Artificial Intelligence*, 151(1):91–154, 2003.

M. Stenmark and J. Malec. Knowledge-based industrial robotics. In *SCAI*, pages 265–274, 2013.

M. Stenmark, J. Malec, K. Nilsson, and A. Robertsson. On distributed knowledge bases for robotized small-batch assembly. *IEEE Transactions on Automation Science and Engineering*, 12(2):519–528, 2015a.

M. Stenmark, J. Malec, and A. Stolt. From high-level task descriptions to executable robot code. In *Intelligent Systems' 2014*, pages 189–202. Springer, 2015b.

M. Stenmark, M. Haage, E. A. Topp, and J. Malec. Supporting semantic capture during kinesthetic teaching of collaborative industrial robots. *International Journal of Semantic Computing*, 12(01):167–186, 2018.

T. A. Stoffregen. Affordances as properties of the animal-environment system. *Ecological Psychology*, 15(2):115–134, 2003.

R. Studer, V. Benjamins, and D. Fensel. Knowledge engineering: Principles and methods. *IEEE Transactions on Data and Knowledge Engineering*, 25(1-2): 161–197, 1998.

I. A. Sucan, M. Moll, and L. E. Kavraki. The open motion planning library. *IEEE Robotics Autom. Mag.*, 19(4):72–82, 2012.

I. H. Suh, G. H. Lim, W. Hwang, H. Suh, J.-H. Choi, and Y.-T. Park. Ontology-based multi-layered robot knowledge framework (omrkf) for robot intelligence. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 429–436. IEEE, 2007.

N. Sünderhauf, O. Brock, W. J. Scheirer, R. Hadsell, D. Fox, J. Leitner, B. Upcroft, P. Abbeel, W. Burgard, M. Milford, and P. Corke. The limits and potentials of deep learning for robotics. *Int. J. Robotics Res.*, 37(4-5):405–420, 2018.

L. Talmy. *Toward a Cognitive Semantics. Volume 2: Typology and Process in Concept Structuring*. Language, Speech, and Communication. MIT Press, Cambridge, MA, 2000. ISBN 978-0-262-20121-6.

M. Tenorth and M. Beetz. Knowrob – knowledge processing for autonomous personal robots. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4261–4266. IEEE, 2009.

M. Tenorth and M. Beetz. A unified representation for reasoning about robot actions, processes, and their effects on objects. In *IROS*, pages 1351–1358. IEEE, 2012.

M. Tenorth and M. Beetz. KnowRob – A Knowledge Processing Infrastructure for Cognition-enabled Robots. *Int. Journal of Robotics Research*, 32(5):566 – 590, April 2013.

M. Tenorth and M. Beetz. Representations for robot knowledge in the knowrob framework. *Artif. Intell.*, 247:151–169, 2017.

M. Tenorth, L. Kunze, D. Jain, and M. Beetz. KnowRob-Map – knowledge-linked semantic object maps. In *10th IEEE-RAS International Conference on Humanoid Robots*, pages 430–435, Nashville, TN, USA, 12 2010a.

M. Tenorth, D. Nyga, and M. Beetz. Understanding and Executing Instructions for Everyday Manipulation Tasks from the World Wide Web. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1486–1491, Anchorage, AK, USA, 05 2010b.

M. Tenorth, A. C. Perzylo, R. Lafrenz, and M. Beetz. Representation and exchange of knowledge about actions, objects, and environments in the roboearth framework. *IEEE Transactions on Automation Science and Engineering*, 10: 643–651, 2013a.

M. Tenorth, S. Profanter, F. Balint-Benczedi, and M. Beetz. Decomposing CAD Models of Objects of Daily Use and Reasoning about their Functional Parts. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5943–5949, Tokyo Big Sight, Japan, November 3–7 2013b.

M. Tenorth, G. Bartels, and M. Beetz. Knowledge-based specification of robot motions. In *ECAI*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, pages 873–878. IOS Press, 2014.

M. Tenorth, J. O. Winkler, D. Beßler, and M. Beetz. Open-ease: A cloud-based knowledge service for autonomous learning. *Künstliche Intell.*, 29(4):407–411, 2015.

M. Thosar, S. Zug, A. M. Skaria, and A. Jain. A review of knowledge bases for service robots in household environments. In *AIC*, volume 2418 of *CEUR Workshop Proceedings*, pages 98–110. CEUR-WS.org, 2018.

I. Tiddi, E. Bastianelli, G. Bardaro, M. d'Aquin, and E. Motta. An ontology-based approach to improve the accessibility of ros-based robotic systems. In

Proceedings of the Knowledge Capture Conference, K-CAP 2017, pages 13:1–13:8, New York, NY, USA, 2017. ACM.

E. A. Topp and J. Malec. A knowledge based approach to user support for robot programming. In *AI for Multimodal Human Robot Interaction Workshop within the Federated AI Meeting 2018 in Stockholm*, pages 31–34, 2018.

E. A. Topp, M. Stenmark, A. Ganslandt, A. Svensson, M. Haage, and J. Malec. Ontology-based knowledge representation for increased skill reusability in industrial robots. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5672–5678. IEEE, 2018.

P. Torres, P. J. S. Gonçalves, and J. Martins. Robotic motion compensation for bone movement, using ultrasound images. *Industrial Robot: An International Journal*, 42(5):466–474, 2015.

F. Toyoshima and A. Barton. A formal representation of affordances as reciprocal dispositions. In *TriCoLore (C3GI/ISD/SCORE)*, volume 2347 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2018.

M. T. Turvey. Affordances and prospective control: An outline of the ontology. *Ecological psychology*, 4(3):173–187, 1992a.

M. T. Turvey. Ecological foundations of cognition: Invariants of perception and action. *American Psychological Association*, 1992b.

A. Umbrico, A. Orlandini, and A. Cesta. An ontology for human-robot collaboration. *Procedia CIRP*, 93:1097–1102, 2020. ISSN 2212-8271. doi: 10.1016/j.procir.2020.04.045.

J. Urbani, S. Kotoulas, E. Oren, and F. van Harmelen. Scalable distributed reasoning using mapreduce. In *ISWC*, volume 5823 of *Lecture Notes in Computer Science*, pages 634–649. Springer, 2009.

M. Uschold and M. Grüninger. Ontologies: Principles, methods and applications. *Knowledge Engineering Review*, 11(2):93–155, 1996.

M. Uschold, M. King, S. Moralee, and Y. Zorgios. The enterprise ontology. *The knowledge engineering review*, 13(1):31–89, 1998.

K. M. Varadarajan and M. Vincze. Afrob: The affordance network ontology for robots. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 1343–1350. IEEE, 2012.

D. Vernon. *Artificial cognitive systems: A primer*. MIT Press, 2014.

D. Vernon, G. Metta, and G. Sandini. A survey of artificial cognitive systems: Implications for the autonomous development of mental capabilities in computational agents. *IEEE transactions on evolutionary computation*, 11(2):151–180, 2007.

L. Vieille. A database-complete proof procedure based on sld-resolution. In *ICLP*, pages 74–103. MIT Press, 1987.

A. Voulodimos, N. Doulamis, A. Doulamis, E. Protopapadakis, and D. Andina. Deep learning for computer vision: A brief review. *Intell. Neuroscience*, 2018, Jan. 2018. ISSN 1687-5265. doi: 10.1155/2018/7068349.

A. Vyas, D. Beßler, and M. Beetz. Inferring dispositions from object shape and material with physics game engine modelling. In *JOWO*, volume 2969 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2021.

M. Waibel, M. Beetz, R. D'Andrea, R. Janssen, M. Tenorth, J. Civera, J. Elfring, D. Gálvez-López, K. Häussermann, J. Montiel, A. Perzylo, B. Schießle, O. Zweigle, and R. van de Molengraft. RoboEarth - A World Wide Web for Robots. *Robotics & Automation Magazine*, 18(2):69–82, 2011.

A. Y. Wang, J. H. Sable, and K. A. Spackman. The SNOMED clinical terms development process: refinement and analysis of content. In *Proceedings of the AMIA Symposium*, page 845. American Medical Informatics Association, 2002.

M. Warnier, J. Guitton, S. Lemaignan, and R. Alami. When the robot puts itself in your shoes. managing and exploiting human and robot beliefs. In *2012 IEEE RO-MAN: The 21st IEEE International Symposium on Robot and Human Interactive Communication*, pages 948–954. IEEE, 2012.

J. Wielemaker, G. Schreiber, and B. Wielinga. Prolog-based infrastructure for RDF: performance and scalability. In D. Fensel, K. Sycara, and J. Mylopoulos, editors, *The Semantic Web - Proceedings ISWC'03, Sanibel Island, Florida*, pages 644–658, Berlin, Germany, October 2003. Springer Verlag. LNCS 2870.

J. Winkler, M. Tenorth, A. K. Bozcuoglu, and M. Beetz. CRAMm – memories for robots performing everyday manipulation activities. *Advances in Cognitive Systems*, 3:47–66, 2014.

N. Yamanobe, W. Wan, I. G. Ramirez-Alpizar, D. Petit, T. Tsuji, S. Akizuki, M. Hashimoto, K. Nagata, and K. Harada. A brief review of affordance in robotic manipulation research. *Journal of the Robotics Society of Japan*, 36: 327–337, 01 2018. doi: 10.7210/jrsj.36.327.

H. A. Yanco and J. L. Drury. A taxonomy for human-robot interaction. In *Proceedings of the AAAI Fall Symposium on Human-Robot Interaction*, pages 111–119, 2002.

H. A. Yanco and J. L. Drury. Classifying human-robot interaction: an updated taxonomy. In *SMC (3)*, pages 2841–2846. IEEE, 2004.

D. Yang, M. Dong, and R. Miao. Development of a product configuration system with an ontology-based approach. *Comput. Aided Des.*, 40:863–878, 2008.

F. Yazdani, G. Kazhoyan, A. K. Bozcuoglu, A. Haidu, F. Balint-Benczedi, D. Beßler, M. Pomarlan, and M. Beetz. Cognition-enabled framework for mixed human-robot rescue teams. In *IROS*, pages 1421–1428. IEEE, 2018.

A. K. Zaidi and L. W. Wagenhals. Planning temporal events using point–interval logic. *Mathematical and Computer Modelling*, 43(9):1229–1253, 2006. ISSN 0895-7177. doi: 10.1016/j.mcm.2005.05.018.

I. Zaplana, J.-A. Claret, and L. Basanez. Kinematic analysis of redundant robotic manipulators: application to kuka lwr 4+ and abb yumi. *Revista Iberoamericana de Automática e Informática industrial*, 15, 11 2017. doi: 10.4995/riai.2017.8822.

A Prior Publications

The work presented herein is in parts based on prior work that has been published in international journals, international conferences, and as book chapters. The related prior work is referred to in the respective sections of this thesis that are based on it. For the sake of completeness, this appendix provides a complete list of my scientific publications.

Book Chapters

- D. Beßler, A. K. Bozcuoglu, and M. Beetz. Information system for storage, management, and usage for embodied intelligent systems. In C. S. Große and R. Drechsler, editors, *Information Storage: A Multidisciplinary Perspective*, pages 135–159. Springer International Publishing, 2020a. doi: 10.1007/978-3-030-19262-4_5

- M. Beetz, S. Stelter, D. Beßler, K. Dhanabalachandran, M. Neumann, P. Mania, and A. Haidu. Robots collecting data: Modelling stores. In L. Villani, C. Natale, M. Beetz, and B. Siciliano, editors, *Robotics for Intralogistics in Supermarkets and Retail Stores*. Springer International Publishing, 2022. Accepted for publication

Journal Papers

- M. Diab, M. Pomarlan, D. Beßler, A. Akbari, J. Rosell, J. A. Bateman, and M. Beetz. Skillman - A skill-based robotic manipulation framework based

on perception and reasoning. *Robotics Auton. Syst.*, 134:103653, 2020a

- A. Olivares-Alarcos, D. Beßler, A. Khamis, P. Gonçalves, M. Habib, J. Bermejo, M. Barreto, M. Diab, J. Rosell, J. Quintas, J. Olszewska, H. Nakawala, E. Pignaton de Freitas, A. Gyrard, S. Borgo, G. Alenyà, M. Beetz, and H. Li. A review and comparison of ontology-based approaches to robot autonomy. *The Knowledge Engineering Review*, 34, 12 2019. doi: 10.1017/S0269888919000237

- G. Bartels, D. Beßler, and M. Beetz. Episodic memories for safety-aware robots - knowledge representation and reasoning for robots that safely interact with human co-workers. *Künstliche Intell.*, 33(2):123–130, 2019

- M. Tenorth, J. O. Winkler, D. Beßler, and M. Beetz. Open-ease: A cloud-based knowledge service for autonomous learning. *Künstliche Intell.*, 29(4): 407–411, 2015

Conference Papers

- D. Beßler, R. Porzel, M. Pomarlan, A. Vyas, S. Höffner, M. Beetz, R. Malaka, and J. Bateman. Foundations of the Socio-physical Model of Activities (SOMA) for autonomous robotic agents. In *FOIS*, volume 344 of *Frontiers in Artificial Intelligence and Applications*, pages 159–174. IOS Press, 2021. doi: 10.3233/FAIA210379

- D. Beßler, R. Porzel, M. Pomarlan, M. Beetz, R. Malaka, and J. A. Bateman. A formal model of affordances for flexible robotic task execution. In *ECAI*, volume 325 of *Frontiers in Artificial Intelligence and Applications*, pages 2425–2432. IOS Press, 2020b. doi: 10.3233/FAIA200374

- M. Diab, M. Pomarlan, D. Beßler, A. Akbari, J. Rosell, J. A. Bateman, and M. Beetz. An ontology for failure interpretation in automated planning and execution. In *ROBOT (1)*, volume 1092 of *Advances in Intelligent Systems and Computing*, pages 381–390. Springer, 2019b

- D. Beßler, M. Pomarlan, and M. Beetz. OWL-enabled assembly planning for robotic agents. In *Proceedings of the 17th International Conference on*

Autonomous Agents and MultiAgent Systems, AAMAS '18, pages 1684–1692, Richland, SC, 2018c. International Foundation for Autonomous Agents and Multiagent Systems. Finalist for the Best Robotics Paper Award

- D. Beßler, M. Pomarlan, A. Akbari, Muhayyuddin, M. Diab, J. Rosell, J. A. Bateman, and M. Beetz. Assembly planning in cluttered environments through heterogeneous reasoning. In *KI*, volume 11117 of *Lecture Notes in Computer Science*, pages 201–214. Springer, 2018b

- M. Beetz, D. Beßler, A. Haidu, M. Pomarlan, A. K. Bozcuoglu, and G. Bartels. Knowrob 2.0 – A 2nd generation knowledge processing framework for cognition-enabled robotic agents. In *2018 IEEE International Conference on Robotics and Automation, ICRA 2018, Brisbane, Australia, May 21-25, 2018*, pages 512–519, 2018. doi: 10.1109/ICRA.2018.8460964

- F. Yazdani, G. Kazhoyan, A. K. Bozcuoglu, A. Haidu, F. Balint-Benczedi, D. Beßler, M. Pomarlan, and M. Beetz. Cognition-enabled framework for mixed human-robot rescue teams. In *IROS*, pages 1421–1428. IEEE, 2018

- A. Haidu, D. Beßler, A. K. Bozcuoglu, and M. Beetz. KnowRobSIM - game engine-enabled knowledge processing towards cognition-enabled robot control. In *IROS*, pages 4491–4498. IEEE, 2018

- J. Crespo, R. Barber, Ó. M. Mozos, D. Beßler, and M. Beetz. Reasoning systems for semantic navigation in mobile robots. In *IROS*, pages 5654–5659. IEEE, 2018

- J. A. Bateman, M. Beetz, D. Beßler, A. K. Bozcuoglu, and M. Pomarlan. Heterogeneous ontologies and hybrid reasoning for service robotics: The EASE framework. In *ROBOT (1)*, volume 693 of *Advances in Intelligent Systems and Computing*, pages 417–428. Springer, 2017

- M. Beetz, D. Beßler, J. O. Winkler, J. Worch, F. Balint-Benczedi, G. Bartels, A. Billard, A. K. Bozcuoglu, Z. Fang, N. Figueroa, A. Haidu, H. Langer, A. Maldonado, A. L. P. Ureche, M. Tenorth, and T. Wiedemeyer. Open robotics research using web-based knowledge services. In *ICRA*, pages 5380–5387. IEEE, 2016

- F. Balint-Benczedi, T. Wiedemeyer, M. Tenorth, D. Beßler, and M. Beetz. A knowledge-based approach to robotic perception using unstructured information management. In *AAMAS*, pages 1941–1942. ACM, 2015
- A. K. Bozcuoglu, D. Beßler, and M. Beetz. Rendering semantically-annotated experiment videos out of robot memories. In *Humanoids*, pages 541–546. IEEE, 2015a
- M. Beetz, G. Bartels, A. Albu-Schäffer, F. Balint-Benczedi, R. Belder, D. Beßler, S. Haddadin, A. Maldonado, N. Mansfeld, T. Wiedemeyer, R. Weitschat, and J. Worch. Robotic agents capable of natural and safe physical interaction with human co-workers. In *IROS*, pages 6528–6535. IEEE, 2015b

Other Publications

- D. Beßler, R. Porzel, P. Mihai, and M. Beetz. Foundational models for manipulation activity parsing. In T. Jung and M. C. tom Dieck, editors, *Extended Reality - XR in Times of Crisis*. Springer, 2022. Accepted for Publication
- D. Beßler, S. Jongebloed, and M. Beetz. Prolog as a querying language for MongoDB, 2021. URL `https://arxiv.org/abs/2110.01284`
- M. Diab, M. Pomarlan, S. Borgo, D. Beßler, J. Rossel, J. A. Bateman, and M. Beetz. Failrecont - an ontology-based framework for failure interpretation and recovery in planning and execution. In *JOWO*, volume 2969 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2021
- A. Vyas, D. Beßler, and M. Beetz. Inferring dispositions from object shape and material with physics game engine modelling. In *JOWO*, volume 2969 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2021
- M. Diab, M. Pomarlan, D. Beßler, S. Borgo, and J. Rosell. "knowing from" - an outlook on ontology enabled knowledge transfer for robotic systems. In *JOWO*, volume 2708 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2020b

- D. Beßler, S. Koralewski, and M. Beetz. Knowledge representation for cognition- and learning-enabled robot manipulation. In *CogRob@KR*, volume 2325 of *CEUR Workshop Proceedings*, pages 11–19. CEUR-WS.org, 2018a
- G. Bartels, M. Beetz, D. Beßler, M. Tenorth, and J. O. Winkler. How to use openEASE: An online knowledge processing system for robots and robotics researchers (demonstration). In *AAMAS*, pages 1925–1926. ACM, 2015
- A. K. Bozcuoglu, F. Yazdani, D. Beßler, B. Togorean, and M. Beetz. Reasoning on communication between agents in a human-robot rescue team. In *Towards Intelligent Social Robots – Current Advances in Cognitive Robotics*, Seoul, Korea, 2015b