

Storing and Predicting User-Specific Action Chains in Assistive Robots

Thesis by
Jeremias Thun

In Partial Fulfillment of the Requirements for the
Degree of
Master of Science



UNIVERSITÄT BREMEN
Bremen

Winter Term 2024/25

First Assesor: Prof. Daniel Leidner PhD
Second Assesor: Prof. Michael Beetz PhD

Supervision: Samuel Bustamante-Gomez
Advisors: Daniel Beßler, Florian Lay

© Winter Term 2024/25

Jeremias Thun

Matrikelnummer: 4433587

All rights reserved

TABLE OF CONTENTS

Abstract	1
Chapter 1: Introduction	2
1.1 Contributions	3
1.2 Thesis Outline	3
1.3 About the use of AI in writing this thesis	4
Chapter 2: Background	5
2.1 The Assistive Robot EDAN	5
2.2 EDAN's Object Database (odb)	6
2.3 Modern Ontologies	8
2.4 Underlying Concepts of the DUL and SOMA Ontology	12
Chapter 3: <i>What just happened?</i> A memory for the robot Edan	15
3.1 Introduction	15
3.2 Related Work	15
3.3 Mapping Existing Concepts	16
Mapping odb Objects with SOMA	17
Mapping odb Tasks with SOMA	19
3.4 Introducing new Entities in the odb Ontology	20
Idle Task	20
Imagining Available and Preferred Tasks	21
Semantic Distinction Between Times of Day	22
3.5 Interfacing with the EDAN Framework	23
3.6 Summary	24
Chapter 4: <i>What would you want?</i> Calculating Preferences	25
4.1 Introduction	25
4.2 Related Work	26
4.3 Overview on Calculating Predictions, Introducing the PREFERENCEORACLE	27
4.4 The Algorithm and its Hyperparameters	31
Comparing Task Types and Object Types	31
Comparing Situations	36
Looking Further Back	37
Comparing Time of Day	40
Multiple Occurrences of the same Candidate Situation in History	40
Looking into the Future, Calculating with $N > 1$ in the N-Gram	41
4.5 Hyperparameter Tuning based on Data	43
4.6 Modeling Predictions	44
4.7 Summary	45
Chapter 5: Evaluation	47
5.1 Introduction	47
5.2 Generating and Curating the Daily Living Dataset	48
5.3 Methodology	50
5.4 Results	54
5.5 Discussion	60
5.6 Overview over the Hyperparameters	62

5.7 Summary	63
Chapter 6: Conclusion	65
6.1 Thesis Summary	65
6.2 Further Research	66
6.3 Future Work on the NEEM Logger	66
6.4 Future Work on the PREFERENCEORACLE	67
Including the Preference Calculations in a User Interface	67
Remove Invisible Byproducts of Changes in the Hyperparameters or the Ontology.	67
Improving the Predictions of a Further Future	68
Including more Robotic Properties in the Preferences	68
Preparing the Environment	69
Evaluating the Prediction Accuracy with a Real User Base	69
Bibliography	70
List of Illustrations	75
List of Tables	78
Appendix A: The NEEM logger	80
Appendix B: Preferences	83

ABSTRACT

Descriptive logging of robotic actions requires a structured knowledge base that is capable of describing a variety of information, ranging from task descriptions over object properties to decision-making processes. Researchers have developed ontologies to structure robotic knowledge and also providing a rich taxonomy of concepts. Integrating such an ontology into an existing robotic framework remains a challenge, unique for different robotic contexts. This thesis describes the work of integrating the SOMA ontology into the assistive robot EDAN and building an algorithm for semantic logging of episodic memories.

This structured knowledge base not only allows ex-post evaluation of robotic experiments, but also enables online algorithms to make sense of the robotic actions. Leveraging the highly structured logging of task chains of the assistive robot EDAN, this thesis proposes a light-weight algorithm to predict the next action a user might want the robot to take based on the previously executed tasks, the available tasks, and the time of day. With privacy issues in mind, the algorithm renounces any large-scale training. With the N-Gram algorithm, which has been well established for next-word prediction, as a role model, this thesis proposes a next-task prediction based solely on a chain of previously executed tasks.

Chapter 1

INTRODUCTION

Assistive robots have received significant attention thanks to their prospective ability to help elderly people, as well as people with motor impairments, live their lives as self-determined as possible. Assistive robots can help perform the necessary tasks, from taking a cup of water over setting a table to cooking a meal. However, a big obstacle to overcome is an often perceived mismatch between what the robot has to offer and what people expect and require in usability. Although parts of this can be overcome by adding more functionality to the robot, the perceived mismatch is partly due to imperfect communication of existing functionality and a lack of considerations of user experience in robotic development (see [10]).

With the very wide area of application for assistive robots on the one hand and the different technical challenges provided by a variety of tasks in a human household, research is required to balance both application-specific considerations and scalability for robotic frameworks. This project focuses on two distinct, yet complementary, projects in this regard to bring both of them together: The robot EDAN by the DLR and the SOMA ontology by the University Bremen.

The German Aerospace Center (DLR) has developed the assistive robot EDAN, that is built on top of a wheelchair for the very specific case of helping motor impaired people depending on a wheelchair to autonomously execute tasks in a normal human household (see [53]). The robotic arm is physically attached to the wheelchair and has been shown to be capable of opening doors for the wheelchair to roll through, picking objects out of drawers or a fridge, or placing drinks right next to a users mouth, allowing the user to drink from a straw without the need to move their head. The internal knowledge representation of the robotic framework has been adapted to support the different areas of research done with the robot and has expanded over the years of research.

The SOMA ontology, developed by University Bremen, has followed the different approach of modeling knowledge of robots and even non-robotic environments and activities done by anyone. The ontology aims to provide a very general taxonomy of activities, their social and physical relationships, and the objects involved (see [12]).

This work is building on both projects and proposes an interface between them, acknowledging on the one hand the application-specific requirements of the EDAN framework and the necessity to keep internal structures of objects and tasks to support all ongoing research projects within the EDAN framework. While keeping the internal logic of the EDAN framework, this project aims at leveraging the more general concepts of the SOMA ontology and incorporating the EDAN concepts into the SOMA taxonomy.

First, this allows the EDAN framework to take advantage of the SOMA concept of narrative-enabled episodic memories (see [15]) and record robotic experiences in a form defined by the SOMA ontology. Such a memory allows not only postprocessing of robotic experiments, but also comparability of experiences by the EDAN robot with experiences by other robotic frameworks that have been built on top of the SOMA ontology.

Such a semantic log of robotic actions lies the foundation for this thesis to also propose a task prediction algorithm that anticipates tasks that a user sitting in EDAN might want to execute next. The semantic log of past actions allows the algorithm to predict next actions without requiring any pre-training or leak of robot usage data to a third party. The algorithm is able to run completely locally. By predicting the next actions, the proposed algorithm can help users navigate through a larger list of available actions by ranking actions that have been executed more often by the user in a similar situation higher than others.

In a different project, the work presented here has also been used to enable EDAN to communicate its current and past states in natural language. The RACOON framework uses modern large language models and a RAG system to read information, among others, from the semantic log presented in this thesis and tell the user what happened at a specific point in time (see [18]).

1.1 Contributions

This thesis aims at improving robotic help for people with motor impairment to allow them to live in a self-determined way. This is done by two main contributions:

1. This thesis integrates the robotic framework EDAN with the ontology SOMA and further develops them into a new knowledge representation for the EDAN robot, which allows for a structured semantic logging of EDAN's past actions in the form of narrative-enabled episodic memories (NEEMs).
2. This log builds the foundation for a novel next task prediction algorithm, proposed in this thesis, that is solely based on the chain of previously executed tasks, their context, and their taxonomic relationships.

Some frameworks have already been developed using the framework presented in this thesis and published independently, as shown in the conclusion chapter, Section 6.2 on page 66.

1.2 Thesis Outline

This thesis first introduces the reader to the preexisting frameworks of a) the robot EDAN and b) the SOMA ontology. In Chapter 2, both frameworks will be presented in relation to their respective fields of research. The following Chapter 3 introduces the field of ontology matching and describes the process and challenges of integrating the SOMA ontology into the EDAN framework. The thereby presented NEEM Logger algorithm automatically integrates both knowledge representations and logs the environment as well as actions and

decisions made by either the robot EDAN or the person sitting in EDAN. Chapter 4 builds on the NEEM logger by suggesting an algorithm that predicts the next task a user sitting in EDAN might want to execute. The algorithm has been evaluated using synthetic data generated by large language models, with the results presented in Chapter 5. Finally, Chapter 6 gives a brief summary of the thesis and concludes with a reference to another paper that has already made use of the NEEM Logger presented herein, as well as a future outlook.

1.3 About the use of AI in writing this thesis

The contents of this thesis have been developed by the author and the thesis has been written by the author. To enhance readability, local Large Language Models have been used to detect errors in grammar and sentence structure. No changes in this thesis were the result of an AI alone. Every mistake regarding grammar or sentence structure detected by an AI has been manually examined by the author and carefully checked and corrected. The author claims full responsibility for all content of this thesis.

Chapter 2

BACKGROUND

2.1 The Assistive Robot EDAN

There are several reasons people develop or are born with various types of disability that severely limit their ability to perform tasks that others might take for granted in their daily life. Assistive robotics is part of robotic research that aims to help people with disabilities achieve those tasks with the help of a robot and thereby try to help regain autonomy. This can range from moving robots that can be externally controlled by disabled people to help them perform some tasks to robotic prostheses or additional exoskeletons that are increasingly part of the user itself. The latter requires a lot more automatic detection of the users intention, as it is designed to seamlessly support the users movements, whereas external robots can lean more in the direction of executing specific commands given by the user.

A DLR research group has developed an "EMG-controlled Daily Assistant" (EDAN) that positions itself somewhere in between both categories [53]. It is designed for users who do not have all their limbs or cannot use them, which limits their ability to perform basic tasks of picking and moving in a normal household. EDAN is an electronic wheelchair with an attached robotic arm (a Light-Weight Robot III) that can be controlled by different input devices, such as joysticks or different Brain-Computer-Interfaces. The robotic arm can perform tasks that non-impaired people could perform with their hands, like reaching objects in front of the user or bringing objects to the user's mouth stopping just at the right distance without physically hurting them. The robotic arm is attached to the wheelchair in which the user is sitting. An intuitive way of using the arm is to control it manually and work towards the goal of enabling users to use the arm as if it were their own. Studies have found that with training sessions of 1-2 hours per day, people can learn to effectively control a robotic arm manually during a learning period of just three weeks. Afterwards, the task execution time for manual control is very close to automatic task execution (see [29]). However, manually controlling a robotic arm can be very difficult when it comes to finer movements like closing the fingers or pouring some liquid from one container to another, and is therefore tiresome to do all day. Even with enough training in manual mode, it takes much more effort to manually execute a task with a robotic arm than to automate it. The solution for the EDAN robot is to allow users to switch between different control modes. If the user feels up for the task, they can activate manual control and move the robotic arm with a joystick or EMG controller. This can be enhanced by a "shared control" mode (using shared control templates, i.e. **sct-files**), which detects the action that the user is attempting to execute and helps them execute that action by guiding the users manual controls: Similar to a lane assistant, the shared control mode limits the degrees of freedom that a user has to help them keep the robotic arm in the desired trajectory. An especially interesting example

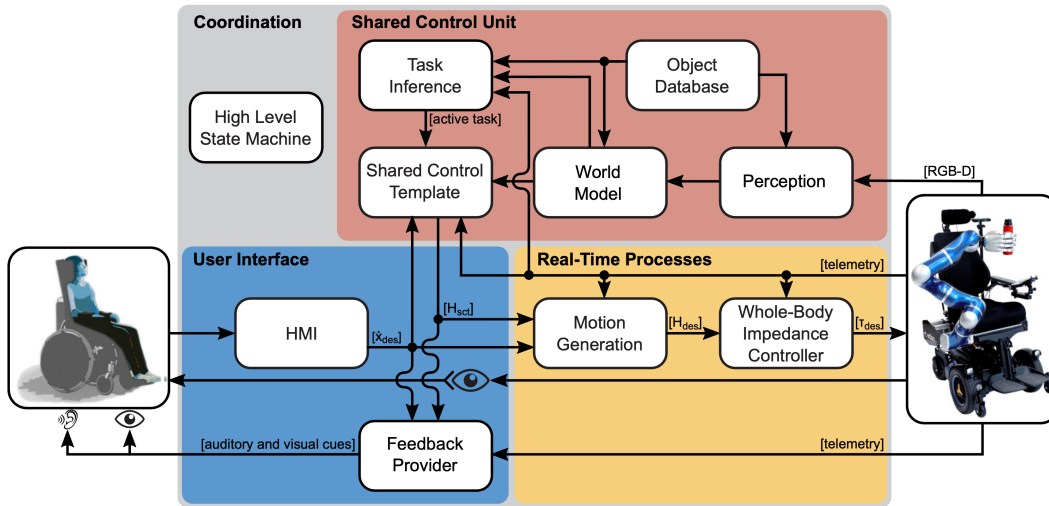


Figure 2.1: Illustration of EDAN's software components

©2020 IEEE. Reprinted, with permission, from Jörn Vogel et. al., *EDAN: An EMG-controlled Daily Assistant to Help People With Physical Disabilities*, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Oct. 2020 [53]

for this is the task of pouring a liquid from one container (typically a bottle of some kind) to another (typically a mug): The shared control mode allows the user to move forward with the pouring task and internally calculates the correct position of the bottle to be above the mug without spilling any liquid (see [47]). The robot internally calculates the desired position for the thermos bottle in order to avoid spillage. In case the user is getting tired of executing tasks manually, there is also an autonomy mode, where the robot takes full control and executes tasks without the user's help. The wheelchair is equipped with a tablet computer that constantly updates the user with a set of technical information as well as available actions. The user could, for example, use the tablet to click on the task to fetch a glass of water. By that the user allows the robot to autonomously move the wheelchair to the sink, grab a glass, fill it with water, and bring it up to the user's mouth.

To accomplish such tasks, EDAN is equipped with a set of high-level software components that allow the robot to perceive and store information about its surroundings and calculate high-level information like a description of the task the user is currently performing, the properties of the objects the user is interacting with, the availability of following tasks, and so on. While a broader illustration of the different software components controlling EDAN can be found in Figure 2.1, we will dive into three components that are most relevant for this thesis:

2.2 EDAN's Object Database (odb)

The robot EDAN is equipped with an **object database (odb)** (see [32]) that stores information about known objects with which the robot is able to interact. The information can vary from physical properties like bounding box information or the weight of the object over technical information like recommended poses the robot should take when grabbing the object to

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <object>
3   <name>thermos</name>
4   <author>...</author>
5   <derive>_bottle</derive>
6   <mass>0.1</mass>
7   <updir>0 0 1</updir>
8   <dimension>0.03 0.03 0.236</dimension>
9
10  <logic>
11    <code_snippet>pick_from_side_sct.yml</code_snippet>
12    <code_snippet>pour_sct.yml</code_snippet>
13  </logic>
14
15  ...
16
17  <geometry>
18    <display>thermosflasche.obj</display>
19    <collision>thermosflasche.obj</collision>
20  </geometry>
21
22  <grasping>
23    <graspset>edan_right_arm_power_clash.grasps</graspset>
24    <graspset>edan_right_arm_power_hit.grasps</graspset>
25  </grasping>
26 </object>

```

Figure 2.2: Example of a `manifest.xml` file for the object `thermos`, describing and linking all information about an object in the odb.

high-level planning information such as information about the tasks EDAN can perform with the object. From a technical point of view, the odb is a directory within the file system, containing subdirectories for every object, where all object-related information can be stored. The semi-structured character of the odb allows it to easily add new kinds of information to the odb and even store image data inside the odb. The odb also introduces an object hierarchy by requiring every object to at least have a `manifest.xml` file, that may point to one or more parent objects. A manifest file looks something like the example in Figure 2.2: It is an XML file that contains the name of an object, its potential parents, physical attributes like its mass and dimensions, a list of tasks that can be performed using the object, machine-readable instructions on how to grasp the object, and much more. All technical information regarding an object is currently collected and stored by hand. Different research projects currently explore ways to auto-generate the required data for interacting with objects, like for example a project to derive suitable grasping positions from visual data, therefore enabling the robot to grasp unknown objects (see [36]).

During EDAN’s runtime, a perception module films the environment and detects objects that are visible to the cameras attached to the wheelchair of EDAN. Those objects instantiate the classes defined by the odb in a belief state, called the **world representation** or the world model. The world representation represents everything the robot assumes about its current

environment and exposes the information to other software components (see [31]). Any object that moves out of EDAN's sight is immediately removed from the world representation, resulting in an image of the surrounding that is limited to the directly accessible area. In the current implementation, the belief state is not stored for later references at any point.

A **task inference** module accesses the belief state in the world representation to calculate the available tasks based on a given world state. Using a PDDL reasoner, the EDAN framework defines requirements for different tasks that need to be fulfilled before the tasks can be executed. For example, a container can only be closed if it is currently open, and an object can only be placed somewhere if the robot is first holding that object in its hand. Such requirements, which are defined as part of the task definitions stored in the odb, are applied to all objects available in the world representation, resulting in a list of available tasks. Based on the distance between the robot and the different objects, as well as parameters such as occlusion, the task inference module calculates a hierarchy for the available tasks from the closest to the farthest (see [53], [31], [9]). This is the basis for a list of actions, shown to the user on their tablet, that they can choose from in autonomy mode.

2.3 Modern Ontologies

"The term ontology has roots in philosophy and describes the study of being. In the computer science domain, an ontology is a 'formal, explicit specification of a shared conceptualization', i.e. an abstract model of real-world concepts that is represented in a computer-readable way and is shared by a group of stakeholders. The definition is technology-independent; conceptually, even an XML Schema could be interpreted as an ontology. While multiple ontology languages are available, most ontologies are typically defined in the W3C Web Ontology Language (OWL)." [46, p. 2643]¹

The ontologies in the OWL standard by the World Wide Web Consortium (W3C) form a graph in which nodes describe entities and edges describe the relationship between them. The most relevant for the scope of this thesis are *classes*, which describe the general concept of an object, *individuals*, which describe instances of classes, and *object properties*, which describe the relation between both. In the same way that a graph is a set of edges and vertices, an ontology is a set of entities and a list of labeled edges, i.e. relations, in the form of triples. Such triples are related to natural language, as they correspond to the form of (subject, predicate, object) (see [13], [42]). An example of a valid OWL triple could be (Mug, isA, Thing) or (me, hasType, Person), where Mug, Thing and Person are classes, me is an individual and hasType and isA are object properties. Both exemplary object properties also define very fundamental relations for ontologies, with isA declaring one class to be the subclass of another class, and hasType declaring an individual to be an instance of a class. However, those triples can also transport more domain-specific

¹Building on [14], [23], and [19].

information, such as `(kitchenCleaning, involvesAgent, Edan)`. In contrast to other forms of hierarchical knowledge representation, ontologies allow entities to have multiple super-classes. Quite in line with the philosophical background of ontologies, the topmost class from which all other classes derive in OWL ontologies is the class `Thing`.

Entities in OWL possess an Internationalized Resource Identifier (IRI), which uniquely identifies every entity. Those IRIs are built from the local name of an entity, such as `Mug` or `Person`, and a prefix that identifies the domain defining the said entity. The prefix and suffix are separated by a `#` symbol, resulting in an entity like `Thing` looking like this: `http://www.w3.org/2002/07/owl#Thing`. Since these domains are usually a long string, an ontology can also define abbreviations for domains, such as the domain of OWL specifications with the term `owl:`, resulting in the unique identifier `owl:Thing`. This allows any ontology to define universally unique entities as well as to define relationships between the local entities and some entities defined by other ontologies. An ontology can, for example, import another ontology and define its entities solely as children, refining some distinctions that another ontology has already defined (see [13]). So, to be precise, the triples mentioned previously usually look more like this: `(odb:kitchenCleaning, dul:involvesAgent, odb:Edan)`.

The knowledge base of autonomous robots is often organized as an ontology. Not just in the conceptual sense in which any kind of organized knowledge is, but in the well-defined sense of an ontology following the OWL standard. While EDAN's object database can (and will for the rest of this thesis) be understood as a form of an ontology, it does not satisfy the OWL standard but is a hierarchical organization system for defining objects, their parental relationships, and other properties. This thesis seeks to find a way to enrich EDAN's current knowledge representation by making it satisfy the OWL ontology standard.

With formalized OWL ontologies building a common backbone for robotic knowledge bases, the question about which ontology to use arises. In its fundamental idea, the semantic web, for which W3C ontologies have been designed, aims at creating a common machine-readable interface for knowledge throughout the world and introducing ontologies that use common concepts, which should make collected data comparable. However, the idea of comparable knowledge relies on researchers using the same ontology or, when using different ontologies, on a formal definition of equivalencies of concepts in the different ontologies. Both directions have been studied.

Unifying different ontologies to a single ontology used by different research teams has been, among others, the project of the DOLCE ontology developers, who are also providing the community with the better known ultra-light version **DUL** (see [21]), and an IEEE consortium suggesting the upper ontology **SUMO** (see [44]). A study in 2011 has found the ontologies DUL and IEEE's suggestion SUMO to be the most widely used ontologies (see [34]). Those ontologies aim at declaring many general relationships between entities. In practice, they mostly function as so-called *upper* ontologies: Research laboratories define

Term	KnowRob 1/2	ROSETTA	ORO	CARESSES	OROSU	PMK
Objects	Yes / Yes	Yes	Yes	No	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 2.1: By [41, p. 24]: List of relevant terms for the autonomous robotics domain, and their coverage in the different chosen works.

their own application ontology containing the formal definitions of concepts and use or focus on them in their laboratories, but define those concepts as subcategorizations of concepts introduced in one of these upper ontologies. Although research laboratories use different formal concepts, they all share a common definition of parent concepts. This shared foundation allows for some level of comparability between their findings. The comparability increases further with the introduction of a general mapping between the two upper ontologies (see [38]), a concept that will be further investigated in the next chapter.

For the more specific application of ontologies in robotics, the IEEE has been working on setting some standards. In 2015, a standard for the context of **Robotics and Automation** was published, which is based on the upper ontology SUMO (see [1]). In 2021, the very general scope of robotics and automation was further specified in another IEEE standard for **Autonomous Robotics** (see [2]). This second standard again builds on the upper ontology SUMO, but also provides an interface for the upper ontology DUL.

When it comes to practical applications in autonomous robots, two recent studies have compared various ontologies and ontology frameworks designed to organize the knowledge base for such robots based on their scope (see [41] and [35]). These studies highlight diverse use cases where publicly available ontologies excel and define a set of classes necessary for a semantic description of everything around autonomous robots, thereby facilitating ontology comparison based on these classes. In terms of usage, [41] identifies the KnowRob framework as the most widely used in autonomous robotics. Furthermore, KnowRob demonstrates the broadest coverage of concepts relevant to autonomous robotics, as evidenced in Table 2.1 by [41, p. 24]. The KnowRob framework (see [8]) has developed its own program and ontology, which has been further refined to form the **Socio-physical**

Model of Activities (SOMA) (see [12]).

This thesis proposes SOMA as the underlying ontology for a new knowledge representation in EDAN. Built upon the upper ontology DUL, SOMA provides comprehensive definitions for tasks, actions, objects, their attributes, including affordances, actors with goals, and much more.

The SOMA ontology also serves as the foundation for narrative-enabled episodic memories (NEEMs), a concept intended to create a detailed robotic memory of past situations that can be replayed with the precision of a cinematic experience. Achieving this requires integrating two distinct components of robotic memory. On the one hand, it includes a detailed protocol of hardware information. To simulate a situation, the memory should include the image data of the robotic vision as well as the movements of its internal hardware components, usually stored as a tf-tree. KnowRob is capable of storing such information within the framework as part of the semantic memory. On the other hand, NEEMs include a detailed *description* of what the situation semantically represents. That is where the SOMA ontology comes in and provides a set of concepts and their relationships to semantically describe everything that happens in a situation. In this context, the SOMA ontology has been designed to annotate low-level technical information with their semantic meaning, enabling learning algorithms of different types to understand what happened in a situation, analyze situations that involved mistakes, or simulate a situation for a user.

A broader ontology like SOMA also enables a new kind of collaboration between research labs. A wide variety of robots with different capabilities can be modeled in the same ontology, making their differences as well as similarities more apparent. More importantly, this allows for a shared ontology between experiments conducted by different robots with different capabilities. While a robot with arms might be able to pick up and move a cup to a different location, another robot without arms might only be able to transport the same cup but would require someone else to place it on the robot. A shared ontology between both robots enables comparability of their actions, even when the experimental setups differ significantly. The project OPENEASE [50] aims to collect a variety of recorded robotic experiments, modeled in the SOMA ontology, in the form of NEEMs. Such a large open-source collection of robotic actions can serve as a basis for learning desired behavior and transferring robotic knowledge from one robot to another. This even includes the possibility of recording and learning from NEEMs that are recorded in a virtual reality setup and are executed manually (see [24]).

The KnowRob framework allows for the recording, storage, and access of such NEEMs. Users can interact with previous memories using first-order logic, as well as information about the current world state or logical implications of the world state [8]. This can be done by prompting in Prolog, in which many predicates already define common queries that internally query the underlying MongoDB database. In recent iterations, KnowRob had also been enabled to recognize queries in a Mongolog language, which combines the first-order

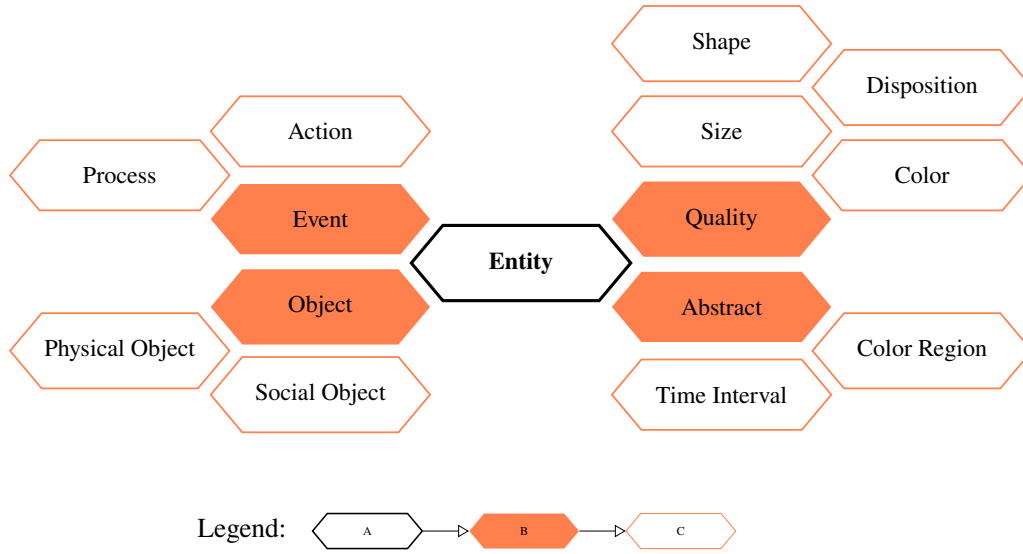


Figure 2.3: High level overview of some of the concepts in SOMA, building on the four basic concepts `Event`, `Quality`, `Object`, and `Abstract`.

predicate logic of Prolog with some aggregate commands for MongoDB (see [11]).

2.4 Underlying Concepts of the DUL and SOMA Ontology

The SOMA ontology, as it builds on the upper ontology DUL, fundamentally distinguishes between four top level concept, as depicted in Figure 2.3. There is the top-level concept of an `dul:Event`, which covers everything that is physically happening. While the most commonly used concept in this category is `dul:Action`, which describes the intentional modification of the environment by an agent, the SOMA Ontology also takes notion of other types of events, like the `dul:Process`, which describes a change that is "not strictly dependent on agents, tasks, and plans" ([21]). Disjoint from an Event is any type of `dul:Quality`. This top level concept describes various kinds of qualities that (mostly) objects might have. In addition to a set of intrinsic qualities, such as `soma:Size` or `soma:Color`, this concept also describes a set of extrinsic qualities such as `soma:Disposition`, which is used to annotate objects with a description of which tasks they can be used for. A third top-level concept in the SOMA ontology is an `dul:Abstract`, which is used to define abstract properties, other entities might have. In particular, this concept also defines possible values for different qualities, called `dul:Region`. For example, to describe a red mug, the color red is modeled as children of the `soma:ColorRegion`, as is already the case in the SOMA ontology. The mug then gets associated with the quality of having a `soma:Color`, which in turn points to the `soma:RedColor`. The fourth and last top level concept is a `dul:Object`, which can either be some `dul:PhysicalObject`, like a `soma:Cup`, or some `dul:SocialObject`, like a `dul:Task`. The distinction between physical and social entities is to be further discussed next.

SOMA aims at modeling various everyday activities, the objects involved, the purpose of tasks, the specific parameters of an action, and much more. For this purpose, the SOMA

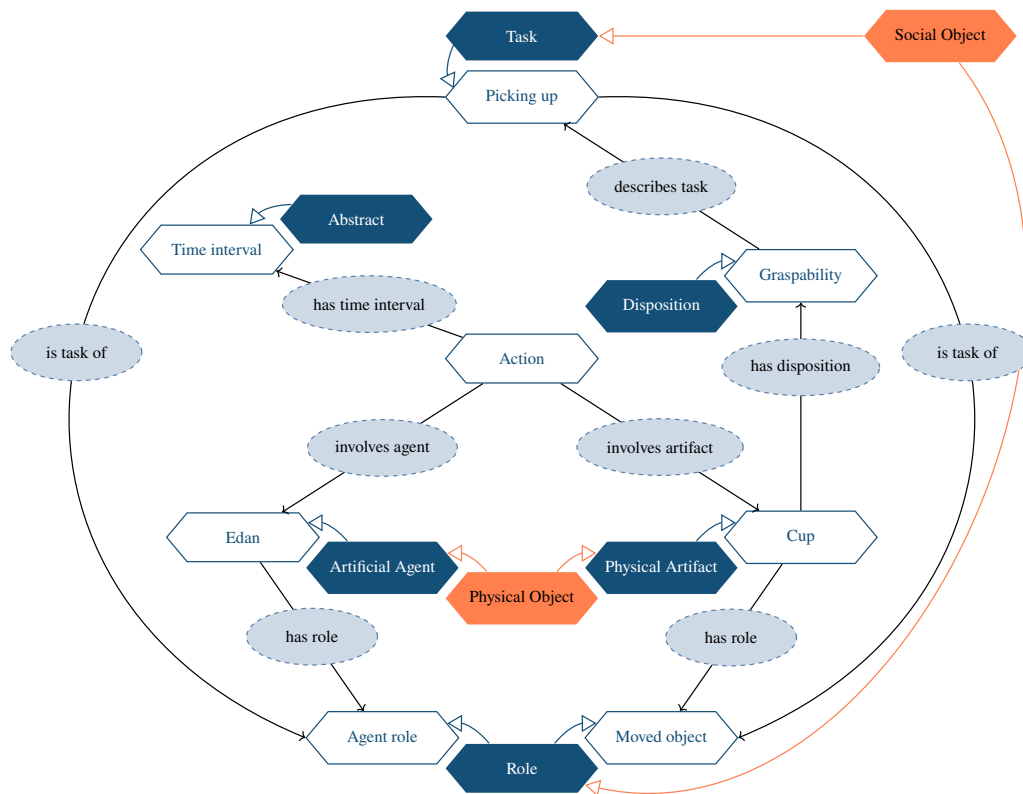


Figure 2.4: Excerpt of the way, an action executing the task of picking up a mug can be modeled in SOMA, highlighting the concepts of *Physical Object* (in blue) vs. *Social Object* (in orange).

ontology is built on top of a fundamental distinction between the general concept of different objects and their *role* in a specific situation. On the one hand, SOMA models the Concept of an object like a mug as a *DesignedContainer*, that is designed to contain liquids to drink from. This includes multiple features that tend not to change, like a color or a weight. However, in the context of different situations, the same mug can be used in different ways. The robot EDAN, for example, was trained not just to grab and help a user drink from a mug, but also to use a mug to roll some dice. In SOMA, this does not classify the mug as a "dice rolling object", but rather gives the mug a specific role in a specific situation: during that experiment, the mug played the *Role* of a container for dice. The *Role* of anything is dependent on the situation, perhaps even dependent on the viewer's perspective, and therefore changes over time. The same mug can, for example, later play the role of a destroyed object if the robot is not trained well and accidentally smashes the mug. In this way, the SOMA ontology contains on the one hand a *physical model* of the world and on the other hand a *social model* of roles and tasks.

To get a better grasp on this broader distinction between the physical model and the social model, we can think about the very simple action of picking up a mug. In the physical model, there is an *Action* happening that involves a robot, let us call it *Edan*, and an object, the *Mug*. We already know about the mug that it has the *Disposition* of being

able to be grasped. This is what is happening physically. But what do we know socially about the situation? We can assume that the robot not only picked up the mug, but someone implemented the *intention* to pick up that mug. Therefore, the Mug does not only physically allow to be grasped, but this Disposition is describing the **Task** that the robot intends to perform (as depicted in Figure 2.4 on the previous page). All technical commands to the robot had a *meaning* that we understand as humans: It is supposed to all culminate in the task of picking up the cup. Although both the robot EDAN and the mug are involved in the task, they both play a different role in it, depending on the intention of the task. When we think about the task of picking up a mug, we think about some agent who is supposed to do the picking and some mug that is supposed to be moved in the process. Those are the roles that both objects are *supposed* to play in the **Task** and they *are* playing those roles in an **Action**, where the task is executed. Depending on the social situation, the same Action, physically executing the same movements, resulting in the same physical changes to the world, could, however, also mean something completely different. In a setup where a researcher wanted to push the mug off the table, the robot picking up the mug is actually making a mistake and failing at a completely different task. It can also be stated that a single physical Action might be serving multiple intentions and completing multiple Tasks. On the other hand, a single Task might also require multiple physical Actions to be executed. To put it in the words of [41]:

"[T]asks 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." [41, p. 8]

Chapter 3

WHAT JUST HAPPENED?

A MEMORY FOR THE EDAN ROBOT

3.1 Introduction

The robotic framework EDAN has been designed to seamlessly interact with a user sitting in a wheelchair and perform several different tasks. While the multimodular system is capable of logging application-specific data within different modules, the framework has no central logging mechanism to record and store data on what the robot EDAN has done at what point in time. This chapter presents a mechanism for accumulating a high-level log that contains the perceived world state, assumptions about the environment at the time, inferred task options that are assumed to be available due to the given world state, and executed tasks.

To allow for efficient post-computing as well as an easy understanding of the log, a structured knowledge representation in the form of an ontology has been chosen. Ontologies provide a taxonomy that is both human- and machine-readable and for this reason well-established for representing data. This chapter presents the work of integrating the preexisting ontology SOMA, which provides a rich taxonomy for the context of autonomous robotics as well as the query framework KnowRob, into the robotic framework EDAN by matching the SOMA ontology with EDAN's pre-existing knowledge representation.

3.2 Related Work

For this chapter, the already mentioned object database (**odb**), in which objects and their properties and relationships are stored in the EDAN framework, will be considered an ontology, even though it lacks the formal representation, an OWL ontology requires. The task of interfacing such an ontology with the very much formalized SOMA ontology with a rich taxonomy that goes far beyond an object hierarchy, can therefore be understood as a merging of two ontologies into one. This is a task that has been researched quite extensively with the Ontology Alignment Evaluation Initiative (OAEI) regularly dedicating workshops to compare different approaches to matching ontologies (see [37]). This research has resulted in a set of approaches to finding similarities between different ontologies, mostly concerning larger ontologies that are too big to compare manually (see [39]). The assumption of ontology matching problems is usually, that the two ontologies to be compared model overlapping concepts, such that an alignment between two representation of either the same or very similar concepts is needed. This alignment is in many cases an equivalence relation between the concepts of different ontologies, but is not required to. Some ontology matching algorithms have also developed algorithms to find more complex relationships between two ontologies (see [51]).

Finding similarities between different ontologies requires some notion of what constitutes a similarity. To determine this similarity, every matching algorithm needs some sources of background information regarding the meaning of different entities. Those sources may range from lexical similarities over factual databases to the association of a machine learning algorithm such as a large language model (see [45]). However, these approaches still require the ontologies to be somewhat well described for a matching algorithm to find similarities. If, for example, a research team always uses the same cup for its robotic experiments, they may represent that cup as `cup` in the ontology, but implicitly expect a `cup` to have the exact properties that the cup in their experiment has. While such a modeling is an obvious bad practice for the example case, this does get more blurry when talking about concepts like a robotic arm or a kitchen: No ontology is able to fully describe every aspect of the world and, therefore, always operates with some implicit assumptions.

3.3 Mapping Existing Concepts

EDAN's odb is an ontology that gets updated regularly to accommodate the requirements of different research topics and competitions. Although some objects are well-described objects from the scope of a human kitchen, such as a `banana`, other objects are more abstract, such as a $10\text{cm} \times 10\text{cm}$ cube that EDAN has shown to be able to pick up in a competition. With the limited scope of the odb, all of those very different objects are modeled to be very closely related, which does not always fit the intuitive perception of those objects. On the other hand, the robotic framework EDAN relies on a certain object hierarchy that should not just be overturned. This has led to the decision to limit the number of changes to be made within the object database drastically and find ways to attach most of the pre-existing hierarchy to the SOMA ontology. On the other hand, the alignment of both ontologies should also serve the purpose of making knowledge, represented in the new EDAN framework, understandable for other projects, implementing SOMA.

The odb with around 100 objects and around 10 tasks in total is very limited in size, allowing for a detailed manual comparison of those concepts with their counterpart in the SOMA Ontology. For such an application, a heavy-weight ontology comparison does not seem necessary, especially considering the goal to change as little as possible in the relative object hierarchy in the odb. Since the odb does mostly describe the very application-specific environment at the DLR research lab as well as some competition setups, it has also been decided that concepts in the odb should not get the equivalence relation to any of the SOMA concepts. Even when not formally described, a mug in the odb is generally a mug with parameters similar to the ones that EDAN was trained to interact with. To be able to keep this implicit knowledge, odb concepts should rather be modeled as children to the corresponding SOMA concept, thus allowing for comparability on the one hand and distinction between a general term and an object in EDAN's world on the other hand. The ontology prefixes are helpful in this manner, as they allow to distinguish between a `soma:mug` and an `odb:mug`.

The result is a novel odb ontology, that is a specification of the more general SOMA

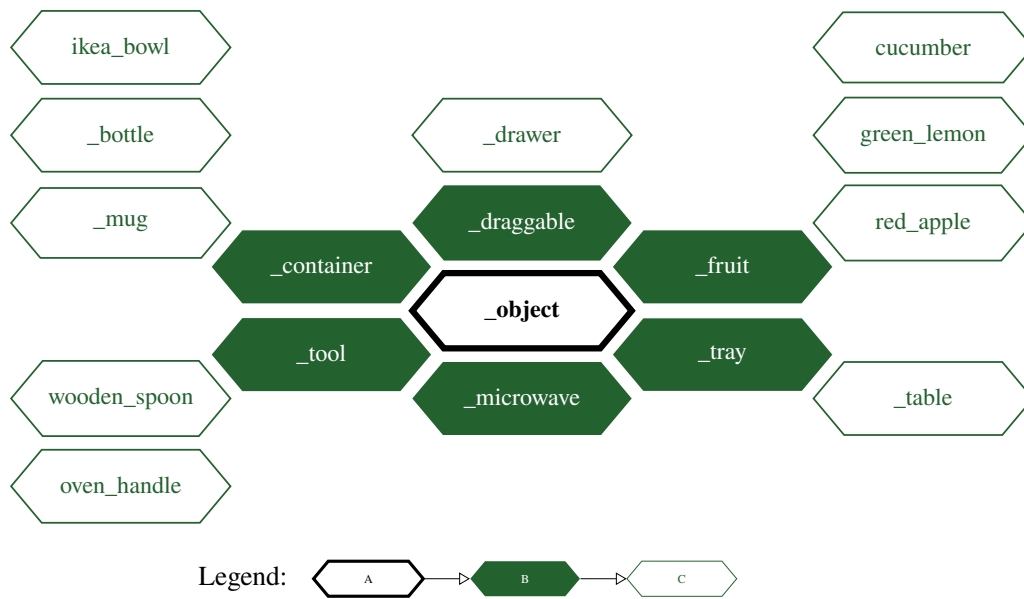


Figure 3.1: Excerpt of the object hierarchy in EDAN's odb.

While objects like `ikea_bowl` have no further children, objects like `_bottle` are themselves parent concepts for several children objects not depicted in this figure.

ontology in that it incorporates most of the information, the odb currently provides, as part of an ontology.

Mapping odb Objects with SOMA

As the odb models both objects to interact with as well as the robot itself and humans as objects, SOMA's distinction between a `dul:DesignedArtifact`, a `dul:PhysicalAgent`, and a `dul:PhysicalBody` poses the most changes to odb's internal hierarchy. SOMA therefore distinguishes between manmade designed objects (`dul:DesignedArtifact`) and objects that are the result of some kind of process, be it chemical, biological, geographical, etc.. In the odb ontology, this distinction concerns for once the objects `_robot` and `human`, which describe the physical form of an agent rather than an object for the robot to interact with. And, on the other hand, this concerns the object type `_fruit`, which is an object that is not designed, therefore varies greatly in size, shape, firmness, etc. across instances of the same object, and should be categorized as a `dul:PhysicalBody`:

```
(odb:_robot, rdfs:subClassOf, soma:ArtificialAgent)
(odb:human, rdfs:subClassOf, dul:NaturalPerson)
(odb:fruit, rdfs:subClassOf, soma:BiologicalObject)
```

To keep the underlying object hierarchy of the odb intact, only the top-most concepts of the odb are being linked with the SOMA ontology. Odb objects that derive from those topmost concepts are automatically being sorted into the ontology as their children.

```

1 _robot: soma:ArtificialAgent
2 _tool: soma:DesignedTool
3 human: dul:NaturalPerson
4 _gripper: soma:Gripper
5 _container: soma:DesignedContainer
6 _coffee_machine: soma:Appliance
7 _content: soma:DesignedSubstance
8 _fruit: soma:BiologicalObject
9 _microwave: soma:Appliance
10 _phone: soma:DesignedTool
11 _tray: soma:Surface
12 _cube: soma:DesignedComponent
13 edans_ikea_kitchen_door: soma:DesignedComponent

```

Listing 3.1: List of manually defined subclass relations between odb objects and SOMA objects

The topmost concept in the odb is called `_object`, even though, not every object derives from that topmost concept. One of the most important concepts that derive from `_object` is the `_draggable`, contains all kinds of drawers, with which EDAN is trained to interact. On the other hand, `_container` describes objects such as bottles and mugs, which contain something. The door, with which EDAN has conducted many experiments, is also modeled as direct children of `_object` together with the abstract concept `_tool`, containing all tools ranging from a door handle to a bottle opener. EDAN has also been trained to operate two machines, a `_coffee_machine` and a `_microwave`, which are also modeled as direct children of `_object`. For a better overview, Figure 3.1 on the preceding page shows an excerpt of the object hierarchy in the odb.

All of these concepts in the odb can be considered subclasses of the SOMA concept `dul:DesignedArtifact`. Without going into too much depth of the SOMA objects, the notion of a `dul:DesignedArtifact` offers a small set of high-level distinctions between different kinds of object that might be beneficial for the new odb ontology: `soma:Appliance` describes objects that have been built to perform a task and be operated by some agent. A `soma:DesignedContainer` covers other objects in itself and is able to limit the object's ability to move. `soma:DesignedSubstance` is describing substances, such as fluids. And `soma:DesignedTool` describes any deliberately made tool that an agent might use to accomplish tasks.

Reading this description already gives some hints on reasonable similarities between some top-level odb and SOMA classes. For every object currently used in the odb that is directly derived from the top-level object `_object` to get matched with a parent concept in the SOMA ontology, there are just 13 assignments to be made in total, which can be found in Listing 3.1.

Figure 3.2 on the following page helps visualize the finished alignment, depicting the SOMA ontology in blue and the newly attached concepts for the odb ontology in green. All objects that are modeled later and derive from `_object` in the odb, are automatically merged as children of the most abstract concept in the object ontology, the `dul:PhysicalObject`.

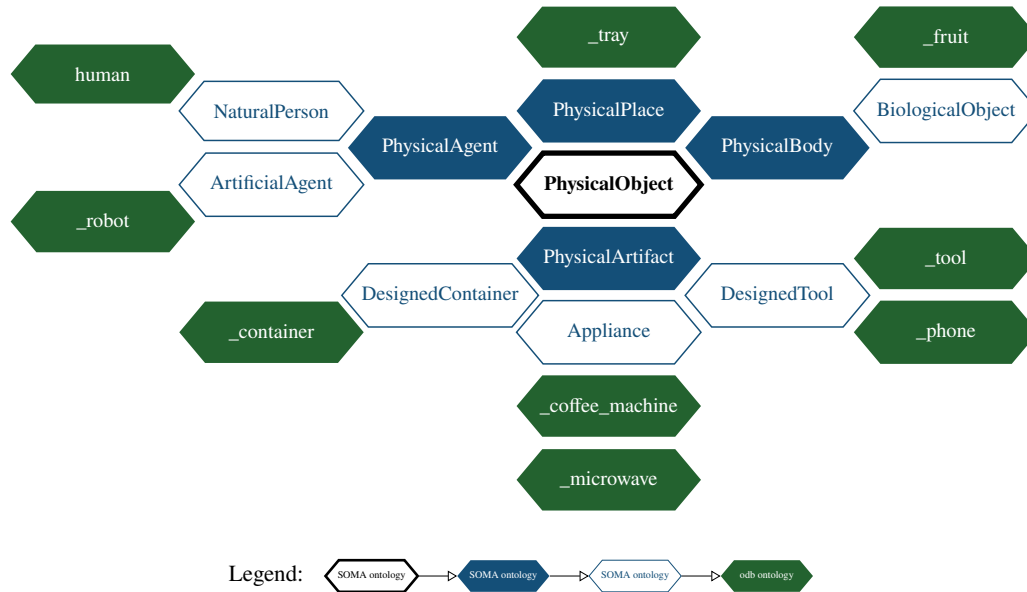


Figure 3.2: Excerpt of the alignment of SOMA objects with odb objects with the newly attached odb concepts in green

Note that some of the object definitions in the odb have a training underscore in their name. In the odb, all abstract object concepts have an underscore as a prefix, whereas every specific object definition derives from such an abstract object and has a name without training underscore. Those specific object definitions differ from an instantiation of an object in that they can be instantiated an arbitrary amount of time by the world representation, depending on whether they are recognized by the perception module (see 2.1 on page 6). For mapping the odb with the SOMA ontology, the odb naming conventions are kept intact as class names in the ontology, while both objects with and without a trailing underscore are considered classes (and not individuals) in the merged ontology.

Mapping odb Tasks with SOMA

Tasks in the odb do not have a hierarchy. Every task is assigned to an object that passes the ability to execute that task on to its children. But tasks in the odb do not have any inherent relationship other than through the objects that provide the task. When executing a task, objects can have one of two roles in the EDAN framework: One object is the *target object* and if the task involves more than one object, the other object is the *reference object*.

That is different for the SOMA ontology, which provides a separate taxonomy just for tasks and also distinguishes between the definition of a `dul:Task`, which describes a users intention to do something, a `soma:Disposition`, which is a property of an object, indicating that the object has features that allow it to be used for a specific task, and a `dul:Role`, that different objects can play in a certain situation. While the roles objects can play in the EDAN framework do resemble the same concept, the SOMA ontology describes with a `dul:Role`, the latter is again thoroughly described by a taxonomy of 94 different Roles that Entities can play in a Task.

To merge the unstructured tasks in the EDAN framework with these different notions in SOMA, for every one of EDAN's tasks there are four mappings to be made: 1) EDAN's task has to be mapped with a child of SOMA's `dul:Task`, 2) Any object in the odb that is able to execute that task has to be mapped with a subcategory of SOMA's `soma:Disposition`, 3) the role that an object plays in a certain task when being the *target object* in EDAN's terms has to be mapped with a subcategory of SOMA's `dul:Role`, and 4) the same mapping is required for any object being the *reference object* in the EDAN framework.

EDAN's task *drink*, for example, is an action that describes EDAN moving a cup with something to drink and a straw close enough to the mouth of the person sitting in EDAN for them to drink from the cup. While the exact task is not modeled in the SOMA ontology, the task `soma:Serving` comes very close in its meaning, as it describes an agent delivering an object to another agent. The person who drinks from the cup will always have the `soma:RecipientRole`, meaning they receive something, in this case a drink, in the task. The mug containing the drink, on the other hand, plays the role of a `soma:MovedObject`, since the mug is getting moved in order for the person to drink from it.

In such a way, all 25 tasks, defined in the odb, have been mapped with those four concepts in the SOMA ontology, and therefore given a taxonomic hierarchy. Since EDAN's tasks have no taxonomic hierarchy on their own, the mapping also included the modeling of some internal references. For example, defining the tasks *pick_from_side* and *pick_from_top* to be two tasks, deriving from the common parent task *pick*. All of these relationships are, together with the object mapping definitions, defined in a separate yaml file, enabling future developers on the odb to easily define a mapping for new odb concepts. An excerpt of this definition can be found in Listing 3.2 on the following page, whereas the complete definition has been moved to the appendix in Listing A.1 on page 80

3.4 Introducing new Entities in the odb Ontology

While most of the information that is to be modeled in the new odb ontology comes from EDAN's odb and the SOMA ontology, the EDAN framework provides some additional knowledge outside of the odb, that is beneficiary to gather and store as part of the recorded NEEMs. Since this data is present neither in the odb nor in the SOMA ontology, the newly built odb ontology has been developed to expand on two SOMA concepts.

Idle Task

Even though the odb has no notion of an idle task, the EDAN framework regularly shows a task called *shared grasping* being executed, which indicates the robot being in a state where a user could manually control its arm, and the robot executing no task on its own. This is the state in which the robot is in when idling. For a logging mechanism that ranges over more than a few tasks but over multiple days, including a lot of idle time, it has been decided to be valuable to explicitly model this robotic state of doing nothing. The new odb ontology therefore includes an `odb:IdleTask` as child of `soma:PhysicalTask` to state that the robot

```

1  drink:
2    task: soma:Serving
3    ref_role: soma:MovedObject
4    target_role: soma:RecipientRole
5    disposition: soma:Storage
6
7  open:
8    task: soma:Opening
9    target_role: soma:ShapedObject
10   ref_role: False
11   disposition: soma:Enclosing
12   open_door:
13     parent: open
14   open_fridge:
15     parent: open_door
16
17  pick:
18    task: soma:PickingUp
19    target_role: soma:MovedObject
20    ref_role: False
21    disposition: soma:Graspability
22  pick_from_side:
23    parent: pick
24  pick_from_top:
25    parent: pick

```

Listing 3.2: Excerpt of the definition for a mapping from tasks in the odb to SOMA tasks, roles and dispositions.

is idle.

Since EDAN is always in this *shared grasping* mode in the time between the completion of one task and the beginning of another, the concept of an `odb:IdleTask` has to distinguish between a short transition period and a state where the robot is not being used. For the NEEM logger, this distinction has been set at a threshold of 20 seconds. As soon as EDAN is in *shared grasping* for 20 seconds or longer, the NEEM logger records this as executing an `odb:IdleTask`.

Imagining Available and Preferred Tasks

The EDAN framework calculates a list of available tasks at any moment, using a set of PDDL status descriptions and a PDDL reasoner, incorporated into the task descriptions. This allows the user to always have an overview of what the robot is capable of doing. When building a logging framework that remembers what EDAN knew and did in past situations, this list of available tasks is a crucial piece of information to be stored. Especially unexpected actions can often be necessary because the robot could not recognize an object or something else failed, resulting in the originally intended task not being available. Thus, the newly built odb ontology also introduces the notion of imagining a situation, i.e. noting the possibility to execute a number of tasks.

The SOMA ontology already models the notion of a `soma:MentalAction`, that is capable of executing a `soma:MentalTask` with one of its subcategories being `soma:Imagining`.

The SOMA ontology also already distinguishes, as discussed earlier, between tasks that are being executed in an `dul:Action` and pure task definitions, which describe the idea of a task without necessarily executing it. Therefore, the SOMA ontology allows modeling a *hypothetical* situation by describing an `soma:ActionExecutionPlan`, which models a plan to execute a `dul:Task`.

The odb ontology introduces a new relation that enables a `soma:Imagining` task to imagine such a hypothetical situation as an available task at a specific point in time. The robot then executes a `soma:MentalAction` that executes the task of `soma:Imagining`, which `odb:imagines_situation` an `soma:ActionExecutionPlan`, which `soma:isPlanFor` a `soma:PhysicalTask`.

A similar approach has been taken for modeling preferred tasks. An algorithm to find user preferences will be presented in the following chapter. To describe a predicted user preference in the ontology, the robot is modeled to execute a task called `soma:Reasoning`, which imagines a preference of the user. In latest updates, the notion of preferences, consisting of a set of elements that are ordered by a preference, has been implemented in the SOMA ontology (see [3]). To describe those preferences not as a general fact but rather as an imagined preference that is the result of a reasoning process at a specific time, the odb ontology introduces the relation `odb:imagines_preference`. This also allows for different preference calculations to take place and be stored separately in the ontology.

The NEEM logger therefore allows for three different types of actions, the same task definition can be a part of: At first, a `dul:Task` will be imagined as part of an `soma:ActionExecutionPlan`, describing the task to be available at that point in time. When equipped with the yet-to-be discussed preference prediction, the NEEM logger orders these task definitions as part of a predicted user preference. If the user decides to execute a task, the same task will also be modeled as being executed in a `soma:PhysicalAction`. In later evaluations, researchers can directly find all task options at a given point in time, their order given the predicted preference, and an annotation whether they have been executed or discarded in the end.

Semantic Distinction Between Times of Day

The later to be discussed PREFERENCEORACLE distinguishes tasks not only by their type and the types of objects that are involved, but also by the time of day they are being executed. The SOMA ontology does already allow to annotate actions with a `dul:TimeInterval`, during which e.g. an action is being executed. However, the values for such a time interval are usually encodings of hours and minutes. To semantically describe the time of day, the odb ontology therefore introduces a child to the `dul:TimeInterval`, called the `odb:TimeOfDay`, allowing a range over

- `odb:Morning`
- `odb:Midday`

- `odb:Evening`
- `odb:Night`

as value. With the NEEM logger in place, both the exact times are stored as a `dul:TimeInterval` of any action, as well as a `odb:TimeOfDay`.

3.5 Interfacing with the EDAN Framework

With the available set of classes in the ontology (also known as terminology component, or TBox) being defined the NEEM Logger is nowadays equipped to the robot EDAN, recording information about every object the robot perceives and every action the robot could and does execute by instantiating the presented classes or TBox with a set of instances (also known as the assertion component, or ABox). The NEEM Logger observes all the internal communication between modules in the EDAN framework and stores the ABox statements in a Mongo database, managed by the KnowRob framework.

Based on the presented mapping between high-level odb concepts and the SOMA ontology, the hereby presented NEEM Logger automatically generates the odb ontology based on the established odb in its classical form. This system allows developers to add new objects and tasks to the odb object hierarchy by simply adding them to the odb file system. The NEEM Logger then generates a merge odb ontology based on the object and task hierarchy that the new entities have been sorted into. This process does come with the drawback, that major changes to the odb do require some manual engineering, i.e. a developer thinking of a mapping between the newly introduces odb concepts and the SOMA ontology. If this process is skipped, the NEEM Logger still keeps the odb hierarchy intact and builds an ontology based on that hierarchy. But any newly introduced odb concepts that have neither been declared to derive from an existing concept nor been matched with a SOMA concept, will not benefit from SOMA's object taxonomy and therefore reach less comparability of concepts across research laboratories.

A properly set up odb ontology, even though it introduces a set of new entities, provides an ontology that is compatible with the SOMA ontology and therefore allows for comparisons between logged robotic experiments with the robot EDAN and experiments with other robots in other research labs. The SOMA ontology has been designed to support such larger-scale comparisons and learning algorithms. Their developers have even built a platform to collect and learn from robotic logging data in the form of SOMA-compatible NEEMs (see [50] and [16]).

While such a large-scale analysis of robotic experiments can be a valuable part of research, end users tend to be very cautious in allowing recordings of robotic interactions with them to be evaluated (see [7]). The NEEM Logger therefore stores any logging data only locally and temporarily and overrides the whole database with every new start. It requires an additional action by the user to permanently store the recorded logs in a non-temporal local space.

3.6 Summary

In this chapter, the NEEM Logger has been presented as a program that merges the pre-existing object and task representation of the robotic framework EDAN with the SOMA ontology by generating a novel odb ontology and records all robotic actions and perceptions in the form of an ontology. It has been shown that a mapping of the two knowledge representations is a task that can be achieved by only a handful of assertions, enabling the semi-structured object hierarchy of the EDAN framework to benefit from the richer taxonomy of the SOMA ontology without losing its internal object hierarchy.

The NEEM Logger allows the robot EDAN to record and later analyze any set of robotic actions and even compare them to the recordings of different robots using the SOMA ontology for knowledge representation. While such comparisons are valuable for the research process, this can also easily be turned off when interacting with an end user, potentially revealing personal information during the interaction with the robot.

Chapter 4

WHAT WOULD YOU WANT?

CALCULATING PREFERENCES

4.1 Introduction

The robot EDAN is already capable of executing a variety of actions automatically and semi-automatically (see [47], [17]) to help users with their daily workflow, as further discussed in Chapter 2. While both manual mode and shared control mode allow users to maintain as much control over the robot as possible and experience the robotic actions as their own, both modes require more concentration and personal energy compared to using full-on automation. Studies have repeatedly underscored the importance of users staying active, especially when they depend on a robot (see [7] and [29]), but daily tasks should also not require the user to be permanently focused. In the current user interface for the robot EDAN, a user can roll up to an object the robot is able to interact with. The computer tablet attached to the wheel chair will prompt the user to choose from different possible actions that allow the robot to automatically interact with the object (see [53]). This task suggestion scheme is calculating and keeping track of a symbolic state of the world (like a thermos bottle being attached to the robot’s hand) and derives from that a list of feasible actions that a user could execute, given this world state (see [9]). This list is ranked solely by the spacial distance between the robotic hand and the objects to interact with, only modified by the notion of occlusion.

This is a suitable user interface for an environment with a limited number of objects the robot is able to interact with, as well as for users who prefer to control the robotic arm manually. For a world setup with more objects, for example, a situation where the user opens a drawer with several different cups, glasses, plates, etc., the prompt-and-response technique might test its limits, as the sheer number of available tasks does not fit comfortably on a tablet screen so the user would potentially need to search for the desired action before activating automation mode. On another note, this workflow is strictly limited to executing single tasks (like picking something up, pouring some liquid from one container to another, etc.) with an event horizon of one and therefore does not include more complex workflows that consist of a chain of single tasks, which users might have in mind when switching to automatic mode.

This thesis introduces an algorithm, called `PREFERENCEORACLE`, that suggests a ranking of available robotic actions that is no longer limited to calculating the spacial distance from the robotic arm to an object. The algorithm aims to calculate a ranking of actions based on the probability of the user’s desires. In cases where the number of available actions is too large, the `PREFERENCEORACLE` helps the user by simplifying the selection of the desired task, therefore omitting the need to search within the list of all available actions. As studies

have shown, a user-specific approach to assistive robots is desirable and preferred over a one-size-fits-all solution (see [7] and [26]). Accordingly, the PREFERENCEORACLE suggests a method to calculate and store user preferences in daily routines, enabling the ranking of a list of available actions specific to each user. In a set of tests with auto-generated datasets and a list of around 25 available tasks, the algorithm succeeded in placing the desired task within its top five suggestions in close to 90% of the cases. In addition to ranking the available actions, the PREFERENCEORACLE also suggests a follow-up workflow with a variable event horizon that enables users to execute tasks that are commonly performed by them.

4.2 Related Work

When working with assistive robots, adjusting the robot to the needs of different users is a multifaceted subject with several different problem definitions and many approaches to make the robot more appealing to an individual human. Different user studies have shown that the acceptance of a robot helping with caregiving tasks is highly dependent on the way the robot is able to deal with different users differently and adapt to their individual needs (see [7]).

On the one hand, this includes different user-specific ways to execute the same task. This is especially important when working with users with motor impairments. Since people have different body parts that behave differently, any robotic help must consider the specific abilities of different users. One study, for example, has found a way to personalize the robotic task of helping people put on their shoes, considering a user who cannot move their sitting position by themselves (see [28], and similar: [56]). A similar but higher-level goal has been studied by a team that tried to leverage standardized occupational therapy assessments with respect to the range of motion of a patient. Those standardized tests, used for therapy, are employed to train a model to learn the area in space that a specific user can interact with or not. The robot is then instructed to enter that user-specific space when handing an object to the user (see [33]). Like the question of *how* to execute a given task, the question of *what* task to execute is another vast field. This area of research spans beyond the context of care-giving robots. The well-known RoboCup competition has, as early as 2016, introduced a section for competitions regarding robots helping in the home environment, regularly challenging competitors to let their robot organize shelves by the meaning of objects or clean up a room based on some inherent organization system of the room (see [22]). Such challenges, where a robot is tasked with understanding the organization system of a room and autonomously deriving task definitions (which object should be placed where?) from that, have been the subject of several studies (see [57], [4], and [55]).

Another direction for such applications is robots that don't just execute some actions that the user desires, but anticipate the requirements of the user's own future actions. Researchers have, for example, worked on a scenario in which a robot helps a human execute some actions by handing tools and objects to the user without an explicit order. For this scenario, the robot observes human actions, categorizes them, and predicts an action order based on

previous experiences (see [43]) in order to make the environment ready for future tasks (see [40]). Such proactive assistance applications can make the use of robots especially seamless by reducing not only the commands required to execute specific tasks but also the visible presence of a robot and active waiting time, which a user typically spends with modern robots.

The predictions have been calculated differently throughout the literature, but most approaches consider previous tasks to be an unordered set of actions that are added up to a frequency of tasks and their contexts without any order of execution, for example, when trying to predict tasks based on an organization scheme for a shelf or based on visual or semantic information on the current task (see [30], [27], [4], [40]). Task chains in their order of execution are, on the other hand, being considered as important context with the latest developments in large language models, which have become increasingly powerful, tempting researchers to find zero-shot solutions for more and more complex problems. All prediction problems seem to be a question of the right prompt, which was supported early on by the developers of the pioneer GPT models (see [52]). This can again take the form of a task planning algorithm that uses the current visual or textual representation of the world as an input parameter, like a project that introduced prompts that directly output the code necessary for a robot to execute a complete plan to achieve a human’s intended goal (see [48]). Or, it can leverage a broader notion of a situation, collecting information about the user’s actions as a string of actions, and reasoning about task plans based on a specific user’s action chains (see [54], [25]). However, adding a lot of contextual information to the prompt of a large language model significantly diminishes its performance (see [18]), leaving the algorithm with a very limited set of information to provide to such a model.

4.3 Overview on Calculating Predictions, Introducing the PREFERENCEORACLE

While the next section will go into more detail on how each aspect of the similarity between two situations is calculated, this section presents a general overview of the ideas on how to calculate preference predictions and their integration in the EDAN framework.

The goal of the PREFERENCEORACLE is to calculate the task preferences of different users, sitting in the wheelchair robot EDAN and using EDAN for their personal daily routines. These preferences can range from a favorite cup that a user might have over preferences regarding different kinds of beverage to routines involving a set of multiple tasks that a user executes regularly. The PREFERENCEORACLE aims on the one hand at an algorithm that is capable of predicting some preferences within the first few tasks that a user is performing. On the other hand, it aims at being quickly adaptable without resorting to larger deep learning algorithms that take time and a lot of (in this case quite personal) data to be trained and fine-tuned. With the PREFERENCEORACLE, data regarding a user workflow is not leaving the machine the robot is operating on for any kind of external training process, the algorithm is designed to be executed completely locally.

The idea of the preference calculation is based on the old and common concept of an

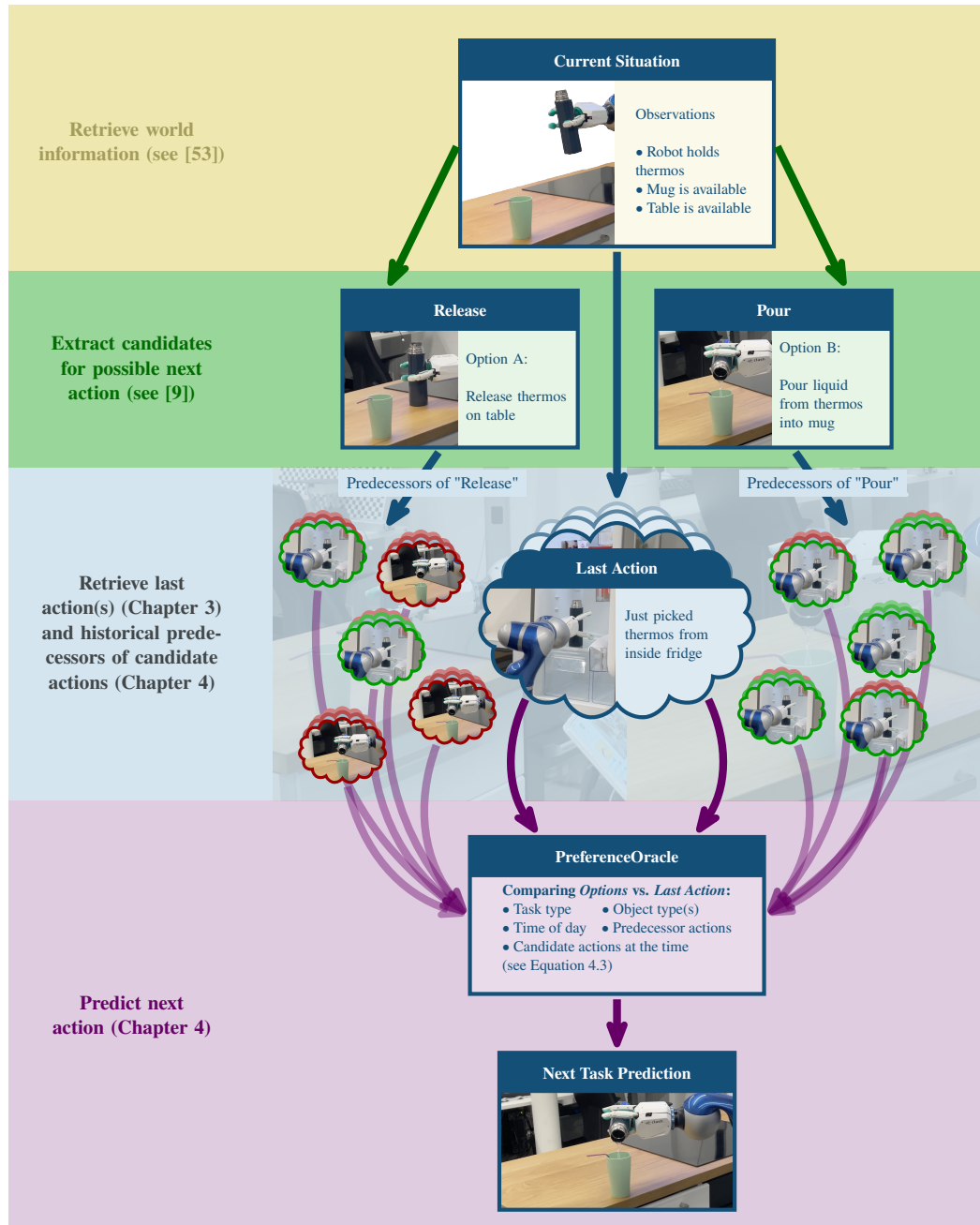


Figure 4.1: Overview of the *PREFERENCEORACLE*.

N-gram, where the statistical frequency of words used directly after a queried word or sentence indicates the probability of those words completing the sentence. Mathematically, the probability of a specific word token sequence following a query text can be thought of as the probability P of that sequence of word tokens w_1^k with length k , given the sequence of word tokens with length N appearing right before the word token in question:

$$P(w_1^k) = \prod_{q=1}^k P(w_q | w_{q-N+1}^{q-1}) \quad (4.1)$$

When calculating the probability of a specific sequence of word tokens in a sentence, a straightforward approach would be to count the number of occurrences of that exact sequence of tokens. In the context of task prediction, the tokens are not natural language words, but tasks. However, since in the context of the robot EDAN the task history per user is often limited, this definition of probability may result in a zero probability for many tasks. Therefore, the PREFERENCEORACLE considers not only identical tasks but also *similar* tasks to compute the probability P .

To determine a degree of similarity between two tasks, the algorithm presented here considers the type of the action, the type of objects that have been interacted with, the time of day the tasks were executed, and the context of the tasks, described by the set of tasks that could have been executed alternatively. With these comparisons as foundation for a model similar to an N-Gram, the PREFERENCEORACLE leverages the previously introduced NEEM Logger to read in the task history and calculate a prediction for the next task executed by the user, as shown in Figure 4.1 on the preceding page:

In any given situation s_c , the robot is in a state where it either holds an object or it does not and observes a number of objects, some of which it has been trained to interact with. The yellow field in Figure 4.1 depicts an example situation that shows EDAN holding a thermos bottle above a counter top with a mug standing on the counter top, ready to be used. As described in [9], the EDAN framework derives a set of candidate tasks for a possible next action $O(s_c)$ from the description of any current situation s_c . In the given example, this includes the two candidate tasks to either release the thermos bottle on the counter top or to pour some liquid from the thermos bottle into the cup (i.e. $O(s_c) = \{release, pour\}$). As described earlier, the EDAN framework in its current form presents the user with those options and allows to automatically execute either one of them by selecting the task on a tablet computer. Resorting to the NEEM Logger described in the previous chapter, the PREFERENCEORACLE then retrieves information on the previously executed tasks, using the framework KnowRob (see [8]). As depicted in the blue section of Figure 4.1, this includes on the one hand the situation that had last been executed $p(s_c) = s_1$ and therefore directly led to the current situation s_c . In the exemplary situation, EDAN picked up the thermos from inside the fridge. On the other hand, the set of all previously executed tasks S is searched for instances, where either one of the candidate tasks $s_o \in O(s_c)$ had been executed

before. Those instances $H(s_o) \exists s_o \in O(s_c)$ together with their respective predecessor tasks $p(s_h) \forall s_h \in H(s_o) \forall s_o \in O(s_c)$ (represented as mini clouds in Figure 4.1) are collected and compared with the last action s_1 , leading to the current situation.

The process of comparing the last action $p(s_c)$ with each of the historic predecessor tasks of the currently available task options $p(s_h) \forall s_h \in H(s_o) \forall s_o \in O(s_c)$ consists first and foremost of a comparison between their task type $tsk(s)$ and the types of objects they interact with $obj(s)$. Objects that take part in an action are only ever compared with other objects that play the same `dul:Role` in their respective task, presented here as an index to the obj function: $obj_0(s)$ and $obj_1(s)$. Both comparisons between objects and between tasks leverage the odb ontology, based on both the SOMA ontology and EDAN's odb as described in the previous chapter, to determine the graph distance between them. The hierarchical taxonomy of tasks and objects allows to determine, for example, a thermos bottle and a mug to be closer related than a mug and a table, because both the thermos bottle and the mug are containers that can hold liquids and be used to drink from. Figure 4.1 shows an example in which the task of releasing the thermos bottle on the table has previously been executed twice after picking a thermos bottle from the fridge (which is the same task, involving the same objects, as the last action s_1 , therefore represented with a green border) and three times after pouring some liquid from a thermos bottle into a mug (which is a different task, involving the same object, and is depicted with a red border). However, the candidate task of pouring liquid from the thermos into the mug has only ever been executed after EDAN picked up a thermos from the fridge. The third aspect in comparing two tasks is the time of day they had been executed. If one of the candidate tasks $s_h \in H(s_o) \forall s_o \in O(s_c)$ has been mainly executed in the morning, whereas the current situation s_c takes place in the evening, the path distance between both tasks is multiplied by a penalty:

$$\begin{aligned}
 path(s_h, s_c) = & (path(tsk(s_h), tsk(s_c)) \times X \\
 & + \sum_{n=0}^{|Obj(s_h)|} path(obj_n(s_h), obj_n(s_c)) \times Y) \\
 & [\times Z]
 \end{aligned} \tag{4.2}$$

All three aspects of the calculation are multiplied by a different hyperparameter (represented as X , Y , and Z in the equation) that can be set in the beginning to adjust the calculations to different goals. More on that later.

Another aspect of comparing situations is the comparison of the candidate actions in any given situation (not visually depicted in Figure 4.1). For every task option $s_o \in O(s)$, the described path distance calculation is repeated. The candidate tasks with the most similarity in both situations (i.e. the two task options with the lowest $path$) are matched and averaged to reach a *situation* path distance:

$$sit(s_a, s_b) = path(s_a, s_b) + \frac{\sum_{s_o \in O(s_a)} \min(path(s_o, s) \forall s \in O(s_b))}{|O(s_a)|} \times W \quad (4.3)$$

The factor W again represents a hyperparameter that is introduced later.

In a last step, PREFERENCEORACLE iterates over the recorded tasks and compares the latest executed task chain with previously executed task chains to find similarities. In a next step, the same calculations are repeated for the respective predecessor tasks (depicted as the shadow behind the clouds in Figure 4.1). The predecessor of the last action $p(p(s_c))$ is compared with the predecessor of every second degree predecessor of every historical occurrence of any of the candidate tasks $p(p(s_h)) \forall s_h \in H(s_o) \forall s_o \in O(s_c)$. With $p(p(x)) = p_2(x)$ and Q and R representing simplified hyperparameters, this can be represented as:

$$score(s_a, s_b) = \sum_{n=1}^Q \frac{sit(p_n(s_c), p_n(s_h)) \forall s_h \in H(s_o) \forall O(s_c)}{n \times R} \quad (4.4)$$

Based on this score, the PREFERENCEORACLE ranks the available tasks $O(s_c)$ and suggests the candidate task with the best score to the user. This allows an EDAN user to ideally find their desired task in a prominent position on the tablet computer and directly select it to be executed.

Finding task chains in history that are highly correlated with the task chain that had just been executed by the user also allows the PREFERENCEORACLE to predict not just one next task, but aggregate the tasks that often followed after the given task chain and predict a longer task chain, consisting of multiple tasks, into the future.

4.4 The Algorithm and its Hyperparameters

With the presented overview in mind, this section further elaborates on the calculations behind the similarity calculations as well as the role of the hyperparameters in those calculations.

Comparing Task Types and Object Types

As mentioned in the previous chapter, tasks and objects are, thanks to the NEEM Logger, represented as part of an ontology, storing narrative-enabled episodic memories (NEEMs). Computing similarities between two individuals (nodes) in an ontology is a well-studied problem with a vast literature. The most basic approach is to count the required steps (**path**) to get from one individual to another in the knowledge graph that is the ontology. For example, if task B and task C are both direct children of task A, it would take two steps to get from task B to task C (one step to get from B to A and a second step from A to C). However, this approach does not account for the granularity of task B and C, that is, their depth in the graph. For instance, in the SOMA ontology the general task **Actuating** has, among others, the children **Pouring** and **Pushing**, which in turn have

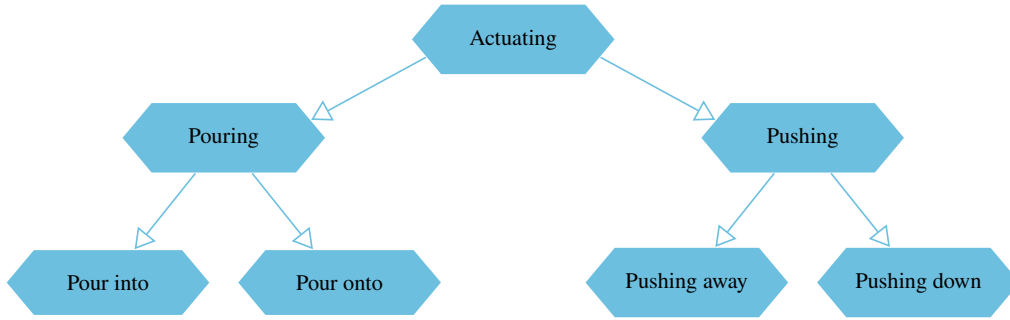


Figure 4.2: Excerpt of the SOMA ontology. Counting the steps from subclasses of `Pouring` leads to the same result as counting steps from `Pouring` to `Pushing`, even though the latter are far more different.

the children `Pouring into` and `Pouring onto` resp. `Pushing away` and `Pushing down` (see Figure 4.2). Taking into account only the number of steps to move from one task to another would result in the same similarity score for $path(\text{Pouring}, \text{Pushing}) = 2$ as for $path(\text{Pouring into}, \text{Pouring onto}) = 2$. However, the two variations of a pouring task are clearly more similar than the two tasks pouring and pushing, and should therefore get assigned a lower distance. To address this, the authors of [5] have developed a metric (originally designed for ontologies in the biomedical domain) to calculate the similarity of two individuals in an ontology, taking into account the granularity of their difference.

The idea is to first identify the *Least Common Subsumer* node (LCS), a common ancestor of both nodes, and measure its *depth* within the knowledge graph, which is considered the *common specificity* (CSpec) of those nodes. If the entire graph looks like the one in Figure 4.2, the LCS of `Pouring into` and `Pouring onto` would be `Pouring`, which is the second highest level concept in an ontology with at most three levels. The depth of `Pouring`, i.e. the common specificity (CSpec) of `Pouring into` and `Pouring onto`, is therefore $2/3$. The LCS of `Pouring` and `Pushing`, on the other hand, would be `Actuating`, leading to a common specificity of $1/3$. To calculate the distance between two nodes, the authors developed the formula that our PREFERENCEORACLE uses to determine the distance (**dist**) between two individuals A and B in the ontology:

$$dist(A, B) := \log(path(A, B) \times CSpec(A, B) + 1) \quad (4.5)$$

Calculating this formula results in an algorithm similar to Algorithm 1 on the next page.

To determine the similarity between two tasks in the ontology, the distance between the two task types as well as the distance between all objects involved in those tasks are being calculated. The total distance can be considered a weighted sum of the distances between the task types and the object types, using two hyperparameters `task factor` and `object factor`. Additionally, a third hyperparameter is used to assign a distance in cases where one task involves more objects than the other: `penalty`. For example, the `Pour` task involves two objects: one that the robot holds in its hand to pour something out of (for example, a

Algorithm 1 Distances in the ontology

```

1: function CALCOWLDISTANCE( $i_a, i_b$ )
2:    $lcs$  : OWLCLASS ▷ Least Common Subsumer
3:    $anc_a$  : LIST  $\leftarrow i_a$  ▷ Ancestors of  $i_a$ , sorted by degree of kinship
4:    $anc_b$  : LIST  $\leftarrow i_b$  ▷ Ancestors of  $i_b$ , sorted by degree of kinship
5:    $path$  :  $\mathbb{N}$  ▷ Path length from  $i_a$  to  $i_b$ 
6:   while not  $lcs$  do
7:      $anc_a \leftarrow directParent(anc_a[-1])$ 
8:      $anc_b \leftarrow directParent(anc_b[-1])$ 
9:     for  $\forall a \in anc_a, \forall b \in anc_b$  do
10:      if  $a = b$  then
11:         $lcs \leftarrow a$ 
12:         $path \leftarrow index(a, anc_a) + index(b, anc_b)$ 
13:      end if
14:    end for
15:  end while
16:
17:   $depth_{lcs}$  :  $\mathbb{N} \leftarrow 1$  ▷ Depth of lcs in the ontology
18:   $last\_parent$  : OWLCLASS  $\leftarrow lcs$ 
19:  while True do
20:    if  $last\_parent = owlRoot$  then
21:      break.
22:    end if
23:     $last\_parent \leftarrow directParent(last\_parent)$ 
24:     $depth_{lcs} \leftarrow depth_{lcs} + 1$ 
25:  end while
26:
27:   $CSpec \leftarrow 1 - \frac{depth_{lcs}}{depth_{ontology}}$  ▷ Common specificity of  $i_a$  and  $i_b$ 
28:
29:  return  $\log(path \times CSpec + 1)$ 
30: end function

```

Thermos), and another object into which the liquid is being poured (for example a Mug). In contrast, tasks such as Pick up an object (for example a Thermos) involve only one object. When comparing such two tasks, the algorithm calculates the $dist(Pour, Pick)$, weighted by the task factor, adds it to the $dist(Thermos, Thermos)$, weighted by the object factor, and as the Pouring task includes one more object that the other does not, the result is added to the penalty, again weighted by the object factor. As task definitions are derived from the EDAN framework, which allows at most two objects to be involved in the semantic representation of a task (a target object and a reference object, see Chapter 3), no task can involve more than two objects. This results in an algorithm similar to Algorithm 2 on the following page.

In any given situation s_c , the EDAN framework provides a list of candidate tasks ready to be executed $O(s_c)$, considering the current world state (depicted as green bar with the tasks Pour and Release as examples in Figure 4.1 on page 28). For each available task

Algorithm 2 Overall distances

```

1: task_factor :  $\mathbb{R}$ 
2: task_factor :  $\mathbb{R}$ 
3: penalty :  $\mathbb{R}$ 
4:
5: function OVERALLDISTANCE( $s_c, s$ )
6:    $dist_{tsk} \leftarrow \text{CALCOWLDISTANCE}(tsk(s_c), tsk(s))$  ▷ see Alg. 1
7:    $Dist_{objs} : \text{List}$ 
8:
9:   if  $obj_1(s_c) \wedge obj_1(s)$  then
10:      $Dist_{objs} \leftarrow \text{CALCOWLDISTANCE}(obj_1(s_c), obj_1(s))$  ▷ see Alg. 1
11:   else if  $obj_1(s_c)$  then
12:      $Dist_{objs} \leftarrow \text{penalty}$ 
13:   end if
14:   if  $obj_2(s_c) \wedge obj_2(s)$  then
15:      $Dist_{objs} \leftarrow \text{CALCOWLDISTANCE}(obj_2(s_c), obj_2(s))$  ▷ see Alg. 1
16:   else if  $obj_2(s_c)$  then
17:      $Dist_{objs} \leftarrow \text{penalty}$ 
18:   end if
19:
20:   return task_factor  $\times dist_{tsk}$  + object_factor  $\times \text{avg}(Dist_{objs})$ 
21: end function

```

$s_o \in O(s_c)$, the PREFERENCEORACLE aims to determine the probability that it is the next item in the task chain, given the chain of previously executed tasks by that user. To achieve this, the algorithm compares each available task $s_o \in O(s_c)$ with every task in the chain of historic tasks S and identifies similar tasks, i.e., tasks with a low distance according to algorithm 2. While the resulting weighted graph distance is being kept for further calculations, the hyperparameter **knockout** determines a threshold that a task $s \in S$ has to pass in order to be considered similar to one of the candidate task options $s_o \in O(s_c)$. The set of historic tasks that are considered similar to a candidate task option $H(s_o)$ is therefore defined as:

$$H(s_o) = \{s \in S \mid dist(s_o, s) < \text{knockout}\} \quad (4.6)$$

If, for example, the algorithm is to compare the current task option s_o to **Pick** a **mug** with a previously executed action s of picking a mug, the distance between both tasks is always 0:

$$\begin{aligned} dist(A, B) &:= \log(\text{Path}(A, B) \times \text{CSpec}(A, B) + 1) \\ \Rightarrow dist(A, A) &= \log(0 \times \text{depth}(A) + 1) = \log(1) = 0 \end{aligned} \quad (4.7)$$

$$\text{task_factor} \times 0 + \text{object_factor} \times 0 = 0$$

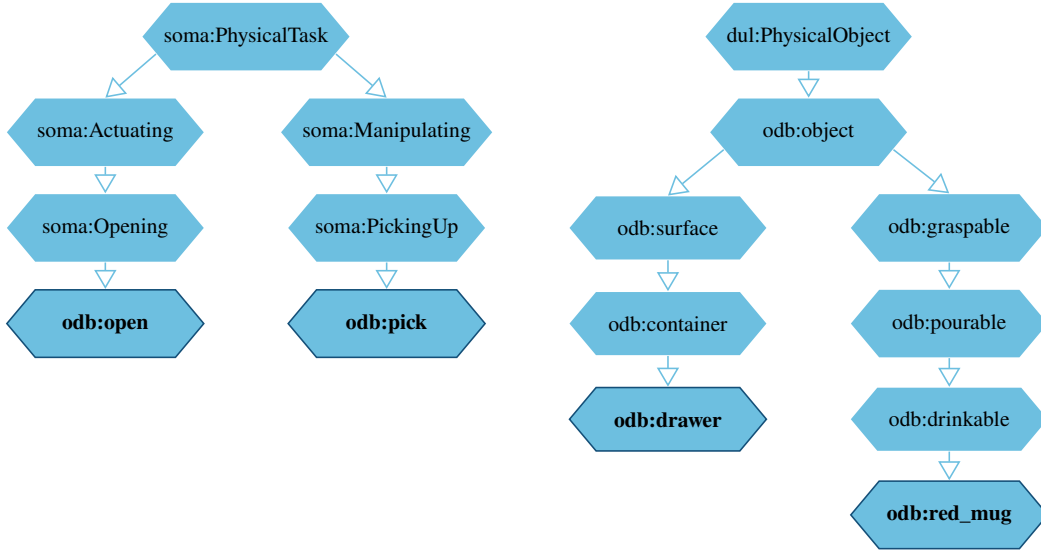


Figure 4.3: Comparing the task close drawer with the task pick mug in the ontology.

On the other hand, when comparing the same current task option s_o with a previously executed task $s \in S$ to **Close** a **drawer**, given an ontology that looks as depicted in Figure 4.3, the distance calculation looks as follows:

$$\begin{aligned}
 \text{Path}(\text{Open}, \text{Pick}) &= 6 \\
 \text{lcs}(\text{Open}, \text{Pick}) &= \text{PhysicalTask} \\
 \text{depth}(\text{PhysicalTask}) &= \frac{1}{5} \\
 \Rightarrow \text{CSpec}(\text{Open}, \text{Pick}) &= 1 - \frac{1}{5} = \frac{4}{5} \\
 \text{dist}(\text{Open}, \text{Pick}) &= \log(\text{Path}(\text{Open}, \text{Pick}) \times \text{CSpec}(\text{Open}, \text{Pick}) + 1) \\
 \Rightarrow \text{dist}(\text{Open}, \text{Pick}) &= \log\left(6 \times \frac{4}{5} + 1\right) = \log(5.8) \\
 \Rightarrow \text{dist}(\text{Open}, \text{Pick}) &\approx 1.758
 \end{aligned} \tag{4.8}$$

$$\begin{aligned}
 \text{Path}(\text{drawer}, \text{mug}) &= 7 \\
 \text{lcs}(\text{drawer}, \text{mug}) &= \text{object} \\
 \text{depth}(\text{object}) &= \frac{2}{7} \\
 \Rightarrow \text{CSpec}(\text{drawer}, \text{mug}) &= 1 - \frac{2}{7} \\
 \Rightarrow \text{dist}(\text{drawer}, \text{mug}) &= \log\left(7 \times \left(1 - \frac{2}{7}\right) + 1\right) = \log(6) \\
 \Rightarrow \text{dist}(\text{drawer}, \text{mug}) &\approx 1.792
 \end{aligned} \tag{4.9}$$

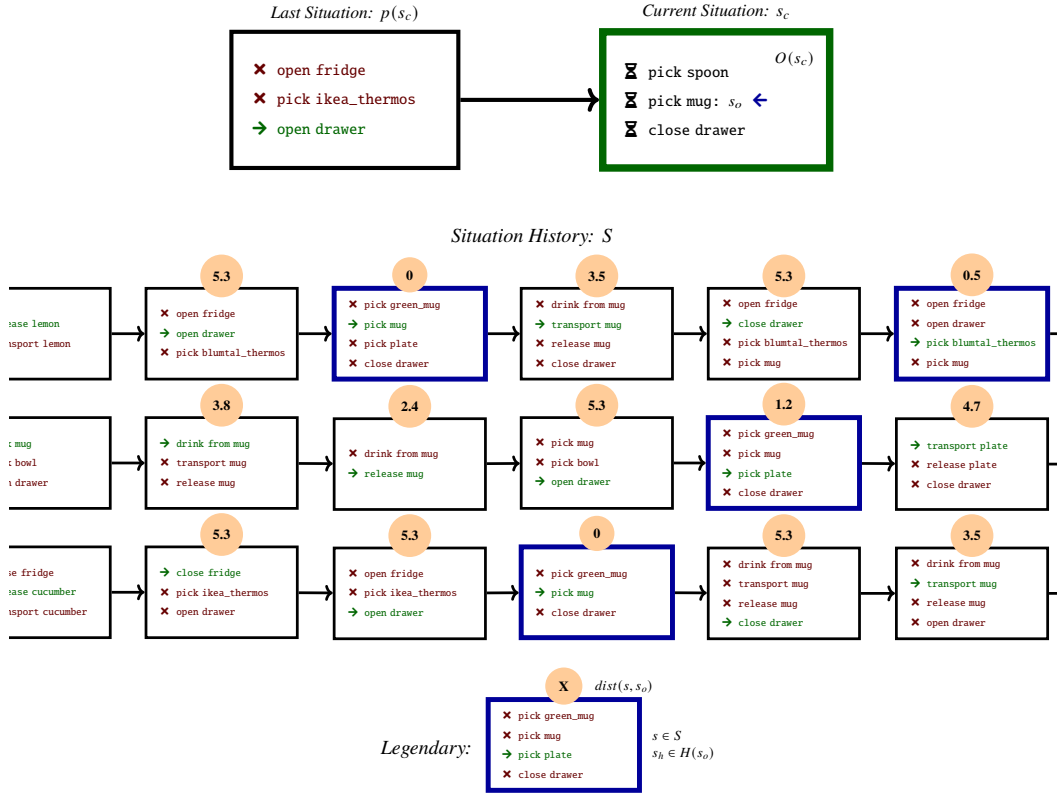


Figure 4.4: Ontological distance values, comparing the task option to pick a mug in the current situation with a history of executed tasks, based only on task types and object types.

Given an **object factor** of 1 and a **task factor** of 2, this would result in:

$$\begin{aligned}
 \text{object factor} &= 1 \\
 \text{task factor} &= 2 \\
 \text{OVERALLDistance} &= \text{object factor} \times 1.792 + \text{task factor} \times 1.758 \\
 \text{OVERALLDistance} &= 5.308
 \end{aligned} \tag{4.10}$$

The resulting distance of ≈ 5.308 is then being compared to the **knockout** value to determine, whether the tasks of closing a drawer and picking up a red mug should be considered similar. For the upcoming examples, a **knockout** value of 1.5 is assumed, leading the presented tasks to be considered *not* similar. To get an idea of other ontological distances for objects as well as tasks, Figure 4.4 depicts an exemplary history of situations S and a distance value for all of the historically executed tasks compared to the task option s_o of picking a mug. Compared to a **knockout** value, four of these situations are considered to be in $H(s_o)$, i.e. similar, and depicted with a thick blue border.

Comparing Situations

A *situation* s , in contrast to a task $tsk(s)$, is defined to link not only the executed task $tsk(s)$ and the involved objects $obj(s)$, but also the candidate tasks, available in that specific point in time $O(s)$. For instance, the user is currently opening the drawer: Usually in such a

situation, the user might want to grab their favorite red mug from the drawer and then take a spoon. Therefore, if the last executed task was opening the drawer, the prediction for the next situation should be picking up the red mug. However, if there is already a mug standing on the countertop, because the user had been drinking coffee all morning, this might change the purpose of opening the drawer (perhaps they just want to get the spoon). To differentiate such situations, the PREFERENCEORACLE not only compares the currently available tasks ($O(s_c)$) with previously executed ones (S), but also the currently available tasks $O(s_c)$ with the available tasks in a previous situation $O(s) \forall s \in S$, determining how similar the available tasks are on average.

Accordingly, Figure 4.4 on the preceding page depicts situations that are compared with every situation containing a list of candidate tasks $O(s)$ in red as well as the executed task $task(s)$ in green. Comparing the list of candidate tasks in a given situation, first requires a matching of candidate tasks. This is achieved by the surjective function $m(s_o, O(s)) \mid s_o \in O(s_c), s \in S$, finding the most similar task option in $O(s) \mid s \in S$ for every $s_o \in O(s_c)$, i.e. the task option with the lowest value for the *dist* function. With every candidate task in the current situation $O(s_c)$ being assigned a candidate task in the historic situation $O(s) \mid s \in S$, their respective *dist* values are averaged, which constitutes the distance between both situations as a whole: $dist(s, s_c)$. This situation distance is weighted by the hyperparameter `compare situations` and added to the overall distance between the current task option and the historically executed task (see Algorithm 3 on the next page).

When computing time becomes a significant issue, the option of comparing situations (i.e. setting `compare situations` > 0) also has its drawback. For distance calculations, which involve comparing only task types and object types with each other, a preference prediction compares the current task options $O(s_c)$ with all previously executed tasks S , resulting in a linear time dependency $O(n)$, depending on n being the number of already stored history tasks¹. For every additional situation in S , the algorithm compares the current situation s_c with one more situation $s \in S$. However, when introducing a comparison of situations, each candidate task $s_o \in O(s_c)$ must be compared to all historical task options as well, effectively multiplying the computing time by the number of available task alternatives $|O(s_c)|$. To prevent this behavior, the PREFERENCEORACLE implements the option to set `compare situations` $= 0$, which causes the program to completely omit comparing situations, reducing the computing time by a factor of $|O(s_c)|$.

Looking Further Back

For a more precise measure, the algorithm considers the current situation s_c as part of an ongoing chain of tasks, i.e. expects every situation to have a predecessor $p(s) \forall s \in S$. When trying to determine what a user might do in a specific situation, their previous actions are a

¹We assume the number of *available* tasks at any current moment to be relatively constant and not excessively high. If the number of available tasks increases faster than the number of tasks already executed, the time complexity would depend primarily on the number of available tasks. However, there does not appear to be any practical situation in which this might be a concern.

Algorithm 3 Calculating situation distance

```

1: compare situations :  $\mathbb{R}$ 
2: knockout :  $\mathbb{R}$ 
3:
4: function CALCULATIONDISTANCE( $s_c, s$ )
5:    $dist \leftarrow \text{OVERALLDISTANCE}(s_c, s)$  ▷ see Alg. 2
6:   if  $dist \geq \text{knockout} \vee \text{compare situations} \leq 0$  then
7:     return  $dist$ 
8:   end if
9:
10:   $D$  : List
11:  for  $\forall s_o \in O(s_c)$  do
12:     $d_{best} \leftarrow \infty$ 
13:    for  $\forall s_{oh} \in O(s)$  do
14:       $d \leftarrow \text{OVERALLDISTANCE}(s_o, s_{oh})$  ▷ see Alg. 2
15:      if  $d \leq d_{best}$  then
16:         $d_{best} \leftarrow d$ 
17:      end if
18:    end for
19:     $D \leftarrow d_{best}$ 
20:  end for
21:
22:  return  $dist + \text{AVG}(D) \times \text{compare situations}$ 
23: end function

```

crucial factor in determining their goal. For example, a user who is opening a drawer with cutlery might want to grab a spoon if they had previously prepared a mug with tea or coffee, whereas the user might be more likely to choose a knife if they previously placed a breakfast plate on a table.

To account for such correlations, after comparing the current situation s_c with a historic one $s \in H(s_c)$, the PREFERENCEORACLE repeats the same process for their respective predecessors $p(s_c)$ and $p(s) \mid s \in S$. If the calculated distance between these predecessor situations exceeds the **knockout** value, the process terminates. Otherwise, the program repeats this loop as many times as specified by the hyperparameter **event horizon backwards** or until it encounters an **odb:IdleTask**, whichever comes first. Since an **odb:IdleTask** marks the beginning and end of a sequence of tasks, examining their predecessors would not provide any relevant information, therefore the design decision has been made to consider them delimiters of the task chain.

Even though the situations leading to the current situation are very relevant for predicting the next task, they are not as relevant as the current situation s_c . The further down in the task chain the predecessor $p_n(s_c)$ is, the less relevant it is to determine the next task. The distance between the current task s_c and its counterpart in history $s \in H(s_c)$ has a greater impact on the final result than the comparison between their predecessors $p(s_c)$ and $p(s) \mid s \in H(s_c)$, which in turn have a greater impact than their predecessors $p(p(s_c)) =$

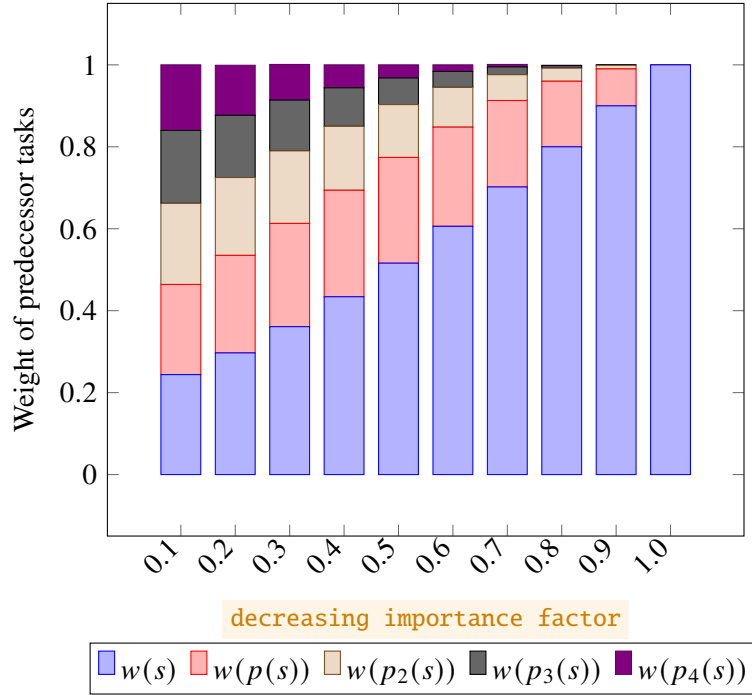


Figure 4.5: Different values for **decreasing importance factor** (x-axis) and the resulting weights of the n -th predecessor situation.

$p_2(s_c)$ and $p(p(s)) = p_2(s) \mid s \in H(s_c)$, and so on. This is achieved by introducing the **decreasing importance factor**, which assigns a decreasing weight to predecessor tasks that lie further back in history (see Figure 4.5), as calculated by this formula (with $p_n(s)$ describing the n -th predecessor of s and $w(p_n(s))$ describing the weight given to $p_n(s)$):

$$\begin{aligned}
 w_x(p_n(s)) &= \text{decreasing importance factor} \times \left(1 - \sum_{k=0}^{n-1} w_x(p_k(s))\right) \\
 N &= \text{event horizon backwards} \\
 w(p_n(s)) &= w_x(p_n(s)) \div \sum_{k=0}^N w_x(p_k(s))
 \end{aligned} \tag{4.11}$$

As can also be seen in Figure 4.5, a lower value for the **decreasing importance factor** approaching 0 leads the algorithm to consider predecessor situations $p_n(s)$ almost as important as the original situations s , whereas a high value approaching 1 reduces the weight given to predecessor situations to almost 0, effectively eliminating them from the calculation.

With a naive implementation, a number of predecessors to be considered might again dramatically increase computing time. After all, the calculations to be executed get multiplied with every additional predecessor task. However, since the *dist* value of any predecessor $p_n(s_c)$ with the predecessors in history $p(s) \mid s \in H(s_c)$ must have already been calculated when calculating predictions for the previous task, the results of those calculations are being stored by the PREFERENCEORACLE. Additional predecessor tasks to be considered therefore

only require additional lookup operations in the stored distance values rather than additional calculations.

Comparing Time of Day

Depending on the user, their habits may change over the course of a day. Most people have a morning routine, an evening routine, etc.. Although the SOMA ontology already incorporates the concept of a `dul:TimeInterval` from the DUL ontology (see [21]) and introduces the new relations `soma:hasEventBegin` and `soma:hasEventEnd` to mark the beginning and end of any event, this does not include a semantic representation of the time of day. The odb ontology therefore introduces the `odb:TimeOfDay` concept with the children `odb:Morning`, `odb:Midday`, `odb:Evening`, and `odb:Night`. These can be annotated to any event to indicate the time of day of that event.

After comparing situations and their predecessors, the PREFERENCEORACLE compares the `odb:TimeOfDay` of the current situation s_c and the historic situation s . The algorithm then applies a penalty to the *dist* value of all situations that differ in their `odb:TimeOfDay`. The penalty itself is another hyperparameter: `time of day penalty`.

Multiple Occurrences of the same Candidate Situation in History

The candidate tasks in $O(s_c)$ can get assigned multiple situations in the task chain S , which are considered similar, i.e. $|H(s_o)| > 1, s_o \in O(s_c)$. Reflecting on the discussions of N-Grams in the beginning, the probability for any candidate task $s_o \in O(s_c)$ to be executed next should be the probability of this task following its predecessor, given the task chain. In boolean terms, this would be as simple as dividing the number of occurrences of similar tasks in history, i.e. $|H(s_o)|$, by the number of occurrences of its predecessor situation $|H(p(s_c))|$. However, since the tasks are not identical but similar, this would cancel out the differences between a number of identical tasks in history vs. a number of vaguely similar tasks in history.

It is not obvious, how the different distance values of the similar tasks should be weighted. Table 4.1 on the next page presents an example of this problem, where the last action $p(s_c)$ was picking a thermos bottle with three proposed successor tasks $O(s_c)$: place the thermos bottle on the kitchen table, drink from the thermos bottle, or pour some liquid from the thermos bottle into a blue cup. In the example, the `Pour` task has been assigned to only two tasks in history $|H(s_o)|, s_o \in O(s_c)$, both with a fairly high distance value, leaving this option to be discarded early on. The candidate task `Place` has four similar tasks $|H(s_o)| = 4$ and the candidate task `Drink` has $|H(s_o)| = 5$. Based on occurrences alone, the `Drink` task would be considered to have the highest probability to be the next task. On the other hand, the average distance value for the `Place` task is significantly lower with $avg(dist(s_o, s_h)) = 0.35$ with $s_h \in H(s_o)$, compared to the `Drink` task with $avg(dist(s_o, s_h)) = 1.22$. Based on the average distance of the similar tasks, the `Place` task should therefore be favored.

...place on kitchen table	: [0.0, 0.1, 0.4, 0.9]
pick thermos... ..drink from thermos	: [0.5, 0.9, 1.3, 1.5, 1.9]
...pour from thermos to blue cup	: [2.0, 0.7]

Table 4.1: Collecting successor tasks poses the question of how to weigh number of occurrences vs. the overall distance between each occurrence and the current task.

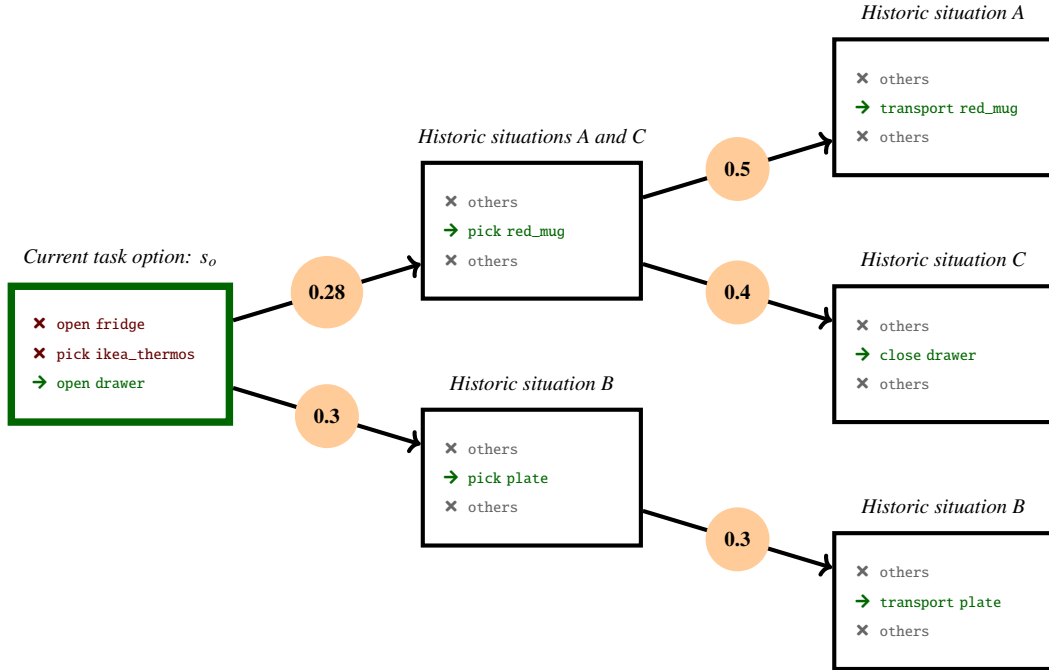


Figure 4.6: Example of a preference tree (calculated with `multiple occurrences factor` = 0.7)

The PREFERENCEORACLE allows the user to weigh both aspects, the number of occurrences and the average distance score of the similar tasks, by providing the hyperparameter `multiple occurrences factor` (x), using the following formula

$$v = \frac{avg + 0.0001}{len^x} \quad (4.12)$$

A higher value for the `multiple occurrences factor` places more emphasis on the number of similar tasks $|H(s)|$, while a lower value prioritizes a low average distance among those similar tasks $avg(dist(s_h \forall s_h \in H(s)))$.

Looking into the Future, Calculating with $N > 1$ in the N-Gram

Reflecting again on the discussion of N-Grams at the beginning of this section, it becomes clear that while the algorithm does calculate with a larger N , i.e. calculates with a larger sub-chain of tasks when looking at the chain of task tokens S , it does up to this point not include a prediction of more than one task token into the future.

To achieve such a prediction of a larger event horizon in the future, the PREFERENCEORACLE

builds on the previously mentioned list of tasks that are considered similar to any task option in the current situation $H(s_o)$, $s_o \in O(s_c)$. For every similar situation $s_h \in H(s_o)$, the algorithm collects the successor tasks, executed in the situations that followed next in the previous situations: $N(H(s_o)) := \{n(s_h) \mid s_h \in H(s_o)\}$ with $n(s)$ representing the next situation, i.e. the inverse of $p(s)$. Just as it weights the situations in $H(s_o)$, the **multiple occurrences factor** also weights their successors in $N(H(s_o))$, resulting in a weighted tree of successor tasks. Each branch represents a different task that could be executed next. An example of this tree can be found in Figure 4.6 on the preceding page: In the example, one currently available task $s_o \in O(s_c)$ is to open the drawer. There are three historic situations, labeled A , B , and C in $H(s_o)$, each with a different overall distance value: $dist(A, s_o) = 0.5$, $dist(B, s_o) = 0.3$, and $dist(C, s_o) = 0.4$. In the historic task corpus S , after the task of opening a drawer was executed in situation B , the next two tasks were to pick up a plate and then transport that plate. After situation A had been executed, a mug was picked and transported, while after situation C was executed, a mug was also picked and then the drawer was closed. Since historic situations A and C both have picking a mug as a direct successor $n(A) = n(C)$, the current task option s_o has two possible successors in $N(s_o)$: picking a mug and picking a plate. Because picking a plate is the successor of historic situation B with an overall distance of $dist(B, s_o) = 0.3$, its successor value based on Equation 4.12 on the previous page is calculated as $v_1 = \frac{B+0.0001}{1} \approx B = 0.3$. However, the successor value for picking a mug must be calculated based on the distance values of both $dist(A, s_o)$, and $dist(C, s_o)$, using the presented formula. Assuming a **multiple occurrences factor** of $x = 0.7$, this results in:

$$v_2 = \frac{\frac{A+C}{2} + 0.0001}{2^x} = \frac{\frac{0.4+0.5}{2} + 0.0001}{2^{0.7}} = 0.28 \quad (4.13)$$

To determine the event horizon for future events, the algorithm allows to set the hyperparameter **event horizon forward**, which indicates the number of successor tasks to be calculated. This is the only parameter that does not change any distance value. Instead, it determines how many future tasks should be suggested (Figure 4.6 shows a value of 3).

Since the only criterion for predicting successor tasks is previously executed tasks as well as a list of currently feasible tasks, the algorithm might suggest successor tasks that are not feasible given the current world state. This could be due to a different world state, where an object present during the execution of the historic task is not available. It could also be because the historic tasks are not necessarily identical to the current ones but similar. For example, a current task option s_o to pick a spoon may be matched with a historically executed task s_h to pick a butter knife (with a distance > 0), since both the spoon and the knife are cutlery and in this sense relatively similar. In such a case, after the user picked up a knife in the historical situation s_h , a cutting task may have followed in $n(s_h)$. However, in the current situation s_c where the user is about to pick up a spoon s_o , cutting something with a knife is not a feasible option. To prevent such infeasible suggestions, the algorithm

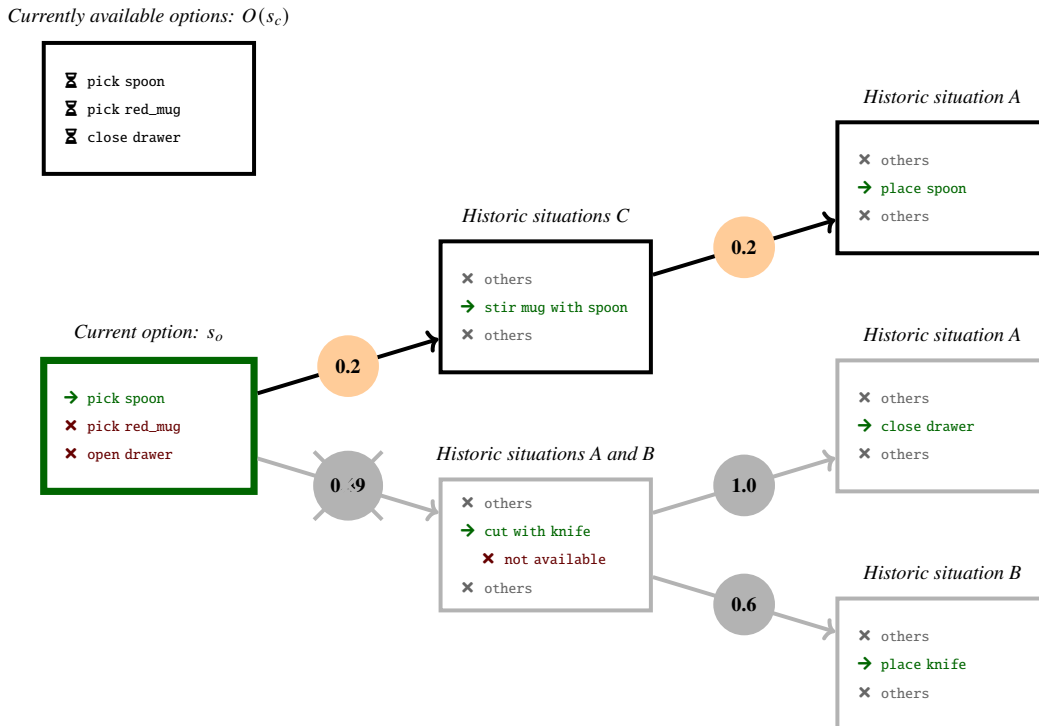


Figure 4.7: Example of finding and removing suggestions from the preference tree that involve objects that are not available in the current world state.

traverses the tree and removes all suggestions that involve objects that are not currently present (see Figure 4.7). An exception applies to tasks of type `soma: 'Opening'`, as these involve opening a container, which may reveal new objects and offer more task options. Consequently, predicted successors of tasks derived from `soma: 'Opening'` are not checked for object existence.

4.5 Hyperparameter Tuning based on Data

Finding good values for all the mentioned parameters can be done automatically using the PREFERENCEORACLE. After some tasks have been executed, the program can reprocess the already executed tasks with different values for the hyperparameters and compare the newly calculated predictions with the executed tasks. For any NEEM, the recorded history can thus be used to create a training scenario, where the history itself serves as ground-truth information. The tuning algorithm is relatively simple. It just iterates over all hyperparameters, recalculates preferences with slightly adjusted parameter values, and focuses in greater detail on parameter values that show promise for improving prediction accuracy. The basis for this fine-tuning process can be either a previously recorded and stored NEEM that is reloaded for tuning purposes, or the calculations can even be performed during runtime while the robot is idling.

The criterion for tuning the hyperparameters can also be adjusted between different top k settings. For a real-world application that involves a tablet screen with large fonts and images

object factor	1.0
task factor	2.0
knockout	1.16
penalty	3.7
event horizon backwards	16
decreasing importance factor	0.81
compare situations	1.0
time of day penalty	1.18
event horizon forward	<i>not trained</i>
multiple occurrence factor	0.25

Table 4.2: *Recommended hyperparameter settings for a good performance in top 5 metric*

that suggests only a few task alternatives, the goal is to have the perfect suggestion within the top five or even the top three suggestions. In scenarios with few alternatives, the goal may be to maximize the number of cases where the first suggestion is already correct, allowing the user to simply confirm with an OK button. For training and evaluation in this thesis, we focus primarily on the top five metric.

Although it seems promising to fine-tune the parameters during run-time and adjust them to the specifics of the current workflow, the overall direction of these values is unlikely to change significantly between workflows. Tuning the parameters using an automated dataset (see the next section) yielded a set of values that provide good default settings. The values that resulted in the most cases with the correct prediction within the top five predictions are listed in table 4.2.

4.6 Modeling Predictions

As mentioned above, SOMA also includes a preference model to store the predicted preferences as part of the ontology (see [3]). The SOMA developers also developed the KnowRob framework (see [8]), which enables querying the generated knowledge base during runtime. By storing the calculated predictions in the SOMA ontology, developers can use KnowRob to query information about previous behavior. This capability provides users with a sense of control and understanding of the robot's behavior. With a history of calculated preferences accessible via KnowRob, the robot can answer questions such as:

Why didn't you do X? This could be something like "Why didn't you pick up the banana?" or "Why didn't you open the door?". In different situations, users might expect the robot to behave in a certain way and may feel that it is obvious what the next task should be. The reason why a desired task was not executed can vary depending on the robot's world state and preference calculations. For example, the robot could explain to the user that it prioritized another task *before* getting to the task the user is asking about. This might even include working on some preconditions for the task that the user forgot about (e.g., boiling the water before making tea with it). On the other hand, the robot may have expected the user to desire a different workflow.

What do you want to do next? Understanding the next tasks that the calculations predict can help the user to make sense of the action that is currently taking place. In situations where robotic actions seem unreasonable at first, users might be able to infer the larger goal of a task based on the next few steps the robot is planning to take. For example, in a situation where the robot starts getting a mug out of the drawer, the user might ask the robot, "What do you want to do with the mug?"

Do not do this, do something else! Although a command interface has not yet been implemented, the preference model is capable of returning the action with the second highest probability of being chosen by the user. If a user wants the robot to automatically infer the next tasks, they might notice that the robot is starting to execute a wrong task. By instructing the robot, "Do not do this, do something else!", KnowRob can review the history and identify the next likely task that had been calculated before starting the current one.

4.7 Summary

This section demonstrates a potential application of integrating NEEMs with the EDAN framework by recording user actions and all their properties in a SOMA compatible knowledge graph, as demonstrated in Chapter 3. This approach enables algorithmic processing of previously executed tasks, their properties, and relationships, thus validating the effectiveness of the integration in improving task anticipation and preference-based decision-making. This study explores how the EDAN framework can leverage recorded actions stored in a knowledge graph to perform a computational analysis on past tasks. In doing so, an algorithm is developed that anticipates user preferences and predicts the next tasks they are likely to execute under various circumstances, effectively demonstrating the utility of integrating NEEMs into the EDAN framework.

The algorithm presented in this chapter combines the traditional notion of preferences with the concept of N-Grams, usually used in next-word prediction. This approach highlights the importance of previously executed tasks by considering all tasks as part of a task chain. The predicted task preferences of a user are calculated using two concepts newly introduced into the EDAN framework by the NEEM logger: (1) the recorded chain of tasks executed by a specific user controlling EDAN and (2) the topological relationship of tasks and objects in the knowledge graph.

By calculating the suggestions solely based on a user's previous actions, the algorithm ensures complete safety regarding user data and does not require users to share their workflows for training purposes. All calculations can be performed directly on the computer attached to EDAN and are deterministic, without using external data. This also ensures the algorithm's compatibility with different world setups for the EDAN robot. Introducing new objects with a new object hierarchy or removing existing ones does not pose additional challenges to the algorithm. Furthermore, the algorithm can accommodate users with diverse behaviors, cultural backgrounds, habits, and abilities without introducing any pre-trained bias into the

preference predictions.

Chapter 5

EVALUATION

5.1 Introduction

To evaluate the predicted user preferences, they can be compared with the actual decisions made by a user. Such a comparison, however, requires a larger dataset of human interactions with the EDAN robot, that can be used to first calculate user-specific preferences and then compare them to the actual human decisions. Such a dataset in the form of the previously mentioned NEEMs has been generated for this chapter:

By equipping the NEEM Logger to the robot EDAN, we could record the actual behavior of users interacting with the robot and assess whether the preference calculations are able to predict that behavior. On a larger scale, this approach would be the gold standard for a field test and provide the best proof of the quality of the calculated preferences. However, the EDAN robot is still under development and has a limited, albeit steadily increasing, set of available actions and objects to interact with. In recent competitions, the team has demonstrated EDAN's ability to interact with various objects, e.g. EDAN can grasp a towel, empty a dishwasher, open a door, or retrieve a package from a physical mailbox (see [20]). Although the list of tasks is impressive from a technical standpoint, the number of different objects and tasks is still quite limited. Clearly, the research team focuses on learning to handle different types of object that present technical challenges, rather than increasing the number of different objects that can be handled in a similar way. Although handling a mailbox, a towel, and a dishwasher is impressive, these tasks are too dispersed to be included in a scenario in which different users behave differently. We need a setup with similar objects and similar tasks to make it challenging to predict different user behavior. With only one cup and one thermos bottle in the kitchen, every user would use that one cup and that one thermos bottle. But what if there were five different fruits, a thermos bottle, a water bottle, and three different cups, which would still be a very limited setup, compared to a real kitchen? These are the questions where preference calculations become interesting and provide meaningful results. However, even with a more equipped laboratory kitchen with a larger set of similar objects, simulating a user's daily workflow with all its repetitiveness would require a significant amount of time and energy (both human and electrical). Since most other research topics working with EDAN focus on refinement of specific task types or identifying and overcoming technical challenges, repeatedly executing and recording a similar workflow every day would not be particularly helpful for researchers.

Rather than using the robot to execute and record a daily workflow, we can also gather data by having people describe their daily routines to us. This enables us to significantly expand the dataset and record a wide variety of people's daily routines. Furthermore, this method allows us to include a much larger number of objects in the hypothetical laboratory kitchen

for users to interact with, since none of the objects need to be physically present.

To collect data on people’s workflows, we could conduct a study and interview individuals on the street. This would provide us with a valuable dataset of actual human beings sharing their daily routines. However, this approach has its limitations when it comes to capturing the repetitive aspect of a typical daily workflow. The research would require participants to describe not only their daily activities, but also variations in their routines over multiple days, including subtle differences in their interactions with the kitchen. For example, they would need to recall how one day they took a mug out of the drawer before turning on the coffee machine, whereas on another day they turned on the coffee machine first and then retrieved the mug. Such nuances are difficult to elicit through oral interviews. This issue becomes even more pronounced when considering very minor tasks, such as entering the kitchen solely to grab a glass of water and then leaving again.

Recent advancements in Large Language Models (LLM) have presented an alternative strategy for collecting the necessary data: leveraging the LLM itself. Since it is trained to concatenate words in a way that mimics human language, it should be able to provide natural language descriptions of how different human beings would describe their daily workflows. Although the model’s output is non-deterministic, meaning it rarely produces identical workflows, it can follow detailed instructions on what to include and exclude from the results.

5.2 Generating and Curating the Daily Living Dataset

To create user workflows that can be used to evaluate the PREFERENCEORACLE, a Large Language Model has been prompted to generate a set of synthetic personas with different daily habits. In a first iteration, the experimenter generated a very abstract persona description in yaml format consisting of a name, a description of the persona’s habits (taking just one to two sentences), a daily routine, and some variations to this routine. The daily routine as well as the variations are a list with every entry consisting of a very brief action description and a time for the action to happen. For illustrative purposes, two example personas used to prompt the LLM are shown in Listing 5.1. The complete list of example personas can be found in the Appendix in Listing B.1 on page 83. The LLM was then prompted to replicate the same basic file structure, but with different personas and varying routines and preferences.

```

1 — file_name: vegan_twenties
2   persona: You are a healthy-living vegan in your twenties, trying to stay
3             away from coffee as best you can.
4   daily_routine:
5     - description: You want to prepare a breakfast with tea
6       time: 10
7     - description: You get home from college and want to relax.
8       time: 15
9     - description: You want to eat dinner.
       time: 20

```



```

10  variations:
11    - description: You find yourself drinking coffee.
12      time: 13
13
14  — file_name: 90_year_drinking_man
15    persona: You are a 90-year old man with a heart condition and a tendency
16             to drink too much.
17    daily_routine:
18      - description: You want to prepare breakfast
19        time: 07
20      - description: You want to snack.
21        time: 14
22      - description: You want to eat dinner.
23        time: 20
24    variations:
25      - description: You are getting home from the doctor
26        time: 15
27      - description: You want to drink something
28        time: 18

```

Listing 5.1: Excerpt of the list of personas, used to prompt an LLM for some auto-generated personas

By incorporating different prompts, such as a persona's love for tea or a favorite cup, the LLM created a more nuanced and realistic set of characters. This resulted in a set of more than one hundred different personas, each with a unique daily routine and some variations on that routine.

Given that this project does not intend to contribute research data to for-profit organizations, the selection of LLMs is limited to those that can be run locally without compromising data security. Furthermore, the chosen model must be capable of adhering to strict boundaries in its text output, specifically interacting only with the predefined kitchen setup outlined in table 5.1. After careful consideration, the model `qwen2.5:32b` was chosen (see [6]), as it was the most competitive open-source model at the time of the test runs, yielding the best results in its more performant 32b version, surpassed only by the less performant Llama 70b (see [49]). Our experience with the model, particularly in the context of this thesis, has shown that it produces the most satisfactory results. However, in instances where performance was of greater concern and prompts were relatively simple, the model `mistral-small` has also been used.

The synthetically generated personas have served as an input parameter for a next prompt, asking the LLM to elaborate on the actions of a user with a specific user description (for example: "*You are a healthy-living vegan in your twenties, trying to stay away from coffee as best you can.*"), acting to achieve a specific goal at a given time (for example: "*You get home from college and want to relax. It is 3 pm.*"). All entries in the previously generated daily routine as well as the variations are hereby included to each define a different prompt. This process is repeated for the equivalent of seven virtual days, during which the daily

routine is being repeated, but occasionally replaced with the stored variations at random.

To limit the synthetic list of actions, a hypothetical laboratory kitchen setup was created, which includes a variety of cups and glasses, different types of beverages, and various food items. Since the evaluation is conducted outside of the robot's actual environment, this setup was able to incorporate objects and tasks that the robot has not yet learned to interact with or to execute, making the calculations more interesting. This hypothetical kitchen setup, comprising a set of objects, their hierarchical structure, and a set of tasks, is presented in table 5.1 on the following page. The setup also incorporates the concept of certain objects being contained within other objects, such as drawers or a dishwasher, which can only be accessed and interacted with if the container object is opened first.

Based on both the description of a situation as well as the description of a kitchen environment including available objects and available tasks with some objects being enclosed in others, the LLM has been prompted to generate a list of actions to achieve the goal, described in the situation. Table 5.2 on page 52 provides an example of the textual description of the user's workflow generated through this method, as well as an excerpt of the prompt.

The resulting textual descriptions of tasks, such as *"I pour the contents of the thermos into the green mug"*, are then transformed into the machine-readable format of `<task>(<obj1>[, <obj2>])`, using a third prompt for an LLM. Tasks like *"I pour the contents of the thermos into the green mug"* have thereby been transformed into `pour(thermos, green_mug,` while tasks like *"I pick up the thermos"* have been transformed into `pick(thermos)`. Although language models excel at following prompted restrictions compared to other non-deterministic models, there have been instances where the prompt did not result in a task that could be executed in our hypothetical kitchen setup. In some cases, the model omitted trivial intermediate tasks necessary for executing a specific task, such as opening a drawer before retrieving an item. In other cases, the model invented or misspelled objects or tasks. While the former issues were relatively easy to resolve using a reasoner that added the necessary intermediate tasks, the hallucinated objects and tasks posed a more significant problem. After making some minor manual adjustments, datasets with an excessive number of errors were discarded.

This results in a dataset, that consists of a number of machine-readable tasks for each synthetically generated persona. Together with a world description as shown in Table 5.1 on the following page, these actions are read and stored as NEEMs by the NEEM Logger, forming a NEEM for every persona.

5.3 Methodology

A synthetically generated dataset of NEEMs, generated by a Large Language Model and recorded by the NEEM Logger, consisting of 105 NEEMs, has been built as described in the previous section. Each NEEM consists of a number of actions by one hypothetical persona, with 105 hypothetical personas each being recorded in their own separate NEEM. This dataset has been split into a subset of 17 NEEMs used for development and fine-tuning

kitchen_table:	
counter_top:	
thermos	pick, place, pour, drink_from
water_bottle	pick, place, pour, drink_from
bread	pick, place, eat
big_drawer:	open, close
knife	pick, place, cut, eat_with
spoon	pick, place, stir
fork	pick, place, stir, eat_with
chamomile_tea_bag	pick, place
peppermint_tea_bag	pick, place
green_tea_bag	pick, place
black_tea_bag	pick, place
chai_tea_bag	pick, place
small_drawer:	open, close
instant_coffee	pick, place, pour
red_mug	pick, place, drink_from
green_mug	pick, place, drink_from
blue_mug	pick, place, drink_from
drinking_glass	pick, place, drink_from
espresso_cup	pick, place, drink_from
cereal_bowl	pick, place, drink_from
cereal_box	pick, place, pour
plate	pick, place
fruit_bowl:	
orange	pick, place, eat
apple	pick, place, eat
lemon	pick, place, eat
banana	pick, place, eat
fridge:	open, close
milk_carton	pick, place, pour, drink_from
juice	pick, place, pour, drink_from
beer_bottle	pick, place, pour, drink_from
cucumber	pick, place, eat
avocado	pick, place, eat
yogurt	pick, place, pour
tap:	turn_on, turn_off
coffee_machine:	turn_on, turn_off
microwave:	turn_on, turn_off
dish_washer:	turn_on, turn_off
stove:	turn_on, turn_off

Table 5.1: List of objects and their afforded tasks in the kitchen setup used for evaluating the preference calculations

User:

You can do the actions

- pick
- place
- ...

With the following objects:

```
"Drinkable": [  
  "blue_mug",  
  "red_mug",  
  ...  
],
```

```
"Pourable": [  
  "milk",  
  "juice",  
  ...  
],
```

You want to drink some water. It is 11am. What do you do?’

Assistant:

- I open the drawer.
- I pick up the green mug.
- I place the green mug on the countertop.
- I pick up the thermos.
- I pour the contents of the thermos into the green mug.
- I place the thermos.
- I pick up the green mug.
- I drink from the green mug
- I place the green mug.

Table 5.2: *Excerpt of the prompt to generate a textual user workflow as well as an example of the answer*

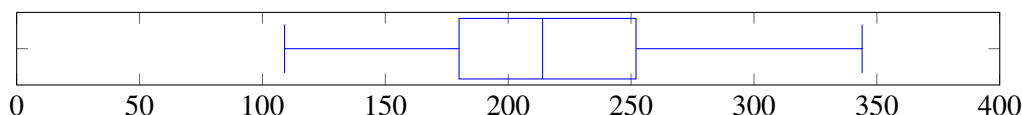


Figure 5.1: *Number of actions per sample NEEM in the test dataset*

purposes (training dataset) and a subset of 88 NEEMs that have not been seen before and used for testing. As described in Section 4.5 on page 43, the `PREFERENCEORACLE` is equipped with a function for self-fine-tuning. This function had been run on the training dataset, resulting in a recommendation for a specific set of parameter settings (see Table 4.2 on page 44).

The remaining test dataset consists of 88 NEEMs / personas with each NEEM / persona containing a different number of actions. As shown in Figure 5.1, the average number of actions per NEEM is around 218 with a very small NEEM consisting of only 109 actions and a very large NEEM consisting of 344 actions. This dataset has then been used to answer the following research questions:

1. With what accuracy does the `PREFERENCEORACLE` predict the next action? With what accuracy does the correct prediction lie within the Top 3 / Top 5 predictions? How does this accuracy compare to chance?
2. How many actions by a specific user does the algorithm need to see in order to have an accuracy of at least 80 % for the correct prediction lying within the Top 5 of predictions?
3. How big is the influence of the different hyperparameters, i.e. how much does the prediction accuracy change for each hyperparameter that is being set to its neutral element?
4. How much time does the algorithm need to calculate a prediction?

To answer questions (1), (2), and (4), the NEEM Logger has loaded each of the NEEMs in the test dataset, one NEEM at a time, progressively. The equipped `PREFERENCEORACLE` has been set to iterate over all actions in a given NEEM with its hyperparameters set according to Table 4.2 on page 44. For each action, the `PREFERENCEORACLE` was instructed to predict the next action solely based on the list of actions prior to the current one. Given only the respective prior actions, the `PREFERENCEORACLE` has predicted a list of successor tasks, ordered by their predicted probability of being executed in the next action. The predicted next action has then been compared with the actual next action in the NEEM, resulting in a hit or a miss for each of the Top 5, Top 3, and Top 1 metrics with a hit defined as the correct next action being predicted as part of the Top 5, Top 3, or Top 1 of predictions.

In another iteration, the `PREFERENCEORACLE` is set to repeat the process with different hyperparameters. To answer question (3), each hyperparameter has been set to its neutral

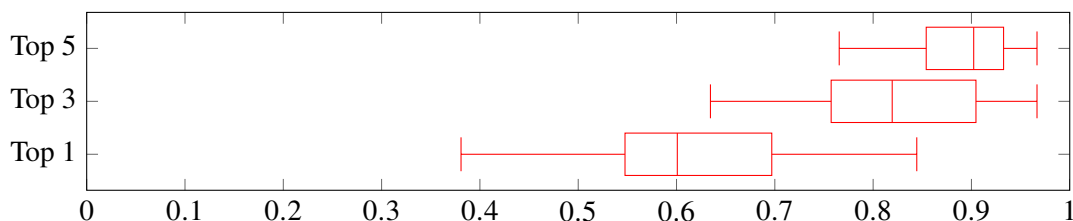


Figure 5.2: Comparison of prediction accuracy for the recommended settings, using different metrics

element, some hyperparameters have also been set to a very high value, with the remaining hyperparameters left at their recommended settings, according to Table 4.2 on page 44.

All calculations have been run on an Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz with x86_64 architecture. Even though the machine is equipped with 56 CPUs, the NEEM Logger and all evaluated algorithms run single-threaded and use only one CPU at a time. The machine is equipped with 250 GB of RAM with an additional swap memory of 5.9 GB. All presented experiments have been run on the Linux OPENSUSE 15.5 system.

5.4 Results

To get a general idea of the accuracy of the prediction of the PREFERENCEORACLE, Figure 5.2 presents a box plot illustration of the overall average accuracy of the PREFERENCEORACLE in the test setup for the Top 1, Top 3, and Top 5 metrics. It shows that all NEEMs achieved at least a 75 % accuracy averaged over all actions in the NEEM for the Top 5 metric, with an overall average of 89.1 % and some NEEMs exceeding 96 % over all actions. For the Top 3 metric, the best-performing samples achieve over 96 % accuracy as well, while the average drops to just above 80 %. When considering only the topmost prediction, the overall average accuracy lies at around 60 %. However, the Top 1 metric shows a much wider variability, with the PREFERENCEORACLE performing on some NEEMs with an accuracy of below 40 % and on some other NEEMs nearing 85 %.

Breaking down the results by the number of actions recorded, Figures 5.3 on the next page and Figure 5.4 on the following page show an evolution of the accuracy of the prediction over time. Figure 5.3 shows that the prediction accuracy across all metrics is near chance for the first 9 predictions with an accuracy of (near) zero for the first actions. After the algorithm has seen the first 9 actions, the 10th prediction already lies significantly above chance. Until around the 30th action, the prediction accuracy continues to increase for all metrics and stabilizes slightly above the overall averages, already shown in Figure 5.2. For comparison, the chance level shows a steady accuracy of around 5 % for Top 1, around 15 % for Top 3, and around 24.5 % for Top 5. Depending on the context of a given situation in a sample NEEM, the robot affords a different number of actions, which is indicated by a slight fluctuation in the chance level across the number of actions.

While Figure 5.3 is cropped after 75 actions to get a closer look at the evolution of the prediction accuracy within the first few actions, Figure 5.4 shows the accuracy for the Top 5 metric over all actions that have been recorded in the test dataset. After approximately

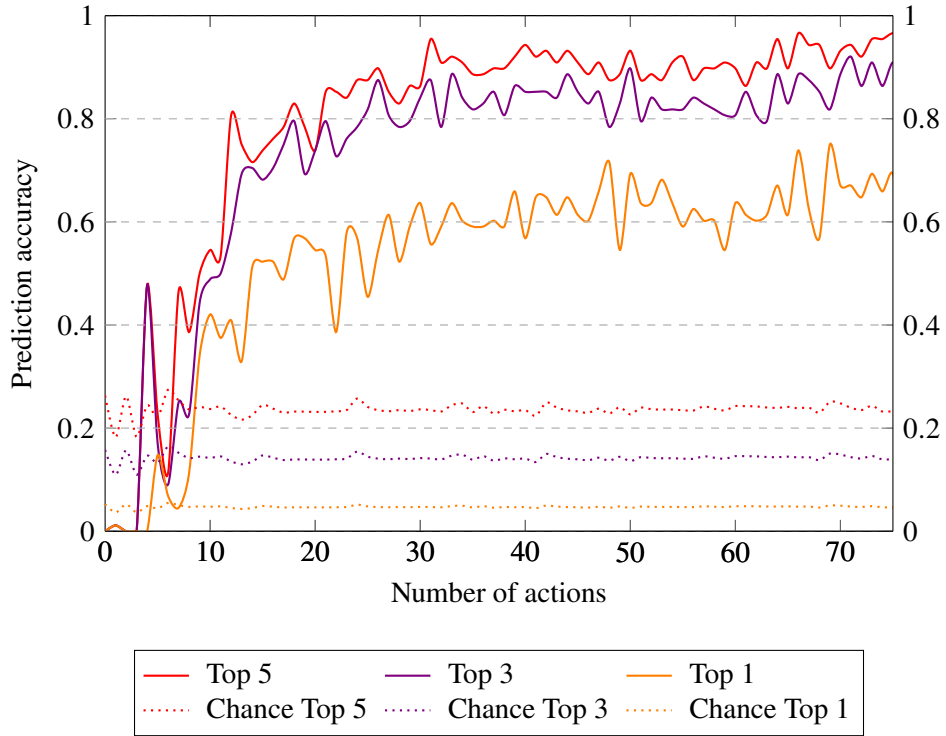


Figure 5.3: Average prediction accuracy after a number of actions in the test dataset.

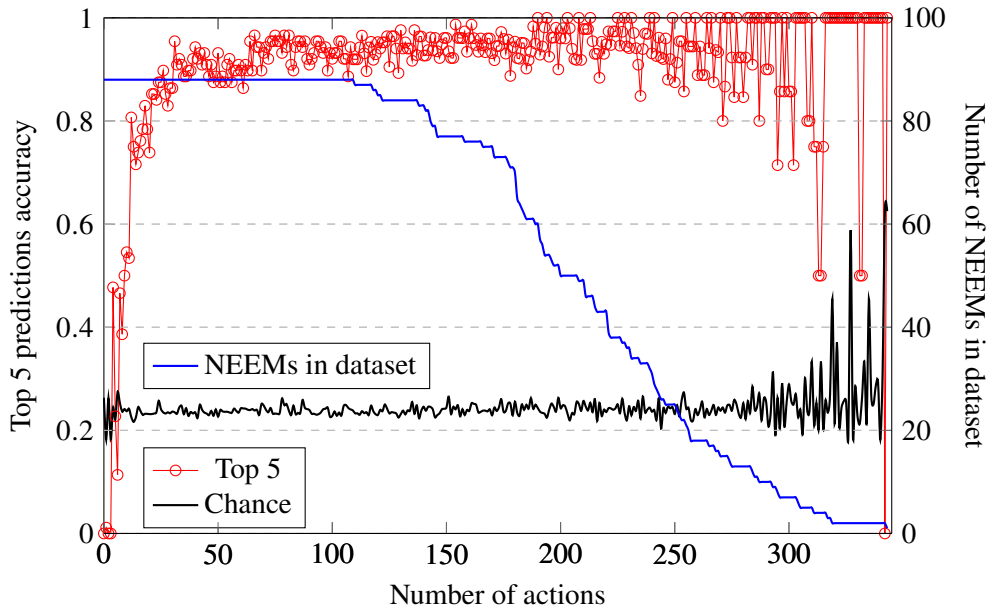


Figure 5.4: Average prediction accuracy after a number of actions in the test dataset, compared to the number of samples providing a different number of actions.

200 to 250 actions, the reader can observe greater fluctuations in accuracy, with some data points showing a lower accuracy of below 80 % after 250 actions and even below 50 % after 300 actions, while the majority of data points show an average accuracy of 100 %. This higher fluctuation correlates with a lower number of NEEMs in the test dataset, which contain more than 200 actions. While all sample NEEMs contain at least 100 actions, the number of NEEMs reaches less than 20 after around 250 actions. This also results in a higher fluctuation in the chance level: Depending on the context of a situation, EDAN has fewer or more actions available to be executed, and therefore different chances for a random suggestion to pick the task that was actually executed.

To assess how various parameters influence the accuracy (question (3)), Figure 5.5 on the next page presents a box plot of the overall Top 5 accuracies when the PREFERENCEORACLE is run with the recommended settings, compared to the results of setting each of the parameters to their neutral element. The corresponding box plot for Top 3 (Figure B.1 on page 85) and Top 1 (Figure B.2 on page 86) metrics are included in the appendix for further reference.

Given the synthesized dataset, the `time of day penalty` fails to show a significant decrease in prediction accuracy when set to its neutral factor 1 (Plot D), which effectively disables the penalty for actions at different times of day. On the other hand, a penalty of factor 10 (Plot C), which completely disregards actions from different times of day, shows significantly worse results with an accuracy of 85.5 % and some outliers having an accuracy of less than 70 %.

Plot H shows a significant decrease in accuracy, down to an average of 55 %, when object types and their roles in a task are ignored. In contrast, Plot I shows a dramatically smaller (yet significant with $p = 0.003$ in the Mann Whitney U test) decrease in accuracy down to 86.6 % on average, when task types rather than object types are ignored.

Changes in the `knockout` parameter, which determines the degree of similarity required between tasks and objects to be considered similar, show a big difference in accuracy. Setting it to a very low value of 0, where only identical tasks with identical objects are considered similar, provides significantly worse results of 81.6 % as shown in Plot L. Conversely, setting it to a very high value of 20, where almost all tasks are considered similar but still ranked by their calculated similarity score, results in an even worse performance of 52.7 % accuracy, as shown in Plot M.

Setting the `penalty` parameter to its neutral element can either mean to set it to be equal to the `knockout` parameter. This leads the algorithm to omit any distinction between the *dist* value of a situation that is beyond `knockout` and a situation that is just reaching `knockout`. Setting the `penalty` parameter in such a way, results in a slight, but not significant ($p = 0.052$ in the Mann Whitney U test) drop in prediction accuracy of on average 87.3 %, as shown in Plot J. On the other hand, the parameter could be set to be very high, resulting in effectively eliminating any situation and element of a situation, that receives the `penalty`. In the test runs for Plot K, the parameter had been set to 20, resulting

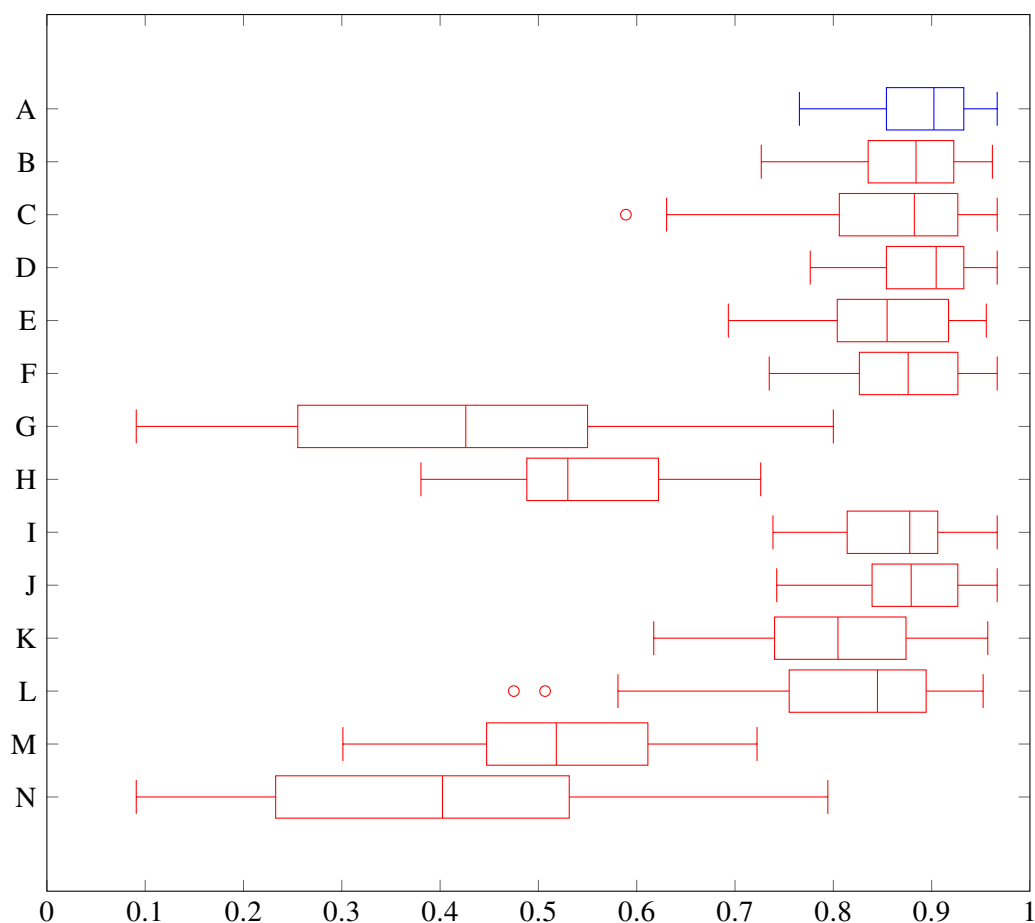


Figure 5.5: Comparison of Top 5 accuracies between different settings.

Legendary:

A: Recommended settings (see tab. 4.2 on page 44)

B: `compare situations` = 0

C: `time of day penalty` = 10

D: `time of day penalty` = 1

E: `multiple occurrences factor` = 1

F: `multiple occurrences factor` = 0

G: `decreasing importance factor` = 1

H: `object factor` = 0

I: `task factor` = 0

J: `penalty` = `knockout`

K: `penalty` = 20

L: `knockout` = 0

M: `knockout` = 20

N: `event horizon backwards` = 0

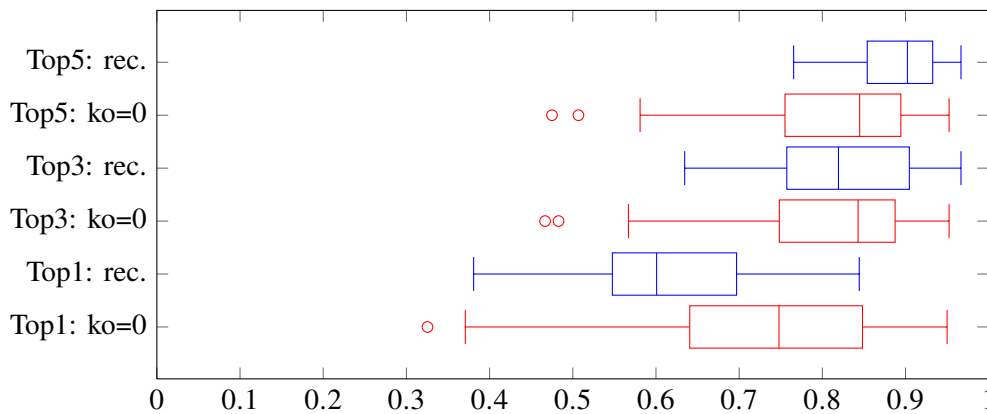


Figure 5.6: Comparison of prediction accuracies for Top 1, Top 3, and Top 5 metrics, comparing recommended settings (see Table 4.2) and `knockout` = 0.

in an average accuracy of 80.9 %.

The parameter with the greatest impact on the prediction outcome is determining, how far the order of task execution should be considered: `event horizon backwards`. Plot N shows the PREFERENCEORACLE to perform poorly when comparing only singular events with the current situation, i.e. setting the parameter to 0 and ignoring all predecessors of situations, resulting in an average accuracy of 39.9 %.

While most settings exhibit similar results for the Top 3 and Top 1 metrics as they do for the Top 5 metric, the `knockout` value is an interesting exception, particularly when `knockout` = 0 (see Figure 5.6). For the Top 5 metrics, a lower `knockout` value generally leads to poorer performance, as already shown in Figure 5.5 on the previous page. However, for the Top 3 metric, the relationship becomes less clear-cut. Although for the the best and worst performing sample NEEMs, the accuracy for Top 3 predictions is higher with the recommended settings (see Table 4.2), the average performance across all NEEMs in the test dataset is with 82.5 % higher for setting `knockout` = 0, than using the recommended settings, which provides an average accuracy of 81.2 %. This trend is even stronger for the Top 1 metric, which provides better overall results with setting `knockout` = 0 resulting in an average accuracy of 73.5 %, than running the PREFERENCEORACLE with the recommended settings, which provides an average accuracy of 61.9 %. On the other hand, Figure 5.6 shows a low `knockout` value to provide more negative outliers, i.e. samples that the algorithm handles significantly poorer than the average sample NEEM.

With different parameter settings, not only does the prediction accuracy vary, so does the computing time. For running the recommended settings in a process as discussed in Section 5.3, the computing time per action ranges from apx. 0.1 seconds up to apx. 0.5 seconds, as shown in Plot A of Figure 5.7 on the following page. On average, these settings require 0.19 seconds per action for the PREFERENCEORACLE to calculate its suggestions. Although many parameters have no impact on computing time (see plots C, D, E, F in Figure 5.7), some parameters do.

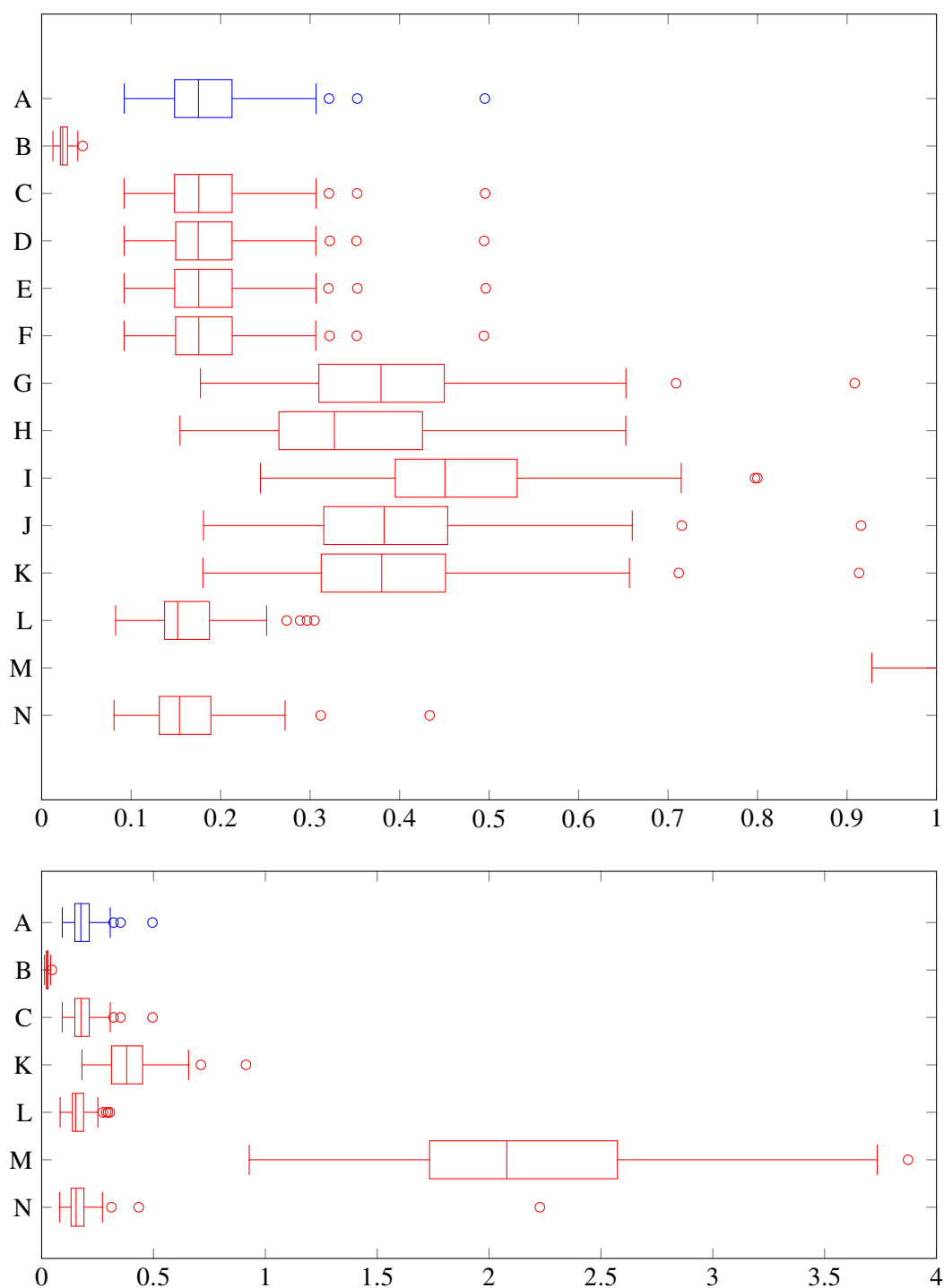


Figure 5.7: Comparison of average computing time per task using different settings

Legendary:

A: Recommended settings (see Tab. 4.2 on page 44)

B: `compare situations` = 0

C: `time of day penalty` = 10

D: `time of day penalty` = 1

E: `multiple occurrences factor` = 1

F: `multiple occurrences factor` = 0

G: `decreasing importance factor` = 1

H: `object factor` = 0

I: `task factor` = 0

J: `penalty` = `knockout`

K: `penalty` = 20

L: `knockout` = 0

M: `knockout` = 20

N: `event horizon backwards` = 0

When setting the `event horizon backwards` to 0 (Plot N in Figure 5.7), the average computing time per action is apx. 0.186 seconds, which is a small, yet statistically significant reduction (with $p = 0.002$ in the Mann Whitney U test).

Perhaps surprisingly significant regarding computing time reduction is a low `knockout` value (see Plot L and M). While a value of 0, which eliminates a higher proportion of tasks early in the calculation process, reduces the computing time down to 0.165 seconds (with $p = 0.004$ in the Mann Whitney U Test) (Plot L), calculating with a high `knockout` value of 20 (Plot M), the computing time dramatically increases up to nearly 4 seconds per task in the worst sample case, which reaches a level that is barely usable in a real life scenario.

Setting a lower value for the `object factor` (Plot H), the `task factor` (Plot I) or the `penalty` (Plot J) effectively lowers all non-perfect *dist* values for situation comparisons and therefore have, regarding computing time, the same effect as choosing a higher `knockout` value: They increase the computing time, but all stay below one second per prediction.

The most significant impact on computing time is associated with the parameter `compare situations`. When set to any value greater than 0, every task in the current situation is compared against all tasks in every historical situation, including not just those that have been executed but also all available alternatives at any moment. In contrast, setting this parameter to 0 completely skips these comparisons, resulting in a substantial reduction in computational effort down to 0.025 seconds per action, ensuring that the computing time remains below 0.1 seconds per prediction even for outliers.

5.5 Discussion

As described above, the evaluation works under the assumption that the output of a large language model, which has been used to generate the NEEM dataset, approximates the statistical distribution of actions in a human workflow. Since large language models do have a bias to follow statistical patterns, there is reason to look at this hypothesis critically, since the model might have generated a much cleaner dataset than human workflows would have. However, testing this hypothesis is beyond the scope of this thesis.

While the PREFERENCEORACLE does show a reasonably high accuracy of close to 90 % in providing the correct next task within its Top 5 of predictions given the generated dataset, the score for a perfect suggestion is less impressive with around 60 %.

Given that the algorithm does not require preexisting knowledge in the form of any training, nor is it limited to a specific context of objects and tasks, the prediction accuracy starts converging reasonably fast. After just 10 actions, the algorithm already provides custom predictions that are well above chance level with the accuracy increasing for the next 20 to 40 actions, where the accuracy seems to converge. This is a time frame that might allow for even smaller demonstrations with the robot to show an interesting prediction.

The hyperparameters show different results regarding their effect on prediction accuracy. The `time of day penalty` does not seem to increase prediction accuracy, which may

be due to the synthetic dataset not providing enough variety of actions depending on the time of day. Assessing the influence of the time of day for a user workflow would require some human data, which goes beyond the scope of this thesis. Different settings for the `multiple occurrences factor` also seem to have very limited effect on the prediction accuracy. For future iterations, it might be reasonable to remove this hyperparameter and set a constant number for this parameter instead.

Recognizing different tasks appears to be highly dependent on recognizing the objects involved in the task. This is evident from the significant decrease in accuracy when object types and their roles in a task are ignored, and the minimal decrease in accuracy when task types are ignored. A possible explanation for this behavior is that a specific setup of objects and their roles in a task (for example: using `odb:Thermos` to perform an unknown task with object `odb:BlueMug`) is typically associated with only one or two task types (the example can only be achieved with the `soma:PouringInto` task). On the other hand, the same task type can often be executed with a number of different objects. Depending on the structure of the object and task hierarchy, as well as the type of available tasks, this behavior might change for different setups.

For the `event horizon backwards`, a higher value provides a higher prediction accuracy, but also increases the computing time. However, the evaluation has shown that the increase in computing time is barely measurable. In future iterations, this parameter might also be omitted in favor of a high constant value for the event horizon.

The most important hyperparameters for a good prediction accuracy seem to be on the one hand the `task factor` and `object factor` and on the other hand the `knockout` parameter, which also depends on the structure of the generated ontology. In a very fine-grained ontology with a lot of subconcepts, the paths between objects are generally all longer and therefore require a higher value for the `knockout` parameter than in an ontology with just a few very general distinctions in objects and tasks. Since EDANs odb has been created with a focus on supporting existing applications, the question must be raised, whether the resulting ontology even returns a meaningful knowledge graph that allows to calculate graph distances, as presented in this chapter. To some extent, this question can be answered Yes, given the prediction accuracy presented. However, a restructuring of the object and task hierarchy might yield some further improvements in accuracy.

The parameter `compare situations` should also be further investigated. In the synthetic dataset, the parameter does not yield a big improvement in prediction accuracy, on the other hand, the parameter controls the single most time-consuming aspect of the algorithm. Comparing the given results with some real-life evaluation could help in deciding whether to remove that part of the algorithm completely.

All in all, the computing time to predict the next action usually stays below 0.5 seconds with an average of 0.19 seconds, which does allow the algorithm to be run in a real life application.

5.6 Overview over the Hyperparameters

Overall, the evaluated test runs show the following effects of the different parameters:

The **compare situations parameter** is a multiplier that controls the weight given to the surrounding situation in the prediction. It can be set to exactly 0, which significantly reduces the computing time. Alternatively, setting it to a higher value between 0 and 1 provides slightly improved prediction accuracy with an optimal value around 1.

The **time of day penalty** is a penalty multiplier for tasks that have been executed at different times of day. The neutral value for this parameter is 1, indicating that there is no penalty. The impact of this parameter changes across datasets and depends on how significantly a user's workflow changes throughout the day. A suitable value ranges from 1 to 2.

The **multiple occurrences factor** weights the number of similar tasks found in history against the average calculated distance to those tasks (see formula 4.12 on page 41). A value of 0 completely disregards the number of hits, considering only the average task distance, whereas a high value of 1 or even 2 gives significant weight to the number of hits and reduces the importance of the distances to the found tasks. A suitable value would be between 0.6 and 0.9

The **decreasing importance factor** weights the predecessor tasks with respect to the main tasks to be compared (see formula 4.11 on page 39). A value of 0.1 makes almost no distinction between the comparison of the main task and the comparison of the predecessor task with the predecessor of a historic task. On the other hand, a value of 1 assigns all the weight to the main task and completely ignores all predecessor tasks. An optimal value for this parameter depends on the **event horizon backwards**. With a high event horizon, this factor needs to be smaller to still recognize events lying further back. A good value is around 0.25.

The **object factor** and **task factor** control the relative weight of task types versus object types when calculating distances between tasks. Experiments have consistently shown that the best performance is achieved with the object factor set to 1 and the task factor to 2.

The **knockout parameter** has the most significant impact among the listed parameters. A low value (near or equal to 0) considers only very similar or identical tasks similar and discards all other tasks. This leads to a shorter computation time and a higher probability that the correct task is the topmost prediction of the algorithm. In contrast, a higher value allows the algorithm to make non-perfect suggestions with a high accuracy of being within the Top 5 even with a limited recorded history. A value of around 1 indicates a higher chance that the correct task is within the Top 5 predictions with the evaluated settings. However,

the optimal value also depends on the structure of the object and task hierarchy, since a more detailed knowledge tree results in higher values for the calculated distances between different tasks and objects and therefore requires a higher **knockout** value to achieve the same effect.

The **penalty parameter** determines the distance assigned to items that exceed the **knockout** value. For example, the algorithm could compare two actions as well as their predecessors with a current task option. While one of the actions has five predecessor actions that are all similar to the predecessor actions of the current task (i.e., they all have a *dist* value that is below the **knockout** parameter), the other action could have only three similar predecessor actions, with the fourth having a *dist* value that reaches beyond the **knockout** parameter. In such a case, the fourth and fifth predecessor action in the latter chain will automatically get assigned this **penalty** value. This parameter should be set slightly higher than the **knockout** value, otherwise, it would not be a penalty but a reward. However, setting the parameter too high reduces the prediction accuracy, since an action would automatically be considered dissimilar just because there is a predecessor that is different from the predecessors of the current action. A suitable value for this parameter would be between 4 and 5 with the **knockout** parameter set between 1.0 and 1.5.

The **event horizon backwards** determines the number of predecessor tasks that are compared as a property of any task. Although a low event horizon slightly reduces computation time, a higher event horizon shows a significant increase in prediction accuracy. A value greater than 10 is recommended.

The **event horizon forward** has no impact on prediction accuracy. Instead, it determines the number of successor tasks that the algorithm proposes after the next task. In terms of an N-Gram, this parameter determines the N, i.e. the number of tokens, for the algorithm.

5.7 Summary

By using both aspects of the SOMA ontology, the algorithm provides the user with a set of task suggestions that have a 90 % chance of the preferred task being in the Top 5 suggestions and a greater than 80 % chance of the preferred task being in the Top 3 suggestions.

However, conducting user studies involving a broader user base over extended time periods, using a comprehensive set of objects and tasks to approximate real-world workflows, is currently infeasible. This limitation arises primarily due to the ongoing prototype status of EDAN and its concurrent development by other researchers. Consequently, the proposed algorithm has been evaluated using a synthetic dataset generated through a prompt-based pipeline with a large language model. The dataset contains several newly introduced objects and actions, making preference prediction more challenging, and includes over 200 actions per user that are used to predict user-specific preferences. The evaluation is carried out by comparing these predictions with previously stored actions within the datasets. Additionally,

this methodology has been used to explore how various hyperparameters influence the algorithm's behavior regarding different user needs. This approach provides insights into the algorithm's adaptability and performance across a range of parameter settings, revealing variations in its predictive capabilities depending on these adjustments.

Chapter 6

CONCLUSION

6.1 Thesis Summary

This thesis has presented the NEEM Logger, which interfaces the knowledge representation for the robotic framework EDAN with the SOMA ontology. The NEEM Logger automatically generates a merged ontology that is based on the SOMA ontology and the latest concepts of EDAN’s internal object database. The merged ontology is the basis for logging not just all actions done by the EDAN robot and the perceived objects, but also all alternatively available actions, given the world state at any given time.

The proposed NEEM Logger records a semantic log in the form of a narrative-enabled episodic memory (NEEM), that can be used for a variety of further calculations. To demonstrate one use case, this thesis has also presented the PREFERENCEORACLE, which reads the semantic log and infers the next tasks based on the notion of task chains. Similar to an N-Gram, the algorithm considers tasks to be tokens in a very long chain of tokens, with the previous tokens determining the probability of the next token. For tokenization of tasks, this thesis proposes a notion of similarity between situations based on the taxonomic distance of task types, object types, the time of day, and alternatively available tasks. By combining the idea of an N-Gram algorithm with this notion of similarity between tasks, allowing the token chain to consist of not just identical, but similar tokens, the algorithm is able to predict a next task to be executed solely based on the chain of previously recorded situations by the same user without requiring any other logging data and without exposing any logging data to the world.

To evaluate the prediction accuracy of the presented algorithm, this thesis has built a synthetic dataset with the use of modern large language models to approximate daily human actions in a kitchen over the course of a week. The models have been instructed to first generate a set of different personas and then a list of simple actions, those different personas might perform. These sets of actions were the groundwork for testing both the reliability of the NEEM Logger, which was used to record the actions as NEEMs, as well as the prediction accuracy of the PREFERENCEORACLE.

The evaluation has shown that the PREFERENCEORACLE is capable of predicting the next action of a user after as little as 10 to 50 actions recorded by that user with a probability of more than 50 % for the prediction to be correct (compared to a chance level of just below 5 %) and a probability of close to 90 % for the correct prediction to be in the Top 5 of predictions (compared to a chance level of 24.5 %). To accommodate different types of user workflows with, for example, some users having very different workflows depending on the time of day and other users acting regardless of the time of day, the algorithm also allows to be customized with hyperparameters, that can be automatically tuned to achieve the highest

possible prediction accuracy based on the previous workflow of that user.

6.2 Further Research

During the development of and in parts based on the work presented in this thesis, the framework RACCOON has been developed at the German Aerospace Center (DLR) (see [18]):

At least one framework at the German Aerospace Center (DLR) has used the NEEM Logger method for recording a semantic log of NEEMs as presented in this thesis. The framework has been published in an independent publication under the name RACCOON (see [18]). RACCOON uses the recorded NEEMs to generate one of the so-called "state summaries" of the robot state, allowing a large language model to speak and answer questions about the robots past actions.

While Large Language Models have more and more been used to make robots explain aspects of the surrounding world, much of these explanations have been based on general knowledge that can be found on the internet and is incorporated in the large models. With Visual Language Models, a robot is also able to describe that can be seen by the cameras of a robot. But while such models can describe a world similar to how a human would perceive it, that description might fundamentally differ from the way, the robot is able to perceive it. For example, such a model is able to classify almost every object in a kitchen according to its name in natural language, while the robot recognizes only a portion of those objects. A talking robot that explains its environment should always reflect on what objects *are there* and what objects *does the robot know about*. In addition, many technical details such as available robotic actions or previous experiences can not be accurately described by a language model based on visual information only. The framework RACCOON has been built to include a set of robotic modules into the reasoning process of a language model, similar to a RAG system. RACCOON classifies a user's question into multiple categories, indicating which robotic modules might be able to give the requested information. Those modules are then internally queried based on the user's prompt with their answers being added to the prompt as "state summaries", that are additional ground truth information for the language model. One of the robotic modules that have been used for the RACCOON framework is the NEEM Logger, presented in this thesis. For all queries regarding the robot's past decisions and actions, the NEEM Logger is used to send queries to the KnowRob (see [8]) database. In future iterations, this also includes queries to the predicted user preferences, calculated by the PREFERENCEORACLE, allowing EDAN to ask the user in natural language, if they want to execute a specific task that is predicted to be their favorite.

6.3 Future Work on the NEEM Logger

1. When further developing the NEEM Logger, researchers could further explore and rethink the option of including some kind of matching algorithm to automatically merge the SOMA ontology with the odb. In the current state of the odb, this matching can be performed

manually with just a few assertions. However, considering a larger odb, this might no longer hold.

2. On the other hand, the EDAN framework could choose a different path and adapt the odb ontology as their native way to represent tasks and object, overcoming the odb in its current form and forcing any developer adding new objects to the odb to think about where to sort in that object in the ontology. Thinking about the taxonomic relationship of any new object with the pre-existing taxonomy of objects might be considered good practice for keeping a well-organized knowledge representation.

3. The NEEM Logger in its current form stores only *semantic* descriptions of actions. However, the general idea of a NEEM, as also discussed in the previous chapter, is to combine both raw technical data with a semantic annotation (see [15]). Future research projects could build on the NEEM Logger and enable it to record raw technical data by the EDAN robot as well.

6.4 Future Work on the PREFERENCEORACLE

Introducing the PREFERENCEORACLE represents a first step toward using the SOMA ontology use NEEMs to calculate user-specific preferences for desired task chains. One key purpose of the PREFERENCEORACLE was to demonstrate a major use case for transforming the robotic infrastructure into an internally represented, richer, and well-defined ontology that is generated by the NEEM Logger. Based on this work, there are several ways to improve and enrich the presented calculations to help users find their preferred actions faster. Some potential ideas are presented below, ordered by and ranging from readily implementable suggestions to more long-term proposals.

Including the Preference Calculations in a User Interface

With a calculated ranking of available tasks, the user interface can be updated to display the suggestions directly to the user. This can be achieved through a ranked list of suggestions, allowing the user to execute the top suggestion or navigate down the list if desired.

This interface can also incorporate a decision making algorithm that determines whether to propose a single action to the user or to suggest components of a workflow. For example, the user can instruct the robot to complete tasks like setting a breakfast table or brewing coffee, rather than proposing each individual task of the workflow separately.

Remove Invisible Byproducts of Changes in the Hyperparameters or the Ontology.

The algorithm for predicting preferences in its current form heavily depends on a set of hyperparameters. Depending on their settings, the calculated graph distance between tasks and objects are greater or lower, requiring the user to change other hyperparameters as well as a side effect. For example, if the `task factor` gets a higher value, all non-zero values for *dist* are automatically higher, requiring the user to also increase the `knockout` value, which in turn requires a different setting for the `penalty` parameter. These unwanted side

effects should be eliminated by normalizing the resulting value for the *dist* function such that a user can change the value for one hyperparameter without thinking about the others.

A similar problem arises with the structure of the ontology. Depending on the depth of the graphs representing the hierarchy of objects and the hierarchy of tasks, the *dist* values for tasks and objects change on average. For example, the difference between a **Cup** and a **Plate** is considered smaller if those are the most specific definitions of their kind in a given knowledge graph, than if the graph also includes **EspressoCup** or even different brands of **EspressoCup** as further differentiation of the general concept **Cup**.

Improving the Predictions of a Further Future

As discussed above, the EDAN framework includes a module that infers a list of available tasks based on a given situation, applying the algorithm presented in [9]. To improve task predictions with an event horizon that is larger than one, this algorithm could also be leveraged to limit the calculations to the set of actions that would be available if a specific action had been executed before. For example, if the PREFERENCEORACLE predicts the next action to be picking up a knife, the algorithm could be leveraged to infer that stirring a pot with a spoon is not an available action after that. Removing such infeasible predictions could improve predictions that look further into the future.

Including more Robotic Properties in the Preferences

Studies have shown that it is important for disabled and elderly people to be active and perform as many tasks as possible on their own or only with the partial help of a robot. Regarding elderly people, a user study has stated:

"The professional caregivers also immediately stated that older adults should not become passive (similar to the informal caregivers) and that the robot should only perform those tasks the users no longer can perform themselves in order to maintain the abilities they still have ('use it or lose it'). It would be ideal if the user and the robot could perform tasks together. When performing a task together, the robot would only need to perform that part of the task the user cannot perform. For example, when the user is still able to make a cup of coffee but unable to carry the cup of coffee to the living room, the robot should do that latter part of the task for the user." ([7])

Addressing this issue can be integrated within the preference prediction by adding the **autonomy mode** as a task property. This would enable the robot to refrain from automatically executing a complete workflow on its own, instead checking which parts of the workflow the user usually performs in manual mode and which ones they leave to the automatic execution. This could enable the robot to help the user stay active and encourage user engagement by incorporating manual task performance in automated workflows.

Another important property that users typically prefer controlling is the robotic behavior regarding its speed of movement and the manner in which it executes a task:

"Elderly participants found it very important that the way the robot executed a certain task would match the user preferences. It was even suggested by one participant that the robot could be trained by the user (i.e. learning the user's preferences over time) in order to have a perfect match. This would include the way the robot should approach the user (e.g. the speed, the side from which it approaches the user, how nearby the robot stops, the warning signals it uses to announce its presence), meaning the robot should be adaptable per user." ([7])

These properties can also be incorporated into the definition of a task and, as a result, become part of the calculated user preferences, enabling user-specific task execution.

Preparing the Environment

Another field of research that the PREFERENCEORACLE touches upon is preparing the environment to support the most probable user actions. For example, if the robotic system can predict that the user wants to start their morning by drinking coffee from their favorite cup, the robot could suggest cleaning the cup the night before. In multirobot setups, this could even involve other robots preparing a drink that the user is likely to want or finding an object that the user will probably interact with.

Evaluating the Prediction Accuracy with a Real User Base

To gain a deeper understanding of the prediction accuracy of the PREFERENCEORACLE in a real environment, the presented evaluation should be repeated with a real user base. Since users have different schedules and workflows, it is important to evaluate how the calculated prediction accuracies hold for different setups. However, this would require a larger study with users willing to work with the EDAN robot (or a similar setup) over an extended period. Moreover, the robot EDAN would need to be available for those experiments. Both could result in challenges regarding the finding of participants and the permission of the DLR. Nevertheless, the results could be highly informative.

BIBLIOGRAPHY

- [1] Ieee standard ontologies for robotics and automation, 2015.
- [2] Ieee standard for autonomous robotics (aur) ontology, 2021.
- [3] Mona Abdel-Keream, Daniel Beßler, Ayden Janssen, Sascha Jongebloed, Robin Nolte, Mihai Pomarlan, and Robert Porzel. An ontological model of user preferences, 2023.
- [4] Nichola Abdo, Cyrill Stachniss, Luciano Spinello, and Wolfram Burgard. Robot, organize my shelves! tidying up objects by predicting user preferences. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1557–1564, 2015. doi: 10.1109/ICRA.2015.7139396.
- [5] Hisham Al-Mubaid and Hoa Nguyen. A cluster-based approach for semantic similarity in the biomedical domain. *Conference proceedings : ... Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in Medicine and Biology Society. Conference*, 1:2713–7, 02 2006. doi: 10.1109/IEMBS.2006.259235.
- [6] Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, Jianxin Ma, Rui Men, Xingzhang Ren, Xuancheng Ren, Chuanqi Tan, Sinan Tan, Jianhong Tu, Peng Wang, Shijie Wang, Wei Wang, Shengguang Wu, Benfeng Xu, Jin Xu, An Yang, Hao Yang, Jian Yang, Shusheng Yang, Yang Yao, Bowen Yu, Hongyi Yuan, Zheng Yuan, Jianwei Zhang, Xingxuan Zhang, Yichang Zhang, Zhenru Zhang, Chang Zhou, Jingren Zhou, Xiaohuan Zhou, and Tianhang Zhu. Qwen technical report, 2023.
- [7] Sandra Bedaf, Patrizia Marti, and Luc Witte. What are the preferred characteristics of a service robot for the elderly? a multi-country focus group study with older adults and caregivers. *Assistive Technology*, 31, 11 2017. doi: 10.1080/10400435.2017.1402390.
- [8] Michael Beetz, Daniel Beßler, Andrei Haidu, Mihai Pomarlan, Asil Kaan Bozcuoglu, and Georg Bartels. Knowrob 2.0 - a 2nd generation knowledge processing framework for cognition-enabled robotic agents. *International Conference on Robotics and Automation (ICRA)*, 2018.
- [9] Mohamed Behery. A knowledge-based activity representation for shared autonomy teleoperation of robotic arms, 2016. URL <https://github.com/Behery/behery.github.io/raw/master/thesis.pdf>.
- [10] Roger Bemelmans, Gert Jan Gelderblom, Pieter Jonker, and Luc de Witte. Socially assistive robots in elderly care: A systematic review into effects and effectiveness. *Journal of the American Medical Directors Association*, 13(2):114–120.e1, 2012. ISSN 1525-8610. doi: <https://doi.org/10.1016/j.jamda.2010.10.002>. URL <https://www.sciencedirect.com/science/article/pii/S1525861010003476>.
- [11] Daniel Beßler, Sascha Jongebloed, and Michael Beetz. Prolog as a querying language for mongodb, 2021.

- [12] Daniel Beßler, Robert Porzel, Mihai Pomarlan, Abhijit Vyas, Sebastian Hoffner, Michael Beetz, Rainer Malaka, and John Bateman. Foundations of the socio-physical model of activities (soma) for autonomous robotic agents. In *In Formal Ontology in Information Systems - 12th International Conference, FOIS 2021, Bozen-Bolzano, Italy, September 13-16, 2021, IOS Press, 2021. Accepted for publication, 2021*. URL <https://ai.uni-bremen.de/papers/bessler21soma.pdf>.
- [13] Conrad Bock, Achille Fokoue, Peter Haase, Rinke Hoekstra, Ian Horrocks, Alan Ruttenberg, Uli Sattler, and Michael Smith. Owl 2 web ontology language structural specification and functional-style syntax (second edition), world wide web consortium (w3c). Online, December 2012. URL <http://www.w3.org/TR/2012/REC-owl2-syntax-20121211/>. Last accessed 12-November-2021.
- [14] W.N. Borst. *Construction of Engineering Ontologies for Knowledge Sharing and Reuse*. Phd thesis - research ut, graduation ut, University of Twente, Netherlands, September 1997.
- [15] Asil Kaan 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.
- [16] Asil Kaan Bozcuoglu, Gayane Kazhoyan, Yuki Furuta, Simon Stelter, Michael Beetz, Kei Okada, and Masayuki Inaba. The exchange of knowledge using cloud robotics. *IEEE Robotics and Automation Letters*, 3(2):1072–1079, 2018. doi: 10.1109/LRA.2018.2794626.
- [17] Samuel Bustamante, Gabriel Quere, Daniel Leidner, Jörn Vogel, and Freek Stulp. Cats: Task planning for shared control of assistive robots with variable autonomy. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 3775–3782, 2022. doi: 10.1109/ICRA46639.2022.9811360.
- [18] Samuel Bustamante Gomez, Markus Wendelin Knauer, Thun Jeremias, Stefan Schneyer, Bernhard Weber, and Freek Stulp. Grounding embodied question-answering with state summaries from existing robot modules. In *RSS (Robotics: Science and Systems) conference 2024, Generative Modeling meets HRI Workshop*, Juli 2024. URL <https://elib.dlr.de/205203/>.
- [19] Isabel Cruz, Huiyong Xiao, and Advis Lab. The role of ontologies in data integration. *Journal of Engineering Intelligent Systems*, 13, 12 2005.
- [20] Cybathlon. Resultate — cybathlon.com. <https://cybathlon.com/de/wettkaempfe/resultate?discipline=5153&event=5267>, 2024. [Accessed 25-02-2025].
- [21] Aldo Gangemi, ver. 3.34. URL <http://www.ontologydesignpatterns.org/ont/dul/DUL.owl>.
- [22] RoboCup Germany. @Home — major.robocup.de. <https://major.robocup.de/ligen/home/>, 2016. [Accessed 28-03-2025].
- [23] Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993. ISSN 1042-8143. doi: <https://doi.org/10.1006/knac.1993.1008>.

- [24] Andrei Haidu and Michael Beetz. Automated models of human everyday activity based on game and virtual reality technology. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 2606–2612. IEEE, 2019.
- [25] Dongge Han, Trevor McInroe, Adam Jelley, Stefano V. Albrecht, Peter Bell, and Amos Storkey. LLM-personalize: Aligning LLM planners with human preferences via reinforced self-training for housekeeping robots. In Owen Rambow, Leo Wanner, Marianna Apidianaki, Hend Al-Khalifa, Barbara Di Eugenio, and Steven Schockaert, editors, *Proceedings of the 31st International Conference on Computational Linguistics*, pages 1465–1474, Abu Dhabi, UAE, January 2025. Association for Computational Linguistics.
- [26] Rajat Kumar Jenamani, Daniel Stabile, Ziang Liu, Abrar Anwar, Katherine Dimitropoulou, and Tapomayukh Bhattacharjee. Feel the bite: Robot-assisted inside-mouth bite transfer using robust mouth perception and physical interaction-aware control, 2024.
- [27] Hochul Jeon, Taehwan Kim, and Joongmin Choi. Ontology-based user intention recognition for proactive planning of intelligent robot behavior. In *2008 International Conference on Multimedia and Ubiquitous Engineering (mue 2008)*, pages 244–248, 2008. doi: 10.1109/MUE.2008.97.
- [28] Aleksandar Jevtić, Andrés Flores Valle, Guillem Alenyà, Greg Chance, Praminda Caleb-Solly, Sanja Dogramadzi, and Carme Torras. Personalized robot assistant for support in dressing. *IEEE Transactions on Cognitive and Developmental Systems*, 11(3):363–374, 2019. doi: 10.1109/TCDS.2018.2817283.
- [29] Dae-Jin Kim, Rebekah Hazlett-Knudsen, Heather Culver-Godfrey, Greta Rucks, Tara Cunningham, David Portee, John Bricout, Zhao Wang, and Aman Behal. How autonomy impacts performance and satisfaction: Results from a study with spinal cord injured subjects using an assistive robot. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 42(1):2–14, 2012. doi: 10.1109/TSMCA.2011.2159589.
- [30] Y. Kuniyoshi, M. Inaba, and H. Inoue. Learning by watching: extracting reusable task knowledge from visual observation of human performance. *IEEE Transactions on Robotics and Automation*, 10(6):799–822, 1994. doi: 10.1109/70.338535.
- [31] Daniel Leidner. *Cognitive Reasoning for Compliant Robot Manipulation*, volume 127 of *Springer Tracts in Advanced Robotics*. Springer, Januar 2019. URL <https://elib.dlr.de/129178/>.
- [32] Daniel Leidner, Alexander Dietrich, Florian Schmidt, Christoph Borst, and Alin Albu-Schäffer. Object-centered hybrid reasoning for whole-body mobile manipulation. In *2014 IEEE International Conference on Robotics and Automation, ICRA 2014, Hong Kong, China, May 31 - June 7, 2014*, pages 1828–1835. IEEE, 2014. doi: 10.1109/ICRA.2014.6907099. URL <https://doi.org/10.1109/ICRA.2014.6907099>.
- [33] Ziang Liu, Yuanchen Ju, Yu Da, Tom Silver, Pranav Thakkar, Jenna Li, Justin Guo, Katherine Dimitropoulou, and Tapomayukh Bhattacharjee. Grace: Generalizing robot-assisted caregiving with user functionality embeddings, 01 2025.

- [34] Liam Magee. *Upper-level ontologies*, pages 235–287. Chandos Publishing, Oxford, UK, 12 2011. ISBN 9781843346012. doi: 10.1016/B978-1-84334-601-2.50009-X.
- [35] Sumaira Manzoor, Yuri Goncalves Rocha, Sung-Hyeon Joo, Sang-Hyeon Bae, Eun-Jin Kim, Kyeong-Jin Joo, and Tae-Yong Kuc. Ontology-based knowledge representation in robotic systems: A survey oriented toward applications. *Applied Sciences*, 11(10), 2021. ISSN 2076-3417. doi: 10.3390/app11104324.
- [36] Elle Miller, Maximilian Durner, Matthias Humt, Gabriel Quere, Wout Boerdijk, Ashok M. Sundaram, Freek Stulp, and Jorn Vogel. Unknown object grasping for assistive robotics, 2024.
- [37] OAEI. Ontology Alignment Evaluation Initiative. <https://oaei.ontologymatching.org>, 2024. [Accessed 13-04-2025].
- [38] Daniel Oberle, Anupriya Ankolekar, Pascal Hitzler, Philipp Cimiano, Michael Sintek, Malte Kiesel, Babak Mougouie, Stephan Baumann, Shankar Vembu, Massimo Romanelli, Paul Buitelaar, Ralf Engel, Daniel Sonntag, Norbert Reithinger, Berenike Loos, Hans-Peter Zorn, Vanessa Micelli, Robert Porzel, Christian Schmidt, Moritz Weiten, Felix Burkhardt, and Jianshen Zhou. Dolce ergo sumo: On foundational and domain models in the smartweb integrated ontology (swinto). *Journal of Web Semantics*, 5(3):156–174, 2007. ISSN 1570-8268. doi: <https://doi.org/10.1016/j.websem.2007.06.002>.
- [39] Peter Ochieng and Swaib Kyanda. Large-scale ontology matching: State-of-the-art analysis. *ACM Comput. Surv.*, 51(4), July 2018. ISSN 0360-0300. doi: 10.1145/3211871.
- [40] Nayoung Oh, Junyong Park, Ji Ho Kwak, and Sungho Jo. A robot capable of proactive assistance through handovers for sequential tasks. In *2021 18th International Conference on Ubiquitous Robots (UR)*, pages 296–301, 2021. doi: 10.1109/UR52253.2021.9494681.
- [41] Alberto Olivares-Alarcos, Daniel Beßler, Alaa Khamis, Paulo Goncalves, Maki K. Habib, Julita Bermejo-Alonso, Marcos Barreto, Mohammed Diab, Jan Rosell, João Quintas, and et al. A review and comparison of ontology-based approaches to robot autonomy. *The Knowledge Engineering Review*, 34:e29, 2019. doi: 10.1017/S0269888919000237.
- [42] Inès Osman, Sadok Ben Yahia, and Gayo Diallo. Ontology integration: Approaches and challenging issues. *Information Fusion*, 71:38–63, 2021. ISSN 1566-2535. doi: <https://doi.org/10.1016/j.inffus.2021.01.007>.
- [43] Maithili Patel, Aswin Gururaj Prakash, and Sonia Chernova. Predicting routine object usage for proactive robot assistance. In Jie Tan, Marc Toussaint, and Kourosh Darvish, editors, *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pages 1068–1083. PMLR, 06–09 Nov 2023. URL <https://proceedings.mlr.press/v229/patel23a.html>.
- [44] Adam Pease, Ian Niles, and John Li. The suggested upper merged ontology: A large ontology for the semantic web and its applications. In *Working notes of the AAAI-2002 workshop on ontologies and the semantic web*, volume 28, pages 7–10, 2002.

- [45] Jan Portisch, Michael Hladik, and Heiko Paulheim. Background knowledge in ontology matching: A survey. *Semantic Web*, pages 1–55, 09 2022. doi: 10.3233/SW-223085.
- [46] Jan Portisch, Michael Hladik, and Heiko Paulheim. Background knowledge in ontology matching: A survey. *Semantic Web*, 15(6):2639–2693, 2024. doi: 10.3233/SW-223085.
- [47] Gabriel Quere, Annette Hagengruber, Maged Iskandar, Samuel Bustamante, Daniel Leidner, Freek Stulp, and Jörn Vogel. Shared control templates for assistive robotics. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1956–1962, 2020. doi: 10.1109/ICRA40945.2020.9197041.
- [48] Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. Progprompt: Generating situated robot task plans using large language models. In *Workshop on Language and Robotics at CoRL 2022*, 2022.
- [49] Gaurav Srivastava, Shuxiang Cao, and Xuan Wang. Towards reasoning ability of small language models, 2025.
- [50] Moritz Tenorth, Jan Winkler, Daniel Beßler, and Michael Beetz. Open-ease: A cloud-based knowledge service for autonomous learning. *KI - Künstliche Intelligenz*, 29(4): 407–411, 2015. doi: 10.1007/s13218-015-0364-1.
- [51] Elodie Thiéblin, Ollivier Haemmerlé, Nathalie Hernandez, and Cassia Trojahn. Survey on complex ontology matching. *Semantic Web*, 11(4):689–727, 2020. doi: 10.3233/SW-190366.
- [52] Sai Vemprala, Rogerio Bonatti, Arthur Buckner, and Ashish Kapoor. Chatgpt for robotics: Design principles and model abilities. *Microsoft Auton. Syst. Robot. Res*, 2: 20, 2023.
- [53] Jörn Vogel, Annette Hagengruber, Maged Iskandar, Gabriel Quere, Ulrike Leipscher, Samuel Bustamante, Alexander Dietrich, Hannes Höppner, Daniel Leidner, and Alin Albu-Schäffer. Edan: An emg-controlled daily assistant to help people with physical disabilities. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4183–4190, 2020. doi: 10.1109/IROS45743.2020.9341156.
- [54] Huaxiaoyue Wang, Nathaniel Chin, Gonzalo Gonzalez-Pumariega, Xiangwan Sun, Neha Sunkara, Maximus Pace, Jeannette Bohg, and Sanjiban Choudhury. Apricot: Active preference learning and constraint-aware task planning with llms, 10 2024.
- [55] Jan Winkler, Ferenc Balint-Benczedi, Thimo Wiedemeyer, Michael Beetz, Narunas Vaskevicius, Christian A. Mueller, Tobias Fromm, and Andreas Birk. Knowledge-enabled robotic agents for shelf replenishment in cluttered retail environments, 2016.
- [56] Bryce Woodworth, Francesco Ferrari, Teofilo E. Zosa, and Laurel D. Riek. Preference learning in assistive robotics: Observational repeated inverse reinforcement learning. In Finale Doshi-Velez, Jim Fackler, Ken Jung, David Kale, Rajesh Ranganath, Byron Wallace, and Jenna Wiens, editors, *Proceedings of the 3rd Machine Learning for Healthcare Conference*, volume 85 of *Proceedings of Machine Learning Research*, pages 420–439. PMLR, 17–18 Aug 2018.

- [57] Jimmy Wu, Rika Antonova, Adam Kan, Marion Lepert, Andy Zeng, Shuran Song, Jeannette Bohg, Szymon Rusinkiewicz, and Thomas Funkhouser. Tidybot: personalized robot assistance with large language models. *Autonomous Robots*, 47(8):1087–1102, 2023. doi: 10.1007/s10514-023-10139-z.

LIST OF ILLUSTRATIONS

<i>Number</i>	<i>Page</i>
2.1 Illustration of EDAN's software components ©2020 IEEE. Reprinted, with permission, from Jörn Vogel et. al., EDAN: An EMG-controlled Daily Assistant to Help People With Physical Disabilities, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Oct. 2020 [53]	6
2.2 Example of a <code>manifest.xml</code> file for the object <code>thermos</code> , describing and linking all information about an object in the odb.	7
2.3 High level overview of some of the concepts in SOMA, building on the four basic concepts <code>Event</code> , <code>Quality</code> , <code>Object</code> , and <code>Abstract</code>	12
2.4 Excerpt of the way, an action executing the task of picking up a mug can be modeled in SOMA, highlighting the concepts of <code>Physical Object</code> vs. <code>Social Object</code>	13
3.1 Excerpt of the object hierarchy in EDAN's odb	17
3.2 Excerpt of the alignment of SOMA objects with odb objects with the newly attached odb concepts in green	19
4.1 Overview of the PREFERENCEORACLE.	28
4.2 Excerpt of the SOMA ontology. Counting the steps from subclasses of <code>Pouring</code> leads to the same result as counting steps from <code>Pouring</code> to <code>Pushing</code> , even though the latter are far more different.	32
4.3 Comparing the task close drawer with the task pick mug in the ontology. . .	35
4.4 Ontological distance values, comparing the task option to pick a mug in the current situation with a history of executed tasks, based only on task types and object types.	36
4.5 Different values for decreasing importance factor and the resulting weights of the n -th predecessor situation.	39
4.6 Example of a preference tree	41
4.7 Example of finding and removing suggestions from the preference tree that involve objects that are not available in the current world state.	43
5.1 Number of actions per sample NEEM in the test dataset	53
5.2 Comparison of prediction accuracy for the recommended settings, using different metrics	54
5.3 Average prediction accuracy after a number of actions in the test dataset. . .	55
5.4 Average prediction accuracy after a number of actions in the test dataset, compared to the number of samples providing a different number of actions. .	55
5.5 Comparison of Top 5 accuracies between different settings.	57

5.6	Comparison of prediction accuracies for Top 1, Top 3, and Top 5 metrics, comparing recommended settings and knockout = 0.	58
5.7	Comparison of average computing time per task using different settings . . .	59
B.1	Comparison of top 3 accuracies between different settings.	85
B.2	Comparison of top 1 accuracies between different settings.	86

LIST OF TABLES

<i>Number</i>	<i>Page</i>
2.1 By [41, p. 24]: List of relevant terms for the autonomous robotics domain, and their coverage in the different chosen works.	10
4.1 Collecting successor tasks poses the question of how to weigh number of occurrences vs. the overall distance between each occurrence and the current task.	41
4.2 Recommended hyperparameter settings for a good performance in top 5 metric	44
5.1 List of objects and their afforded tasks in the kitchen setup used for evaluating the preference calculations	51
5.2 Excerpt of the prompt to generate a textual user workflow as well as an example of the answer	52

LISTINGS

3.1	List of manually defined subclass relations between odb objects and SOMA objects	18
3.2	Excerpt of the definition for a mapping from tasks in the odb to SOMA tasks, roles and dispositions.	21
5.1	Excerpt of the list of personas, used to prompt an LLM for some auto-generated personas	48
A.1	The configuration file, describing all manually defined matchings from concepts in EDAN's odb to concepts in the SOMA ontology.	80
B.1	Example personas, used to prompt an LLM for some auto-generated personas	83

Appendix A

THE NEEM LOGGER

```

1 # namespace of this ontology
2 namespace: https://ontology.dlr.de/soma/odb
3
4 # these also define namespaces that can be used in this file
5 imports:
6   soma: http://www.ease-crc.org/ont/SOMA.owl#
7   dul: http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#
8   knowrob: http://knowrob.org/kb/knowrob.owl#
9
10 # map odb objects to soma object classes
11 soma_odb_map:
12   _robot: soma:ArtificialAgent
13   _tool: soma:DesignedTool
14   human: dul:NaturalPerson
15   _gripper: soma:Gripper
16   _container: soma:DesignedContainer
17   _coffee_machine: soma:Appliance
18   _content: soma:DesignedSubstance
19   _fruit: soma:BiologicalObject
20   _microwave: soma:Appliance
21   _phone: soma:DesignedTool
22   _tray: soma:Surface
23   _cube: soma:DesignedComponent
24   edans_ikea_kitchen_door: soma:DesignedComponent
25   oven_knob: soma:DesignedTool
26
27 # default soma object for objects without mapping
28 root_object: dul:PhysicalObject
29
30 # default action classifications for scts without mapping
31 action_defaults:
32   task: soma:PhysicalTask
33   ref_role: False
34   target_role: soma:Patient
35   disposition: soma:Disposition
36
37 # map sct-names to soma classification
38 actions:
39   drink:
40     task: soma:Serving
41     ref_role: soma:MovedObject
42     target_role: soma:RecipientRole
43     disposition: soma:Storage
44

```



```

45  pour:
46      task: soma:Pouring
47      ref_role: soma:PouredObject
48      target_role: soma:RecipientRole
49      disposition: soma:Storage
50
51  release:
52      task: soma:Releasing
53      target_role: soma:MovedObject
54      ref_role: False
55      disposition: soma:Graspability
56  release_into_mug:
57      parent: release
58  place:
59      parent: release
60
61  pick:
62      task: soma:PickingUp
63      target_role: soma:MovedObject
64      ref_role: False
65      disposition: soma:Graspability
66  pick_from_side:
67      parent: pick
68  pick_from_top:
69      parent: pick
70
71  transfer:
72      task: soma:Transporting
73      target_role: soma:MovedObject
74      ref_role: False
75      disposition: soma:Graspability
76  transfer_tube:
77      parent: transfer
78  transfer_washer:
79      parent: transfer
80
81  open:
82      task: soma:Opening
83      target_role: soma:ShapedObject
84      ref_role: False
85      disposition: soma:Enclosing
86  open_door:
87      parent: open
88  open_fridge:
89      parent: open_door
90
91  open_drawer:
92      parent: open
93      disposition: soma:Storage
94  open_drawer_down:

```

```

95     parent: open_drawer
96 open_cabinet:
97     parent: open_drawer
98
99 close:
100     task: soma:Closing
101     target_role: soma:ShapedObject
102     ref_role: False
103     disposition: soma:Enclosing
104 close_door:
105     parent: close
106
107 push:
108     task: soma:Pushing
109     target_role: soma:ShapedObject
110     ref_role: False
111
112 rotate:
113     task: soma:Orienting
114     target_role: soma:ShapedObject
115     ref_role: False
116 rotate_timer:
117     parent: rotate
118 turn_knob:
119     parent: rotate
120
121 stir_broth:
122     task: soma:Stirring
123
124 roll_dice: {}

```

Listing A.1: The configuration file, describing all manually defined matchings from concepts in EDAN's odb to concepts in the SOMA ontology.

Appendix B

PREFERENCES

The full list of example personas, used for prompting LLMs to generate personas:

```

1 — file_name: grandma_teeth
2 persona: You are a 98-year old grandma, having problems with your teeth.
3 daily_routine:
4   - description: You want to prepare breakfast.
5     time: 06
6   - description: You want to eat a snack for bedtime.
7     time: 18
8 variations:
9   - description: You are coming home from your dental appointment.
10    time: 14
11 — file_name: programmer_red_coffee
12 persona: You are a programmer, living on coffee, always preferring your
13    red coffee mug.
14 daily_routine:
15   - description: You want to prepare breakfast.
16     time: 09
17   - description: You are having a little break from work.
18     time: 13
19   - description: You are preparing a snack for bedtime.
20     time: 20
21 variations:
22   - description: You are getting home late, looking for something to eat
23     .
24     time: 23
25 — file_name: 90_year_drinking_man
26 persona: You are a 90-year old man with a heart condition and a tendency
27    to drink too much.
28 daily_routine:
29   - description: You want to prepare breakfast
30     time: 07
31   - description: You want to snack.
32     time: 14
33   - description: You want to eat dinner.
34     time: 20
35 variations:
36   - description: You are getting home from the doctor
37     time: 15
38   - description: You want to drink something
39     time: 18
40 — file_name: vegan_twenties
41 persona: You are a healthy-living vegan in your twenties, trying to stay
42    away from coffee as best you can.
43 daily_routine:

```

```

40 - description: You want to prepare a breakfast with tea
41   time: 10
42 - description: You get home from college and want to relax.
43   time: 15
44 - description: You want to eat dinner.
45   time: 20
46 variations:
47   - description: You find yourself drinking coffee.
48     time: 13
49 — file_name:
50 persona: You are an injured person, your favourite cup is the blue one,
51   your goal is to get better.
52 daily_routine:
53   - description: You want to prepare something to eat for breakfast.
54     time: 09
55   - description: You want to drink something
56     time: 13
57   - description: you want to drink something
58     time: 17
59   - description: You want to prepare dinner.
60 variations:
61   - description: You are getting home from a doctors appointment.
62     time: 16

```

Listing B.1: Example personas, used to prompt an LLM for some auto-generated personas

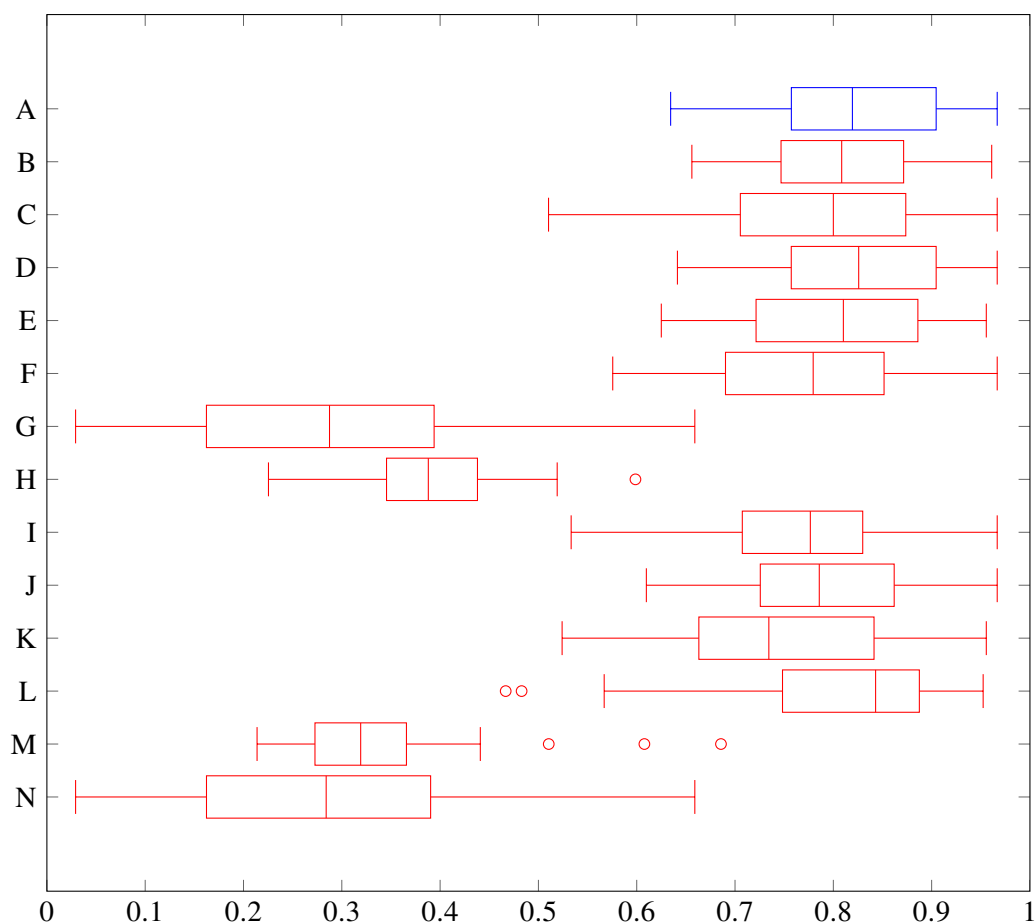


Figure B.1: Comparison of top 3 accuracies between different settings.

Legendary:

A: Recommended settings (see tab. 4.2 on page 44)

B: `compare situations` = 0

C: `time of day penalty` = 10

D: `time of day penalty` = 1

E: `multiple occurrences factor` = 1

F: `multiple occurrences factor` = 0

G: `decreasing importance factor` = 1

H: `object factor` = 0

I: `task factor` = 0

J: `knockout penalty` = `knockout`

K: `knockout penalty` = 20

L: `knockout` = 0

M: `knockout` = 20

N: `event horizon backwards` = 0

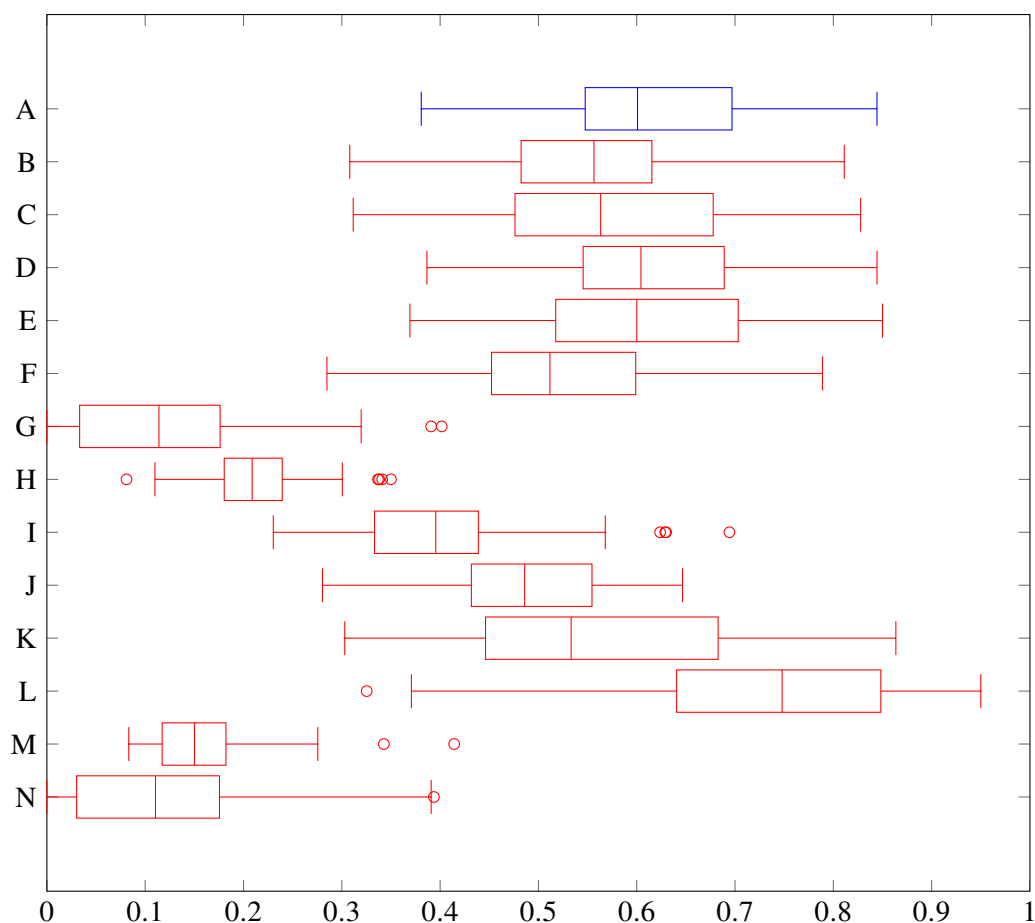


Figure B.2: Comparison of top 1 accuracies between different settings.

Legendary:

A: Recommended settings (see tab. 4.2 on page 44)

B: `compare situations` = 0

C: `time of day penalty` = 10

D: `time of day penalty` = 1

E: `multiple occurrences factor` = 1

F: `multiple occurrences factor` = 0

G: `decreasing importance factor` = 1

H: `object factor` = 0

I: `task factor` = 0

J: `knockout penalty` = `knockout`

K: `knockout penalty` = 20

L: `knockout` = 0

M: `knockout` = 20

N: `event horizon backwards` = 0