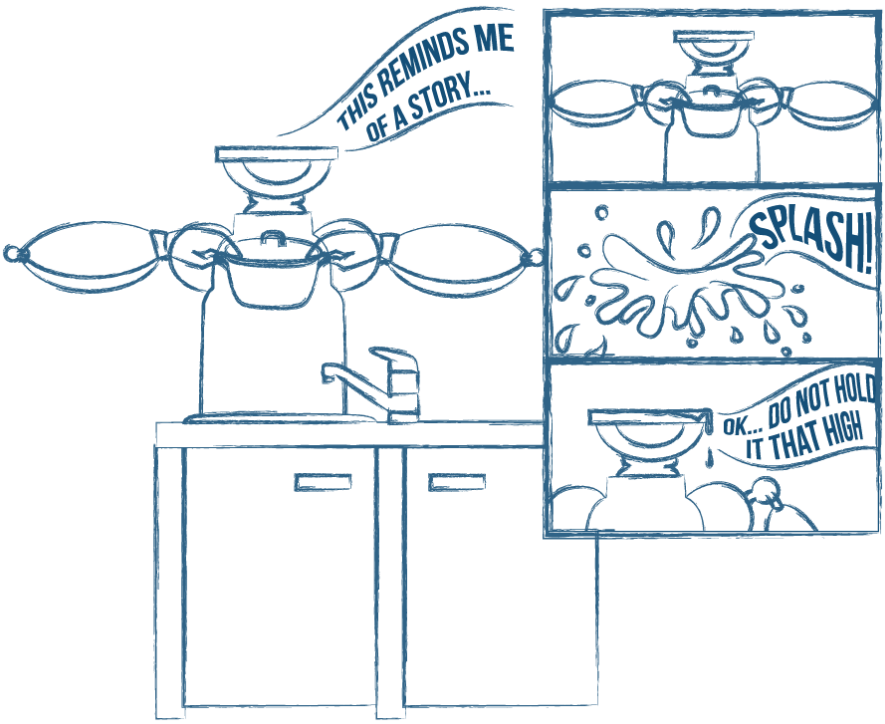
NEEM Handbook

Michael Beetz, Daniel Beßler, Sebastian Koralewski, Mihai Pomarlan, Abhijit Vyas, Alina Hawkin, Kaviya Dhanabalachandran, Sascha Jongebloed

CRC Everyday Activity Science and Engineering (EASE)
University Bremen, Am Fallturm 1, 28359 Bremen
ai-office@cs.uni-bremen.de

Summary. The Collaborative Research Center EASE is an interdisciplinary research initiative at the University of Bremen that attempts to advance our understanding of how human-scale manipulation tasks can be mastered by robotic agents. The challenge is that the same task needs to be executed by the robot in different ways depending on, for example, what tools are available, and how the environment is shaped. The key to solve this issue is *generalization*. However, the robot needs to know more then what step it needs to execute next – it further needs to decide on how the next step is carried out through motions of its body, and interactions with its environment. In this document, we will describe how these types of information are represented in the EASE system, how such data-sets are acquired, and how they are stored, maintained, and curated using a centralized web-service. The goal of this effort is to establish representations and infrastructure for a shared experience storage with annotated data-sets of agents performing everyday activities, and to use these data-sets as ground truth data to find generalizations that do not abstract away from movements, and naive physics.



Contents

1 Introduction **1**
- 1.1 Notation . . . 3
- 1.2 Scope . . . 4
- 1.3 Overview . . . 5

2 NEEM-Background **7**
- 2.1 Types of Objects . . . 8
- 2.2 Properties of Objects . . . 9
- 2.3 Views on Objects . . . 10
  - 2.3.1 Appearance . . . 10
  - 2.3.2 Structure . . . 12
  - 2.3.3 Kinematics . . . 13
  - 2.3.4 Dynamics . . . 16
  - 2.3.5 Naive physics . . . 17
- 2.4 Data Formats . . . 18
  - 2.4.1 URDF . . . 18
  - 2.4.2 DAE . . . 19

3 NEEM-Narrative **21**
- 3.1 Types of Events . . . 21
- 3.2 Roles of Objects . . . 24
- 3.3 Views on Events . . . 24
  - 3.3.1 Occurrence . . . 24
  - 3.3.2 Participation . . . 25
  - 3.3.3 Composition . . . 26
  - 3.3.4 Transformation . . . 26
  - 3.3.5 Conceptualization . . . 28
  - 3.3.6 Contextualization . . . 29

4 NEEM-Experience **31**
- 4.1 Kinematics . . . 32
  - 4.1.1 Pose data . . . 32

5 NEEM-Hub **35**

5.1 Prerequisite . . . 36
5.2 Downloading . . . 37
5.3 Publishing . . . 38

6 NEEM-Acquisition 41
6.1 Data Structure . . . 41
6.1.1 Triple data as JSON object . . . 42
6.2 Robot NEEMs . . . 43
6.2.1 Prerequisite . . . 43
6.2.2 Recording Narrative Enabled Episodic Memories . . . 44
6.2.3 Add Semantic Support to your Designed Plans . . . 44
6.2.4 Summary . . . 46
6.3 VR NEEMs . . . 47
6.3.1 Prerequisite . . . 47
6.3.2 Recording Virtual Reality Narrative Enabled Episodic Memories 48
6.3.3 Transferring VR-NEEMs into the Knowledge Base . . . 48
6.3.4 Using VR-NEEM Data in CRAM plans . . . 49

7 NEEM Quick-start Guide 51
7.1 NEEM Checklist . . . 51
7.1.1 Kinematic information with visualization meshes . . . 51
7.1.2 NEEM Data format . . . 52
7.1.3 Semantic Annotation . . . 52
7.1.4 Semantic Annotation: KnowRob . . . 53

8 Appendix 55
8.1 Agent owl file . . . 55
8.2 Environment owl file . . . 62
References . . . 65

1

Introduction

D. BESSLER, S. KORALEWSKI, M. POMARLAN

This document, referred to as the "NEEM Handbook" hereafter, describes the EASE system for episodic memories of everyday activities. It is thought to provide EASE researchers with compact but still comprehensive information about what information is contained in NEEMs , how it is represented, acquired, curated and published.

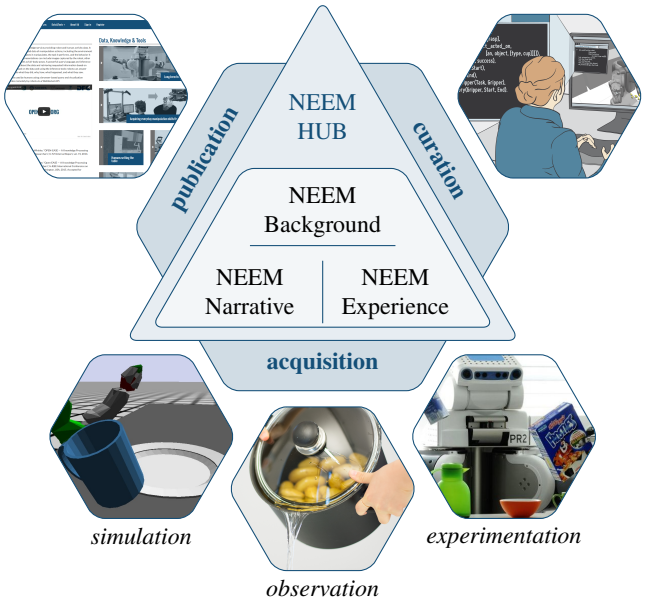


Fig. 1. The EASE system for acquisition, curation and publication of episodic memories

Narrative Enabled Episodic Memories

When somebody talks about the deciding goal in the last soccer world championship many of us can "replay" the episode in our "mind's eye". Those episodic memories can be seen as abstract descriptions that allow us to recall detailed pieces of information from any experienced activity. Having those detailed memories, we can use them to learn general knowledge or map similar memories to unknown situations, so we know how to behave in the given situation.

EASE integrates episodic memories deeply into the knowledge acquisition, representation, and processing system. For every activity the agent performs, observes, prospects and reads about, it creates an episode and stores it in its memory. An episode is best understood as a video recording that the agent makes of the ongoing activity. In addition, those videos are enriched with a very detailed story about the actions, motions, their purposes, effects and the agent's sensor information during the activity.

We define the episodic memories created by our system narrative-enabled episodic memories (NEEMs). A NEEM consists of the *NEEM experience* and the *NEEM narrative*. The NEEM experience captures low-level data such as the agent's sensor information, e.g. images and forces, and records of poses of the agent and its detected objects. NEEM experiences are linked to NEEM narratives, which are stories of the episode described symbolically. These narratives contain information regarding the tasks, the context, intended goals, observed effects, etc. The NEEM-experience and NEEM-narrative combined are so rich of information that the agent can replay an episode to experience the seen activity anytime again.

NEEMs are representations of experiences acquired through experimentation, reading, observing, mental simulation, etc. The main goal is to establish a common vocabulary used to annotate experience data across different tasks, scientific disciplines, and modalities of acquisition, and to define models for the representation of experience data. The vocabulary is not just a set of atomic labels, but each label has a formal definition in an ontology. These definitions are done such that a set of *competency questions* about an activitiy can be answered by a knowledge base that is equipped with the ontology and a collection of NEEMs .

The NEEM model is formally defined in form of an OWL ontology which is based on the DOLCE+DnS Ultralite (DUL) upper-level ontology [1]. DUL is a carefully designed ontology that seeks to model general categories underlying human cognition without making any discipline-specific assumptions. Our extensions of DUL mainly focus on characterizing different aspects of activities that were not considered in much detail in DUL, but are relevant for the autonomous robotics scope. These extensions are part of an ontology that we have called SOMA [1]. A NEEM is made of several patterns defined either in DUL or in SOMA .

[1] `https://ease-crc.github.io/soma`

While it is possible to create the representations listed in this document through a custom exporter, it is not advised to do so. Instead, it is advised to interface with the KnowRob knowledge base [2]. KnowRob provides an interface based on predicate logics that allows to interact with NEEMs . The language is a collection of predicates that can be called by users to ask certain types of competency questions covering different aspects of activitiy, or to add labels and relationships in the NEEM-narrative . We will provide example expressions in this document that highlight how the knowledge base can be used to interact with NEEMs .

1.1 Notation

In this Section, we will shortly introduce the notions and notations that are important to follow this document.

NEEMs are formally represented using an *ontology*. An ontology is a collection of logical axioms in some formal language such as description logic (DL). The entities that can be described in DL can be either *concepts* (sometimes known as *classes*), and *instances*. An individual may belong to one or more concepts. A concept may be subsumed by another concept. Between individuals there may be relations called *object properties*, and, in addition, an individual can also have *data properties* that link it to some data values. As an example, let us assume that Alice and Bob are both individuals belonging to the concept *Human*, and that the object property *hasChild* connects Alice to Bob, i.e. the relation asserts that Bob is a child of Alice. We may also know the height of Alice, which would be represented by a data property *hasHeight* whose value could be a string such as *1,7m* to represent that she is 1.7m tall. In the following, to make clear when we are talking about concepts and when about individuals, we will denote the set of all concepts as $\mathscr{T}$ (called the TBox), and the set of all individuals as $\mathscr{A}$ (called the ABox).

It is useful when describing concepts to emphasize the concept names such that it is clear we reference the concept, and not the colloquial word. As such, *Concepts* and *relations* will be written in a different font. Note that the name of a concept always starts with an uppercase letter, whereas the name of a relation with a lowercase one. Any word appearing in a concept or relation name after the first one will always begin with an uppercase letter.

Ontologies are meant to build on one another, and it is not uncommon for an ontology to collect thousands of concepts from external ontologies it imports. To prevent name clashes, in actual usage the names of concepts, relations, and individuals are often name-spaced. In this document, since we mostly talk about concepts from the SOMA ontologies, the namespace will not be made explicit. An exception will be made in some diagrams where we reference concepts defined in more basic ontologies, such

[2] `https://github.com/knowrob/knowrob`

as those used to define the Ontology Web Language (OWL). An example is a name such as *xsd:double*; in this case, *xsd* is the namespace.

1.2 Scope

The broad scope of this work is to provide information about how robotic manipulation activities are represented, acquired, curated and published in the EASE system for episodic memories. We are in particular interested in aspects of interaction forces and motion characteristics of objects participating in an action, since it is these physical and geometric considerations that are crucial for successfull action execution. The goal is to learn models from collections of recorded data semantically annotated through concepts defined in the NEEM model. The rich semantic annotations enable querying and filtering the data, such that a robot can formalize a learning problem for itself and curate its training data to be appropriate for it. Information about how the data is collected, with what methods, from what agents, in which contexts, is important for this process, as machine learning techniques are sensitive to training data biases. Note that in principle episodes can be stored of any agent performing any activity, and in actuality many of the NEEMs we expect to store will come from humans demonstrating how to perform a task. NEEMs are therefore not simply intended as a kind of self-practice journal, but rather as a store of practical knowledge of a variety of agents, useful for a variety of autonomous, humanoid robots.

The kinds of knowledge a robot needs for competent performance of its tasks are varied. Usually, knowledge modelling in robotics and AI has focused on a symbolic level, of actions treated as black boxes that relate to a larger plan by means of their preconditions and effects. Actions are also very underspecified when described in spoken commands. This abstract level of description however is insufficient; the physical details of the actions matter. For example, the angle and speed with which a pitcher is moved, and the amount of liquid in it, determines whether there will be spillage. A robot needs to choose appropriate parameters for its actions, and infer these parameters when they are left unspecified in a command.

Such inference requires the robotic agent to be equipped with common-sense and intuitive physics knowledge, as well as an abstract task and object model, and knowledge of how to apply these models in a given situation. The NEEM model attempts to support each of these requirements. A brief list of some of the over-arching competency questions follows.

- *How are actions conceptualized?* What is an action, how does it relate to other concepts an agent might have about the world? What is the purpose of an action?
- *What is the structure of an action?* How do several actions make up another? What objects participate in an action and with what roles?

- *How are qualitative and quantitative features of the world represented?* What is the parameter set of an action? What regions can values for these parameters occupy? What is a good parameterization and how can one be found?
- *How are the physical interactions that underlie an action described?* What are the involved forces, and how are they parameterized? What are relevant qualitative, and thus more general, descriptors for interactions, such as balance, blockage, compulsion? How are qualitative aspects of interaction grounded in quantitative physical phenomena?
- *How are objects conceptualized?* What roles can an object play? What actions can it take part in? What kinds of objects are necessary for an action?
- *How is an action recorded and described?* What is the relevant data to capture how an action unfolded? What are the relevant pieces of contextual information for describing an action that has actually occurred? What was the outcome of the action, in particular, to what extent did it match the goal?
- *How is a learning problem formalized?* What is the optimization goal? What assumptions were in effect when collecting the training data? What sort of influence might biases have upon the learned model? What should be essential features that a learned model should use? What would be sanity checks on the learned model to verify it does not abuse spurious correlations?

1.3 Overview

NEEMs are the central data structures that link research results of various subareas within the collaborative research center EASE . EASE is an interdisciplinary institution headed by leading researchers in the fields robotics, human cognition, formal logics, and linguistics. The overall goal is to make a robot more competent in performing everyday activities. This is accomplished by equipping the robot with models learned over experiences represented as NEEMs . The purpose of this document is to provide detailed information about the EASE system for episodic memories. That is how NEEMs are represented as knowledge bases linked to time-series data, acquired through experimentation, observation or simulation, stored on a centralized server, and maintained as a dataset for the research community. The architecture is shown in Figure 1, and will be summarized in the remainder of this section.

At the core of the EASE system for episodic memories is the NEEM data structure. It is a heterogenous datastructure that contains data in different formats to represent different categories of information about everyday activities. Each NEEM is made of three parts: background (Chapter 2), narrative (Chapter 3) and experience (Chapter 4). The background represents physical activity context by characterizing the environment, and agents that play a role during the activity. A single background

may be shared in mutiple NEEMs . The narrative is a representation of events that happened, their characterization and contextualization. That is, for example, that an event occurred, what roles objects played during the event, how the event was carried out through motions and interactions, and what the reason of its occurrence is. The narrative provides labels used to annotate the time-series data stored in the experience of the NEEM . This is done by associating the event time intervals to slices in the time-series database. The experience data is used to capture some aspects of kinematics and dynamics of an activity, that is how objects moved, how they got into contact with each other, and how forces act upon objects.

NEEMs are stored on a publicly accessible infrastructure that we have called the NEEM-hub (Chapter 5). The NEEM-hub builds on top of common infrastructure used in data science to continuously update models learned from NEEMs . Uploading a NEEM requires to create a new data set on the NEEM-hub GitLab interface where users can provide documentation, usage examples, additional links and references for their NEEM data set. Once a user is satisfied with the state of the data set, it may be published. This will make the data set accessible via the knowledge service openEASE where users may search for data sets given some keywords, download the data set, or investigate it in an interactive environment.

As EASE is an interdisciplinary effort, there are also different modalities under which NEEMs can be acquired. We haved developed multiple acquisition infrastructures that support researchers from different domains to acquire NEEMs (Chapter 6). This is, first of all, an interface that integrates with a robot control system either in a simulated or real-world scenario where the robot senses its surroundings, and executes specific plans through motions of its body and interactions with its environment. A second acquisition interface integrates with simulated virtual reality environments in which humans perform everyday activities. In this case, the intentions are not certainly known because even when told to do something specific, a human may do some unrelated experimentation in the virtual reality. The state of the environment including force characteristics can, however, fully be monitored.

2

NEEM-Background

D. BESSLER, M. POMARLAN, A. VYAS

The NEEM-background represents the (physical) context of NEEMs . More concretely, the NEEM-background represents the environment where events took place, and the agents that are involved. These are representations of physical objects, their parts, properties, and relationships between them.

Each NEEM must have exactly one associated NEEM-background . This is important as only the objects and their properties represented in the NEEM-background may be involved in events that occur in NEEMs . Consider, for example, a robot fetching a cup in a kitchen environment to prepare a coffee. The cup would be part of the NEEM-background while the fetching event carried out by the robot would be represented in the NEEM-narrative (Chapter 3).

The way how a task can be solved best depends on what is available in the environment. The suitability of an object to be used to perform a certain task is often derived from the class of objects it belongs to, e.g., a knife can be used for cutting. The NEEM model defines a set of more general object classes such as *agent* and *artifact* (Section 2.1). These are used to classify each object represented in the NEEM-background . The usability of an object is, however, ultimately grounded in its properties, e.g., a dulled knife may prove to be unusable to perform a cutting task. It is thus also relevant to characterize object properties as they correlate with how an agent may solve its task. Consequently, we treat types of object properties as classes organized in a taxonomy (Section 2.2).

NEEMs may characterize different aspects of the environment depending on what information is accessible when the NEEM is acquired. We organize different characteristics in so called *views* (Section 2.3). Each view has its own set of types and relationships to represent the environment from a specific viewpoint such as appearance or kinematics.

NEEMs are heterogeneous representations that may include additional data files. These representations are time series data that are annotated by the NEEM-narrative . In addition, some widely used data formats for the representation of objects are supported (Section 2.4). Such data files may be stored within the NEEM-background , associated to objects it represents, and used to enrich knowledge about the environment.

2.1 Types of Objects

Objects and agents that appear in an environment are classified as *physical objects*. Physical objects are exactly the objects you can point to, as they have a location in space.

The most common physical objects in non-natural environments are *artifacts*. An artifact is an item that has certain structure, often to serve a particular purpose such as to use it in a certain way, or to enjoy looking at it in case of, e.g., an art piece. Artifacts that were created with a purpose in mind are called *designed artifacts*. Most objects in human-made environments belong to this category. Note that, e.g., a *container* is not a designed artifact, as also objects that were not designed as such may serve as containers. Consequently, the class *designed container* is used for the objects that were designed to be used as a container. Other examples of designed artifacts are *tools* and *appliances* designed for specific tasks or agents and the *components* designed to fit together to form a larger whole.

Another category of objects are *physical bodies*. Most commonly one would use this category for *substances* that appear in the environment such as a blob of dough, or the coffee inside of a cup. However, it is more appropriate to classify the substance as *designed substance* in case it was created with a purpose in mind, for e.g., in the case for the dough that is made according to a recipe and supposed to be eaten after being baked.

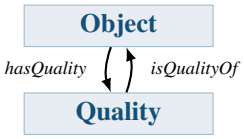
Agents that appear in a NEEM are classified as *physical agents*. The difference from other types of objects is that the agents have intentions, execute actions, and attempt to achieve goals, e.g., by following a plan and moving their body in a way to generate interactions with the environment to cause intended effects. Each agent is composed of *functional parts* organized in a skeletal structure. Interactions with the environment are carried out through *effectors* such as arms, legs, or hands. Effectors that are used for grasping are called *prehensile effectors*.

The last top-level category in our object taxonomy is *physical place*. Places are objects with a specific location such as the surface of a table, or the campus of the University. Each NEEM refers at least to the place where it was acquired, which is usually a room in a building with objects that can be used to perform certain everyday tasks.

2.2 Properties of Objects

Qualities are the properties of an object that are not part of it, but cannot exist without it. This is, for example, the quality of having a shape – a quality inherited by all physical objects. Another example is the quality of a floor having a certain surface friction and thus being slippery or not. A robot navigating on such a floor could use this knowledge to avoid, for example, spillage when moving on the floor with a coffee-filled cup. The quality concept does not directly encode the value of the object property, but only focuses on characteristics of the property itself. This is mainly useful in cases where individual aspects of an entity are considered in the domain of discourse.

Intent	To represent the qualities of an object.
Competency Questions	*What qualities does this object have? What objects have this quality?*
Defined in	DUL.owl



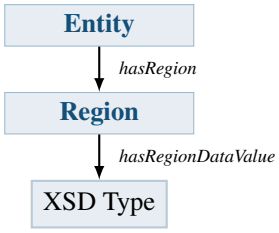
Expression	Meaning
has_quality(o,q)	$q \in \mathscr{A}$ is a quality of $o \in \mathscr{A}$

Several sub-classes of *Quality* and corresponding sub-relations of *has-quality* are defined in the NEEM model. Some of them will be described later in this chapter.

Each object property has one value at a time. The value is an element in a dimensional space. Such a dimensional space is called *Region* in the NEEM model. A region may have an infinite number of elements, or, in the other case, may enumerate all its elements. The color of an object, for example, may have a value encoded as RGB vector which is an element of the RGB colorspace (which is a region). Regions may further be decomposed into sub-regions, for example, to represent the sub-region of RGB colorspace with dominant red color. Note that the domain of the relation *has-region* is not *Quality* but *Entity*. This is to allow assigning regions to entities without requiring an explicit *Quality* individual as an intermediate. Instead, the property connecting the *Entity* specifies what information the *Region* conveys about the object.

As an example, a *PhysicalObject* would be linked via a *hasMassAttribute* to a *MassAttribute*, that is, to a *Region* individual containing information about the object's mass. It is the relation *hasMassAttribute* that specifies what information the *Region* contains.

Intent	To represent values of attributes of things.
Competency Questions	*What is the value for the attribute of that entity? Which entities have a certain value on that parameter/attribute/feature?*
Defined in	DUL.owl



Expression	Meaning
has_region(x,y)	*y* is a region of *x*
has_data_value(x,y)	*y* is a data value of *x*

2.3 Views on Objects

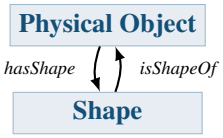
The NEEM-background may represent several different *views* on the same object highlighting different characteristics that are fused in the NEEM-background to form a more complete representation of the environment. Each view has its own vocabulary to describe objects including view-specific types of objects, qualities, and relations, and has a distinct set of competency questions that may be answered in case a NEEM represents the view. A NEEM may not represent each supported view, however, it is recommended to represent as many as possible.

2.3.1 Appearance

SOMA defines several concepts to represent qualities relating to an object's appearance. The list includes, but is not limited to, *Color*, *Shape*, and *Size* . A quality belongs to an object, and can take values only from regions of an appropriate type.

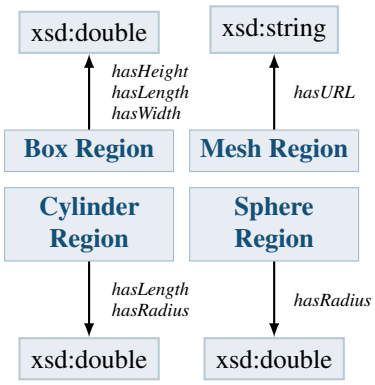
The shape of an object can either be represented as primitive geometry (e.g., box or cylinder), or as mesh. Primitive shapes are described by their geometric parameters, such as height, width and length for a box, and radius and length for a cylinder. A mesh shape, on the other hand, has a data property that is a URI of the file that contains the mesh data.

Intent	To represent the quality of having a shape.
Competency Questions	*Does this objects have a shape?*
Defined in	SOMA.owl



Expression	Meaning
has_shape(o)	$o \in \mathscr{A}$ has a shape
has_shape(o,s)	$s \in \mathscr{A}$ is the shape of $o \in \mathscr{A}$

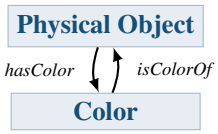
Intent	To represent the region of shapes.
Competency Questions	*What geometric parameters has this shape? What is the URL where a mesh file of this shape can be retrieved?*
Defined in	SOMA.owl



Expression	Meaning
has_bbox(o,d,w,h)	$d, w, h \in \mathbb{R}$ are the depth, width and height of the bounding box of $o \in \mathscr{A}$
has_shape_data(o,sphere(r))	$r \in \mathbb{R}$ is the radius of the sphere shape of $o \in \mathscr{A}$

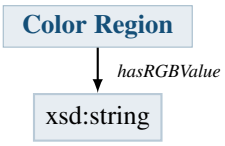
The color of an object is a quality that may take values from a *ColorRegion*. Color regions may be qualitative, such as *GreenColor* or *RedColor*, which correspond to sets of color values; color regions may also be specified as a single datapoint, i.e. a string representing the color's components in some color space.

Intent	To represent the quality of having a color.
Competency Questions	*Does this objects have a color?*
Defined in	SOMA.owl



Expression	Meaning
has_color(o)	$o \in \mathscr{A}$ has a color
has_color(o,c)	$c \in \mathscr{A}$ is the color of $o \in \mathscr{A}$

Intent	To represent the color of physical objects.
Competency Questions	*What is the color of this object? Which objects have this color?*
Defined in	SOMA.owl

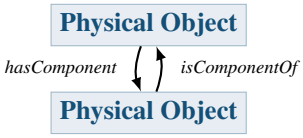


Expression	Meaning
object_color_rgb(o,[r,g,b])	$r,g,b \in \mathbb{R}$ is the RGB color data of $o \in \mathscr{A}$

2.3.2 Structure

Parthood represents that objects are composed of smaller things. These things may be physical objects themselves, and a *component* of their direct parent in the partonomy. Parthood is transitive, that is, parts of parts are parts again, but componency is not. So one would say that the arm is a component of the robot, and that the elbow is component of the arm, but not that the elbow is a component of the robot – however the elbow is a part of the robot due to the transitivity of the parthood relation.

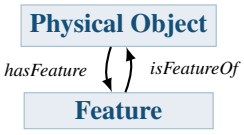
Intent	To represent proper parthood of objects.
Competency Questions	*What is this object component of? What are the components of this object?*
Defined in	DUL.owl



Expression	Meaning
has_component(x,y)	$y \in \mathscr{A}$ is a component of $x \in \mathscr{A}$

However, another parthood type is needed for objects such as holes, bumps, boundaries, or spots of color that are physical parts but not a proper component of their parent in the partonomy. These are *features* of the object. Features are usually localized in the object reference frame, and may carry additional properties describing, for example, the size of the hole, or the color of the spot.

Intent	To represent features of objects.
Competency Questions	*What are the features of this object? What are the objects with this feature?*
Defined in	SOMA.owl

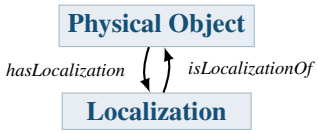


Expression	Meaning
has_feature(x,y)	$y \in \mathscr{A}$ is a feature of $x \in \mathscr{A}$

2.3.3 Kinematics

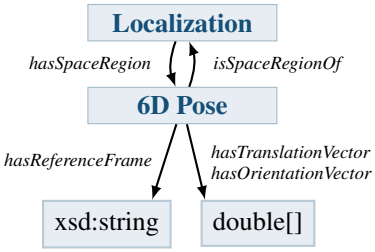
Kinematics, also often referred to as *geometry of motion*, describes how objects may move without considering the influence of forces. The kinematic state of an object is given by its pose over time, stored in the NEEM-experience as time-series data, and accessed via a dedicated predicate *is_at*. Poses are expressed within a frame of reference. There is one reference frame at the origin of each object, and possibly more at the various locations of interest. In addition, there is one dedicated root reference frame, the origin of the map. Each other frame must be, possibly indirect, connected to the map origin frame. The pose itself is given as 6D pose including the objects' orientation as quaternion vector.

Intent	To represent that objects are localized in a map.
Competency Questions	*Is this object localized? In which map is this object localized? What are the objects localized in this map?*
Defined in	SOMA.owl



Expression	Meaning
is_localized(o)	$o \in \mathscr{A}$ is localized wrt. some map origin
is_localized(o,m)	$o \in \mathscr{A}$ is localized wrt. the origin of $m \in \mathscr{A}$

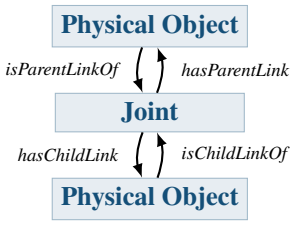
Intent	To represent kinematic trees of objects connected via joints.
Competency Questions	*Are these objects kinematically coupled? Which objects are linked through this joint?*
Defined in	SOMA.owl



Expression	Meaning
*is_at(o,[x,***p**,**q**]) during **ti**	**p** $\in \mathbb{R}^3$ is the position, and **q** $\in \mathbb{R}^4$ the orientation of $o \in \mathscr{A}$ within the reference frame $x \in \mathscr{F}$ during time interval **ti** $\in \mathbb{R}^2_{\geq 0}$

However, objects often can not move freely but are constrained in their movement with respect to some reference object. This is, for example, the case for walls without doors preventing movement from one room to another, or for two objects that are attached to each other via a *joint* and thus restricting movement relative to each other (kinematic coupling). Kinematically coupled objects are often part of a bigger hierarchical structure, and one of the linked objects, the parent link of the joint, is the one closer to the root of the structure then followed by the child link of the joint.

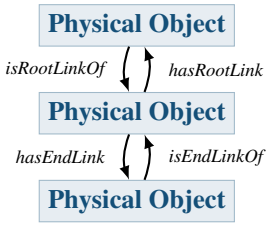
Intent	To represent kinematic trees of objects connected via joints.
Competency Questions	*Are these objects kinematically coupled? Which objects are linked through this joint?*
Defined in	SOMA.owl



Expression	Meaning
has_child_link(j,o)	$o \in \mathscr{A}$ is the child link of joint $j \in \mathscr{A}$
has_parent_link(j,o)	$o \in \mathscr{A}$ is the parent link of joint $j \in \mathscr{A}$

The pattern above is used to represent kinematical structures, however these representations must also be linked to the various typed objects that are represented in the NEEM-background . Such objects may be referred to directly in kinematical structures in case of not being composed of movable parts. In the case of having movable parts, the object corresponds to chains of links connected via joints in the kinematics representation. This is, for example, that the kinematical chain from shoulder to wrist forms an arm component. Each kinematic object component has exactly one root link, and may have many end links such as a hand component having its root in the wrist, and ending at each fingertip.

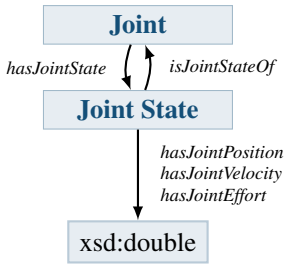
Intent	To represent the kinematic chain of object components.
Competency Questions	*What is the kinematic root of this object? What are the end links of this object?*
Defined in	SOMA.owl



Expression	Meaning
has_base_link(o_1,o_2)	$o_2 \in \mathscr{A}$ is the first link of $o_1 \in \mathscr{A}$
has_end_link(o_1,o_2)	$o_2 \in \mathscr{A}$ is one of the last links of $o_1 \in \mathscr{A}$

The position of a joint determines the position of the child link relative to the parent. Depending on whether the joint is either *hinged* (rotation around an axis) or prismatic (sliding along an axis), the position is measured in radians or meter respectively. When the position changes over time, velocity (measured in rad/s or m/s) and effort applied in the joint (measured in Nm or N) can be measured. Joint states are recorded as time-series data and stored in the NEEM-experience . However, the NEEM model defines a set of data properties used to access joint state data.

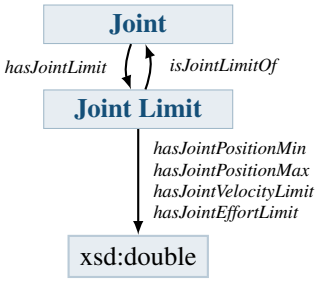
Intent	To represent the position of a joint.
Competency Questions	*What is the state of this joint? What is its position and velocity? How much effort is applied?*
Defined in	SOMA.owl



Expression	Meaning
has_joint_position(j,x)	$x \in \mathbb{R}$ is the position of joint $j \in \mathscr{A}$ given in m (prismatic joints) or *rad* (hinged joints)
has_joint_velocity(j,v)	$v \in \mathbb{R}$ is the velocity of joint $j \in \mathscr{A}$ given in $\frac{m}{s}$ (prismatic joints) or $\frac{rad}{s}$ (hinged joints)
has_joint_effort(j,x)	$x \in \mathbb{R}$ is the applied force of a prismatic, or the torque of a hinged joint $j \in \mathscr{A}$ given in N or $N \cdot m$ respectively.

The movement of a joint may be restricted by physical limits. This is the case for *revolute* and *prismatic* joints. The joint position is bounded between a minimum and maximum value, expressed as radians for revolute joints, and meters for prismatic joints. In addition, maximum values for the velocity and effort of a joint may be provided.

Intent	To represent the hard limits of a joint.
Competency Questions	*How far can this joint move into some direction?*
Defined in	SOMA.owl

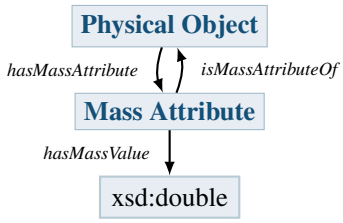


Expression	Meaning
*has_joint_position_limit(j,**x**)*	$\mathbf{x} \in \mathbb{R}^2$ is the minimum and maximum position of joint $j \in \mathscr{A}$ given in m (prismatic joints) or *rad* (hinged joints)
has_joint_velocity_limit(j,v_{max})	$v_{max} \in \mathbb{R}$ is the maximum velocity of joint $j \in \mathscr{A}$ given in $\frac{m}{s}$ (prismatic joints) or $\frac{rad}{s}$ (hinged joints)
has_joint_effort_limit(j,x_{max})	$x_{max} \in \mathbb{R}$ is the maximum force of a prismatic, or the maximum torque of a hinged joint $j \in \mathscr{A}$ given in N or $N \cdot m$ respectively.

2.3.4 Dynamics

The dynamics view in the NEEM-background is used to characterize how objects move under the influence of force. The NEEM model only considers solid objects with constant mass and dynamics governed by Newton's laws.

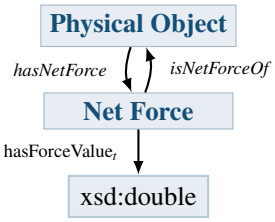
Intent	To represent the quantity of matter which a body contains.
Competency Questions	*What is the mass of this object?*
Defined in	SOMA.owl



Expression	Meaning
has_mass(o,v)	$v \in \mathbb{R}_{>0}$ is the mass of $o \in \mathscr{A}$ in kilograms

At each point in time, the sum of forces influencing an object determines how its movement will change. The accumulated force may be stored as time-series data in the NEEM-background , and accessed via an attribute defined in the NEEM model.

Intent	To represent the quantity of force influencing a solid object.
Competency Questions	*What is the force acting upon this object?*
Defined in	SOMA.owl



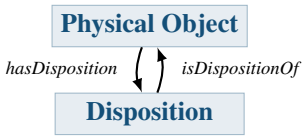
Expression	Meaning
*has_net_force(o,***f***) during* **ti**	**f** $\in \mathbb{R}^3$ is the accumulated force, measured in Newton, that acts upon $o \in \mathscr{A}$ during time interval **ti** $\in \mathbb{R}^2_{\geq 0}$

2.3.5 Naive physics

This view on objects is comprised of qualitative descriptions of the interactions of these objects, with a focus on how the objects could be arranged so as to constrain each other's behavior. The prototypical examples of such interactions are support and containment, but many other interactions are possible. Note, formalizing actual manifestations of such interactions as they occur in an episode will be done in chapter 3. In this chapter, we focus instead on ontological modelling about what kinds of interactions an object could take part in.

This is achieved by the concept of *Disposition*, which is a quality that, by virtue of being possessed by an object, enables that object to participate in certain roles in certain relations or events. E.g., *Deposition* and *Containment* are the dispositions necessary to enable an object to act as a support or container for another.

Intent	To represent what kinds of interactions an object can participate in.
Competency Questions	*What can this object be used for? Can this object interact with others in a particular way?*
Defined in	SOMA.owl



Expression	Meaning
has_disposition(x,y)	$y \in \mathscr{A}$ is a disposition of $x \in \mathscr{A}$

By and large, researchers making use of the SOMA ontology to create NEEM backgrounds and NEEMs can rely on the ontology to already provide a rich store of object knowledge, including dispositions. As such, it should usually be sufficient for the researchers adding knowledge about a new type of object, to specify the object classes defined in SOMA, to which the new object type is a subclass of. However, in case new dispositions need to be added for a new object type in the NEEM background, the above pattern illustrates how.

2.4 Data Formats

Representations in the NEEM-background may be enriched through additional data files. Data files are stored with the NEEM-background and loaded by the EASE knowledge base when a NEEM is activated. They may encode information that can be directly represented in the NEEM , however, it is not necessary in such a case to duplicate the information. A data file may be loaded at runtime, and used by the knowledge base in combination with other representations to answer questions about an activity.

2.4.1 URDF

The Unified Robot Description Format (URDF) is widely used for the representation of kinematics in robotics. That is how objects are organized in a skeletal structure of links and joints, and how links may move relative to each other in case of being connected via a flexible joint. URDF was designed to represent robot kinematics. It is, however, also often used for other types of objects with movable parts, for example to represent how a door is attached to a shelve via a joint, and what limits the joint has, but can also be used to represent completely static environments (via fixed joints). URDF further allows to represent a set of properties for links and joints, such as what the mass of a link is, or what the hard and soft limits of a joint are.

URDF organizes objects and their parts in a common coordinate system, and represents an initial configuration of all links and joints. The origin of this coordinate system is often called *world* or *map* frame. Each object has an associated frame in this coordinate system with a position relative to the parent frame in the skeletal structure. Frame names are further used to identify entities in the knowledge base, and logged position data that corresponds to objects described in the URDF file.

From the point of view of URDF, the world is only made of links and joints. Joints are further classified based on how they operate, and have different sets of parameters quantifying their kinematics depending on their type. It is, however, not possible to represent that links belong to a certain category, or that a chain of links forms a component of some type. However, we can use the information encoded in URDF files to enrich NEEM-background representations, and on the other hand, use the NEEM-background to provide classifications for links in URDF files.

Links in URDF files may have multiple associated shapes. Two different shape types are distinguished: collision and visualization shapes. Each link has usually one shape of each kind. Shapes are either represented as geometrical primitives such as spheres or boxes, or refer to an external mesh file in which case this mesh file needs to be stored as an additional data source in the NEEM-background (next Section).

2.4.2 DAE

The preferred format for meshes is Collada (DAE). The reason primarily being that it is widely supported by modeling tools and rendering engines. If possible, the mesh should be accompanied by high-resolution textures in PNG or JPG format. The more detailed a mesh, the more immersive the experience of humans interacting with it in VR and better the perception models that can be trained by images generated by placing the object in a virtual scene. Of course, this must be balanced with the computational demands imposed by mesh rendering and collision checking. When storing mesh and texture data for objects, researchers who produce NEEM background should make sure the meshes match the demands and resources of their applications.

3

NEEM-Narrative

D. BESSLER, M. POMARLAN, A. VYAS, S. JONGEBLOED

The narrative part of NEEMs represents a "story" of what has happened. The story is more detailed then it would be when told from one human to another as it includes details about movements and interactions that would usually not be spelled out in a human conversation. The reason to include this type of information is that it is extremely difficult to generalize robotic behaviour, and in particular how the robot needs to move its body to accomplish its goal with varying context such as different environemnts. The vision is that a library of *contextualized* motions and interactions will help to uncover models underlying everyday activities. With contextualized we mean that each motion and interaction is associated to the context of its occurrence. That is the environment where it occurred, objects that were involved, the goals and plans of agents, their behaviour etc.

The story represented in the NEEM-narrative provides context for the NEEM-experience acquired during an activity. This is, first of all, that labels are assigned to the time intervals where something relevant has happened such as the execution of a task, the interaction between objects, or the occurrence of a motion. Labels correspond to classes that are organized in a taxonomy (Section **??**). This allows to learn models at different levels of abstraction, e.g., for more general or specific variants of a task. The NEEM-narrative further represents relationships between instances of these classes, such as that an object plays a role during an event, or that a movement has happened as part of executing an action (Section 3.3).

3.1 Types of Events

There are several ways of classifying events including based on *agentivity* (e.g. intentional, natural), or on *typical participants* (e.g. human, physical, abstract, food).

The NEEM model does not take any of these directions. The reason is that events are related to observable situations, and that they can be viewed in different ways at the same time. For example, when seeing someone strolling while some piece of paper appears to slip out of a pocket one may view this event as an accident while in reality it was an intentional action to dispose trash on the street. The reason being that intentions of agents can usually not be observed directly, and hence an interpretation is required. Another aspect is that events may contribute to several goals, and, in such a case, may not be associated to a single category. This is, for example, the case when fetching a glas of wine from a fridge in order to first pour some wine into the Bolognese sauce, and second to drink the remaining wine from the glass. In this case the fetching event contributes to multiple goals and thus would belong to different classes in an action taxonomy organized based on agentivity.

The NEEM model defines a very shallow event taxonomy with rough distinctions that are not dependent on how an event is interpreted, executed or seen. These categories are: *action*, *process*, and *state*. Actions are exactly the events that are performed intentionally by an agent executing a task by following a plan or workflow. A process, on the other hand, is an event that causes state changes without the necessity of an agent driving or causing the process. Movement being one example, as it does not strictly depend on an agent executing a motion. For example, in case of the earth rotating around the sun, or some letter in a bottle being transported through the ocean. Finally, a state is defined as an event where an object has a dedicated stable configuration which is usually represented as having a certain property value over a fixed time interval.

However, it is possible to represent how an event is interpreted, executed or seen using a taxonomy of event *Concepts* which is included in the NEEM model. Concepts are used to classify events in some context. On the highest level, the concept taxonomy distinguishes between concepts used to classify the different event categories action, process and state.

Tasks

A *task* is defined as a concept that classifies actions based on what goals an agent intents to achieve when executing an action. Goals may be achieved in different ways depending on, for example, what is available in the environment, or what skills an agent has. Tasks do not imply a particular way of doing something, however, *plans* can be used to decompose a task into several sub-tasks and thus to represent a particular way how the task can be organized. For example, beer brewing is a task with a dedicated goal which can be accomplished through different workflows which are slightly different at each brewery. Tasks are further classified into three main categories: *physical task*, *mental task* and *communication task*. We are primarily interested in *physical tasks*. These are the tasks that require a physical agent to execute them. That is an agent with its own body and the ability to move meaningfully to achieve a designated goal by interacting in some way with the physical world.

This category includes tasks such as *actuating*, *constructing*, *modifying*, *placing* and *navigating*. A task that requires to execute an action through which an agent has to manipulate representations stored in its own cognition is seen as a *mental task*. *Dreaming*, *imagining*, *prospecting*, *reasoning*, and *retrospecting* are examples of this category. Finally, a *communication task* is seen as a task in which two or more agents exchange information. The scope of interest is to identify which agents communicate, and what kind of information is exchanged.

Process Types

The NEEM model also defines concepts that classify processes. One aspect is that of what a process does to objects. This is represented through the concepts *alteration*, *destruction* and *creation*. Note that this may also be view-dependent, one person may see only the destructive aspects of a process, but another person may see the creation of something new too. Another aspect we are interested in are the *force characteristics* of processes. That is how objects interact with each other with a reference to force. Each process may be labeled as *alterative* or *preservative*, meaning that there was either a tendency to set an object into motion, or to keep it still. Finally, in the NEEM model, motions are also seen as concepts that classify processes. The reason being, as explained before, that motions such as a message in a bottle moving in the ocean are not executed by an agent. However, it is ok to say that an event is an occurrence of a task and a motion at the same time as actions and processes are not disjoint. The motion taxonomy distinguishes between *directed* and *undirected* motions on the highest level. A directed motion is carried out towards a destination and following a directed path, while there is no destination for an undirected motion. An Oscillation being an example of an undirected motion. However, more relevant for goal-directed behaviour are directed motions such as *locomotion* where an agent navigates to a dedicated target, and *prehensile motion* which is directed towards grasping an object, or releasing the grasp again. The NEEM model defines another concept *fluid flow* that can be used to label a process where some fluid moves or has been moved from one location to another following the laws of fluid dynamics.

State Types

Finally, there are concepts defined in the NEEM model that can be used to classify states. We are in particular interested in the states where objects are in contact with each other. These are classified as *contact states*. Our interest in these states arises from the fact that they capture the interaction with the environment caused by the manipulation of objects where objects get into contact with the agent, or other objects in the environment. Other examples of state concepts used for the classification of state events are being *contained* within, and being *supported* or *blocked* by some other object.

3.2 Roles of Objects

Roles are concepts used to describe how an object participates in an event or a relation. An object can play different roles throughout its lifetime. For example, a knife may play the *PatientRole* of a grasping action, i.e. the object mainly affected by the action, and play an *InstrumentRole* in a cutting action. Other roles include *AgentRole*, *CausalProcessRole*, etc. Further role sub-categories will be defined under appendix section.

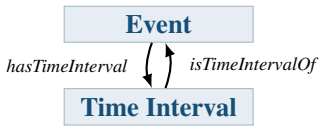
3.3 Views on Events

Events are characterized in the NEEM-narrative through relationships and attributes. We encapsulate certain characteristics in so called *event views*. Each view focusses on specific characteristics such as when the event occurred, or what roles objects have played in order to cover a set of competency questions that can be answered about an activitiy in case the corresponding view is represented. In the following, we will present the different views on events considered in the NEEM-narrative .

3.3.1 Occurrence

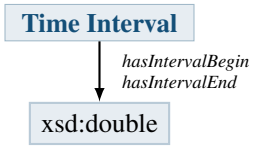
The NEEM model defines *Events* as things that unfold over time which is the main difference to *Objects* which are things that are wholly present at each time instant. This definition of events implies that an event is not instantaneous, but that it has an associated time interval during which it occurred.

Intent	To represent the temporal extension of events.
Competency Questions	*What is the time interval associated to this event?*
Defined in	DUL.owl



Expression	Meaning
occurs(x)	*x* is an occurence
is_time_interval(x)	*x* is a time interval
has_time_interval(y,x)	*x* is the time interval of event *y*

Intent	To quantify when something has happened.
Competency Questions	*When did it happen?*
Defined in	SOMA.owl

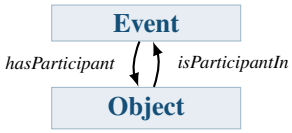


Expression	Meaning
occurs(x) during [y,z]	x occurs between the occurrences of y and z
occurs(x) since y	x and y begin at the same time
occurs(x) until y	x and y end at the same time

3.3.2 Participation

Events always involve some objects that play a certain role during the event. The role of being the *patient* of some event being an example. This is that the event is directed towards the object. It is not always directly observable what the role of an object might be, however, it is less problematic to just state that the object *has participated* in some event without naming a role.

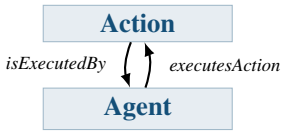
Intent	To represent participation of an object in an event.
Competency Questions	*Which objects do participate in this event? In which events does this object participate in?*
Defined in	DUL.owl



Expression	Meaning
has_participant(x,y)	y is involved in event x

Agents are defined as *agentive objects, either physical (e.g. a robot, a human or a whale) or social (e.g. a corporation, an institution or a community)*, Actions are defined as events with *at least one agent that is participating in it, and that is executing a task*. An example would be a robot that is grasping an object. In that case the robot is the agent and grasping would be the a task executed in an action. Actions can be executed by multiple agents.

Intent	To represent that an agent has executed an action.
Competency Questions	*Which agent did execute this action? Which actions are executed by this agent?*
Defined in	SOMA.owl



Expression	**Meaning**
is_executed_by(x,y)	*y* is executed by *x*

3.3.3 Composition

Because an *Event* often has relevant internal structure, it is necessary to represent relations between it and the other events that make up its composition. In general, the parts of an *Event* are themselves *Events*. For example, the parts of an *Action* may be *Actions* or *Processes*. To help identify when an *Event* is to be understood as a part of some larger, we will use the word "phase": an *Event* which is a part of another. Note that, as we discussed above, any *Event* may be part of another so the word "phase", while useful for our presentation here, does not correspond to a concept in the ontology.

To represent the parthood relation between *Events*, we define the *hasPhase* object property. This is a transitive and asymmetric property; no *Event* may be a part of itself.

Intent	To represent how an event is composed of phases.
Competency Questions	*What are the phases of this action? Is this a phase of some action?*
Defined in	SOMA.owl

Expression	**Meaning**
has_phase(x,y)	*y* is a phase of *x*

3.3.4 Transformation

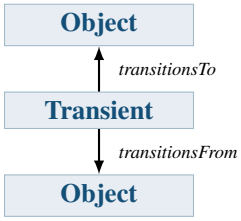
Objects typically undergo changes by taking part in actions or processes. Such changes typically take one of two forms: either some property of an object changes value, or the object itself modifies its ontological characterization. As an example, a wad

of dough being transported from the table to the inside of an oven has changed its position, but is still, at this point, a wad of dough. Left in a hot oven for enough time however, the wad of dough becomes bread.

The variation with time of object qualities has been described in section **??**. The object transformation pattern, described here, handles the changes of an object's ontological classification through time. An immediate problem is that ontological characterizations are necessarily discrete – there is a finite number of classes an object can belong to – while change in the physical world is continuous. In the example above, the wad of dough ceases to be dough after a few moments in the oven, because its chemical composition is changing away from the composition of dough. Nonetheless, it is not bread yet.

For practical purposes however, human beings often do not care about the exact classification of an object undergoing ontological change; only the endpoints are important. Also, there is a tendency to loosely apply ontological classification to the changing object, as if it were on either side of the transformation. The cooking wad of dough could be referred to as dough, or it could be referred to as bread. It is both, and neither. While sufficient for causal discourse, such carelessness would not work in a formal system. Hence, the approach in SOMA is to define a class of objects called *Transient*, and it is to this class that objects undergoing ontological classification change belong to. A Transient *transitionsFrom* some object with a specific classification (e.g. *Dough*) and *transitionsTo* another object with a specific classification (e.g. *Bread*). These two relations can be combined into the *transitionsBack* relation, to cover situations where an object changes in essential ways during an event, but returns to being itself after the event completes, such as a catalyst in a chemical reaction.

Intent	Ontological classification for objects undergoing type changes.
Competency Questions	*What sort of object is this? What objects "went" into the making of another? What is the outcome of some process of change acting on an object? Does an object preserve or restore its identity after change?*
Defined in	SOMA.owl



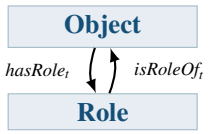
Expression	Meaning
transitionsFrom('A','B')	By entering some process of change, object 'B' becomes Transient object 'A'.
transitionsTo('A','B')	By completing some process of change, transient 'A' becomes object 'B'.
transitionsBack('A','B')	Object 'B' entered some process of change during which its ontological classification is unclear and it is replaced by Transient 'A', but after the process completes object 'B' is restored.

3.3.5 Conceptualization

The classification of entities is done from multiple viewpoints. The most essential one is *what the entitiy really is*. This is reflected in the taxonomy. However, entities may further be classified according to social aspects such as intention, purpose, etc. This type of classification is based on the *conceptualization* of entities. In particular, conceptualizations of objects and events are used to classify them in the scope of some activity.

An object that participates in an event usually plays a certain role during the event. Some objects are designed to be used in certain ways, thus playing certain roles in specific tasks. This is, for example, a box which is designed to be used as a container to store items. However, roles may also be taken by objects that are not designed to be used as such. The box could, for example, also be used as a door stopper, but surely it would be inappropriate to classify it as such taxonomically. Instead, roles are defined as concepts that are used to classify the objects that particpate in an event from a conceptual viewpoint.

Intent	To represents objects and the roles they play.
Competency Questions	*What role does this object play? Which objects do play that role?*
Defined in	DUL.owl

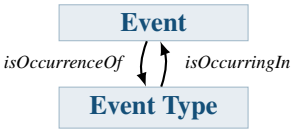


Expression	Meaning
has_role(x,y)	y is a role of x
has_role(x,y) during z	y is a role of x during the occurrence of z

The conceptualization of an event is about how it should be interpreted, executed, expected, seen, etc. One aspect is that a single event may contribute to multiple goals, such as when an ingredient is fetched that is partly used in a step of a cooking recipe,

and partly eaten raw to satisfy hunger. In such a case, the taxonomical category of the event would be unclear in case it should describe the goal to which the event contributes. Another aspect is that intentions of agents may not always be known such that the classification of events based on their goals is difficult, and different viewpoints on the same event may exist. Hence, when referring to goals, intentions, etc. we rather employ conceptual classification. This allows us to represent several different classifications of the same event in one or more situational contexts, for example an interpretations of the same event from different viewpoints.

Intent	To represent how events should be interpreted, executed, expected, seen, etc.
Competency Questions	*What are the events that are classified by this concept? What are the concepts that classify this event?*
Defined in	SOMA.owl

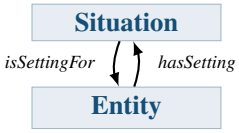


Expression	Meaning
is_occurring_in(x,y)	*y* is an occurrence that is classified by *x*
is_executed_in(x,y)	*y* is an action that executes the task *x*

3.3.6 Contextualization

An episode is seen as a *relational context* created by an observer that creates a view on a set of entities such as actions that were performed, and objects that played a role. We say that an episode is a *setting for* each entity that is relevant for the scope of the episode. As an example, consider the statement *"this morning the robot made a mess on the floor while preparing coffee"*, where the preparation of coffee in the morning is the setting for the robot, the floor, and the actions that were performed. Several specializations of the general *is setting for* relation exist that can be used to distinguish between entities based on their type – these are, among others, *includesObject* and *includesAgent*. However, assuming hierachical organization of objects (i.e. all objects are part of some map), and events (i.e. an event is composed into sub-events) only those entities at the root of the composition (e.g. the map) are required to be included explicitely in the episode.

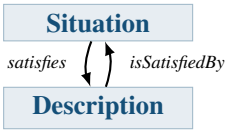
Intent	To represent that entites are included in a situation.
Competency Questions	*What are the entities that are relevant for this situation? What are the situations where this entity is relevant?*
Defined in	DUL.owl



Expression	Meaning
is_setting_for(x,y)	*x* is a setting for *y*

Episodes refer to concrete occurences with actual objects that are involved, and actual events that occur. The conceptualization of an episode is an abstraction that refers to concepts instead that are used to classify entities that are included in the episode. Such conceptualizations are called *descriptions*. We say that an episode *satisfies* a description in case the view represented by the episode is consistent with the conceptualization given by the description. Diagnosis being one example of a description of an episode. Stating that the performance of the robot was *amateurish* when it made a mess on the floor while preparing coffee is one example for describing an epsiode. Another type of descriptions are plans that are used to conceptualize the structure of an activity.

Intent	To represent a conceptualization of a situation.
Competency Questions	*How can this situation be conceptualized? What are the situations that are consitent with this conceptualization?*
Defined in	DUL.owl



Expression	Meaning
satisfies(x,y)	*x* is consistent with *y*

4

NEEM-Experience

D. BESSLER, S. KORALEWSKI

NEEM-experience captures low-level information about experienced activities represented as time series data streams. This data has often no or only unfeasible lossless representation as facts in a knowledge base. To make this data *knowledgable*, procedural hooks are defined in the ontology to compute symbols from the experience data, and to embed these symbols in logic-based reasoning.

The data is stored in a NoSQL database using JSON documents. Each individual type of data is stored in a separate collection named according to the type of data stored in the collection. When imported, the knowledge system stores the data in a MongoDB [3] server, for which the knowledge system implements a client for querying the data during question answering. The query cursor concept employed in MongoDB integrates nicely with backtracking based search employed in the knowledge system. It further scales well to large amount of data and can be distributed amongst clusters through built-in automatic sharding.

The data in NEEM-experiences is represented as time series and indexed in time order. The different experience data types need to define a dedicated time key for computing the search index.

The experience data in NEEMs has individual characteristics regarding the format, compressed representation, and what symbols the knowledge system can abstract from the data. In this chapter, we describe these aspects for the experience data types covered in NEEM version 1.0 .

[3] https://www.mongodb.com/

4.1 Kinematics

4.1.1 Pose data

A robotic system typically has many mobile components arranged in a kinematic chain. Each component in a kinematic chain has an associated named coordinate frame such as world frame, base frame, gripper frame, head frame, etc. Coordinate systems are always 3D, with x forward, y left, and z up, which follows the **right handed** coordinate system. 6 DOF relative poses are assigned to the different frames. These are usually updated with about 10 Hz during movements, and expressed relative to the parent in the kinematic chain to avoid updates when only the parent frame moves. The transformation tree is rooted in the dedicated world frame node (also often called map frame).

The data is used by the EASE knowledge system to answer questions such as:

- Where was the head frame relative to the world frame, 5 seconds ago?
- What is the pose of the object in the gripper relative to the base?
- What is the pose of the base frame in the map frame?

Pose data is saved in MongoDB collections named “tf”, the format is described below.

Format

The pose data structure has fields for encoding the translation and rotation of a coordinate frame. The parent frame and time stamp of pose estimation are stored in the *header* field of the data structure. The transform coordinate frame is assigned to the *child_frame_id* field.

Note that static frames may be recorded at lower frequency – about every two seconds. This usually reduces the data size significantly. At the moment, no other motion data compression, such as motion JPEG, is supported.

Field	Type	Description
tf	dict	–
header	dict	–
seq	uint32	consecutively increasing ID
stamp	time	time stamp of this transform
frame_id	string	parent coordinate frame of this transform
child_frame_id	string	coordinate frame of this transform
transform	dict	–
translation	dict	position of child frame with respect to parent frame in meters
x	float64	*x* axis translation
y	float64	*y* axis translation
z	float64	*z* axis translation
rotation	dict	–
x	float64	*x* component of quaternion
y	float64	*y* component of quaternion
z	float64	*z* component of quaternion
w	float64	*w* component of quaternion


```
{
 "header": {
  "seq": 0,
  "stamp": {
   "$date": "2018-06-22T09:07:54.072Z"
  },
  "frame_id": "map"
 },
 "child_frame_id": "spoon",
 "transform": {
  "translation": {
   "x": -0.05888763448188089,
   "y": -0.23866299023536164,
   "z": 0.13978717740253815
  },
  "rotation": {
   "x": 0.03631169636551407,
   "y": 0.974464008041022,
   "z": 0.1092500799183137,
   "w": 0.1927842778276488
  }
 }
}
```


Fig. 2. The pose data structure in the EASE system.

5

NEEM-Hub

S. KORALEWSKI, S. JONGEBLOED

Our CRC aims to acquire a huge amount of data, make the data accessible to the research community, allow to analyze the data, create machine learning models from the data and support version control for the data and models. With our NEEM-hub concept we are covering all those requirements with one system.

To implement the version control of large data sets and machine learning models, we are using DVC[4]. We use Hadoop[5] and its file system HDFS to store the data and models. Hadoop is a cluster system which creates automatically replicas of the data once it is uploaded and allows parallel processing of data to speed up transforming or querying the data.

On a high level perspective we want to realize two pipelines with our NEEM-hub as depict in Figure 3. The first pipeline can be seen as the acquisition and analytic pipeline. The first step in the pipline is storing raw data, such as videos, text, images and electrocardiogram (ECG) data. In the next step, so called neemifier are transforming this raw data into NEEMs which are utilizing the semantically representation provided by SOMA . Since we are using Hadoop, multiple instances of the same neemifier can neemify the raw data in parallel. There is also the possibility to upload NEEMs directly to the NEEM-hub , so the neemifing step can be skipped. The stored NEEMs can be accessed by openEASE . openEASE allows the inspection and visualization of each individual NEEM . A direct download of the NEEMs to a local system is also supported.

The second pipeline should be used as a pipeline for learning from the acquired data. NEEMs are so rich full on information that one NEEM can be used for multiple learning problems. In addition, NEEMs allow to generate models with different levels

[4] `https://dvc.org/`

[5] `https://hadoop.apache.org/`

of abstraction. One can learn general models e.g. the likely location of perishable items or/and specialized models e.g. how an agent should grasp my favorite mug in my kitchen. The procedure to generate the models is that transformers are used to extract the required features from the NEEMs and store them in a data representation, e.g. CSV, required for the machine learning model. The learners, which create the models, store the data in the model library. This model library can be used afterwards by an agent to perform reasoning.

In some scenarios those two pipelines can be combined to create a closed-loop. A possible scenario can be a robot acquiring NEEMs , uploading them to the NEEM-hub and download afterwards the new models to improve itself for the next experiment.

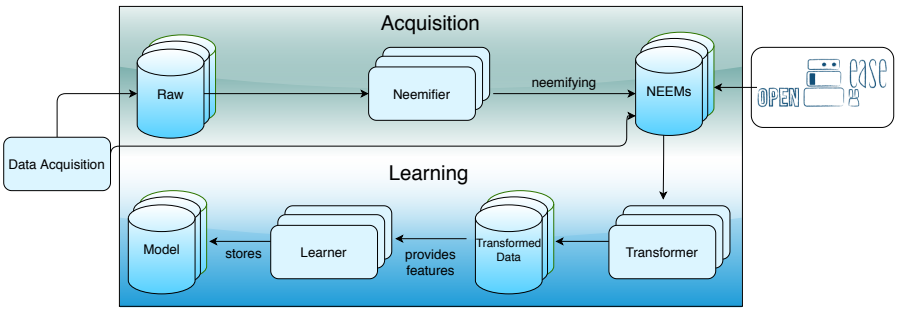


Fig. 3. The NEEM-Hub Architecture

Currently, we are only supporting data upload and hosting, like raw data, NEEMs and transformed data. In future, we will provide the feature to share your neemifier and transformers with the community and create your own pipelines directly on the NEEM-hub . In the rest of this section we will describe how you can upload the data to the NEEM-hub .

5.1 Prerequisite

1. Due to security reasons our Hadoop cluster can be only accessed via the university's intranet right now.
2. To be able to publish your data, you are required to have a git account on our GitLab system `https://neemgit.informatik.uni-bremen.de`
3. To support version control with our NEEMs we have to install DVC `https://dvc.org/`, if possible use a precompiled package `https://github.com/iterative/dvc.org/blob/master`.
4. Get familiar with DVC `https://dvc.org/doc/start`.

5. Install Hadoop or a client which can interact with our HDFS filesystem. To install Hadoop on Ubuntu 18.04 you can follow these steps:

   a) Install Java 8 if it is not already installed:

```
add-apt-repository ppa:webupd8team/java
apt update
apt install -y oracle-java8-set-default
```

Listing 5.1. Install Java

   b) Download the Hadoop Binaries, untar the archive and move it to /usr/local

```
wget http://apache.claz.org/hadoop/common/hadoop-3.3.1/
    hadoop-3.3.1.tar.gz
tar -xzvf hadoop-3.3.1.tar.gz
mv hadoop-3.3.1 /usr/local/hadoop
```

Listing 5.2. Install Hadoop Binaries

   c) Set the following environment variables so that DVC can find the Hadoop binaries:

```
export PATH="/usr/local/hadoop/lib/native:/usr/local/
    hadoop/bin:/usr/local/hadoop/sbin:$PATH"
export ARROW_LIBHDFS_DIR="/usr/local/hadoop/lib/native"
export CLASSPATH=`/usr/local/hadoop/bin/hdfs classpath
    --glob`
```

Listing 5.3. Export environment variables for Hadoop

5.2 Downloading

To test if you setup your system successful, you can try to download NEEMs :

1. You can explore a DVC repository using *dvc list*:

```
dvc list https://neemgit.informatik.uni-bremen.de/neems/
    ease-2020-pr2-setting-up-table
```

Listing 5.4. Exploring a DVC repository using *dvc list*

2. To download specific episodes you can download them using *dvc get*:

```
dvc get https://neemgit.informatik.uni-bremen.de/neems/ease
    -2020-pr2-setting-up-table episodes/1599727087.4392.zip
```

Listing 5.5. Downloading a episode using *dvc get*

3. To download a complete NEEM you can download it by first cloning the repository using *git* and then using *dvc pull*:

```
git clone https://neemgit.informatik.uni-bremen.de/neems/
    ease-2020-pr2-setting-up-table
cd ease-2020-pr2-setting-up-table
dvc pull
```

Listing 5.6. Downloading a NEEM using *dvc pull*

5.3 Publishing

To organized the raw and transformed data and the NEEMs , we created 3 groups on GitLab. Each group corresponds to the specific data set type. The general procedure to publish your data set is that you create a git repository in the corresponding group. The following is an example step by step guide on how to publish a dataset, in this case a NEEM:

1. Create a repository in the NEEM group `https://neemgit.informatik.uni-bremen.de/neems`
2. Clone the empty repository and initialize DVC in this repository

```
git clone git@neemgit.informatik.uni-bremen.de:neems/hello-
    neem2.git
cd hello-neem2
dvc init
```

Listing 5.7. Initialize DVC using *dvc init*

3. Define the remote storage, to let DVC store your data set:

```
dvc remote add -d origin hdfs://hadoop@134.102.137.65:9000/
    neems/hello-neem2
```

Listing 5.8. Define remote storage using *dvc remote add*

4. You can now add your episodes via DVC and push it:

```
dvc add episodes/1599727087.4392.zip
dvc push
```

Listing 5.9. Add data and pushing using *dvc add* and *dvc push*

5. After pushing your data, you still need to push the DVC data placeholder[6]:

[6].dvc files: https://dvc.org/doc/user-guide/project-structure/dvc-files

```
git add episodes/1599727087.4392.zip.dvc
git commit -m "Some commit message"
git push
```

Listing 5.10. Downloading a NEEM using *dvc pull*

Publishing other data set types will work in a familiar fahion, but you need to create the repository in the corresponding group and define the remote storage based on your data set type:

Raw data
`hdfs://hadoop@134.102.137.65:9000/raw/<your-repo-name>`

NEEMs
`hdfs://hadoop@134.102.137.65:9000/neems/<your-repo-name>`

Transformed Data
`hdfs://hadoop@134.102.137.65:9000/transformed_data/<your-repo-name>`

In future, you will be able to publish your neemifier and transformers. This will allow to share your tools easily with the community and also automatize your data transformation and learning pipeline.

6

NEEM-Acquisition

S. KORALEWSKI, A. HAWKIN

This chapter focuses on the acquisition process of NEEMs . At first, we will provide the tools and procedures to acquire episodic memories from robots performing experiments. The second section focuses on the NEEM acquisition from virtual reality.

6.1 Data Structure

We are using MongoDB to capture the data structures of the NEEMs . If you will use the KnowRob interface to create your NEEMs then your NEEM will consist of at least 3 folders - *annotations*, *inferred* and *triples*. The NEEM-narrative and NEEM-experience are stored as a collection of BSON [7] files. Each folder should contain a BSON file and metafile stored as JSON. The metafile will include additional information related to NEEMs . This additional meta information is useful for searching NEEM on openEASE platform and hence needs to be provided by NEEM creator while NEEM acquisition time. An example of such information is as displayed below:

Each generated NEEM stores also the complete state of the SOMA ontology which was used during the acquisition process. The benefit of this is that while loading a NEEM , it is not required to keep track to load the correct SOMA version. In the following, we will give an overview which information is contained in those folders generated by KnowRob :

annotations The annotations collection contains annotations(comments) which are asserted to the concepts of the ontology.

[7] http://bsonspec.org/faq.html

```
{
  "_id" : ObjectId("5f22b1f512db5aed7cd1961b"),
  "created_by" : "seba",
  "created_at" : "2020-07-21T06:54:25+00:00",
  "model_version" : "0.1",
  "description" : "NEEM for robot making pizza.",
  "keywords" : [
  "Pizza",
  "Robot"
  ],
  "url" : "Placeholder for the NEEM hub repository url",
  "name" : "NEEM for robot making pizza",
  "activity" : {
    "name" : "Pizza making",
    "url" : "Placeholder for the url/uri of Activity concept
    defined in ontology"
  },
  "environment" : "Kitchen",
  "image" : "placeholder for image url for showing neem image
    on openEASE",
  "agent" : "Robot"
}
```

Fig. 4. The meta data structure.

inferred The inferred collection contains triples which were inferred and not asserted during the logging process. Inference processes can be triggered when triples are asserted directly to the knowledge base.

triples The triples collection contains all triples which were asserted into the knowledge base during run time.

6.1.1 Triple data as JSON object

Triple data can also be provided in form of JSON documents, where triples are represented as subject, predicate and object. Subjects and objects are identified by an Internationalized Resource Identifier (IRI), which is pointing to concepts or instances defined in the SOMA ontology. A triple can either link subject with an object, or can link subject with data value which is represented using one of the base types: string, boolean, and a number. Whereas, the predicate is named by the IRI pointing to property concepts mentioned in the SOMA ontology. It is also possible to provide additional time scope fields ‘since‘ and ‘until‘ to indicate that the given triple is valid for the given time scope. These values are considered here in seconds from when an experiment has started being recorded. An example of such a JSON document is given below where the *Salad_PMRVYPJH* has a *patient* role from 27.739th second

till 29.075. By default triple is valid for the infinite time when the scope parameters are not specified.

```
[
 {
   "s": "http://www.ease-crc.org/ont/SOMA.owl#Salad_PMRVYPJH",
   "p": "http://www.ontologydesignpatterns.org/ont/dul/DUL.owl
    #hasRole",
   "o": "http://www.ease-crc.org/ont/SOMA.owl#Patient_YGJUVNDR
    ",
   "since": 27.739,
   "until": 29.075
 }
]
```

Fig. 5. The triple data structure.

It is important to note that, this is an intermediate data format which is not equivalent with how the NEEM narrative is actually stored in databases. The format described here rather serves as an easy-to-use interchange format.

6.2 Robot NEEMs

This section describes how to generate NEEMs from experiments performed by the robot either in simulation or in the real world.

6.2.1 Prerequisite

Before you are intending to generate your episodic memories, make sure you are familiar with the Cognitive Robot Abstract Machine (CRAM)[8] system and it is installed on your machine. CRAM is a planning framework which allows to design high-level plan for robots. This section requires that your robot plan is written in CRAM to be able to generate NEEMs . However, once you are familiar with our planning and logging components, you will be able to port those components to your preferred planning tool.

To generate NEEMs you will need also the following software components to be installed:

[8] http://cram-system.org/cram

- A MongoDB server with at least version 3.4.10[9]
- KnowRob [10]
- CRAM ontology[11]

6.2.2 Recording Narrative Enabled Episodic Memories

Our recording mechanism captures every executed CRAM action and its parameter. In addition, the logger puts the actions in relation to each other by creating a hierarchy which is described in Section 3.3.3.

Before you can begin to record your own NEEMs , you need to include the ”cram-cloud-logger” package into your CRAM plan. This package is by default included with your CRAM installation. To start and finish the logging process, run *ccl::start-episode* before your plan execution and when you are finished with your experiment run *ccl::stop-episode*. It can look like the following:

```
(ccl::start-episode)
(urdf-proj:with-simulated-robot (demo::demo-random nil ))
(ccl::stop-episode)
```

Listing 6.1. Steps to Record an Episode for a CRAM Plan

The generated NEEM will be stored per default in ”~/knowrob-memory”. Keep in mind to have KnowRob launched before starting the NEEM recording. You can start KnowRob via:

```
roslaunch knowrob_memory knowrob.launch
```

Listing 6.2. How to Start KnowRob

By default your NEEM will be stored under ”~/knowrob-memory/<timestamp >”.

6.2.3 Add Semantic Support to your Designed Plans

The disadvantage of having a strong semantic knowledge representation is that the used ontology requires updates when semantic knowledge has to be extended. Currently, the EASE project focuses on the support to record table setting/cleaning-up experiments. If you want to create NEEMs for e.g. autonomous cars, you will need to extend the SOMA ontology and the recording mechanism with your required

[9] https://www.mongodb.com/
[10] https://github.com/knowrob/knowrob
[11] https://github.com/ease-crc/cram_knowledge

actions, parameters and objects. How to extend the SOMA ontology is described in Section **??**. In the following subsection, we will describe how you can extend the CRAM recording mechanism to support your required semantic knowledge.

New Tasks Definition

If you want to semantically record new tasks such as e.g. accelerating or breaking, you need to define them in the ontology as described in Section **??**. Tasks which are not defined in the ontology, will be logged as *PhysicalTask*. The *PlanExecution* instance pointing to the *PhysicalTask* will have a comment attached with the statement "Unknown Action: <CRAM-ACTION-NAME>", where <CRAM-ACTION-NAME>is the action name you defined in your plan. After you defined your new actions in SOMA , please open the "knowrob-action-name-handler.lisp" in the "cram-cloud-logger" package and add your new actions in the format:

```
(defun init-action-name-mapper ()
   (let ((action-name-mapper (make-instance 'ccl::
    cram-2-knowrob-mapper))
     (definition
         '(("reaching" "Reaching")
           ("retracting" "Retracting")
           ("lifting" "Lifting")
           ("opening" "Opening")
           ("<CRAM-ACTION-NAME>" "<SOMA-ONTOLOGY-NAME>")
```

Listing 6.3. Linking the CRAM Action to the Ontology Concept

where <CRAM-ACTION-NAME> is the name of your action used in the cram plan and <SOMA-ONTOLOGY-NAME> is the concept name of the task defined in SOMA . With this step you added successfully the new action to the CRAM NEEM episodic memory logger.

Adding New Objects

Unknown objects will be logged as instance of *DesignedArtifact*. In addition, a comment like "Unknown Object: <CRAM-OBJECT-TYPE>" will be attached to this instance. <CRAM-OBJECT-TYPE> indicates which object you need to define in the SOMA ontology. After you added your object to the ontology, open the "utils-for-perform.lisp" in the "cram-cloud-logger" package and include the new object in the hash table generate in "get-ease-object-lookup-table" like:

```
(defun get-ease-object-lookup-table()
  (let ((lookup-table (make-hash-table :test 'equal)))
    (setf (gethash "BOWL" lookup-table) "'http://www.ease-crc.
   org/ont/SOMA.owl#Bowl'")
    (setf (gethash "CUP" lookup-table) "'http://www.ease-crc.
   org/ont/SOMA.owl#Cup'")
```

```
    (setf (gethash "<CRAM-OBJECT-TYPE>" lookup-table) "'<
   SOMA-ONTOLOGY-ENTITY-IRI>'")
lookup-table))
```

Listing 6.4. Linking the CRAM Object to the Ontology Concept

where the key is <CRAM-OBJECT-TYPE>, the object type used in the CRAM plan, and <SOMA-ONTOLOGY-ENTITY-URI>the value which is the uri pointing to the object concept created in SOMA .

Adding New Failure

Unknown CRAM failures will be logged as instance of *Failure*. In addition, a comment like "Unknown failure: <CRAM-FAILURE-NAME>" will be attached to the instance. <CRAM-FAILURE-NAME> indicates which failure you need to define in the ontology. After you added your failure to the ontology, open the "failure-handler.lisp" in the "cram-cloud-logger" package and add your new failure in the format:

```
(defun init-failure-mapper ()
  (let ((failure-mapper (make-instance 'ccl::
    cram-2-knowrob-mapper))
    (definition
      '(("cram-common-failures:low-level-failure" "
    LowLevelFailure")
        ("cram-common-failures:actionlib-action-timed-out" "
    ActionlibTimeout")
        ("<CRAM-FAILURE-NAME>" "<SOMA-ONTOLOGY-NAME>")
```

Listing 6.5. Linking the CRAM Action to the Ontology Concept

where <CRAM-FAILURE-NAME> is the name of your failure defined in the cram plan and <SOMA-ONTOLOGY-NAME>. With this step you added successfully the support of the new failure to CRAM NEEM episodic memory logger.".

6.2.4 Summary

This section provided you the fundamentals, how you can utilize the SOMA ontology and CRAM to create your own NEEMs . Under the following link (`https://neemgit.informatik.uni-bremen.de/neems/ease-2020-pr2-setting-up-table`), you can download some robot NEEMs to have an overview of a generated NEEM .

6.3 VR NEEMs

This section will describe how NEEMs can be generated within a Virtual Reality environment and how they can be utilized within CRAM plans to help a robot perform everyday household activities. The use of VR allows us as humans to show the robot an action we want it to perform within a variety of different environments.This facilitates learning of a lot of common sense knowledge, e.g. where the objects necessary to perform a specific action are commonly stored within the environment , which objects are needed for a specific action, where the human user was standing when he was grasping a certain object, how the objects were arranged on a surface relative to one another and how the human user grasped them.

The example scenario used here is the breakfast setting scenario. This means that the robot is supposed to set up the table with a bowl, cup and a spoon in preparation of a breakfast cereal meal.

6.3.1 Prerequisite

Before VR-NEEM generation can take place, the proper VR-environment needs to be set up within the Unreal Engine, including the installation of the USemLog Plugin, which records the NEEMs and generates the appropriate .owl files. The plugin and a setup of a kitchen environment can be found within the RobCog project. The KnowRob and MongoDB installation are the same as in the section above. In order to be able to use the NEEMs within CRAM , the data first needs to be transferred into the MongoDB and KnowRob. This can be achieved by running the *vr neems to knowrob* scripts. Please refer to the README.md for execution examples.

To summarize, you will need to install the following components:

- Unreal Engine[12] Version 4.22.3
- RobCog[13]
- vr_neems_to_knowrob[14]
- knowrob_robcog
- CRAM[15] Branch: boxy-melodic

[12] `https://www.unrealengine.com/`
[13] `https://github.com/robcog-iai/RobCoG`
[14] `https://github.com/ease-crc/vr_neems_to_knowrob`
[15] `https://github.com/cram2/cram.git`

6.3.2 Recording Virtual Reality Narrative Enabled Episodic Memories

Check if all items that you want to appear in the NEEMs , have a tag under which they will be represented within the ontology. You can check the tag by clicking any item within the kitchen environment, and going to the *Actor* section in the *details* pane. Click the little arrow to expand the section, and also expand the *Tags* section. There you should see something like this:

```
SemLog;Class,IAIIslandArea;Id,tpzV61885UGL785BwZFHYQ;
```

This string defines the class of the item and it's id. More information can be added here, depending on the item and the ontology and the level of detail desired within the classification. This is also very important when introducing new items to the Virtual Reality setup. Items which do not have these tags, will not be included in the recorded NEEM data.

Once everything is set up and all potentially new items are within the virtual reality environment, the recording can begin. First, the semantic map can be automatically generated by going to the *RobCog* pane within the *Unreal Engine* editor, and clicking the *SemanticMap* button. After this, the generated semantic map should be located in the *RobCog/Episodes* directory. It contains the initial state of the virtual reality environment, including the position and rotation of all the furniture objects and also their classifications.

The next step is then to start the virtual reality simulation and perform some actions within it, e.g. setting up a breakfast table. The position of the VR-headset and controllers is being tracked the entire time, as well as the interactions of the virtual hands with the environment and objects. Picking up an object would generate a *Grasping-Something*-action. Placing an object down on a table would generate a *Contact*-Action between the object and the surface it has been placed upon. All these interactions can later on be queried for.

Once all the desired actions are complete, the simulation can be stopped and an *EventData_ID* directory appears in the *RobCog/Episodes* directory. It contains and *EventData_ID.owl* file, an *EventData_ID.html* file, which visualizes all the occurred events, and a *RawData_ID.json*, which contains all the information about the performed actions and events. The last file is the one that needs to be uploaded into the *MongoDB*.

6.3.3 Transferring VR-NEEMs into the Knowledge Base

Please refer to the README of these scripts `https://github.com/ease-crc/vr_neems_to_knowrob` in order to import the VR-NEEMs into KnowRob and MongoDB. More information about the import, how it generally works and why the scripts were created the way they are, please refer to: `http://cram-system.`

org/tutorials/advanced/unreal#importing_new_episode_data_into_mongodb_and_knowrob_additional_information.

6.3.4 Using VR-NEEM Data in CRAM plans

There is a demo within CRAM which uses the data collected in VR, including a tutorial on how to run it. It can be found here: http://cram-system.org/tutorials/advanced/unreal. In this demo the robot performs a pick and place task, picking up a cup, bowl and a spoon from the sink counter, and placing them onto the kitchen island. In order to do this, CRAM queries KnowRob for the following information:

- where/from which surface was the object picked up?
- where was the human user standing when he was picking up/placing the object?
- on which surface and where was the object placed? (In relation to other objects)
- with which hand was the object grasped?
- from which direction (top,left,right...) was the object grasped?

Since the virtual reality kitchen can look very different than the one the robot is acting in, all of the poses are calculated relative to the respective surfaces and each other. For example, the spoon is always placed to the right of the bowl.

For more information on how CRAM interacts with KnowRob and how json-prolog can be used within CRAM , please refer to http://cram-system.org/tutorials/intermediate/json_prolog

7

NEEM Quick-start Guide

A. VYAS, S. JONGEBLOED

In this chapter we present a checklist for the NEEMs creation process to help the users in generating NEEMs in case no existing NEEM-logger can be used.

7.1 NEEM Checklist

7.1.1 Kinematic information with visualization meshes

In order to visualize NEEM experiment in openEASE , you should have the following files available:

- Agent meshes and urdf files
- Agent owl file corresponding to urdf. Refer Figure 8.1 for agent owl file example.
  - An owl file should contain all of the links from urdf.
  - An agent structure should be created with appropriate SOMA concept.
  - Required kinematic information should be provided in owl file pointing to correct urdf file name. Refer Figure 8.1, individuals for kinematics information.
- Environment meshes and urdf files
- Environment owl file corresponding to urdf. Refer Figure 8.2 for agent owl file example.
  - An owl file should contain all of the links from urdf.
  - An Environment structure should be created with appropriate SOMA concept.

  - Required kinematic information should be provided in owl file pointing to correct urdf file name. Refer Figure 8.2, individuals for kinematics information.

7.1.2 NEEM Data format

NEEM data is represented in tf and triple collections. This data should automatically be in the correct format if KnowRob is used to log the information. But in case you want to create NEEM data without Knowrob, please follow the checklist below:

- The following steps will make sure the correct tf data format is provided. An example of such format is presented in Figure 2.
  - Tf data is provided as individual json documents not as the list/array of json documents.
  - The coordinate system is right handed.
  - Correct tf tree is presented in the data.
  - Joint rotation is provided in quaternion.
  - Position data is logged in meters.
- The following steps will make sure the correct triple data format is provided. An example of such format is presented in Figure 5.
  - Triple data is provided as an array of json document.
  - Correct SOMA concepts used from NEEM-narrative part.

7.1.3 Semantic Annotation

In this chapter we discuss the necessary semantic annotation that is stored in the triple collection. First we will list the semantic information that is necessary to generate a simple NEEM:

- Necessary steps when starting the logging:
  - Create the episode and add dul:isSettingFor relation between the episode and the robots and locations (see 3.3.6)
- Create an hierachy of actions
  - Add the task that is executed during the action (see 3.3.5)
  - Add start and endtime (as unix timestamps) to action (see 3.3.1)

  - Repeat the above points for all sub-actions of the Action-Hierachy, and link them to the parent-actions (see 3.3.3)
  - For the top-level action: Link the action to the created Episode

Now we we will list some additional semantic annotation that would be helpful for the future use of the logged NEEM:

- Add additional informations to better classify an action:
  - Add the performing agents (see 3.3.2)
  - Add objects that are participating in the action (see 3.3.2)
  - Add conceptualization to the objects in an action by adding roles (see 3.2)
  - Add executed motions to an action (see 3.1)

In general, additional semantic annotations can be added as needed. In the next chapter we show how this annotation can be implemented with KnowRob.

7.1.4 Semantic Annotation: KnowRob

The easiest way to generate a correct semantic annotation described in 7.1.3 is using KnowRob. First we will describe which queries are necessary to generate a simple NEEM. The used concepts for agents, objects, roles etc. are examples. Please find the correct concepts for your usage in SOMA[16].

- Necessary steps when starting the logging:
  - Load the OWL Files collected according to 7.1.1, e.g.:

```
tripledb_load('package://knowrob/owl/robots/PR2.owl')
```

  - Load the URDF Files and link them to the corresponding robot/location from the OWL File, e.g.:

```
urdf_load('http://knowrob.org/kb/PR2.owl#PR2_0', '
    package://knowrob/urdf/pr2.urdf', [load_rdf])
```

  - Create the episode: tell(is_episode(Episode))
  - Add setting_for relations for robots and locations:

```
is_setting_for(Episode,'http://knowrob.org/kb/PR2.owl#
    PR2_0')
```

[16] https://ease-crc.github.io/soma

- Log the Action-Hierachy
  - Create an action, e.g.:

```
tell(is_action(Action))
```

  - Add the task that is executed during the action, e.g.:

```
tell([has_type(Tsk,soma:'Transporting'),
executes_task(Action,Tsk)])
```

  - Add start and endtime (as unix timestamps) to action, e.g.:

```
tell(occur(Act) during [Start, End])
```

  - Repeat the above points for all sub-actions of the Action-Hierachy, and link them to the parent-actions:

```
tell(has_subevent(ParentAct,Action))
```

  - For the top-level action: Link the action to the created Episode, e.g.:

```
tell(is_setting_for(Episode,Action))
```

Now we we will list some additional semantic annotation to add more information to the logged NEEM:

- Add additional informations to better classify an action:
  - Add the performing agents, e.g.:

```
tell(is_performed_by(Action,pr2:'PR2_0'))
```

  - Add objects that are participating in the action, e.g.:

```
tell(has_participant(Action,soma:'Milk_0'), \)
```

  - Conceptualize objects and agents in an action by adding roles, e.g.:

```
tell([has_type(RobotRole, soma:'AgentRole'),'
    has_role(pr2:'PR2_0', RobotRole) during Action,'])
```

  - Add executed motions to an action, e.g.:

```
tell([has_type(Mot,soma:'LimbMotion'),
      is_classified_by(Action,Mot)])
```

8

Appendix

8.1 Agent owl file

```
<?xml version="1.0"?>
<rdf:RDF xmlns="http://knowrob.org/kb/avatar-skeleton.owl#"
         xml:base="http://knowrob.org/kb/avatar-skeleton.owl"
         xmlns:srdl2="http://knowrob.org/kb/srdl2.owl#"
         xmlns:srdl2-cap="http://knowrob.org/kb/srdl2-cap.owl#
    "
         xmlns:owl="http://www.w3.org/2002/07/owl#"
         xmlns:owl2xml="http://www.w3.org/2006/12/owl2-xml#"
         xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
         xmlns:urdf="http://knowrob.org/kb/urdf.owl#"
         xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
         xmlns:qudt-unit="http://qudt.org/vocab/unit#"
         xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns
    #"
         xmlns:soma="http://www.ease-crc.org/ont/SOMA.owl#"
         xmlns:xml="http://www.w3.org/XML/1998/namespace"
         xmlns:srdl2-comp="http://knowrob.org/kb/srdl2-comp.
    owl#"
         xmlns:knowrob="http://knowrob.org/kb/knowrob.owl#"
         xmlns:dul="http://www.ontologydesignpatterns.org/ont/
    dul/DUL.owl#">

    <!-- ========================================== -->
    <!-- |   Ontology Imports                     | -->
    <!-- ========================================== -->

    <owl:Ontology rdf:about="http://knowrob.org/kb/
    avatar_skeleton.owl">
        <owl:imports rdf:resource="http://www.ease-crc.org/ont
    /SOMA.owl"/>
```

```
    </owl:Ontology>

    <!--
    //////////////
    //
    // Classes
    //
    //////////////
     -->


    <!-- http://www.ontologydesignpatterns.org/ont/dul/DUL.owl
    #PhysicalObject -->
    <owl:Class rdf:about="http://www.ontologydesignpatterns.
    org/ont/dul/DUL.owl#PhysicalObject"/>



    <owl:Class rdf:about="http://knowrob.org/kb/
    avatar_skeleton.owl#avatar_skeleton">
        <rdfs:subClassOf rdf:resource="http://www.
    ontologydesignpatterns.org/ont/dul/DUL.owl#Agent"/>
    </owl:Class>


    <!--
    //////////////
    //
    // Individuals
    //
    //////////////
     -->

    <owl:NamedIndividual rdf:about="http://knowrob.org/kb/
    avatar_skeleton.owl#avatar_skeleton_1">
        <rdf:type rdf:resource="http://www.
    ontologydesignpatterns.org/ont/dul/DUL.owl#PhysicalBody"/>
        <urdf:hasBaseLinkName rdf:datatype="http://www.w3.org
    /2001/XMLSchema#string">Hip</urdf:hasBaseLinkName>
        <dul:hasComponent rdf:resource="http://knowrob.org/kb/
    avatar_skeleton.owl#avatar_skeleton_leg_l"/>
        <dul:hasComponent rdf:resource="http://knowrob.org/kb/
    avatar_skeleton.owl#avatar_skeleton_leg_r"/>
        <dul:hasComponent rdf:resource="http://knowrob.org/kb/
    avatar_skeleton.owl#avatar_skeleton_middle_body"/>
        <dul:hasComponent rdf:resource="http://knowrob.org/kb/
    avatar_skeleton.owl#avatar_skeleton_head"/>
        <dul:hasComponent rdf:resource="http://knowrob.org/kb/
    avatar_skeleton.owl#avatar_skeleton_arm_r"/>
```

```
62          <dul:hasComponent rdf:resource="http://knowrob.org/kb/
    avatar_skeleton.owl#avatar_skeleton_arm_l"/>
63          <dul:hasComponent rdf:resource="http://knowrob.org/kb/
    avatar_skeleton.owl#avatar_skeleton_hand_l"/>
64          <dul:hasComponent rdf:resource="http://knowrob.org/kb/
    avatar_skeleton.owl#avatar_skeleton_hand_r"/>
65
66           <urdf:hasEndLinkName rdf:datatype="http://www.w3.org
    /2001/XMLSchema#string">RToe</urdf:hasEndLinkName>
67  <urdf:hasEndLinkName rdf:datatype="http://www.w3.org/2001/
    XMLSchema#string">LToe</urdf:hasEndLinkName>
68  <urdf:hasEndLinkName rdf:datatype="http://www.w3.org/2001/
    XMLSchema#string">Chest</urdf:hasEndLinkName>
69      </owl:NamedIndividual>
70
71      <owl:NamedIndividual rdf:about="http://knowrob.org/kb/
    avatar_skeleton.owl#avatar_skeleton_leg_l">
72          <rdf:type rdf:resource="http://www.ease-crc.org/ont/
    SOMA.owl#Leg"/>
73          <urdf:hasBaseLinkName rdf:datatype="http://www.w3.org
    /2001/XMLSchema#string">Hip</urdf:hasBaseLinkName>
74          <urdf:hasEndLinkName rdf:datatype="http://www.w3.org
    /2001/XMLSchema#string">LToe</urdf:hasEndLinkName>
75      </owl:NamedIndividual>
76
77      <owl:NamedIndividual rdf:about="http://knowrob.org/kb/
    avatar_skeleton.owl#avatar_skeleton_leg_r">
78          <rdf:type rdf:resource="http://www.ease-crc.org/ont/
    SOMA.owl#Leg"/>
79          <urdf:hasBaseLinkName rdf:datatype="http://www.w3.org
    /2001/XMLSchema#string">Hip</urdf:hasBaseLinkName>
80          <urdf:hasEndLinkName rdf:datatype="http://www.w3.org
    /2001/XMLSchema#string">RToe</urdf:hasEndLinkName>
81      </owl:NamedIndividual>
82
83      <owl:NamedIndividual rdf:about="http://knowrob.org/kb/
    avatar_skeleton.owl#avatar_skeleton_middle_body">
84          <rdf:type rdf:resource="http://www.
    ontologydesignpatterns.org/ont/dul/DUL.owl#PhysicalBody"/>
85          <urdf:hasBaseLinkName rdf:datatype="http://www.w3.org
    /2001/XMLSchema#string">Hip</urdf:hasBaseLinkName>
86          <urdf:hasEndLinkName rdf:datatype="http://www.w3.org
    /2001/XMLSchema#string">Chest</urdf:hasEndLinkName>
87      </owl:NamedIndividual>
88
89      <owl:NamedIndividual rdf:about="http://knowrob.org/kb/
    avatar_skeleton.owl#avatar_skeleton_head">
90          <rdf:type rdf:resource="http://www.ease-crc.org/ont/
    SOMA.owl#Head"/>
```

```
91        <urdf:hasBaseLinkName rdf:datatype="http://www.w3.org
    /2001/XMLSchema#string">Chest</urdf:hasBaseLinkName>
92        <urdf:hasEndLinkName rdf:datatype="http://www.w3.org
    /2001/XMLSchema#string">Head</urdf:hasEndLinkName>
93    </owl:NamedIndividual>
94
95    <owl:NamedIndividual rdf:about="http://knowrob.org/kb/
    avatar_skeleton.owl#avatar_skeleton_arm_l">
96        <rdf:type rdf:resource="http://www.ease-crc.org/ont/
    SOMA.owl#Arm"/>
97        <urdf:hasBaseLinkName rdf:datatype="http://www.w3.org
    /2001/XMLSchema#string">Chest</urdf:hasBaseLinkName>
98        <urdf:hasEndLinkName rdf:datatype="http://www.w3.org
    /2001/XMLSchema#string">LHand</urdf:hasEndLinkName>
99    </owl:NamedIndividual>
100
101   <owl:NamedIndividual rdf:about="http://knowrob.org/kb/
    avatar_skeleton.owl#avatar_skeleton_arm_r">
102       <rdf:type rdf:resource="http://www.ease-crc.org/ont/
    SOMA.owl#Arm"/>
103       <urdf:hasBaseLinkName rdf:datatype="http://www.w3.org
    /2001/XMLSchema#string">Chest</urdf:hasBaseLinkName>
104       <urdf:hasEndLinkName rdf:datatype="http://www.w3.org
    /2001/XMLSchema#string">RHand</urdf:hasEndLinkName>
105   </owl:NamedIndividual>
106
107   <owl:NamedIndividual rdf:about="http://knowrob.org/kb/
    avatar_skeleton.owl#avatar_skeleton_r_hand_null">
108       <rdf:type rdf:resource="http://www.ease-crc.org/ont/
    SOMA.owl#Hand"/>
109       <urdf:hasURDFName>RHand</urdf:hasURDFName>
110       <dul:hasComponent rdf:resource="http://knowrob.org/kb/
    avatar_skeleton.owl#avatar_skeleton_r_index_null"/>
111       <dul:hasComponent rdf:resource="http://knowrob.org/kb/
    avatar_skeleton.owl#avatar_skeleton_r_middle_null"/>
112       <dul:hasComponent rdf:resource="http://knowrob.org/kb/
    avatar_skeleton.owl#avatar_skeleton_r_pinky_null"/>
113       <dul:hasComponent rdf:resource="http://knowrob.org/kb/
    avatar_skeleton.owl#avatar_skeleton_r_ring_null"/>
114       <dul:hasComponent rdf:resource="http://knowrob.org/kb/
    avatar_skeleton.owl#avatar_skeleton_r_thumb_null"/>
115   </owl:NamedIndividual>
116
117   <owl:NamedIndividual rdf:about="http://knowrob.org/kb/
    avatar_skeleton.owl#avatar_skeleton_r_index_null">
118       <rdf:type rdf:resource="http://www.ease-crc.org/ont/
    SOMA.owl#Finger"/>
119       <urdf:hasBaseLinkName rdf:datatype="http://www.w3.org
    /2001/XMLSchema#string">RIndex1</urdf:hasBaseLinkName>
```

```
120         <urdf:hasEndLinkName rdf:datatype="http://www.w3.org
    /2001/XMLSchema#string">RIndex3</urdf:hasEndLinkName>
121     </owl:NamedIndividual>
122
123
124     <owl:NamedIndividual rdf:about="http://knowrob.org/kb/
    avatar_skeleton.owl#avatar_skeleton_r_middle_null">
125         <rdf:type rdf:resource="http://www.ease-crc.org/ont/
    SOMA.owl#Finger"/>
126         <urdf:hasBaseLinkName rdf:datatype="http://www.w3.org
    /2001/XMLSchema#string">RMiddle1</urdf:hasBaseLinkName>
127         <urdf:hasEndLinkName rdf:datatype="http://www.w3.org
    /2001/XMLSchema#string">RMiddle3</urdf:hasEndLinkName>
128     </owl:NamedIndividual>
129
130     <owl:NamedIndividual rdf:about="http://knowrob.org/kb/
    avatar_skeleton.owl#avatar_skeleton_r_pinky_null">
131         <rdf:type rdf:resource="http://www.ease-crc.org/ont/
    SOMA.owl#Finger"/>
132         <urdf:hasBaseLinkName rdf:datatype="http://www.w3.org
    /2001/XMLSchema#string">RPinky1</urdf:hasBaseLinkName>
133         <urdf:hasEndLinkName rdf:datatype="http://www.w3.org
    /2001/XMLSchema#string">RPinky3</urdf:hasEndLinkName>
134     </owl:NamedIndividual>
135
136     <owl:NamedIndividual rdf:about="http://knowrob.org/kb/
    avatar_skeleton.owl#avatar_skeleton_r_ring_null">
137         <rdf:type rdf:resource="http://www.ease-crc.org/ont/
    SOMA.owl#Finger"/>
138         <urdf:hasBaseLinkName rdf:datatype="http://www.w3.org
    /2001/XMLSchema#string">RRing1</urdf:hasBaseLinkName>
139         <urdf:hasEndLinkName rdf:datatype="http://www.w3.org
    /2001/XMLSchema#string">RRing3</urdf:hasEndLinkName>
140     </owl:NamedIndividual>
141
142     <owl:NamedIndividual rdf:about="http://knowrob.org/kb/
    avatar_skeleton.owl#avatar_skeleton_r_thumb_null">
143         <rdf:type rdf:resource="http://www.ease-crc.org/ont/
    SOMA.owl#Finger"/>
144         <urdf:hasBaseLinkName rdf:datatype="http://www.w3.org
    /2001/XMLSchema#string">RThumb1</urdf:hasBaseLinkName>
145         <urdf:hasEndLinkName rdf:datatype="http://www.w3.org
    /2001/XMLSchema#string">RThumb3</urdf:hasEndLinkName>
146     </owl:NamedIndividual>
147
148     <owl:NamedIndividual rdf:about="http://knowrob.org/kb/
    avatar_skeleton.owl#avatar_skeleton_l_hand_null">
149         <rdf:type rdf:resource="http://www.ease-crc.org/ont/
    SOMA.owl#Hand"/>
```

```
150         <urdf:hasURDFName>LHand</urdf:hasURDFName>
151         <urdf:hasBaseLinkName rdf:datatype="http://www.w3.org
    /2001/XMLSchema#string">LHand</urdf:hasBaseLinkName>
152         <urdf:hasEndLinkName rdf:datatype="http://www.w3.org
    /2001/XMLSchema#string">LIndex1</urdf:hasEndLinkName>
153         <urdf:hasEndLinkName rdf:datatype="http://www.w3.org
    /2001/XMLSchema#string">LMiddle1</urdf:hasEndLinkName>
154         <urdf:hasEndLinkName rdf:datatype="http://www.w3.org
    /2001/XMLSchema#string">LPinky1</urdf:hasEndLinkName>
155         <urdf:hasEndLinkName rdf:datatype="http://www.w3.org
    /2001/XMLSchema#string">LRing1</urdf:hasEndLinkName>
156         <urdf:hasEndLinkName rdf:datatype="http://www.w3.org
    /2001/XMLSchema#string">LThumb1</urdf:hasEndLinkName>
157         <dul:hasComponent rdf:resource="http://knowrob.org/kb/
    avatar_skeleton.owl#avatar_skeleton_l_index_null"/>
158         <dul:hasComponent rdf:resource="http://knowrob.org/kb/
    avatar_skeleton.owl#avatar_skeleton_l_middle_null"/>
159         <dul:hasComponent rdf:resource="http://knowrob.org/kb/
    avatar_skeleton.owl#avatar_skeleton_l_pinky_null"/>
160         <dul:hasComponent rdf:resource="http://knowrob.org/kb/
    avatar_skeleton.owl#avatar_skeleton_l_ring_null"/>
161         <dul:hasComponent rdf:resource="http://knowrob.org/kb/
    avatar_skeleton.owl#avatar_skeleton_l_thumb_null"/>
162     </owl:NamedIndividual>
163
164
165     <owl:NamedIndividual rdf:about="http://knowrob.org/kb/
    avatar_skeleton.owl#avatar_skeleton_l_index_null">
166         <rdf:type rdf:resource="http://www.ease-crc.org/ont/
    SOMA.owl#Finger"/>
167         <urdf:hasBaseLinkName rdf:datatype="http://www.w3.org
    /2001/XMLSchema#string">LIndex1</urdf:hasBaseLinkName>
168         <urdf:hasEndLinkName rdf:datatype="http://www.w3.org
    /2001/XMLSchema#string">LIndex3</urdf:hasEndLinkName>
169     </owl:NamedIndividual>
170
171     <owl:NamedIndividual rdf:about="http://knowrob.org/kb/
    avatar_skeleton.owl#avatar_skeleton_l_middle_null">
172         <rdf:type rdf:resource="http://www.ease-crc.org/ont/
    SOMA.owl#Finger"/>
173         <urdf:hasBaseLinkName rdf:datatype="http://www.w3.org
    /2001/XMLSchema#string">LMiddle1</urdf:hasBaseLinkName>
174         <urdf:hasEndLinkName rdf:datatype="http://www.w3.org
    /2001/XMLSchema#string">LMiddle3</urdf:hasEndLinkName>
175     </owl:NamedIndividual>
176
177     <owl:NamedIndividual rdf:about="http://knowrob.org/kb/
    avatar_skeleton.owl#avatar_skeleton_l_pinky_null">
```

```
178         <rdf:type rdf:resource="http://www.ease-crc.org/ont/
    SOMA.owl#Finger"/>
179         <urdf:hasBaseLinkName rdf:datatype="http://www.w3.org
    /2001/XMLSchema#string">LPinky1</urdf:hasBaseLinkName>
180         <urdf:hasEndLinkName rdf:datatype="http://www.w3.org
    /2001/XMLSchema#string">LPinky3</urdf:hasEndLinkName>
181     </owl:NamedIndividual>
182
183     <owl:NamedIndividual rdf:about="http://knowrob.org/kb/
    avatar_skeleton.owl#avatar_skeleton_l_ring_null">
184         <rdf:type rdf:resource="http://www.ease-crc.org/ont/
    SOMA.owl#Finger"/>
185         <urdf:hasBaseLinkName rdf:datatype="http://www.w3.org
    /2001/XMLSchema#string">LRing1</urdf:hasBaseLinkName>
186         <urdf:hasEndLinkName rdf:datatype="http://www.w3.org
    /2001/XMLSchema#string">LRing3</urdf:hasEndLinkName>
187     </owl:NamedIndividual>
188
189     <owl:NamedIndividual rdf:about="http://knowrob.org/kb/
    avatar_skeleton.owl#avatar_skeleton_l_thumb_null">
190         <rdf:type rdf:resource="http://www.ease-crc.org/ont/
    SOMA.owl#Finger"/>
191         <urdf:hasBaseLinkName rdf:datatype="http://www.w3.org
    /2001/XMLSchema#string">LThumb1</urdf:hasBaseLinkName>
192         <urdf:hasEndLinkName rdf:datatype="http://www.w3.org
    /2001/XMLSchema#string">LThumb3</urdf:hasEndLinkName>
193     </owl:NamedIndividual>
194
195     <!-- Individuals for kinematics information -->
196     <owl:NamedIndividual rdf:about="http://knowrob.org/kb/
    Kitchen-clash-agent.owl#InformationObject_KFLDFLKH">
197         <rdf:type rdf:resource="http://www.ease-crc.org/ont/
    SOMA.owl#KinoDynamicData"/>
198         <dul:isAbout rdf:resource="http://knowrob.org/kb/
    avatar_skeleton.owl#avatar_skeleton_1"/>
199     </owl:NamedIndividual>
200
201     <owl:NamedIndividual rdf:about="http://knowrob.org/kb/
    Kitchen-clash-agent.owl#InformationRealization_1RTUIRGH">
202         <rdf:type rdf:resource="http://www.
    ontologydesignpatterns.org/ont/dul/IOLite.owl#
    DigitalResource"/>
203         <dul:realizes rdf:resource="http://knowrob.org/kb/
    Kitchen-clash-agent.owl#InformationObject_KFLDFLKH"/>
204         <soma:hasPersistentIdentifier rdf:datatype="http://www
    .w3.org/2001/XMLSchema#string">ElanHumanSkeleton</
    soma:hasPersistentIdentifier>
205         <soma:hasDataFormat rdf:datatype="http://www.w3.org
    /2001/XMLSchema#string">URDF</soma:hasDataFormat>
```

```
    </owl:NamedIndividual>

</rdf:RDF>
```

Listing 8.1. Agent owl file

8.2 Environment owl file

```
<?xml version="1.0"?>
<rdf:RDF xmlns="http://knowrob.org/kb/elan-map.owl#"
         xml:base="http://knowrob.org/kb/elan-map.owl"
         xmlns:srdl2="http://knowrob.org/kb/srdl2.owl#"
         xmlns:srdl2-cap="http://knowrob.org/kb/srdl2-cap.owl#"
         xmlns:owl="http://www.w3.org/2002/07/owl#"
         xmlns:owl2xml="http://www.w3.org/2006/12/owl2-xml#"
         xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
         xmlns:urdf="http://knowrob.org/kb/urdf.owl#"
         xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
         xmlns:qudt-unit="http://qudt.org/vocab/unit#"
         xmlns:iai-Player="http://knowrob.org/kb/elan-map.owl#"
         xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:soma="http://www.ease-crc.org/ont/SOMA.owl#"
         xmlns:xml="http://www.w3.org/XML/1998/namespace"
         xmlns:srdl2-comp="http://knowrob.org/kb/srdl2-comp.owl#"
         xmlns:knowrob="http://knowrob.org/kb/knowrob.owl#"
         xmlns:dul="http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#">
    <owl:Ontology rdf:about="http://knowrob.org/kb/elan-map.owl">
        <owl:imports rdf:resource="http://www.ease-crc.org/ont/SOMA.owl"/>
    </owl:Ontology>

    <!--
    ///////////////
    //
    // Classes
    //
    ///////////////
     -->



    <!--
```

```
    //////////////
    //
    // Individuals
    //
    //////////////
     -->


    <!-- http://knowrob.org/kb/elan-map.owl#map_1 -->

    <owl:NamedIndividual rdf:about="http://knowrob.org/kb/elan-map.owl#map_1">
        <rdf:type rdf:resource="http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#PhysicalObject" />
        <urdf:hasBaseLinkName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">floor</urdf:hasBaseLinkName>
        <dul:hasComponent rdf:resource="http://knowrob.org/kb/elan-map.owl#table1_1"/>
        <dul:hasComponent rdf:resource="http://knowrob.org/kb/elan-map.owl#table2_1"/>
        <urdf:hasEndLinkName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">table1</urdf:hasEndLinkName>
         <urdf:hasEndLinkName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">table2</urdf:hasEndLinkName>
    </owl:NamedIndividual>

    <owl:NamedIndividual rdf:about="http://knowrob.org/kb/elan-map.owl#table1_1">
        <rdf:type rdf:resource="http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#PhysicalObject" />
        <urdf:hasBaseLinkName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">table1</urdf:hasBaseLinkName>
    </owl:NamedIndividual>

    <owl:NamedIndividual rdf:about="http://knowrob.org/kb/elan-map.owl#table2_1">
        <rdf:type rdf:resource="http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#PhysicalObject" />
        <urdf:hasBaseLinkName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">table2</urdf:hasBaseLinkName>
    </owl:NamedIndividual>

  <!-- Individuals for kinematics information -->
  <owl:NamedIndividual rdf:about="http://knowrob.org/kb/Kitchen-clash-agent.owl#InformationObject_KFLDFGHj">
```

```
65          <rdf:type rdf:resource="http://www.ease-crc.org/ont/
    SOMA.owl#KinoDynamicData"/>
66          <dul:isAbout rdf:resource="http://knowrob.org/kb/elan-
    map.owl#map_1"/>
67      </owl:NamedIndividual>
68
69      <owl:NamedIndividual rdf:about="http://knowrob.org/kb/
    Kitchen-clash-agent.owl#InformationRealization_1RTUPDFG">
70          <rdf:type rdf:resource="http://www.
    ontologydesignpatterns.org/ont/dul/IOLite.owl#
    DigitalResource"/>
71          <dul:realizes rdf:resource="http://knowrob.org/kb/
    Kitchen-clash-agent.owl#InformationObject_KFLDFGHj"/>
72          <soma:hasPersistentIdentifier rdf:datatype="http://www
    .w3.org/2001/XMLSchema#string">ElanMap</
    soma:hasPersistentIdentifier>
73          <soma:hasDataFormat rdf:datatype="http://www.w3.org
    /2001/XMLSchema#string">URDF</soma:hasDataFormat>
74      </owl:NamedIndividual>
75
76
77  </rdf:RDF>
```

Listing 8.2. Environment owl file

References

[1] Claudio Masolo, Stefano Borgo, Aldo Gangemi, Nicola Guarino, and Alessandro Oltramari. WonderWeb deliverable D18 ontology library (final). Technical report, IST Project 2001-33052 WonderWeb: Ontology Infrastructure for the Semantic Web, 2003.