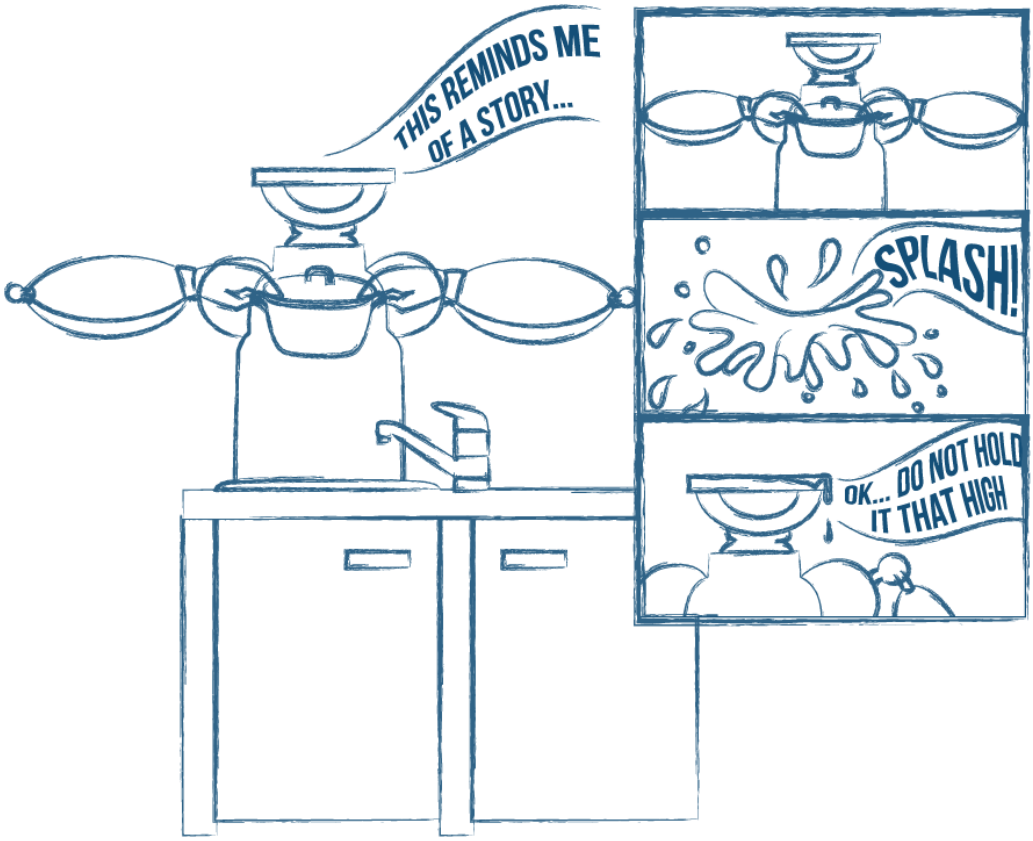
NEEM Handbook

The EASE Researchers

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



Contents

1

Introduction

This document, referred to as the "NEEM Handbook" hereafter, describes the EASE system for episodic memories of everyday activities: what they are, how they are represented and acquired. The NEEM Handbook will be updated along the progress in the CRC EASE . It is thought to provide EASE researchers with compact but still comprehensive information about what information is contained in NEEMs , and how it is represented.

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". The memory mechanism that allows us to recall these very detailed pieces of information from abstract descriptions is our episodic memory. Episodic memory is powerful because it allows us to remember special experiences we had. It can also serve as a "repository" from which we learn general knowledge.

EASE integrates episodic memories deeply into the knowledge acquisition, representation, and processing system. Whenever an agent performs, observes, prospects, and reads about an activity, it creates an episodic memory. An episodic memory is best understood as a video that the agent makes of the ongoing activity coupled with a very detailed story about the actions, motions, their purposes, effects, the behavior they generate, the images that are captured, etc.

We term 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 is a detailed, low-level, time-indexed recording of a certain episode. The experience contains records of poses, percepts, control signals, etc. These can be used to replay an episode in detail. NEEM experiences are linked to NEEM narratives, which are stories that provide more abstract, symbolic descriptions of what is happening in an episode. These narratives contain information regarding the tasks, the context, intended goals, observed effects, etc.

TODO: Are we providing a definition of term episode?

1 Scope

The broad scope of our work is everyday object manipulation tasks in autonomous robot control, and in particular the motion and force characteristics of objects that interact with each other. The research question driving us is whether a single general control program can be written that can generate adequate behavior in many different contexts: for different tasks, objects, and environments.

One of the challenges is that, using such a general plan, the agent needs to fill the knowledge gaps between abstract instructions included in the plan and the realization of context specific behavior. That is, for example, the many ways of how humans perform a pouring task depending on the source from which is poured, the destination, and the substance that is to be poured. Another challenge is that object manipulation tasks may fail if the agent does not perform the motions competently and well. This is caused by the agent choosing inappropriate parametrization of its control-level functions.

The employment of a general plan thus requires an abstract task and object model, and a mechanism to apply this abstract knowledge in situational context. To achieve this, an agent needs to be equipped with the necessary common-sense and intuitive physics knowledge, which is what SOMA**TODO**: The acronym is no where defined attempts.

TODO: write about competency questions

2 Outline

First, this document provides an overview about version 0.1 of the NEEM Handbook, and outlines how we, the EASE researchers, envision the evolution of the NEEM Handbook throughout the CRC EASE . The next two sections are dedicated to the two distinct parts of NEEMs : experience and narrative. For both, we describe data format in detail, providing a compact tabular view on data structures and meaning. Finally, we provide a concrete example for a table setting activity, a highly relevant example for research in EASE .

2

Requirements

A good idea would be to have here a table with all requirements which we want to achieve with NEEMs. I assume it will be a nice overview for the reader(partner), if they will open the document, they can see right away what feature is supported by the current NEEM version, which are planned for the next version and which will be added in the late future. We can create the requirement table based on the EASE proposal.**TODO**: Seba: Good idea but I would wait until the whole document is finished.

3

Representation

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 purpose of this chapter is not to provide a full overview about the ontological modelling underlying NEEMs , but rather to concentrate on aspects that are directly relevant when a NEEM is created. These are the representations that are used in the NEEM-narrative (Section 3) and NEEM-experience (Section 4). In addition, NEEMs are situated in an environment, and acquired by some agent performing a task by interacting with its environment. However, the agent and the environment may be involved in many different NEEMs such that their representation is seperated from the representation of NEEMs (Section **??**).

3 NEEM-Narrative

The narrative part of NEEMs can be seen as a story of what has happened in terms of events that occurred, goals that were followed, and objects that were involved. The NEEM-narrative representation contains two basic elements: classified entities such as events and objects, and relationships between them. The events being particularly important as their time interval is used for time-indexed data access. Data represented in the NEEM-experience may further be correlated to the classes and structures in the NEEM-narrative in order to train models that can predict the NEEM-narrative given the NEEM-experience and possibly some contextual parameter.

The NEEM-narrative 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. These extensions are part of an ontology that we have called SOMA [1]. A NEEM-narrative is made of several patterns defined either in DUL or in SOMA .

While it is possible to create the representations listed in this chapter 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 section that highlight how the knowledge base can be used to interact with NEEMs .

3.1 Taxonomic classification

TODO: Seba: Somehow here or before this seciton is some introduction text missing. It just starts randomly with a taxonomy section and jumps right away to Actions. This Action section does not mention any action taxonmoy but rather a task taxonomy

Here in the following section, we will discuss taxonomy for the concepts used to describe NEEM-narrative part. A taxonomy provides the hierarchical relationships among various concepts and defines the terminology for each concept. One way to classify an entity in ontology is through taxonomies, which concentrates on *what there actually is?* as compared to conceptual classification which is more about *how an entity is interpreted?*.

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

Actions

An action can be defined as an event where at least one agent participates, such that this agent performs a dedicated task, defined by a plan or workflow, which it executes through the Action. Here workflow refers to a plan that defines role(s), task(s), and specific structure of tasks that needs to be executed.

Task

Task is an EventType that classifies an Action to be executed, where an event type describes how an event should be interpreted, executed, expected, seen, etc[3]. For example cleaning a table is a task that can be executed by performing certain actions. However, sometimes it is not easy to say what an event is all about taxonomically, in conceptual classification section 3.5, we explain how the task taxonomy used to classify actions.

TODO: Seba: Where is the real difference between Task and Action ? Can I say also the action grasping executes a task grasping ? Do I only represent high level ”actions” such as setting up a table as tasks ? If yes, where is the end of the abstraction ? At the motion level ?

task taxonomy

A task can be further classified into three main categories, physical task, mental task and communication task. A task which requires to execute an action through which an agent has to manipulate representations stored in its own cognition can be categorized as mental task, dreaming, imagining, prospecting, reasoning, retrospecting falls under mental task. Where as physical task requires an agent to perform certain physical activities such as actuating, constructing, modifying physical object, looking at, placing, navigating and perceiving. At last, communication task in which two more more agents shares information. This is used to classify special kind of events that have participants as agents and social objects. The means of information exchange is physical however the scope of interest here is to identify which agent communicates and what kind of information it has? In the appendix section, we further describe all three tasks into subcategories.

Motions

An event type which deals with movements of an agent and motions it makes during task execution.**TODO**: Seba: In the introduction there is mentioned there are only 2 basic elements: classified entities (events and objects) and the relationships. A simple question I have to ask, are tasks also event types? Maybe a real definition

[3] `http://www.ontologydesignpatterns.org/ont/dul/DUL.owl`

of event types would be really useful. Motion taxonomy includes concepts such as 'body movement' which concentrates on the movements of agents body parts. 'Directed motion' which involves a destination and directed path for an agent to follow, however, an 'un-directed motion' is opposite to directed where agent does not have any particular destination but it is important to know that the agent has moved and motion has occurred. And at last 'fluid flow' is the process by which fluid moves or has moved from one location to other. All four concepts are further sub-categorized in appendix section.

Objects

According to DUL [4] upper level ontology, an object participates in an event during its lifetime and has its own spatial location. describes objects into several subcategories which includes agent, digital object, feature, physical object, social object and transient. It further includes taxonomy of dispositions which describes the potential way an object can be used and design taxonomy which considers functional, structural and aesthetic aspect of object design.**TODO**: Seba: An example with using those terms would be nice.

Roles

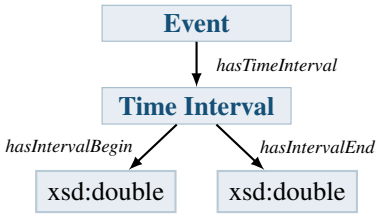
A social concept which is basically used for an object classification. An object can have different roles when it participates in the event during its lifetime. Role taxonomy includes agent role, answer, causal process role, communication topic, existing object role, explanation, instrument, linguistic function, location, locatum role, path role, patient, question, relatum role, and software type. Further role sub-categories will be defined under appendix section.**TODO**: Seba: An example with using those terms would be nice.

3.2 Occurrences

The basic building blocks of NEEMs are the events that occur during the course of an activity**TODO**: Seba: Is activity a general term for experiments or is it also a definition form the ontology?. An event is defined as *any physical, social, or mental process, event, or state*. Events have an associated time interval that determines the time at which the event occurs. Time data is represented as unix timestamps using XSD types.

[4] `http://www.ontologydesignpatterns.org/ont/dul/DUL.owl`

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

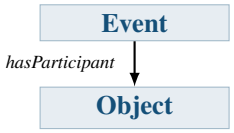


Expression	Meaning
occurs('EV')	*EV* is an occurence
occurs('EV') during [10,14]	*EV* occurs between the times 10 and 14
occurs('EV') since 10	*EV* occurs since the time 10
occurs('EV0') since 'EV1'	*EV0* and *EV1* begin at the same time

3.3 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.

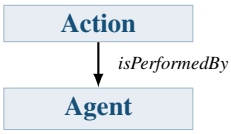
Intent	To represent participation of an object in an event.
Competency Questions	*Which objects do participate in this event? In which events does this object participate in?*
Defined in	DUL.owl



Expression	Meaning
has_participant('EV','O')	*O* is involved in event *EV*
has_participant('EV',X)	what are the entities involved in *EV*?
has_participant(X,'O')	what are the events involving *O*?

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 an 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 performed by multiple agents.

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



Expression	Meaning
is_performed_by('ACT','AG')	*ACT* is performed by *AG*
is_performed_by('ACT',X)	what are the agents that performed *ACT*?
is_performed_by(X,'AG')	which actions are performed by *AG*?

3.4 Composition

Because an Entity **TODO**: Seba: What is an Entity ? Is this an object? often has relevant internal structure, it is necessary to represent relations between it and the other entities that make up its composition. DUL provides two such structuring relations: *hasConstituent* and *hasPart*. We will focus on *hasPart* in this section, but we will briefly discuss the difference between the two relations as well. **TODO**: Seba: Why only hasPart ?

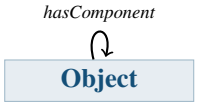
The *hasConstituent* relation is intended to connect entities from different ontological layers or levels of abstraction of looking at the world. A material is a constituent of the object it makes up; individual persons are constituents of a corporation. While these constituency relations could often be in colloquial language described as parthood, DUL reserves parthood for more restrictive connections between entities closer to each other in their ontological characterization. For example, a corporation is a *SocialObject* and as such can only have *SocialObjects* as parts, whereas human persons are *PhysicalAgents*, disjoint from *SocialObjects*.

The *hasPart* relation is transitive and defined to have domain and range *Entity*, that is, in principle it could link entities from any concepts together. However, various concepts restrict what can be parts of their instances as exemplified above; as another example, *PhysicalObjects* can only have other *PhysicalObjects* as parts. In SOMA, we define more specialized subproperties of *hasPart* to deal with parts of *Events* vs. parts of *Objects*. The relevant properties are *hasPhase* and *hasComponent* respectively; both are subproperties of *hasPart*.

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('A','B')	*B* is a phase of *A*

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('A','B')	*B* is a proper part of *A*
has_component('A',X)	What are the components of *A*?
has_component(X,'A')	What are the entities *A* is component of?

TODO: Seba: For me this section makes sense. However, I have the feeling that compared to the previous sections, this section requires maybe more knowledge in ontologies to be understand. Maybe we should give it to someone how does not have any knowledge about ontologies.

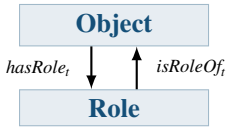
3.5 Conceptual Classification

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. For this purpose, a comprehensive collection of concepts is used to classify objects and events from a conceptual viewpoint (**todo: insert link to list of all concepts**).

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.

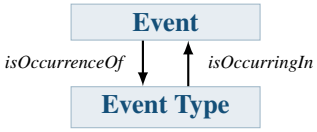
Intent	To represents objects and the roles they play.
Competency Questions	*What role does this object play? Which objects do play that role?*
Defined in	DUL.owl



Expression	Meaning
has_role('A','B')	*B* is a role of *A*
has_role('A','B') during 'C'	*B* is a role of *A* during the occurrence of *C*
has_role('A','B') during [t_0,t_1]	*B* is a role of *A* during the timepoints t_0 and t_1

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. **TODO**: Seba: How the conceptualization of an event looks in the end? Do we have only one classification or can it have multiple classifications if the goal is not clear?

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

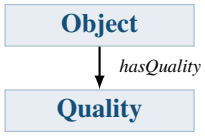


Expression	Meaning
is_occurring_in('C0','Evt0')	*Evt0* is an occurrence that is classified by *C0*
is_executed_in('Tsk0','Act0')	*Act0* is an action that executes the task *Tsk0*

3.6 Object Properties

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 being slippery. 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 focusses 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.

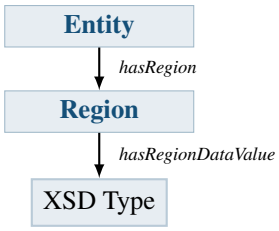
Intent	To represent the qualities of an object.
Competency Questions	*What qualities does this object have? That are the objects with this quality?*
Defined in	DUL.owl



Expression	Meaning
has_quality('A','B')	*B* is a quality of *A*
has_quality('A',X)	What are the qualities of *A*?
has_quality(X,'A')	What entity is the host of *A*?

Each object property has one value at a time. The value of an object property is called *region*. The value itself is an element, or a sub-region in some dimensional space such as *time interval* or *space region*. A region may be a finite set of discrete labels, allowing for "qualitative" descriptions, but more often a region is some dimensional space allowing "quantitative" descriptions. A Region may contain a single point, in cases where the value of a property is known precisely. Note that the domain of the relation *hasRegion* is not *Quality* but *Entity*. This is to allow assigning regions to entities without explicating the quality as a concept (quality-as-relation). **TODO**: Seba: maybe some more example would be nice. Or refering to the example with the slippery floor.

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('A','B')	*B* is a region of *A*
has_region('A',X)	What is the region of *A*?
has_data_value('A','B')	*B* is a data value of *A*

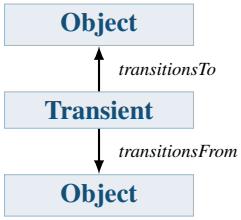
3.7 Object Transformation

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 3.6. 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

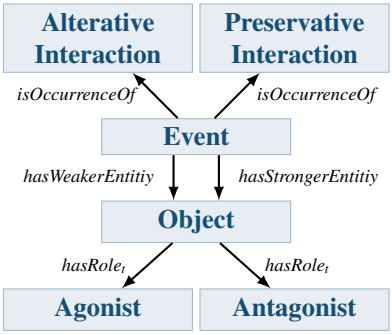


Expression	Meaning
transitionsFrom('A','B')	By entering some process of change, object 'B' becomes Transient object 'A'.
transitionsTo('A','B')	By completing some process of change, transient 'A' becomes object 'B'.
transitionsBack('A','B')	Object 'B' entered some process of change during which its ontological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

3.8 Force Interaction

todo: Daniel write this todo: preservative vs. alterative interaction todo: how to represent who is stronger? can measured forces be included in the NEEM? todo: isProcessTypeOf relation is not nice Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

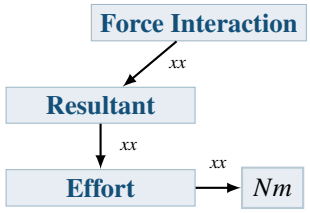
Intent	To represent force dynamical event characteristics.
Competency Questions	*What entity is the agonist in this event? What are the events where this entity is an antagonist? Which entity is stronger in this event? What is the force dynamical tendency in this event?*
Defined in	SOMA.owl



Expression	Meaning
??	??

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Intent	To quantify force dynamical event characteristics.
Competency Questions	*What is the effort of the agonist in this event?*
Defined in	SOMA.owl



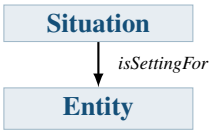
Expression	Meaning
??	??

3.9 Episodes

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 perfomed, 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 perfomed. 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*. **TODO**: Seba: Do I have to use is_setting_for each action and object created in the episode ?

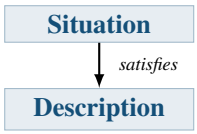
Intent	To represent that entites are included in a situation.
Competency Questions	*What are the entities that are relevant for this situation? What are the situations where this entity is relevant?*
Defined in	DUL.owl



Expression	Meaning
is_setting_for('Episode1', 'Obj1')	*Episode1* is a setting for *Obj1*
is_setting_for('Episode1', X)	What is *Episode1* setting of?
is_setting_for(X, 'Evt1')	What are the episodes that include *Evt1*?

Episodes refer to concrete occurences with actual objects that are involved, and actual events that occur. The conceptualization **TODO**: Seba: can you provide a definition ? Before readining this section I assumed that a synomy for classificationof an episode is an abstraction that refers to concepts instead. 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 **TODO**: Seba: Will be description somewhere defined ?). 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('Episode1', 'Descr1')	*Episode1* is consistent with *Descr1*
satisfies('Episode1', X)	What are conceptualization of *Episode1*?
satisfies(X, 'Descr1')	What are the episodes that are conceptualized by *Descr1*?

4 NEEM-Experience

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 [5] 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 0.1 .

4.1 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. 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?

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

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.

The data is stored in the DB collection as array. Each array holding pose estimates for distinct frames at the same time stamp. For indexed search, it is important that each array member has the same time stamp.

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`	–
`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

Fig. 1. The pose data structure in the EASE system.

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.

Symbol Abstraction

Module	`knowrob_objects`[6]
Symbols	`object pose`
Implementation	`Prolog`

belief_at(Object, [Parent,Child,Translation,Rotation], Instant) This temporal predicate computes a Prolog-based pose representation (2nd argument) for a named object (1st argument). Time is supplied to the predicate as time stamp (3rd argument).

Module	`comp_spatial`[7]
Symbols	`spatial relations`
Implementation	`Prolog`

comp_inCenterOf(Inner,Outer,Interval) Check if *Inner* is in the center of *Outer*. Currently does not take the orientation into account, only the position and dimension. Computes the *inCenterOf* relation.

comp_inFrontOf(Front,Back,Interval) Check if *Front* is in front of *Back*. Currently does not take the orientation into account, only the position and dimension. Computes the *inFrontOf-Generally* relation.

comp_inContGeneric(Inner,Outer,Interval) True iff the object *Inner* is inside of the bounding box of container *Outer* during the specified interval. Computes the *in-ContGeneric* relation.

comp_onPhysical(Top,Bottom,Interval) Check if *Top* is in the area of and above *Bottom*. Computes the *on-Physical* relation

comp_above(Top,Bottom,Interval) Check if *Top* is in the area of and above *Bottom*. Computes the *above-Generally* relation.

comp_below(Bottom,Top,Interval) Check if *Top* is in the area of and above *Bottom*. Computes the *below-Generally* relation.

comp_toTheLeftOf(Left,Right,Interval) Check if *Left* is to the left of *Right*. Currently does not take the orientation into account, only the position and dimension. Computes the *toTheLeftOf* relation.

comp_toTheRightOf(Right,Left,Interval) Check if *Left* is to the left of *Right*. Currently does not take the orientation into account, only the position and dimension. Computes the *toTheRightOf* relation.

4

Acquisition

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.

Each section will contain an example NEEM to provide insights how the representation, described in Chapter 3, is utilized to represent performed activities by robots or by humans. In addition, each example NEEM is available on the NEEM-hub for downloading.

5 Narrative Enabled Episodic Memories for Robotic Agents

This section focuses on describing how to generate NEEMs from experiments performed by the robot.

5.1 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 installed it on your machine. CRAM is a cognitive-enabled planning framework which allows to design high-level plan for robots.**TODO**: @Seba: Add more detailed description about CRAM 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.

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

In addition to having a valid CRAM plan, you will need the following software components to be installed:

- A MongoDB server with at least version 3.4.10[9]
- KnowRob [10]
- SOMA ontology[11]
- CRAM ontology[12]

5.2 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 **??**.

Before you can begin to record your own NEEMs , you need to include the ”cram-cloud-logger” package into your CRAM plan. After you included the package, you need to enable the logging via:

```
(setf ccl::*is-logging-enabled* t)
```

Listing 4.1. Enabling NEEM Logging in a CRAM plan

The only things left to do start the logging before the plan execution and after the execution to finishing it. It can look like the following:

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

Listing 4.2. Steps to Record an Episode for a CRAM Plan

The generate 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 4.3. How to Start KnowRob

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

5.3 Data

Per default your generated NEEM will be stored under ”~/knowrob-memory/<timestamp >”. Your NEEM will consist of at least 5 folders - *annotations*, *inferred*, *roslog*, *ros_tf* and *triples*. In the following, we will give an overview which information is contained in those folders:

TODO: @Seba add proper descriptions

annotations description

inferred description

ros_tf description

triples description

5.4 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, SOMA focuses on the support to record table setting/cleaning-up experiments. If you want for instance create NEEMs for e.g. autonomous cars, you will need to extend the SOMA ontology and the recording mechanism with your required 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 **??**. Unknown tasks 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 4.4. 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>. With this step you added successfully the support of the new action to 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 ontology **TODO**: set reference. 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/EASE-OBJ.owl#Bowl'")
    (setf (gethash "CUP" lookup-table) "'http://www.ease-crc.
    org/ont/EASE-OBJ.owl#Cup'")
    (setf (gethash "<CRAM-OBJECT-TYPE>" lookup-table) "'<
    SOMA-ONTOLOGY-ENTITY-URL>'")
 lookup-table))
```

Listing 4.5. 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-URL>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**TODO**: set reference. 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 4.6. 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.".

Adding New Rostopic

Per default, we log the rostopics tf and tf_static. If you need to log additional topics, open "memory.pl" in the "knowrob_memory" package from KnowRob and include your topic in the "mem_episode_start(Episode)" function. After the NEEM generation, the data will be stored in the created NEEM folder under the file "roslog/¡rostopic¿.bson".

Adding New Parameters

Unknown parameters will be logged as comment attached to the corresponding *PlanExecution* instance. The comment statement "Unknown Parameter: PARAMETER-NAME -####- PARAMETER-VALUE/¿" The current parameter types are represented

TODO: @Ontology group: Please make sure that is available in the ontology

1. Integer/Floats
2. Posen
3. Spatial Relations
4. Link to entities of other ontologies such as http://knowrob.org/kb/PR2.owl#pr2_right_arm

Before you want to model your parameter what data type your parameter is. If it is a complex object, you need to consider how you want to to represent it an the ontology. For simpler representaion such has a discrete domain representation, you might represented as the domain values as *Region* and add the model the parameter as a subconcept of *Parameter*. More information about the concepts *Region* and *Parameter* can be found in **TODO**: reference to Region and parameter.

5.5 Adding New Reasoning Tasks

TODO: @Ontology group: How to log the result of the reasoning query ?

6 Next steps

After you have generate your NEEM , you can use the tool **TODO**: Add neem2narrative to generate an cvs file for your NEEM . Keep in mind that the csv is a abstraction of NEEM-narrative and can be used to make data-mining on explicit knowledge. For more sophisticated analysis, you will need to use KnowRob . We use this general analysis to identify bottlenecks in our plan execution. We also showed that with a collection of NEEMs we are able to improve the robot's performance. **TODO**: Af The tools for the feature extraction can be found here **TODO**: Add link Now that you we encourage you to generate your NEEMs and share them via our NEEM-hub .

6.1 Example

TODO: Add belief state example **TODO**: What about reference to semantic map ? In this chapter we will show a generated NEEM based on the version 0.1 . You can download the log used in this example here. **TODO**: Provide link for downloading real log file We will not go into the details about the NEEM-experience meaning the tf since we just stored the exact tf message in the database. Therefore it is not required to have detail explanation since the idea of tf is already understood.

This log file represents a experiment where the PR2 tired to grasp five objects which were located on a kitchen counter and bring each object to our table and place them there. We defined grapsing + placing as a transporting. This experiment was performed in projection meaning in simulation. CRAM is not used using a belief state during execution in projection, therefore we will not show how a belief state is represented. The object which were precived or grapsed are represent as strings.

Figure 2 show how a transporting task for a cup is represented. To summarize what happend during the transporitng task in the experiment, The task was performed not successfully in projection. The failure was that the cup was unfetchable for the PR2 . The PR2 used the left arm to grasp the cup from a connected space region and transported to object to a specific pose in the map. **TODO**: How we can differ between those two goal Locations ? This task has also one sub action. Given the task the agent inferred that to achieve this task, it has to perform a PickingUpAnObject task. You can also read from the log that the PR2 2 tried already an transporting task on another object and that this task is also not the last transporitng task.

```
Individual: TRANSPORTING_KVRJlsOG
Facts:
goalLocation knowrob;ConnectedSpaceRegion_SgfniCWn
goalLocation knowrob;Pose_eLrKlpcE
arm pr2;pr2_left_arm
endTime knowrob;timepoint_1526640320.499945
failure CRAM-COMMON-FAILURES:OBJECT-UNFETCHABLE
nextAction knowrob;TRANSPORTING_ysHEUIwW
objectType "CUP"\todo{add figure to connected space region
}
performedInProjection true
previousAction knowrob;TRANSPORTING_FkRlsFGa
startTime knowrob;timepoint_1526640316.535484
subAction knowrob;PickingUpAnObject_iLTYaxBE
taskContext TableSetting
taskSuccess false
```

Fig. 2. Example for a Transporting Task

The next transporting represented in . . . 3 represents a transporting task of a bowl object. In contrast to the previous task, this task was succuessfully performed. Again the PR2 used only the left arm. The PR2 grasped the object from a connected space region which is an kitchen counter 4. It placed the object in the specific pose in the map which is displayed in 5.

```
Individual: TRANSPORTING_ysHEUIwW
Facts:
  goalLocation knowrob;ConnectedSpaceRegion_YOXaigqs
  goalLocation knowrob;Pose_GIcRIGdP
  subAction knowrob;PickingUpAnObject_CTGdupxa
    subAction knowrob;PuttingDownAnObject_uBRtvLcg
    arm pr2;pr2_right_arm
    endTime knowrob;timepoint_1526640332.733415
    nextAction knowrob;TRANSPORTING_HRgQoeEw
    objectType rdf:resource="BOWL"
    performedInProjection true
    previousAction knowrob;TRANSPORTING_KVRJlsOG
    startTime rdf:resource="&knowrob;timepoint_1526640320
  .561752
  taskContext "TableSetting"
  taskSuccess true
```

Fig. 3. Example for a second Transporting Task

```
Individual: ConnectedSpaceRegion_YOXaigqs
Facts:
  onPhysical knowrob:iai_kitchen_sink_area_counter_top
```

Fig. 4. Example for Connected Space Region

```
Individual: Pose_GIcRIGdP
  Facts:
  quaternion 0.0 0.0 1.0 0.0
  translation -0.7599999904632568 1.190000057220459
  0.9300000071525574
```

Fig. 5. Example for a Pose

7 VR NEEMs

5

NEEM-Hub

8 Publishing

In this chapter we have to describe, how the NEEM narrative and NEEM experience should be represented e.g. using OWL and MongoDB and how the partners can upload those file to openEASE.

9 Maintaining

6

Application

7

Future Work

12 NEEM-Experience

1. Logging Images
2. CostMaps

13 NEEM-Narrative

1. Logging Reasoning Tasks

A Taxonomy

Here we will discuss in detail taxonomies for various concepts.

A.1 Task taxonomy

Communication Task

Communication Task deals with communication related tasks.

Mental Task

Mental Task deals with mental related tasks.

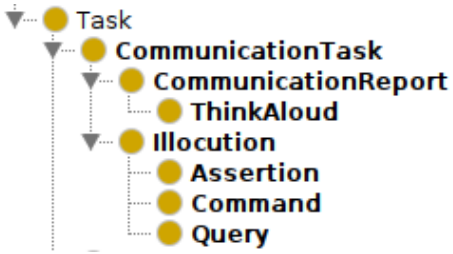


Fig. 6. Communication task taxonomy.

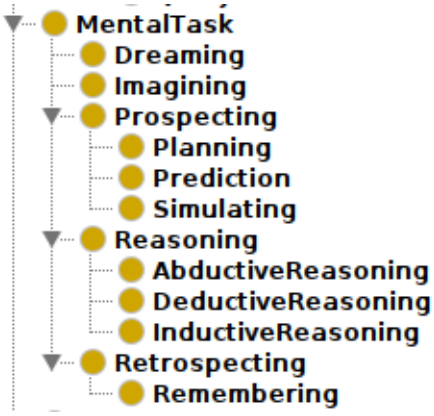


Fig. 7. Mental task taxonomy.

Physical Task

Physical Task deals with Physical related tasks.

A.2 Motion taxonomy

A.3 Object taxonomy

A.4 Role taxonomy

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.

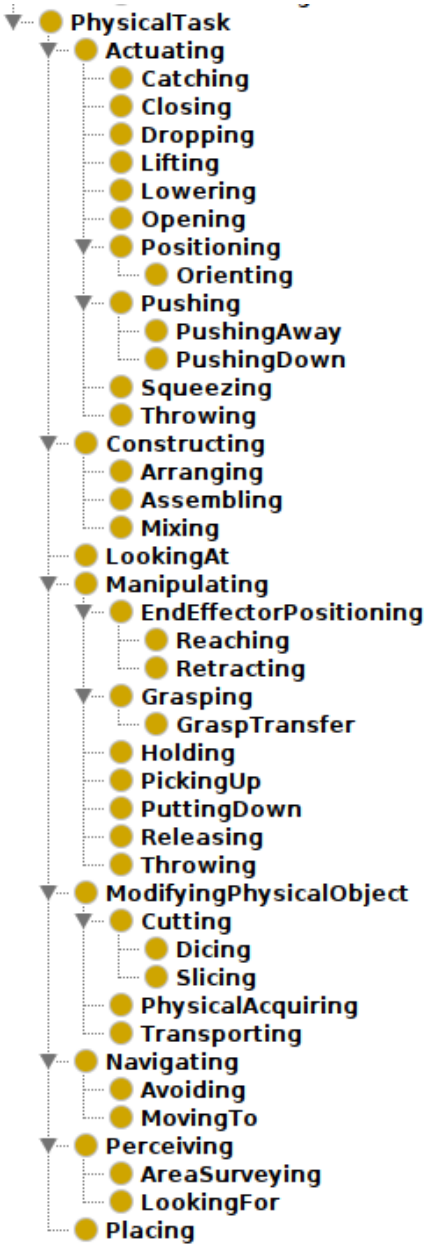


Fig. 8. Physical task taxonomy.