Modeling Knowledge with the Concept Hierarchy for Household Action Recognition and Task Representation*

Andrei Costinescu[1], Luis Figueredo[2] and Darius Burschka[1]
andrei.costinescu@tum.de, figueredo@ieee.org, burschka@tum.de

***Abstract*— The Concept Hierarchy is a knowledge modeling framework for representing geometric, semantic, and dynamic scene elements in household environments. It stores necessary information for autonomous systems to plan and reason in indoor environments and serves typical applications thereof: environment modeling, action modeling and recognition, and task planning. Its hierarchical structure supports generalization and knowledge transfer to new entities thanks to the non-monotonic modeling of concept properties. We define tasks, actions, skills, and affordances that enable human-understandable and -explainable household applications. We validate the framework for action and skill recognition in human manipulation and for verifying a correct task execution in the environment.**

I. Introduction

Making sense of the world in which we humans live is a difficult problem, more so for a robotic system, see Figure 1. Acting intelligently and interacting with objects for purposeful changes in environments requires knowledge of objects, agents, actions, and skills, as well as algorithms that use this knowledge and consider the task goal, the abilities of the performing agent(s), and the environment in which the task is to be performed. Ontologies [1], [2] are methods to represent semantic knowledge of the world. A well-known robotics ontology is knowrob [3], which includes knowledge about executing skills. Task planning also needs knowledge of the changes in the world to create solution plans. In the STRIPS [4] model, actions change the state of entities. They have preconditions to be satisfied and effects on the entities on which they are performed.

We consider it important to model the verification of skill execution as well in a knowledge base, which is one of the applications of our proposed ontology: the Concept Hierarchy (**CH**). Furthermore, we add knowledge about more complex data types, called *ValueDomains*, and about operators that are applied to these data types, called *Functions*. We also distinguish between the *Actions* and *Skills* and propose a method to recognize the performed *Skills* in an environment. The environment state, also modeled as a *ValueDomain*, is the basis for defining a task. We define a task goal as a set of valid environment configurations and present an application for recognizing whether an environment satisfies a task.

Probabilistic learning methods for action recognition [5] can speedily generate action hypotheses and a human-like description or segmentation of the recognized actions in a demonstration. Such approaches lack, however, an understanding of **why** motions "look like" actions. A model-based approach enables richer uses of knowledge, such as recognizing failures and pinpointing the reason for an unsuccessful skill execution by modeling the physical world. Also, a model-based approach can identify the missing steps or needed circumstances to successfully execute a skill. A skill model can verify whether it is truly happening in the scene, not just if it "looks like" the skill is executed.

*This work was supported by the Lighthouse Initiative Geriatronics by StMWi Bayern (Project X, grant no. 5140951).

[1]School Of Computation, Information, and Technology at the Technical University of Munich (TUM). [2]School of Computer Science, University of Nottingham, UK; also Associated Fellow at the MIRMI, TUM.

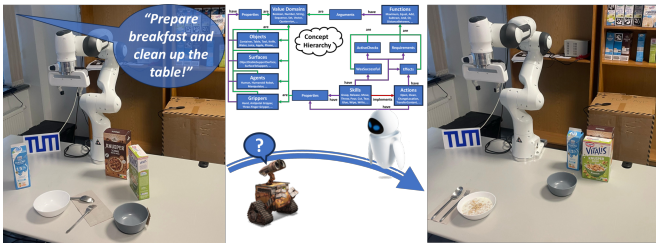

Fig. 1: "How to transform the left environment into the right one?" The **CH**'s knowledge enables household robots to represent environments and to create a plan to execute tasks.

II. The Knowledge in the Concept Hierarchy

In every problem domain, one must quantify the state of the environment. One must also describe how to change the state. Finally, one must specify desired environment states, i.e., goals. The **CH** models these three steps.

An *Environment* is the collection of all instances in it, and an instance has a *Concept* and a collection of properties that the instance's *Concept* defines. The **CH** stores the *Concepts*, whose definition contains a list of properties specified as $(p, pType)$. p is the property name, and $pType$ is a *ValueDomain* subconcept that represents its type, i.e. the set of allowed property values. For example, a *PhysicalEntity* has a location property defined with the *Location ValueDomain*, that contains a reference *Instance* entity and a *Pose* displacement to that entity's origin. Another example is the *Container* concept having contentLevel and contentVolume properties, both represented as *Number ValueDomains*.

Changes to the states are represented hierarchically as well. *Functions* are the highest abstraction level of change-operators affecting all *ValueDomains*. *Add* and *Subtract* are *Function* examples operating on two *Number ValueDomains* and returning a *Number* as well. *Compose* applied on two *Pose*s is also a *Function*. On a more specific level, *Actions* and *Skills* are change-operators that affect only *Concept* properties. The difference between an *Action* and a *Skill* is what they represent. **The change** of a *Concept* property is represented by *Actions*. **How the change is executed** in an environment is represented by *Skills*. *Actions* do not know the, e.g., geometric details and particularities or the instances whose property is

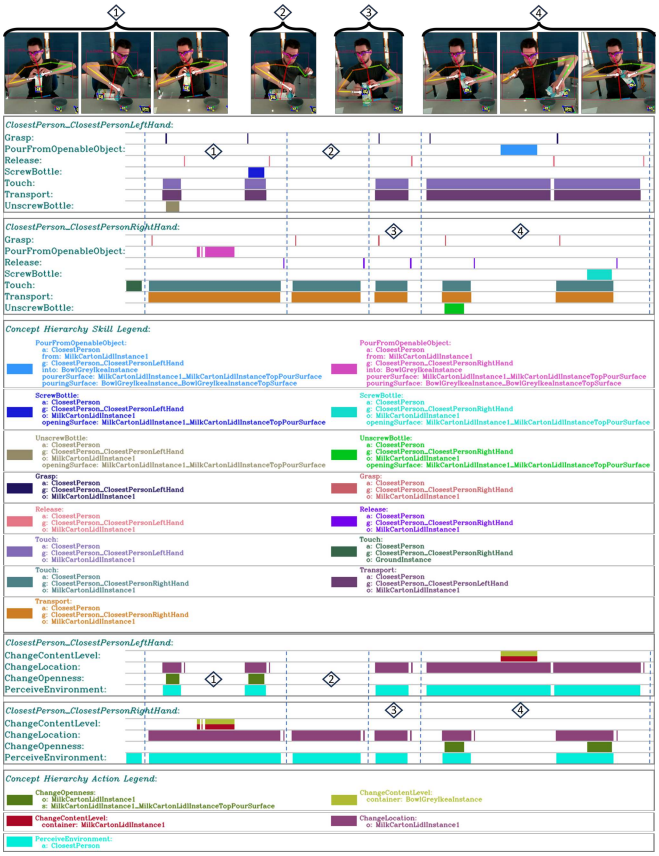

Fig. 2: *Skill* and *Action* recognition on a *pouring milk into a bowl* interaction, divided into four segments. The upper part shows the recognized *Skill* instances for each hand and *Skill* type. Colors distinguish *Skills* of the same type with different properties. The lower part shows the correspondence between *Skills* and *Actions*.

being changed. *Skills* do consider *Instance* properties as well as the *Abilities* of the executing *Agent(s)* and the *Environment* arrangement. The effects, prerequisites, and active-checks of *Skills* are modeled using *Functions*. *Actions* and *Skills* also have properties. E.g., the properties of the *ChangeLocation Action* are $(e, Instance\langle PhysicalEntity\rangle)$ and $(newLocation, Location)$. Its effects set e's location to $newLocation$. A *Skill* implementing *ChangeLocation* is *Carry*, with properties $(a, Instance\langle Agent\rangle)$, $(g, Instance\langle Gripper\rangle)$, $(e, Instance\langle PhysicalEntity\rangle)$, $(destination, Location)$, etc. Instance affordances indicate as which action or skill properties can the instance be used. The action affordances of an instance e is the set of pairs $Aff(e) = \{(a,p) \mid a \in Actions, p \in E(a), e$ is a subconcept of the entity-property p of the action $a\}$, where $E(a)$ is a's entity-properties. Skill affordances are defined similarly. *Abilities*, abstraction of motion primitives, are *Agent*-specific change-operators at the bottom of the operator abstraction hierarchy. *Skills* define behavior trees for each *Agent* that specify how to chain its *Abilities* to perform the *Skill's* change with given *Instances* and in a given *Environment*.

Finally, for representing desired states, i.e. task goals, we define *ValueDomain*-specific *Variations*. A *Variation* of a *ValueDomain* is a subset of values contained in that *ValueDomain*. Thus, a task goal is an *Environment Variation*, a set of allowed configurations of the *Environment* state. By allowing conjunctions, disjunctions, and negations of *Variations*, this

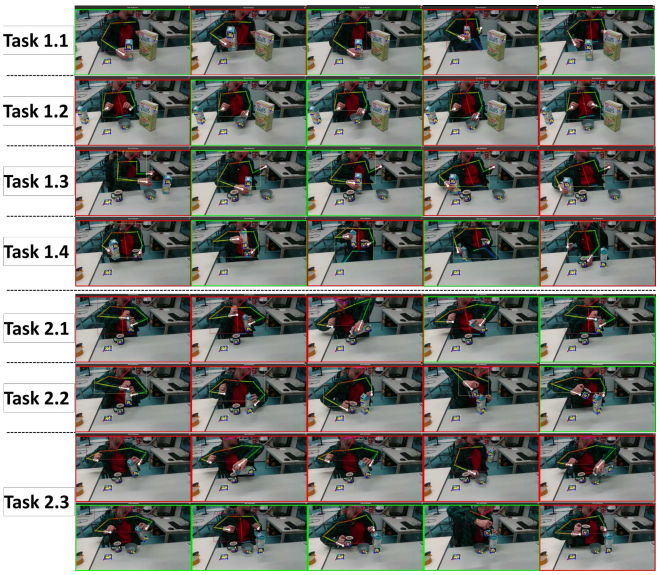

Fig. 3: The image frame is green when the goal is satisfied; it is red otherwise. In all tasks 2.*, the *MilkCartonInstance* m starts with contentLevel $= 0.9$L. In 2.1, pouring into the cup decreases m's contentLevel by 0.1L, satisfying the target variation. In 2.2, pouring from the cup adds 0.1 to m's contentLevel, which satisfies the goal. In 2.3, the cup's contentVolume can not contain all 0.9L of content from m, so the bowl is added to the scene. Once m's contentLevel is 0, m gets the *Trash* concept. When content is poured back into m, it loses its *Trash* concept, making the goal unsatisfied.

is a functionally complete [6] method of specifying all possible subsets of *ValueDomains*. Also using *Functions*, *Variations* check whether a specific value is contained and, if not, also return the reasons why a value is not contained, highlighting the framework's explainability.

III. Applications

We use OpenPose [7] and AprilTags [8] to get human hand 3D positions and object 3D poses from a Realsense [9] camera. The *Objects* and *Agents* are recognized and localized in each image frame, which updates their location property. The result of the *Action* and *Skill* recognition process is shown in Figure 2. This process updates the entity properties based on the effects of the recognized *Skills*. The *Environment* is checked framewise against task goals.

In Figure 3, the tasks 1.* define location goals with conjunctions, disjunctions, and complements of *Location Variations*. The following cases are presented: 1) a *LiquidContainer* should be on the *TableTopSurface*; 2) a *Bowl* should not be on the table; 3) the *MilkCarton* should be to the right of the *CupInstance* and on to the left of the *BowlInstance*; 4) a *LiquidContainer* should be on the *CupInstance* or below the *TableTopSurface*. The tasks 2.* define contentLevel goals on *Containers*: 1) the *contentLevel* of the *MilkCartonInstance* is requested to be $\leq$ 0.8L; 2) a *LiquidContainer* should have $\geq$ 1L of content; 3) a *Trash* instance is requested to be in the scene. None of the entities are *Trash* instances. However, the *MilkCartonInstance* is a *PerishableContainer* instance, which defines that if the contentLevel $= 0$, the instance receives the new concept of *Trash*. If the conteneLevel is changed again to something > 0, the instance loses its *Trash* concept.

REFERENCES

[1] E. Prestes, J. L. Carbonera, S. Rama Fiorini, V. A. M. Jorge, M. Abel, R. Madhavan, A. Locoro, P. Goncalves, M. E. Barreto, M. Habib, A. Chibani, S. Gérard, Y. Amirat, and C. Schlenoff, "Towards a core ontology for robotics and automation," *Robotics and Autonomous Systems*, vol. 61, no. 11, pp. 1193–1204, 2013, ubiquitous Robotics. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0921889013000596

[2] J. I. Olszewska, M. Barreto, J. Bermejo-Alonso, J. Carbonera, A. Chibani, S. Fiorini, P. Goncalves, M. Habib, A. Khamis, A. Olivares, E. P. de Freitas, E. Prestes, S. V. Ragavan, S. Redfield, R. Sanz, B. Spencer, and H. Li, "Ontology for autonomous robotics," in *2017 26th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, 2017, pp. 189–194.

[3] M. Tenorth and M. Beetz, "Knowrob: A knowledge processing infrastructure for cognition-enabled robots," *The International Journal of Robotics Research*, vol. 32, no. 5, pp. 566–590, 2013. [Online]. Available: https://doi.org/10.1177/0278364913481635

[4] R. Fikes and N. J. Nilsson, "Strips: A new approach to the application of theorem proving to problem solving," *Artif. Intell.*, vol. 2, pp. 189–208, 1971. [Online]. Available: https://api.semanticscholar.org/CorpusID:8623866

[5] Y. Kong and Y. Fu, "Human action recognition and prediction: A survey," *International Journal of Computer Vision*, vol. 130, no. 5, pp. 1366–1401, May 2022. [Online]. Available: https://doi.org/10.1007/s11263-022-01594-9

[6] H. Enderton, *A Mathematical Introduction to Logic*. Elsevier Science, 2001. [Online]. Available: https://books.google.de/books?id=JO4_NVZ_NqkC

[7] Z. Cao, G. Hidalgo Martinez, T. Simon, S. Wei, and Y. A. Sheikh, "Openpose: Realtime multi-person 2d pose estimation using part affinity fields," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.

[8] E. Olson, "AprilTag: A robust and flexible visual fiducial system," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, May 2011, pp. 3400–3407.

[9] L. Keselman, J. I. Woodfill, A. Grunnet-Jepsen, and A. Bhowmik, "Intel realsense stereoscopic depth cameras," *CoRR*, vol. abs/1705.05548, 2017. [Online]. Available: http://arxiv.org/abs/1705.05548