Traceable Robotic Handling of Sterile Medical Products

TRACEBOT

Software models (Final): Final definition of the software interfaces and software creating the Audit Trail

Deliverable 5.3

Horizon 2020

TraceBot receives funding from the European Union's H2020-EU.2.1.1. INDUSTRIAL LEADERSHIP programme (grant agreement No 101017089), 2021-2025

Deliverable Title	D5.3 Software models (Final): Final definition of the software interfaces and software creating the Audit Trail
Deliverable Lead:	University of Bremen (UOB)
Related Work Package:	WP5: Traceable Semantic Twin: Planning, reasoning, Audit Trail
Related Task(s):	T5.1: Definition of the domain process structure and taxonomy T5.2: Replication of medical lab environments into Traceable Semantic Twin Knowledge Bases for Reasoning T5.3: Traceability-aware process (re-)planning, reasoning and regulatory digital audit trail T5.4: Learning and reasoning about recorded process memories for accessible task-context information
Author(s):	Prof. Michael Beetz
Dissemination Level:	Public
Due Submission Date:	30/11/2023
Actual Submission:	29/11/2023
Project Number	101017089
Instrument:	Research and innovation action
Start Date of Project:	01.01.2021
Duration:	51 months
Abstract	We are describing our final definition of the interfaces that are used to interact with the hybrid knowledge representation and reasoning components. This also includes the simulation aspects in the Semantic Digital Twin and descriptions of the employed models for reasoning tasks.

Horizon 2020

TraceBot receives funding from the European Union's H2020-EU.2.1.1. INDUSTRIAL LEADERSHIP programme (grant agreement No 101017089)

Versioning and Contribution History

Version	Date	Modified by	Modification reason
v.01	14.11.2023	Prof. Michael Beetz (UOB)	Ready for internal revision
v.02	26.11.2023	Prof. Michael Beetz (UOB)	Revised version read for submission

Horizon 2020

TraceBot receives funding from the European Union's H2020-EU.2.1.1. INDUSTRIAL LEADERSHIP programme (grant agreement No 101017089)

Table of Contents

Versioning and Contribution History 3
Table of Contents 4
1 Executive Summary 5
2 Introduction 6
3 Reasoning Framework 8
3.1 Reasoning Framework overview 8
3.2 A Formal Query Language as Interface with the TST in TraceBot Robotic System 9
3.2.1 Loading distributed ontologies 9
3.2.2 Unload Ontology (WP5) 10
3.2.3 Percept Grounding into Ontology (WP5) 10
3.2.4 Handling articulated and compound objects (WP5) 11
3.2.5 Retrieval of common poses for grasp and motion planning (WP5) 12
3.2.6 Retrieval of common object properties (WP5) 13
3.2.7 Recording of Neem Narratives and Experiences (WP5) 14
3.2.8 Automated Planning and Execution of Verification Tasks (WP4) 17
3.2.9 Audit Trail Generation (WP5) 22
3.3 Simulation-enabled reasoning 23
3.4 Physical Reasoning based Embodied Probabilistic Simulations 26
3.5 Perception Executive 32
4 Deviations from the workplan 37
5 Conclusion 38
6 References 39

Horizon 2020

TraceBot receives funding from the European Union's H2020-EU.2.1.1. INDUSTRIAL LEADERSHIP programme (grant agreement No 101017089)

1 Executive Summary

This document serves as an extension to our previous deliverable 5.2 which laid out the basic software interfaces and simulation aspects as developed by the end of 2022. This deliverable incorporates key modifications aimed at addressing new and more complex use cases.

We offer a comprehensive interface description crucial for the interaction between the components developed in Work Package 5, primarily focusing on the reasoning apparatus, and other integral modules of the TraceBot ecosystem designated for the usage on the real world demonstrators of the project. The provided interfaces to the simulation aspects in our Semantic Digital Twin (semDT) and the foundational symbolic knowledge representation forms the main entry to our reasoning functionality.

One significant addition in this document is the introduction of the audit trail interfaces as well as the interfaces to the verification framework whose outputs are essential in the audit trail, a component essential for generating audit trails, ensuring transparency, traceability, and accountability in all operations. Another important enhancement is the handling of compound objects - compositions of multiple objects potentially attached to or removed from each other during task execution.

The key components are presented at a functional level, highlighting the expected inputs and anticipated results. We have structured the description into the primary building blocks focusing on knowledge representation & reasoning, simulation-based reasoning, physical reasoning, and imagistic reasoning in the perception executive. These aspects are articulated in a query-answering scheme, bridging use case specific reasoning tasks to the capabilities provided by the components of Work Package 5.

Horizon 2020

TraceBot receives funding from the European Union's H2020-EU.2.1.1. INDUSTRIAL LEADERSHIP programme (grant agreement No 101017089)

2 Introduction

In the rapidly evolving landscape of robotics, one key determinant for real world success and adaptability is the deployment of robust knowledge representation and reasoning systems. These systems, at their core, empower robots to process, interpret, and react aptly to complex environmental configuration and tasks. In the context of TraceBot, such a system must enable the robot to model knowledge about the task and its related objects, but also provide means to subsymbolically model the desired effects of the robots' actions. To combine this functionality and enable traceability, we develop a hybrid knowledge representation and reasoning system which is the key component for the reasoning based on semantic digital twins and symbolic knowledge.

In this report, we finalize our definition of the interfaces and software models. We do this by describing our additions and changes with respect to the previous Deliverable 5.2, which was defined as a first iteration of the description of our system.

Beginning with an analysis of developments since the last deliverable, we identified that enhancements were needed in our software models and their associated interfaces to cope with the challenges that revealed when integrating all the software components of the real world demonstrators of the project. These revisions were not arbitrary; they have risen from the project's ongoing demands and the investigation of additional use cases in the domain of sterility testing. Many of the changes were largely addressed by adapting the software models instead of the interfaces, to ensure continued compatibility with the other TraceBot components developed against our services.

Building upon this integration-driven approach, we introduced a novel model that improves the constraining of one object to another. Such a model is important when dealing with compound objects. To illustrate, consider the scenario of managing a needle with its cap attached to a fixture. The needle always comes from the bag of consumables with a cap attached, which will be removed during the process. To improve the reliability of the grasping of delicate needle, the consortium has used fixtures during development to place the needle in a well-graspable pose. This demonstrates that compound objects (e.g. needle with cap), can also be constrained to other objects in the environment. The importance of this functionality became clearly evident during the system integration phase and led to an improved handling of compound objects in our hybrid system.

Large effort was also spent on the improvement of the software model for robot simulation, necessary to update the semantic digital twin of the system in realtime, independent of the employed robot platform of the consortium.

Horizon 2020

TraceBot receives funding from the European Union's H2020-EU.2.1.1. INDUSTRIAL LEADERSHIP programme (grant agreement No 101017089)

Compared to the previous deliverable, we substantially extended the knowledge representation and reasoning capability of our framework. This added functionality does not only facilitate the creation of detailed audit trails but also empowers the system to introspectively reason about the outcomes of its task executions. Introducing this capability required us to not only consider the whole skill process driving the execution, but also the modalities of verification provided by the individual components of all partners.

In addition to the previous enhancements, we also proposed a new underlying model of our perception executive to handle differing perception tasks more flexibly. With the inclusion of a state-of-the-art task execution model, our robotic system can now effortlessly change between intricate imagistic reasoning tasks as well as common perception tasks, such as object detection. This integration ensures a generalized approach to support diverse task requirements.

All of the changes mentioned above led us to new, integrated demonstrations of the TraceBot robots with our digital twin technology. A recent video recording of these demonstrations can be found here https://ai.uni-bremen.de/tbintegration53

Horizon 2020

TraceBot receives funding from the European Union's H2020-EU.2.1.1. INDUSTRIAL LEADERSHIP programme (grant agreement No 101017089)

3 Reasoning Framework

In this section, we briefly situate the reasoning framework in the TraceBot robotic system and provide a revision of the interfaces between the framework and the rest of the system, initially defined and presented in the previous deliverable 5.2.

3.1 Reasoning Framework overview

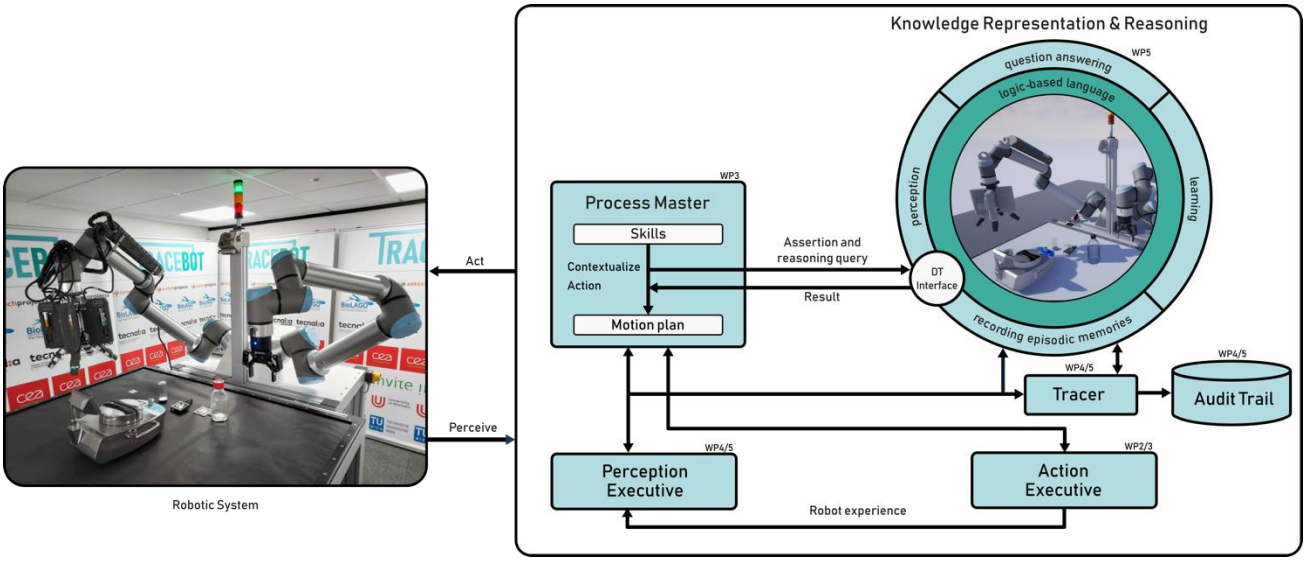


Figure 1 Overview of the interaction scheme between the different components in the TraceBot ecosystem, highlighting the connection between the robotic system and the digital twin (shown in the circle on the right).

As you can see from the architecture Figure 1 above, the reasoning framework acts as an information provider to other modules of the system for the successful completion of the TraceBot processes. The overall conceptual framework for the knowledge representation and reasoning system in TraceBot, also termed as Traceable Semantic Twin (TST), was presented in the former deliverable 5.1. In deliverable 5.2, we defined and proposed an initial version of the interfaces between the reasoning framework and the rest of the system. It mainly consisted of a logical query language that allows to tell and ask for any knowledge, be it symbolic, subsymbolic, procedural, declarative, episodic or semantic to the reasoning framework. However, as we get deeper into the development of the TraceBot robotic system, there starts to appear a clear subset of the proposed language that encompasses the set of queries circumscribing the necessary and concrete interfaces so far. In this section, we present this set of queries in terms of inputs, outputs, goals and demonstrations. We organize the presentation of these interfaces into two sections for the sake of simplification. In the first section, we present the top-level (i.e., less details) queries covering the whole spectrum of interactions between the reasoning framework and the rest of the TraceBot robotic system. Then, we present in a second section more detailed queries mostly dedicated to specific simulation and reasoning processes that are encapsulated by some top-level queries.

Horizon 2020

TraceBot receives funding from the European Union's H2020-EU.2.1.1. INDUSTRIAL LEADERSHIP programme (grant agreement No 101017089)

3.2 A Formal Query Language as Interface with the TST in TraceBot Robotic System

In order to ease the understanding and use of the developed interfaces, especially in the robotics community, we wrapped them within ROS[1] as action interfaces. This being said, the interfaces exposed below are also compatible with and can be understood in the ROS terminology.

3.2.1 Loading distributed ontologies

Any reasoning task takes place within the frame of an ontology, which itself describes the set of fundamental truths that hold for a certain domain. This means, an affirmation can be true according to one ontology and false according to another. This being said, it is essential to provide a mechanism to dynamically load the ontology of interest into the reasoning framework. On the other hand, it might be difficult to build a sufficiently rich ontology from scratch, hence the necessity to combine multiple existing ones: we talk about distributed ontology. This is achieved by loading the ontologies one after the other.

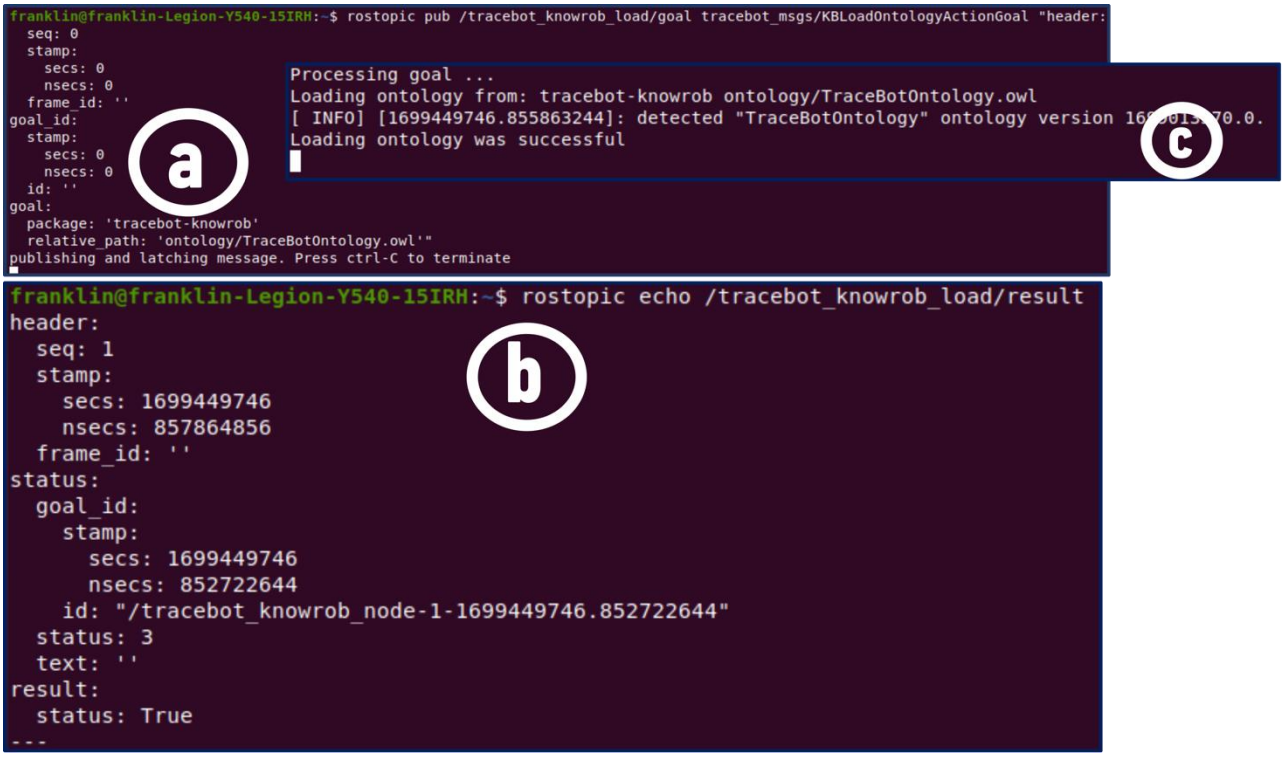


Figure 2: (a) the query with info about ontology file location, (b) success of the ontology load, (c) confirmation of loader.

[1] ROS stands for Robot Operating System

Horizon 2020

TraceBot receives funding from the European Union's H2020-EU.2.1.1. INDUSTRIAL LEADERSHIP programme (grant agreement No 101017089)

`Load_Ontology(ontology_file_path)`. This interface allows to load a specific ontology located at `ontology_file_path` into the reasoning engine. Actually, it supports ontology formatted within the well-established OWL (Ontology Web Language). Within an .owl file, knowledge is stored in the form of triple `triple(subject, predicate, object)`. e.g., `triple(Canister, subclass_of, Container)` to state that a canister is a subclass of container. Note that this is an infix representation of the triple which can also be written in the prefix form as `subclass_of( Canister, Container)`. And to load the whole .owl file, the triples are basically projected into the reasoning system one after the other using the more fundamental assertion `kb_project (subclass_of(Canister, Container))`. the loading returns true if successful and false otherwise. This interface is demonstrated in Figure 2.

3.2.2 Unload Ontology (WP5)

`KB_Reset(namespace)`. Once that the ontologies of interest can be loaded into the reasoning engine, it is important to be able to unload undesired ontologies from the engine. We then defined this interface that takes as parameter the namespace of the target ontology for instance `http://www.ease-crc.org/ont/SOMA.owl` and retracts from the knowledge all the triple facts that involve such namespace. Note that the identification of any concept in an ontology is prefixed by the namespace of that ontology, e.g., `http://www.ease-crc.org/ont/SOMA.owl#Canister`. To retract a triple fact from the knowledge base, the more fundamental query `kb_unproject(triple_fact)` is used where `triple_fact` is a triple.

3.2.3 Percept Grounding into Ontology (WP5)

In order to reason and understand what is going on as the robot performs, it is a crucial step to link the ongoing activities to the ontology. This mainly consists in attributing meaning to robot motions, scene objects, events, etc. by grounding them into specific concepts. Since these percepts are handled by different modules in TraceBot (e.g., perception executive, planning executive, action executive), they come with their own conceptual classifications of these percepts. The arising problem is how to uniformize these classifications in the TraceBot ontology. For instance, when the robot comes into a safe pose after completing a task or before starting a task, the planning executive called it `move_home` (local class), it is referred in the ontology to as `http://www.ease-crc.org/ont/SOMA.owl#ParkingArms` (ontological class).

Horizon 2020

TraceBot receives funding from the European Union's H2020-EU.2.1.1. INDUSTRIAL LEADERSHIP programme (grant agreement No 101017089)

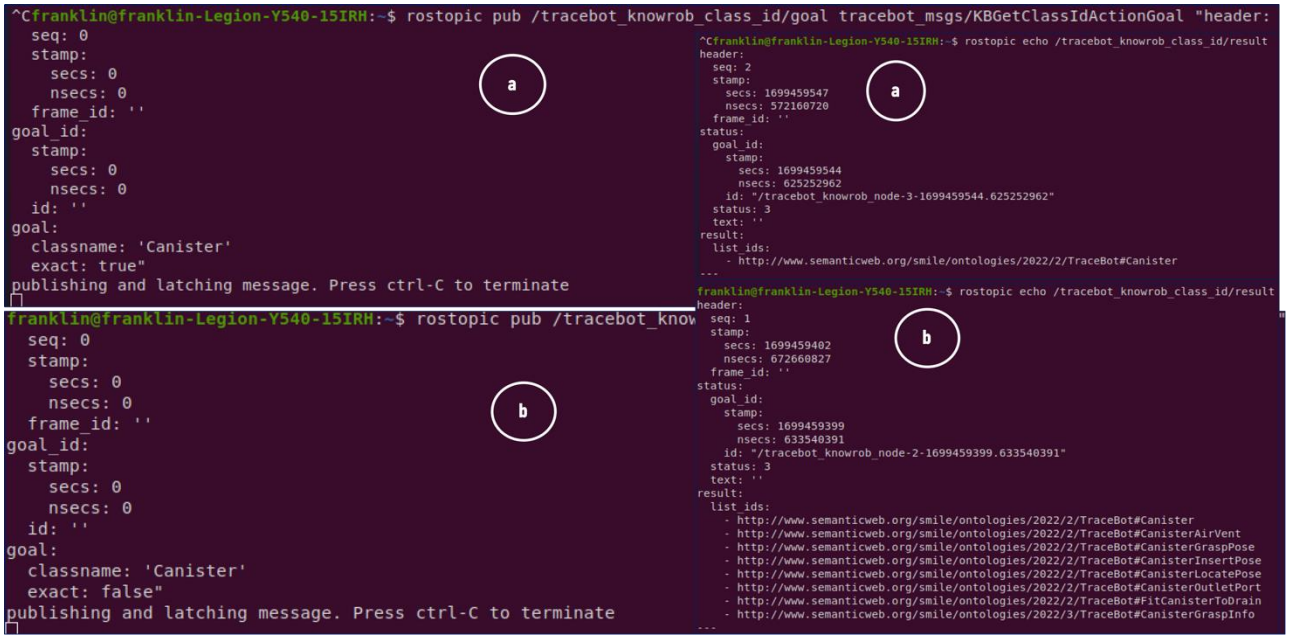


Figure 3: (a) the query and result for exact ontological classes, (b) the query and result for ontological classes from lexical field i.e., concepts that are closely related to the concept of canister.

`Get_Class_Id(local_concept_key, exact)`. This interface takes as parameters the local class `local_concept_key` of a percept and returns the corresponding ontological classes. When the second boolean parameter `exact` is set to false, the interface returns all the ontological classes from the lexical field of the local class. The point here is that one might not be sure about whether the local class is correct. Figure 3 illustrates the demonstration of this interface.

3.2.4 Handling articulated and compound objects (WP5)

In TraceBot, objects are generalized to articulated and compound objects in the sense that one should inherently reason about them with respect to their parts. And one fundamental step towards reasoning about the states of compound objects is to be able to instantiate them properly.

Horizon 2020

TraceBot receives funding from the European Union's H2020-EU.2.1.1. INDUSTRIAL LEADERSHIP programme (grant agreement No 101017089)

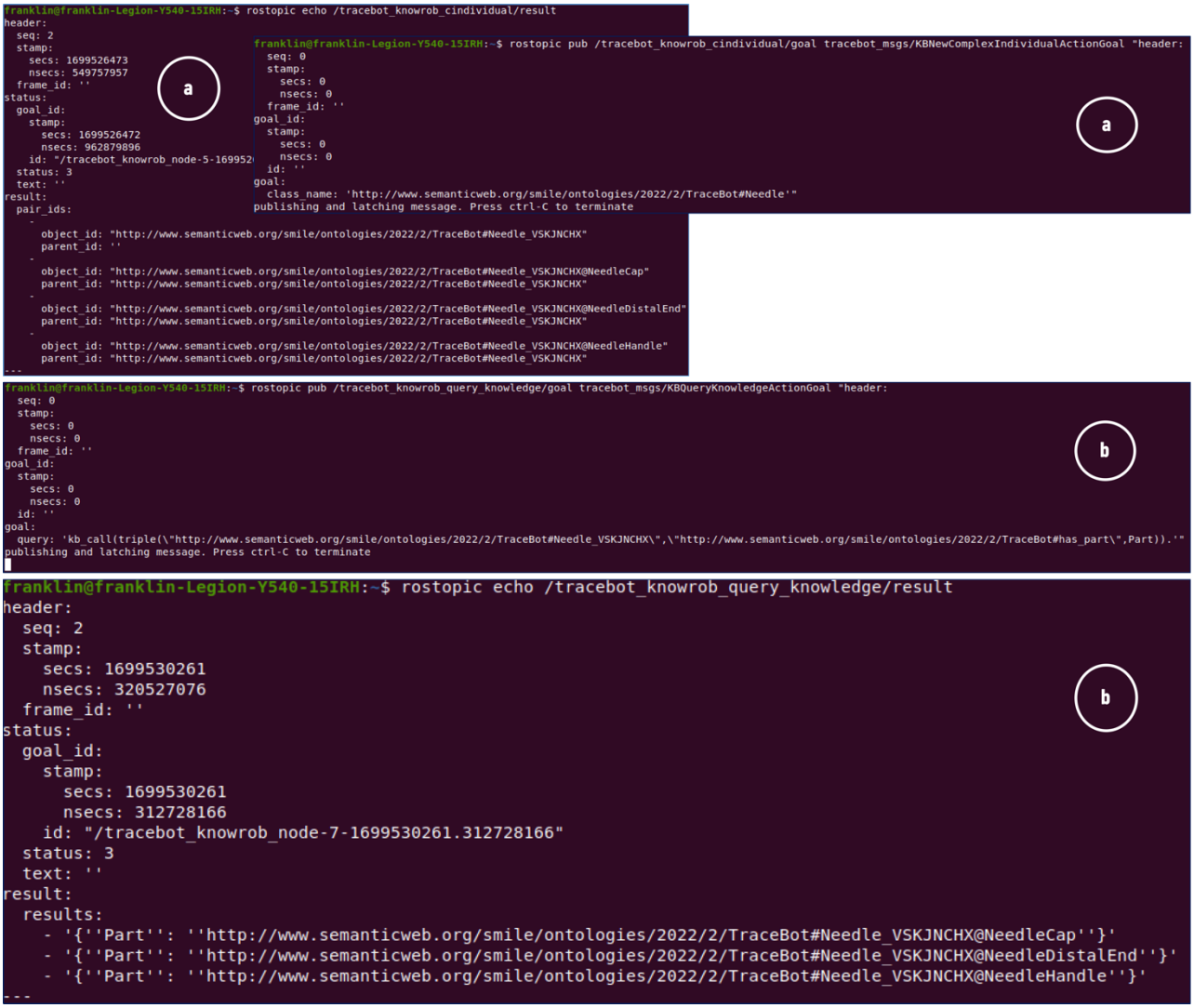


Figure 4: (a) the query and result for instantiating a needle, (b) the query and result for checking the aggregation of part instances to parent instance.

For instance, when a needle is instantiated, all the needle parts (distal end, handle, cap, etc) are instantiated along with it and the instances are connected with each other. Later on, the cap will become an integral object if it is to be removed and laid on the table.

And given the position of the cap instance and that of other parts such as the distal end, then one would be able to infer that the cap has been removed. This is also essential for tracking parts of an object to be later on able to reassemble it, e.g., closing the bottle after opening it.

`New_Complex_Individual(class_id)`. This interface generalizes the instantiation of objects by viewing them as compound and then generating a tree of instance where child instances are defined as parts of the parent node. It only takes as argument the ontological id of the object to be instantiated such as `http://www.semanticweb.org/smile/ontologies/2022/3/TraceBot#Needle`. This interface is demonstrated by Figure 4.

3.2.5 Retrieval of common poses for grasp and motion planning (WP5)

Horizon 2020

TraceBot receives funding from the European Union's H2020-EU.2.1.1. INDUSTRIAL LEADERSHIP programme (grant agreement No 101017089)

Though the natural world is dynamic, in the sense that the state of entities changes over time, it maintains however a considerable stable structure whose knowledge is commonly referred to as commonsense. This considerably reduces the entropy of the natural data the robot has to process in order to successfully interact with the world. Some of these stable structures are the common poses of entities (e.g., the typical standing pose of a long-tail glass) after an action is performed and the common poses of the grippers when grasping an object for a particular action execution (e.g., grasping a standing bottle of milk to pour some milk into the mug). `Get_Common_Pose(object_id, pose_type)`: This interface returns the common pose of an object after an action, or the grasp poses of the gripper for a given object and a given envision action as shown by Figure 5.

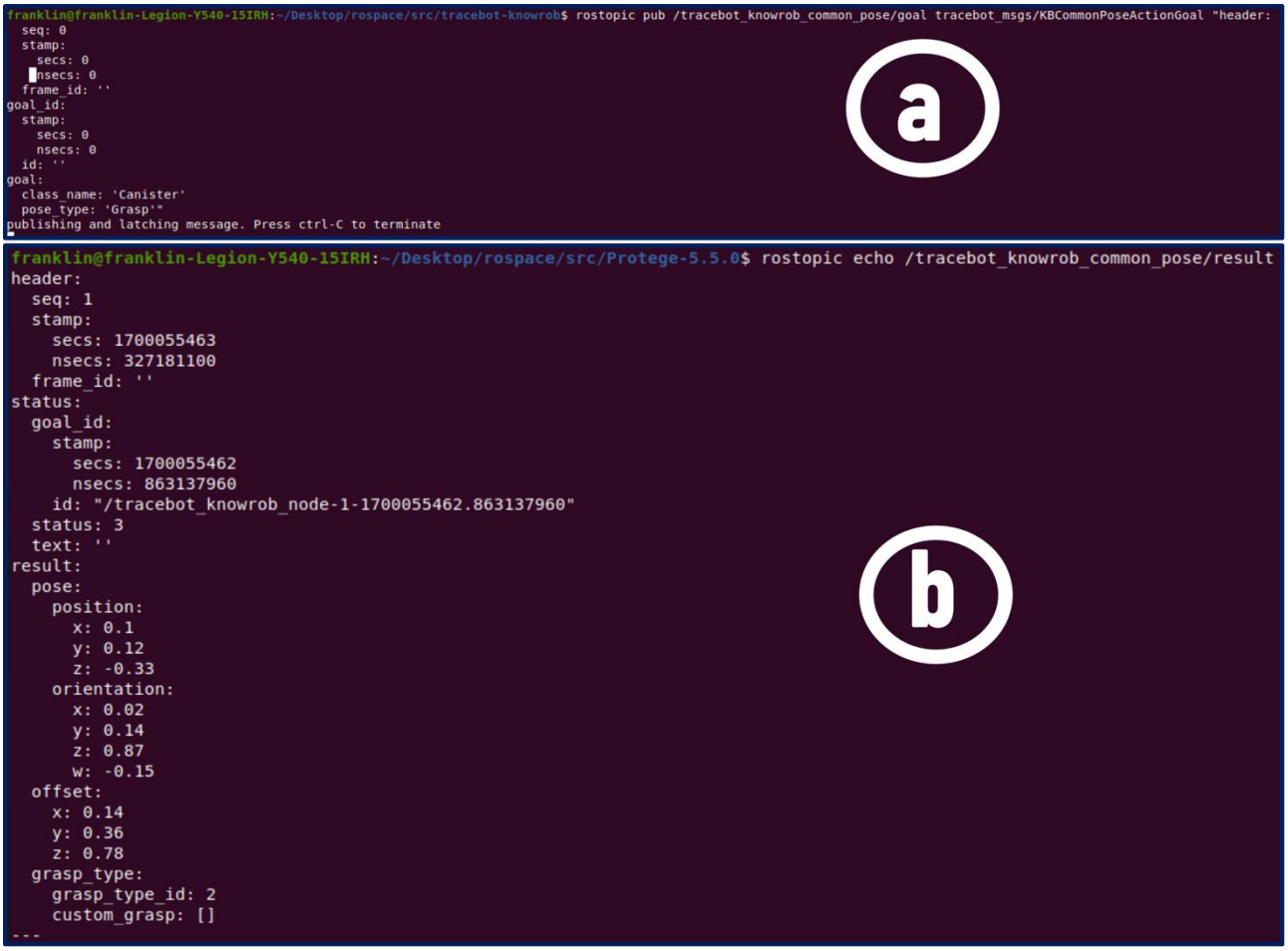


Figure 5: (a) The query about relative grasp poses for canister, (b) answer about relative grasp pose for the canister.

3.2.6 Retrieval of common object properties (WP5)

Inspecting the general or specific attribute of a concept is key to handle entities that are classified by that concept. For instance, knowing that canisters are transparent would cause the perception executive to choose a dedicated algorithm for its detection.

`Get_Properties(object_id, property_id, quantifier)`. This interface returns the value of a given attribute of an entity that is classified by a given concept, for instance the diameter of a specific bottle, the color, the object parts. Figure 6 demonstrates the interface.

Horizon 2020

TraceBot receives funding from the European Union's H2020-EU.2.1.1. INDUSTRIAL LEADERSHIP programme (grant agreement No 101017089)

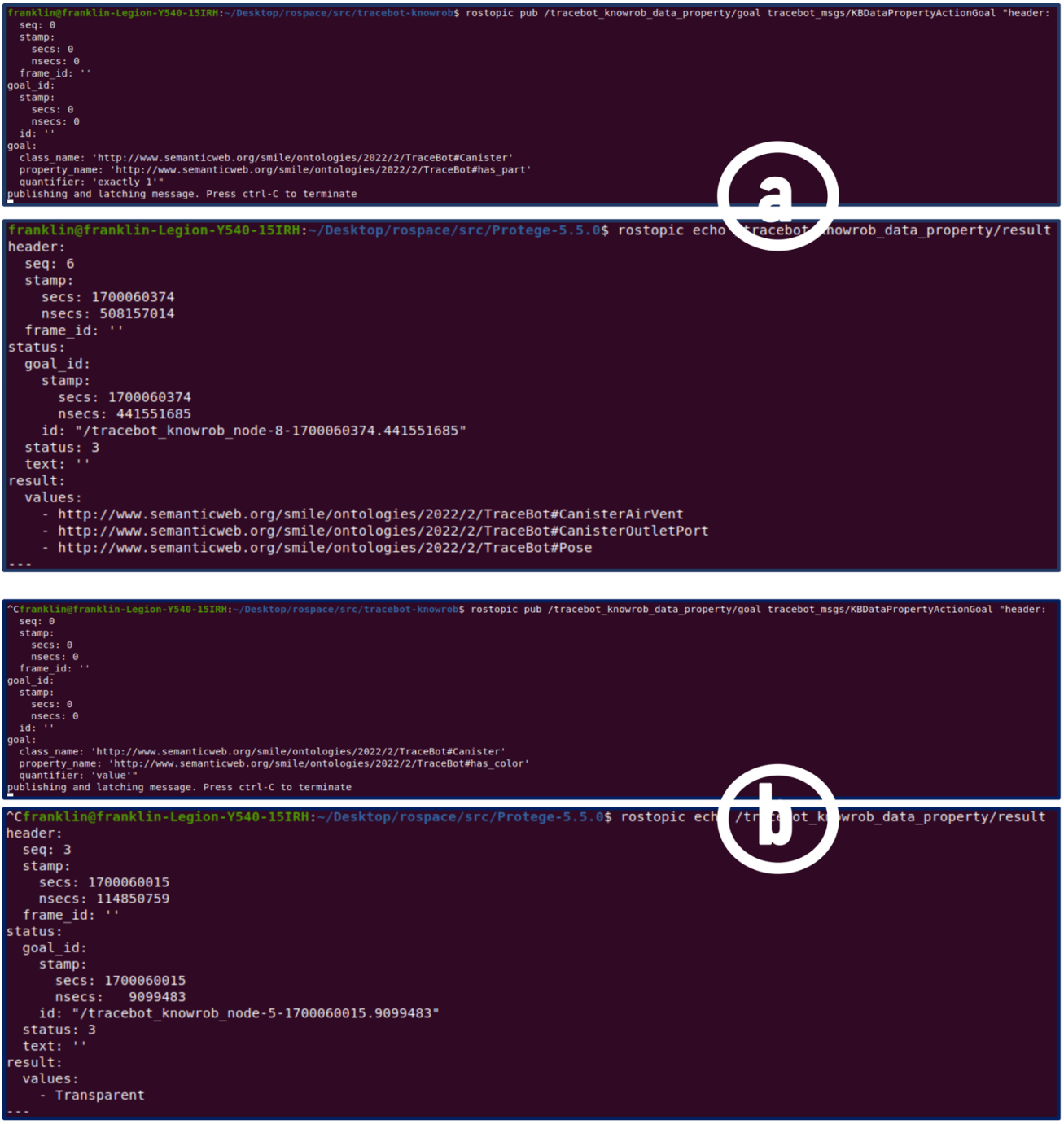


Figure 6: (a) The query and answer about canister's parts, (b) query and answer about canister's color.

3.2.7 Recording of Neem Narratives and Experiences (WP5)

As the robot performs sterility testing tasks, it memorizes what it is doing in terms of narratives i.e., what (tasks?), when (execution time?), who (actuators? sensors? objects? roles?) and in terms of experiences i.e., sensor data (sensations? feelings?). Such memory is known as NEEM for Narrative-Enabled Episodic Memory and is grounded into the ontology to enable understanding. These NEEMs

Horizon 2020

TraceBot receives funding from the European Union's H2020-EU.2.1.1. INDUSTRIAL LEADERSHIP programme (grant agreement No 101017089)

constitute the informational foundations of subsequent generated audit trails of the robot activity, but also rich, not to say universal, training datasets for multi-purpose robot learning. The concept of NEEM is illustrated by Figure 7. In order to record the NEEM, the interface below makes use of the above primitive interfaces as follows.

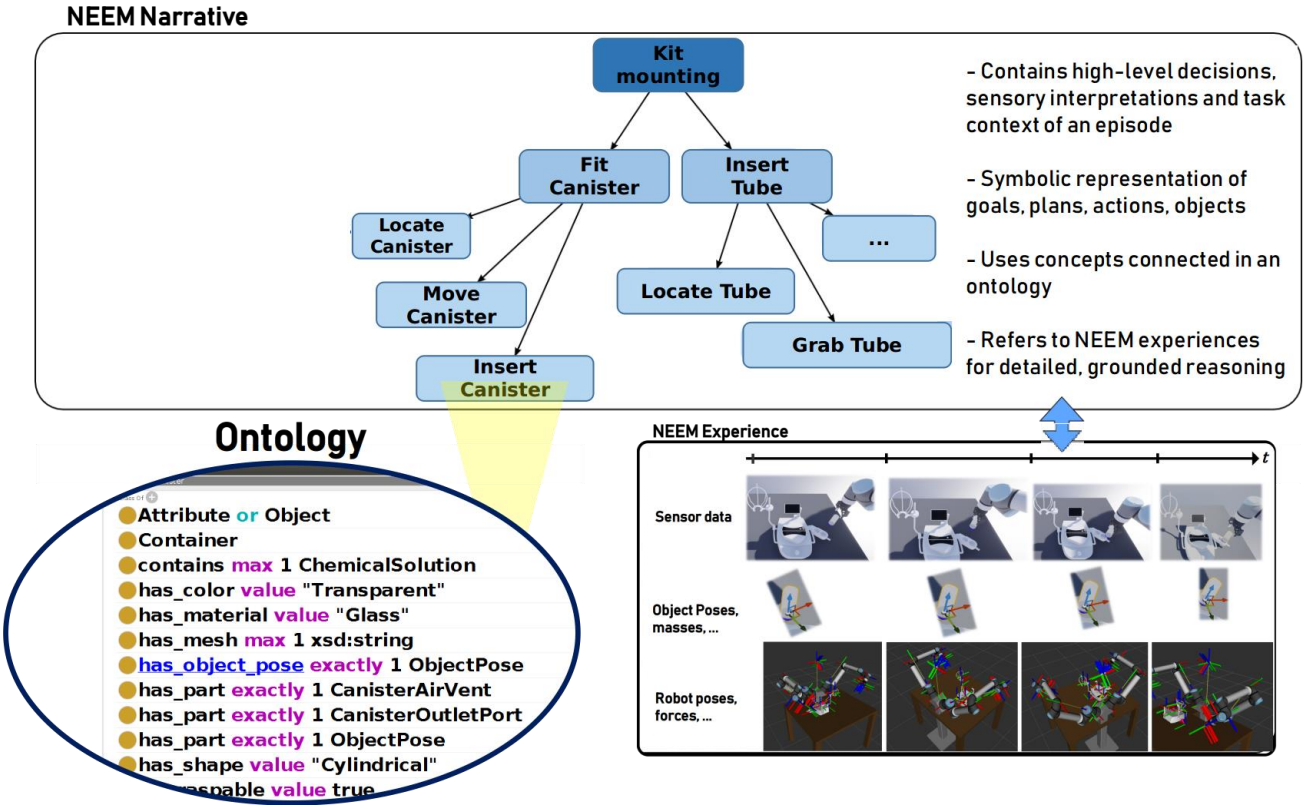


Figure 7: (bottom-right) The sensor data as robot experience, (top-right) the robot activity narrative and (bottom-left) the grounding of the symbols from activity narratives into ontology.

`Begin_Episode(root_task_id, env_model, agent_model).` An episode for a given task is a complete execution of that task, which might be composed of subtasks. The recording of an episode is the smallest unit of NEEMs and starts with a call to this interface, taking as parameter the type of action being performed, the environment model (i.e., where the action takes place such as lab or kitchen which is described in the ontology) and the model of agent or robot performing (also described in the ontology). This interface returns the id of the instance of the created root action.

`Set_Episode_Title(episode_id, title).` This interface sets the title of a given episode. This is an important annotation of the episode for retrieving, presenting the corresponding NEEM but also to learn from it.

`End_Episode(neem_path).` This function terminates the recording of a NEEM by storing it into a specific location of the disk specified by neem_path. This will contain NEEM narrative as triples (object, predicate, object).

`Begin_Event(action_id).` In TraceBot-SOMA ontology, actions are regarded as a subclass of events which themeselves describes any phenomenon that characterizes a state change. When an action is about to be executed, this interface instantiates it and allocates a begin execution time to it. Beyond the beginning execution time, this interface can also trigger events that cause the execution

Horizon 2020

TraceBot receives funding from the European Union's H2020-EU.2.1.1. INDUSTRIAL LEADERSHIP programme (grant agreement No 101017089)

of critical actions such as the verification of the action feasibility and the storage of the verification results.

`End_Event(action_instance_id).` Once the action has been executed, this function registers the termination in the NEEM by indicating the end execution time. Beyond the end execution time, this interface can also trigger events that cause the execution of critical actions such as the verification of the executed action and the storage of the verification results.

`Set_SubEvent(sub_action, parent_action).` Given that some actions might include other actions, this interface asserts a given sub-action as a phase of a given parent action. This is essential to generate the action tree of the robot activity which represents the skeleton of the activity narrative.

`Set_Participant_with_Role(entity_id, role_id, action_id).` As already mentioned above, a narrative is also about the participants and the roles they play in the story. For instance, the canister can play the role of stimulus in the action *perceiving* but the role patient in the action grasping or inserting. This interface asserts a given entity as participant in a given action with a given role.

`Set_Event_Status(action_id, status).` Once an action has been verified either for feasibility or success, the decision of verification (e.g., succeeded), noted here as status, is also asserted.

`Set_Event_Confidence(action_id, status).` Beside the status of the action verification, there is also the confidence of the result which somehow describes the probability that the decision is correct. This characterizes the multiple sources of uncertainty during reasoning.

`Set_Comment(entity_id, comment).` This interface allows generic annotations of symbols from the NEEM narrative. As the `Set_Episode_Title` seen earlier, the goal is manifold, but mostly either for learning or understanding purposes.

`Begin_Episode_Experience(episode_id).` What has been recorded so far is just symbolic and therefore part of the NEEM narrative. Along with the recording of narratives, the beginning of an episode also triggers the recording of the robot experiences through this interface. Given the episode id, the interface knows which environments, agents and objects are involved and will automatically fetch the sources of sensor data from the ontology, listen to them and store them within the knowledge base while keeping track of their chronological order. In the ROS ecosystem, these sources of sensor data are referred to as topics and can be listened to and recorded within the so-called bag files.

`Belief_Perceived_At(object_id, pose).` Besides sensor data that are part of the NEEM experiences, there are also object poses. When an object is perceived for the first time a stream is created to advertise its pose at regular intervals of time. This creates a trajectory of the object pose over the course of the task execution. The same pose is published to the stream if no change occurs. And when a change occurs, this is notified by this interface and the new pose of the object is advertised in the stream. In the ROS ecosystem, these pose streams are called TF (TransForm) topics.

`Stop_Episode_Experience(episode_id).` When the episode ends, the recording of the experiences is also terminated and stored knowledge is exported from the knowledge base into the location specified by neem path for portability and exploitability.

Figure 8 illustrates the recording of the NEEM for an episode of the generic fitting an object into another one called *FitInsertableIntoInsertee*.

Horizon 2020

TraceBot receives funding from the European Union's H2020-EU.2.1.1. INDUSTRIAL LEADERSHIP programme (grant agreement No 101017089)

Figure 8: (a) The action tree of the NEEM narrative of an episode of FitInsertableIntoInsertee action, (b) the assertions of symbolic knowledge in the NEEM narrative, (c) the extraction of experience sources from the ontology, the listening of these streams

3.2.8 Automated Planning and Execution of Verification Tasks (WP4)

We iterated many times that the ultimate goal of verification in the TraceBot project, may it be visual-based, tactile-based, DT-based, acoustic-based, odometry-based, symbolic-knowledge-based etc., could be reduced to the verification of action feasibility and success. And we termed such ultimate verification as functional verification, which makes use of the recorded NEEMs grounded into the ontology and enriched by the results from primitive verification modalities (e.g., visual-based, DT-based) to ultimately decide the feasibility and success of actions with respect to the formulated action pre-conditions and post-conditions. However, there are three core challenges: **(Challenge 1)** As an action becomes more and more complex in structure and length, it becomes difficult to formulate its pre-conditions and post-conditions comprehensively. **(Challenge 2)** Secondly, as the action complexity grows up, it becomes difficult to figure out how to make use of the dynamic content of NEEMs (e.g., tactile sensor available in this episode and not in the other one) to check against the pre- and post-conditions of such actions. Notice that a brute force, i.e., an attempt to store everything in the NEEM is not realistic and only makes the robot control program very rigid as not all knowledge modalities are usually available and the set of knowledge they can provide is practically unbounded. **(Challenge 3)** Finally, notice also that though two action instances of the same type (e.g., Grasping) do very likely have the same pre- and post-conditions, the actual procedures to attest that those conditions are met might significantly differ (e.g., checking paper in robot gripper vs checking bottle in robot gripper). In order to address these three challenges, we propose a highly scalable (w.r.t. process structure, length and diversity of knowledge modalities) framework for automated planning and execution of verification tasks whose architecture is depicted on Figure 9.

Horizon 2020

TraceBot receives funding from the European Union's H2020-EU.2.1.1. INDUSTRIAL LEADERSHIP programme (grant agreement No 101017089)

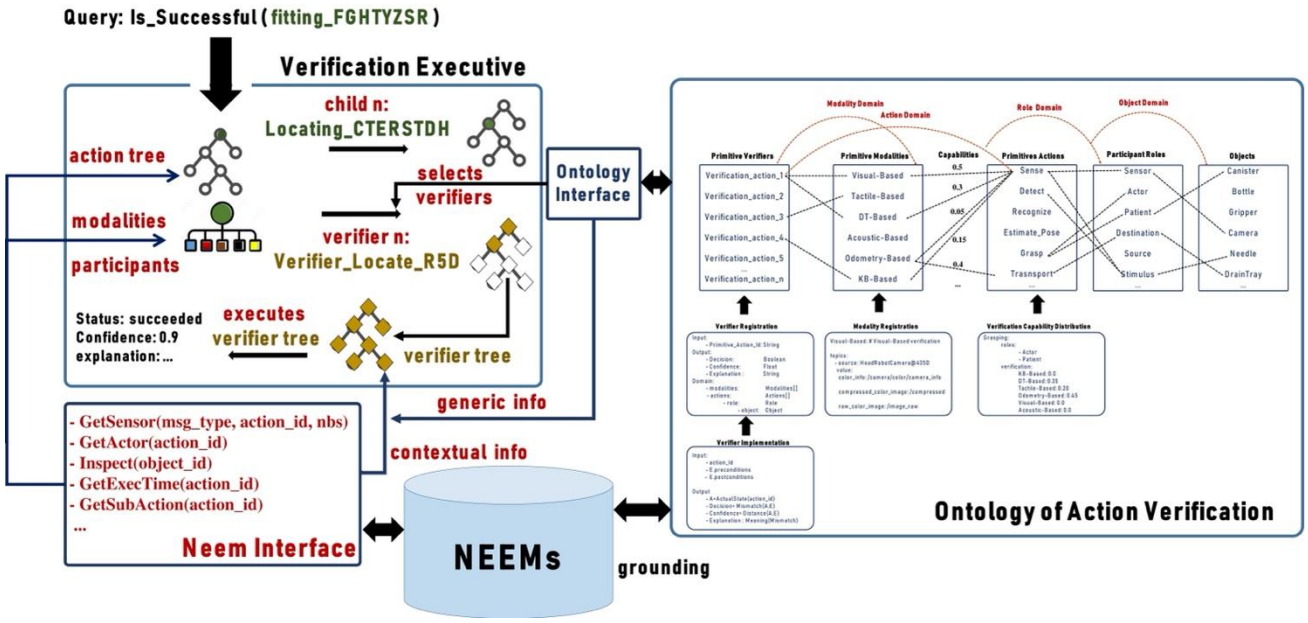


Figure 9: Automated planning and execution of action verification tasks.

In the ontology, primitive actions are defined in terms of their participants which themselves are defined in terms of roles they play in the action. Notice that roles are more stable in terms of action participants than direct object names. In the former case, one can specify the participants of an action ahead.

Action k

```
Recognizing:
  roles:
    - Sensor
    - Stimulus
  verification:
    KB-Based: 0.0
    DT-Based: 0.35
    Tactile-Based: 0.20
    Odometry-Based: 0.0
    Visual-Based: 0.45
```

...

Action n

```
Transporting:
  roles:
    - Actor
    - Patient
    - Source
    - Destination
  verification:
    KB-Based: 0.0
    DT-Based: 0.35
    Tactile-Based: 0.20
    Odometry-Based: 0.30
    Acoustic-Based: 0.0
    Visual-Based: 0.15
```

Figure 10 Revising the specification of actions in the ontology for flexible and automated action verification

Horizon 2020

TraceBot receives funding from the European Union's H2020-EU.2.1.1. INDUSTRIAL LEADERSHIP programme (grant agreement No 101017089)

Moreover, the distribution of capabilities over knowledge modalities to enable the verification of the underlying action is provided based on commonsense (e.g., tactile sensors are more likely to detect a slippage than visual sensors). Whereas the specification of actions' participants allows the fine-grained specification of verification algorithms as we will see, the distribution of verification capabilities over knowledge modalities informs about how to combine concurrent verification algorithms and consolidate their outputs. To ease the specification of primitive actions in terms of participants and distribution of verification capabilites, yaml configuration files are provided such as illustrated by Figure 10. This being done, the framework assumes a bag of verification experts called verifiers where each verifier is competent at verifying a specific action for a given set of objects playing some given roles under a set of well-specified knowledge modalities. This restriction of the verifier w.r.t. actions, actions' participants and necessary knowledge modalities is termed as the verifier's domain. Also important is the detailed specification of knowledge modalities in terms of nature (e.g., symbolic/subsymbolic), type (e.g., visual, acoustic, knowledge base), source (e.g., head camera) and the knowledge stream (e.g., ontology or sensor channels). For the specification of verifiers and modalities, templates of yaml configuration files are also proposed as interface with the ontology as shown by Figure 11 and Figure 12.

Verifier k

```yaml
# the verifier is a ros action
verifier:
  name: TransparentObjectRecognitionCheck
  description: Check the success of a recognition action
  mode: Success
domains:
  - Recognizing:
      objects:
        - Support: []
        - Container: []
        - Actor: []
        - Sensor:
            - Robot@Tracebot.Head.Camera@D435
        - Patient:
            - Canister
            - RinseBottleA
            - RinseBottleK
        - Destination: []
        - Source: []
        - Tool: []
        - Accommodation: []

      modalities: [Visual-Based, DT-Based]

# the inputs and outputs of the verifiers are fixed
#inputs:
#  action_id: string
#outputs:
#  decision: bool
#  confidence: float64
```

...

Verifier n

```yaml
# the verifier is a ros action
verifier:
  name: KBGrabFeasibilityCheck
  description: Check the success of a grasp action
  mode: Feasibility
domains:
  - Grasping:
      objects:
        - Support: []
        - Container: []
        - Actor:
            - Robot@Tracebot.LeftArm@UR10.Gripper@Robotiq
            - Robot@Tracebot.RightArm@UR10.Gripper@Robotiq
        - Sensor: []
        - Patient:
            - Canister
        - Destination: []
        - Source: []
        - Tool: []
        - Accommodation: []
      modalities: [KB-Based, Visual-Based, Odomedry-Based]
  - PickingUp:
      objects:
        - Actor:
            - Robot@Tracebot.LeftArm@UR10.Gripper@Robotiq
            - Robot@Tracebot.RightArm@UR10.Gripper@Robotiq
        - Patient:
            - Canister
      modalities: [KB-Based, Visual-Based, Odomedry-Based]

# the inputs and outputs of the verifiers are fixed
#inputs:
#  action_id: string
#outputs:
#  decision: bool
#  confidence: float64
```

Figure 11 Specification of action verifiers. On the left is a uni-domain verifier and on the right is a multi-domain verifier.

Finally, the implementation of each verifier is provided beside its specification (i.e., input, output). Notice that the pre- and post-conditions of an action are then moved to the implementation of the specific verifier (See green verifier signature from Figure 11's bottom). Such a description constitutes an ontology of action verification tasks. Given a recorded NEEM of a task's episode, the robot control program is given two interfaces namely `Is_Successful(action_id)` and `Is_Feasible(action_id)` to flexibly (e.g., at any time of the program execution) and respectively

Horizon 2020

TraceBot receives funding from the European Union's H2020-EU.2.1.1. INDUSTRIAL LEADERSHIP programme (grant agreement No 101017089)

check whether a given action is successful or feasible. To compute these two predicates, the verification executive first infers the action tree of the given action as well as the participants of each primitive sub-action and the knowledge modalities under which it took place (i.e. action structure), which somehow delineates the domains of potential verifiers. In order to retrieve this information about an executed action from the NEEM, the NEEM interface described above is used after being augmented with a few more specific interfaces that we describe below.

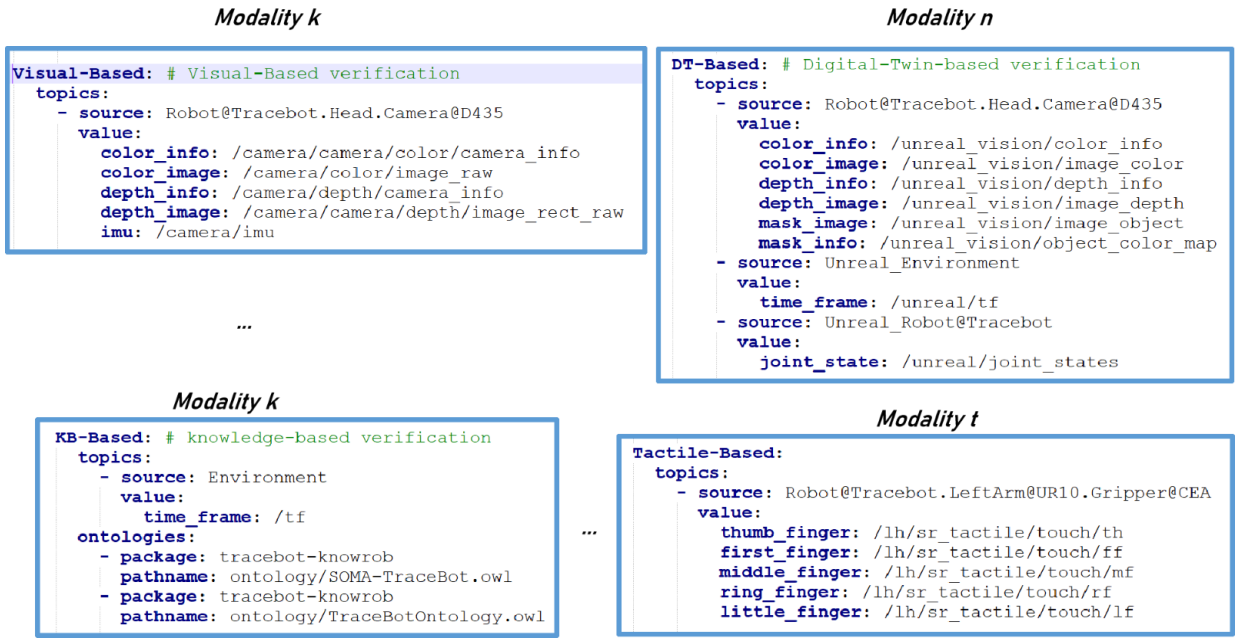


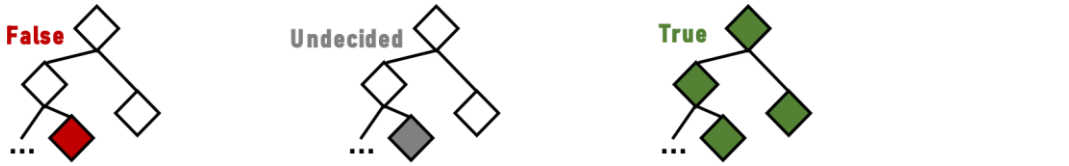
Figure 12 Specification of some knowledge modalities. For each modality, the knowledge might be flowing through topics or stored in a knowledge base such as ontologies. And for each modality, there might be multiple sources of information (e.g., three cameras in

Given these verifiers' domain constraints, the verification executive searches for each primitive sub-actions one or if possible, a group of satisfiable verifiers to build a verifier tree isomorphic to the target action's action tree. To access the list of verifiers registered into the ontology as well as their properties, the generic ontology interfaces described earlier are used. Once the verifier tree has been built, the implementation i.e. program for each verifier node is loaded and executed using the NEEM interface for accessing contextual information (e.g., object's pose). At the end, the tree of individual results is processed as if the tree was a complex boolean operator for the decision output (e.g., action successful if all sub-actions were successful, false if at least one action failed and undecided otherwise), and as a joint probability tree for the confidence output (e.g., the confidence of an action in the tree is the product of the confidence of all the direct child actions. If multiple modalities contributed to the verification of an action, then the ultimate confidence for the verification of that action is the weighted average of the verifiers' individual confidence where the weights are their prior capabilities to verify the given action. Since some modalities could not contribute, the weights of the contributing modalities are renormalized (i.e., sum up to one) before application. However, if a single verifier makes use of multiple modalities, then the previous rule is applied while keeping verifier's individual confidence identical for each of these modalities. Given that multiple verifiers can

Horizon 2020

TraceBot receives funding from the European Union's H2020-EU.2.1.1. INDUSTRIAL LEADERSHIP programme (grant agreement No 101017089)

participate in the verification of a single action, it is possible to obtain contradictory results. To address this issue, we first highlight the following remark. That is, if a verifier a True with a confidence of *C*, this means that it returns False with a confidence of *1-C* and vice-versa. This being said, given the different individual results from different verifiers for a single action, the confidence of probability of the overall verification decision being True is computed as well as for it be False. Then, the decision with the highest confidence is selected (i.e., argmax). If both decisions (i.e., True and False) come with the same confidence, then the final decision is returned as Undecided. This consolidation of individual verification results is illustrated by Figure 13.



$P^T(R = True) < P^T(R = False)$ $P^T(R = True) = P^T(R = False)$ $P^T(R = True) > P^T(R = False)$

$$P^T(R = V) = \sum_i P_i^T(R = V) \times C_i^T$$

Figure 13 Consolidating the verification results from individual verifiers for a given action or task *T*. P^T is the probability of having a decision for a given task *T*, $P^T{}_i$ is the probability of having a decision when using the knowledge modality *i* and $C^T{}_i$ is the capability of knowledge modality i to inform the verification of the action *T*.

Finally, the explanation of an action verification decision if the action is not atomic is the fact that all the sub actions were successful if the said action is successful, at least one of the sub actions failed if the said action failed and at least one of the sub actions was undecided if the action was undecided. For the atomic actions, the explanation is returned by the verifier as an output. In this case, the explanation derives directly from the verifier's intention and control flow, exactly as a compiler will generate error messages to explain while a program compilation failed. And if multiple verifiers are working in the verification, then the explanation is generated by concatenating the explanations from individual verifiers while using contrastive connectors for opposite decisions (e.g., but, however, although, ...) and supportive connectors for similar decisions (e.g., moreover, furthermore, additionally, ...). To conclude this section, let us describe succinctly the interfaces needed for this automated planning and execution of action verification tasks. Apart from the `Is_Successful (action_id)` and `Is_Feasible(action_id)` interfaces, as well as the templates of yaml configuration files interfaces, most of them are just getter counterparts to the setter interfaces discussed in the previous section.

`Is_Successful(action_id)`. This interface verifies whether a given action was successful and returns a boolean decision, a confidence score and a verbal explanation of the decision. `Is_Feasible(action_id)`. This interface verifies whether a given action is feasible and returns a boolean decision, a confidence score and a verbal explanation of the decision.

`Get_Participant(action_id)`. This interface returns the participants in terms of entities and roles of a given action.

Horizon 2020

TraceBot receives funding from the European Union's H2020-EU.2.1.1. INDUSTRIAL LEADERSHIP programme (grant agreement No 101017089)

`Get_Action_Execution_Time(action_id)`. This interface returns the execution timeslot of a given action.

`Get_Action_Tree(action_id)`. This interface returns the action tree of a given action.

`Inspect(entity_id)`. This interface returns the properties of a given entity from the NEEM (e.g., pose of an object, its color, …).

`Get_Experience(action_id, msg_type, nb_msgs)`. Finally, this interface retrieves from the NEEM experience a given number of experience samples of a certain given type after an action took place (e.g., one color image after the robot performed the insertion action). This is also crucial for the so-called non-verbal explanation of verification outputs in the audit trail.

3.2.9 Audit Trail Generation (WP5)

`Generate_Audit_Trail(audit_trail_path)`. As you can notice, NEEMs contain so much information that it will not be trivial for a human investigator to get at first glance an idea of what happened during the robot performance. The idea of the audit trail is to summarize in a human-understandable manner what the robot has done, when, who participated, how it went and why it went so. This interface generates the audit trail from a given NEEM and saves it as a pdf document to a specific given location. An overview of the audit trail's structure and content is illustrated by Figure 14. Note that the action hierarchy is encoded by each action's index but as well as the color of the corresponding row's background color.

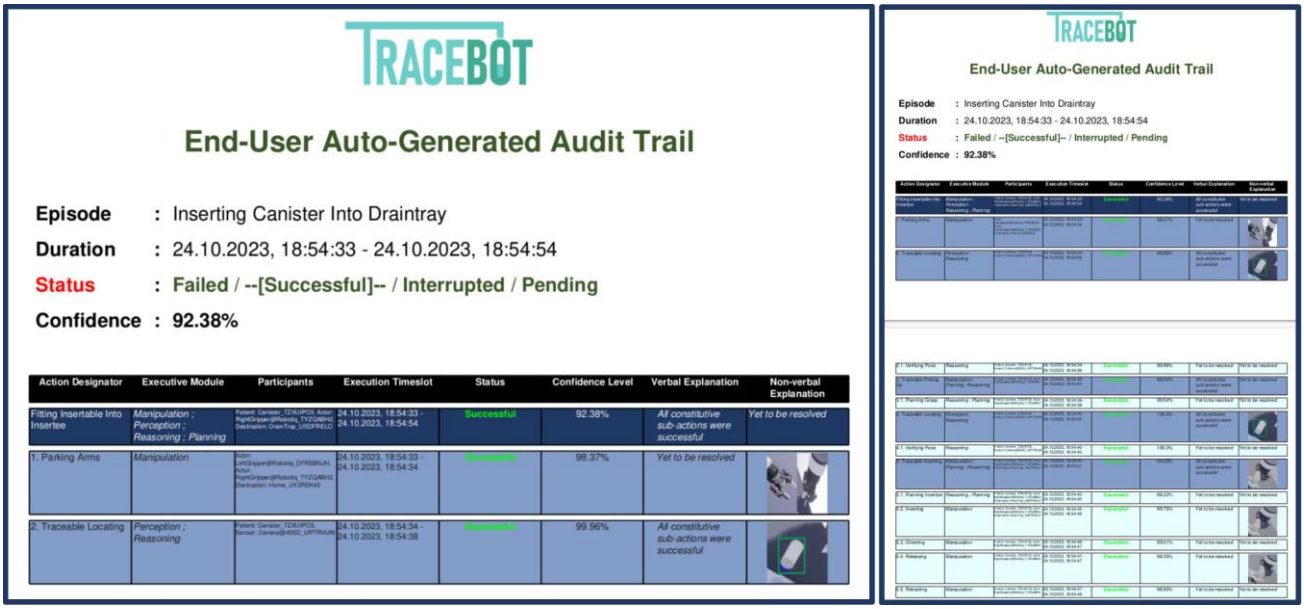


Figure 14 Audit trail of a fitting insertable (canister) into insertee (drain tray) action.

Horizon 2020

TraceBot receives funding from the European Union's H2020-EU.2.1.1. INDUSTRIAL LEADERSHIP programme (grant agreement No 101017089)

3.3 Simulation-enabled reasoning

The fundamental definitions and software modules of deliverable 5.2 are still valid. The developments during the last reporting period were focusing on the improvement of the simulation environment and the extension the software modalities to tackle the integration of these components with the real world system. We have to ensure that the used meshes resemble the real world objects, so that visual comparison between the real and simulated world can be used. The changes that were made include, among others, the different tables used by the different partners, as well as the pump as shown in Figure 15 and Figure 16.

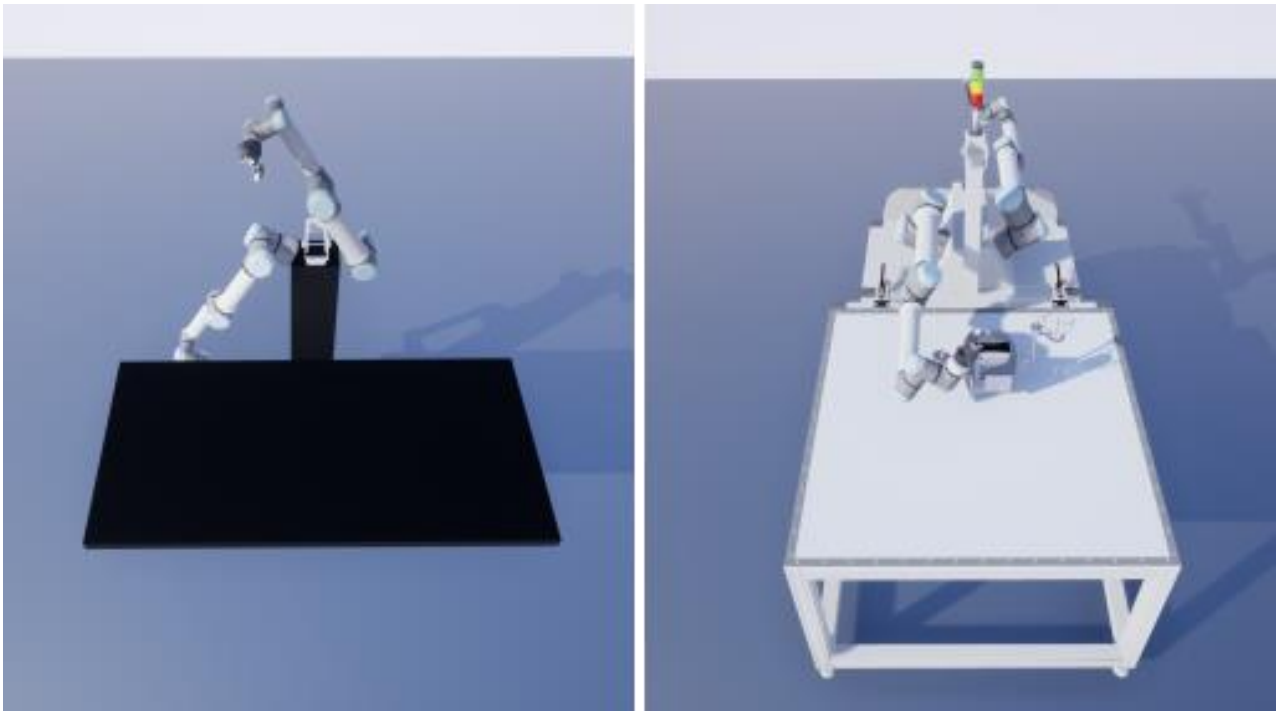


Figure 15 The digital twin of the current revision of the TraceBot demonstrator constructed by Tecnalia (left) and Astech (right) in our simulation component.

Horizon 2020

TraceBot receives funding from the European Union's H2020-EU.2.1.1. INDUSTRIAL LEADERSHIP programme (grant agreement No 101017089)

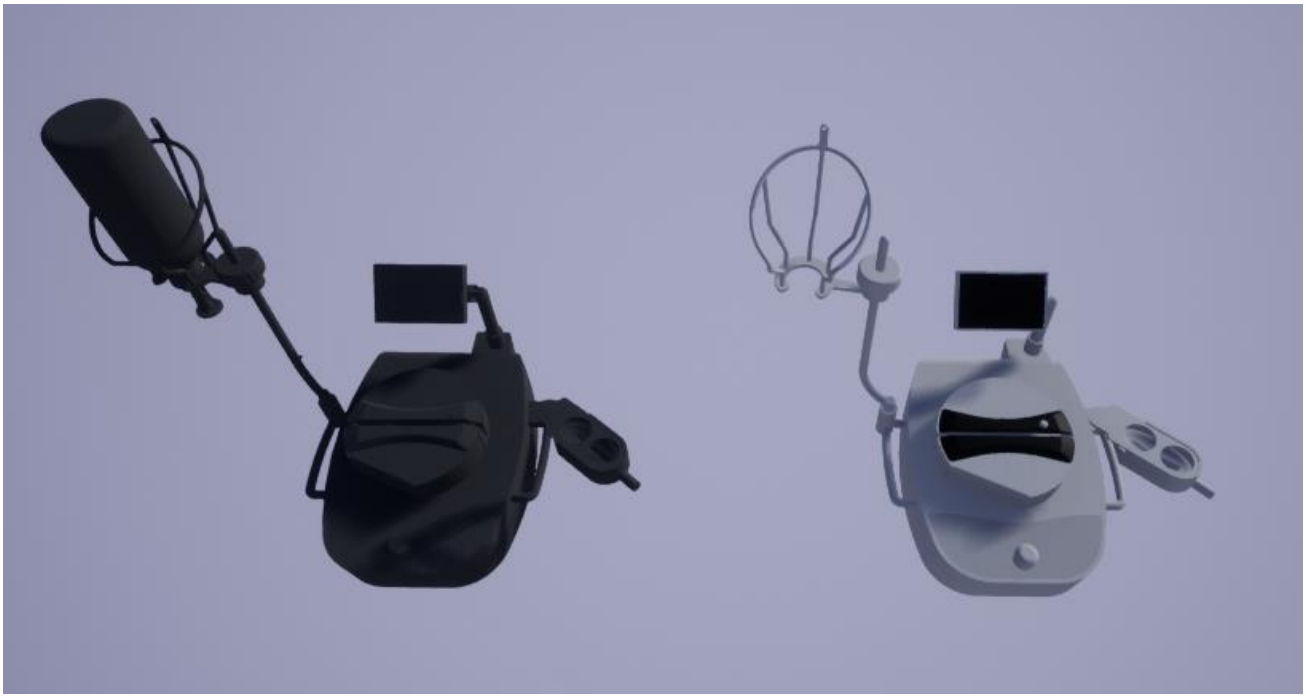


Figure 16 The digital twin of the old pump (left) and current pump (right) in our simulation component.

For the needle insertion use-case we finished the constraint-based object interaction. Figure 17 shows a schema for how this interaction works. They act independently, as long as needle and bottle are separate. As soon as the needle starts to overlap with the bottle, the linear constraint is activated, and the needle can only move linearly into or out of the bottle. If the needle is pulled out and the overlap stops, both objects will be independent again and the constraint is deactivated. If the needle is pushed further into the bottle, the fixed constraint is activated. Now, the needle cannot be moved. Only if the applied force exceeds a specified limit will the fixed constraint be deactivated. This functionality is a property of the needle only. This means it is possible to insert the needle into other objects, as long as they allow the needle to overlap with said object.

Horizon 2020

TraceBot receives funding from the European Union's H2020-EU.2.1.1. INDUSTRIAL LEADERSHIP programme (grant agreement No 101017089)

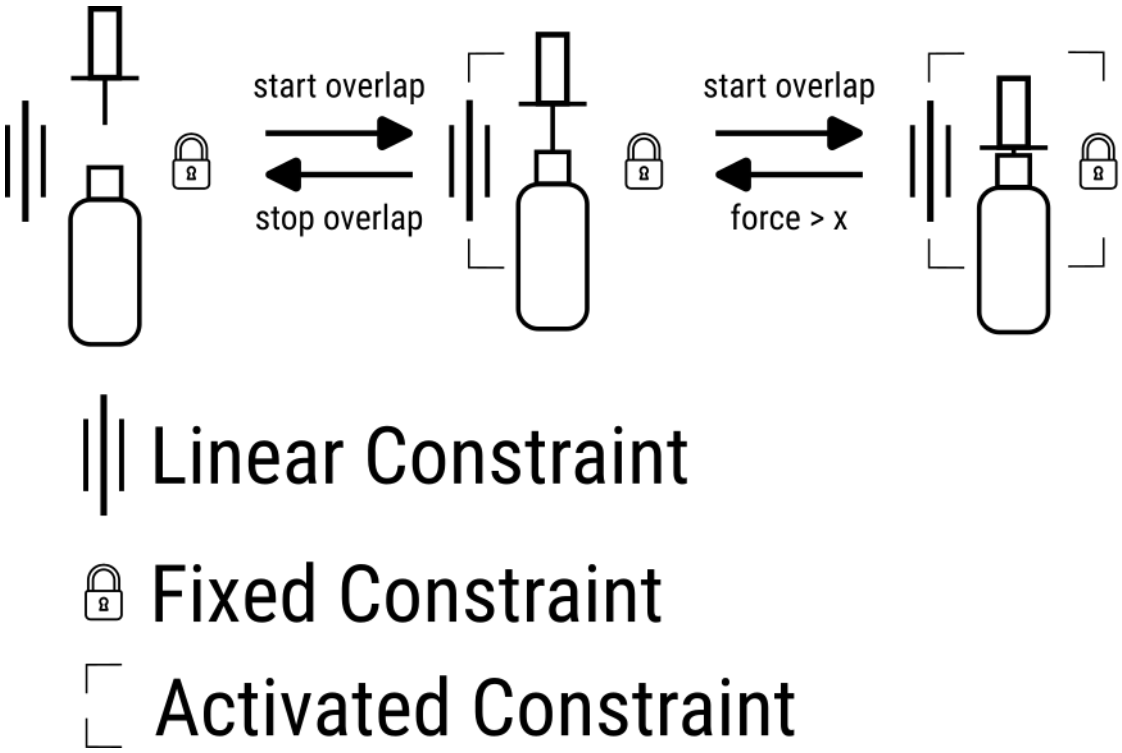


Figure 17 Schema of constraint-based object interaction. The Semantic Digital Twin supports different constraint types to model the attachment and interaction possibilities of objects during manipulation actions.

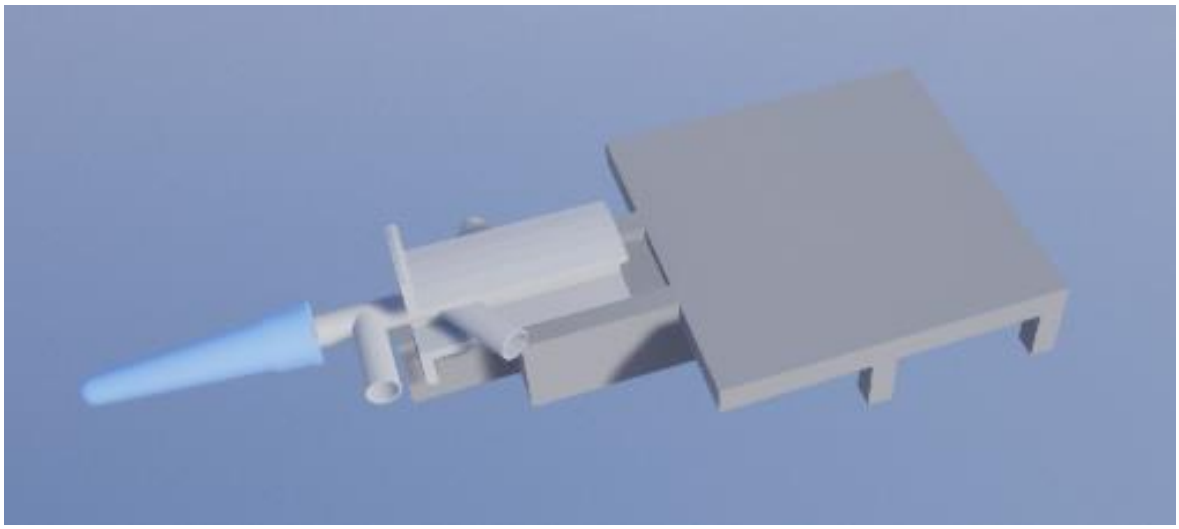


Figure 18 The digital twin of the current revision of the needle with support and needle cap in our simulation component.

In addition to the constraint-based object interaction, we needed to add the support of spawning compound objects. As shown in Figure 18, the needle is supported by a holder and is covered by a needle cap. To assert these objects into the semDT, the pose of those three objects must be known. Instead of trying to perceive their pose, and risking to introduce perception errors, we are using our knowledge of the relative poses to spawn them together, thus only needing the pose of one of the

Horizon 2020

TraceBot receives funding from the European Union's H2020-EU.2.1.1. INDUSTRIAL LEADERSHIP programme (grant agreement No 101017089)

objects. The different objects are connected through the constraint-based physics. Another important development was ensuring that the IDs used in the different components of the semDT and the rest of the system are consistent across representations.

3.4 Physical Reasoning based Embodied Probabilistic Simulations

We perform physical reasoning to safely handle the increasing uncertainty about the state of scene entities in realistic and mission-critical interactions, enabling therefore scene understanding. This physical reasoning is concretely realized through cognitive emulation where mental probabilistic embodied simulations are essential. We designed and published a framework coined as NaivPhys4RP (Naive Physics for Robot Perception) [3] (Humanoids 2022) to realize such physical reasoning. The architecture of NaivPhys4RP is presented by Figure 19. As applications of NaivPhys4RP to TraceBot are the points below:

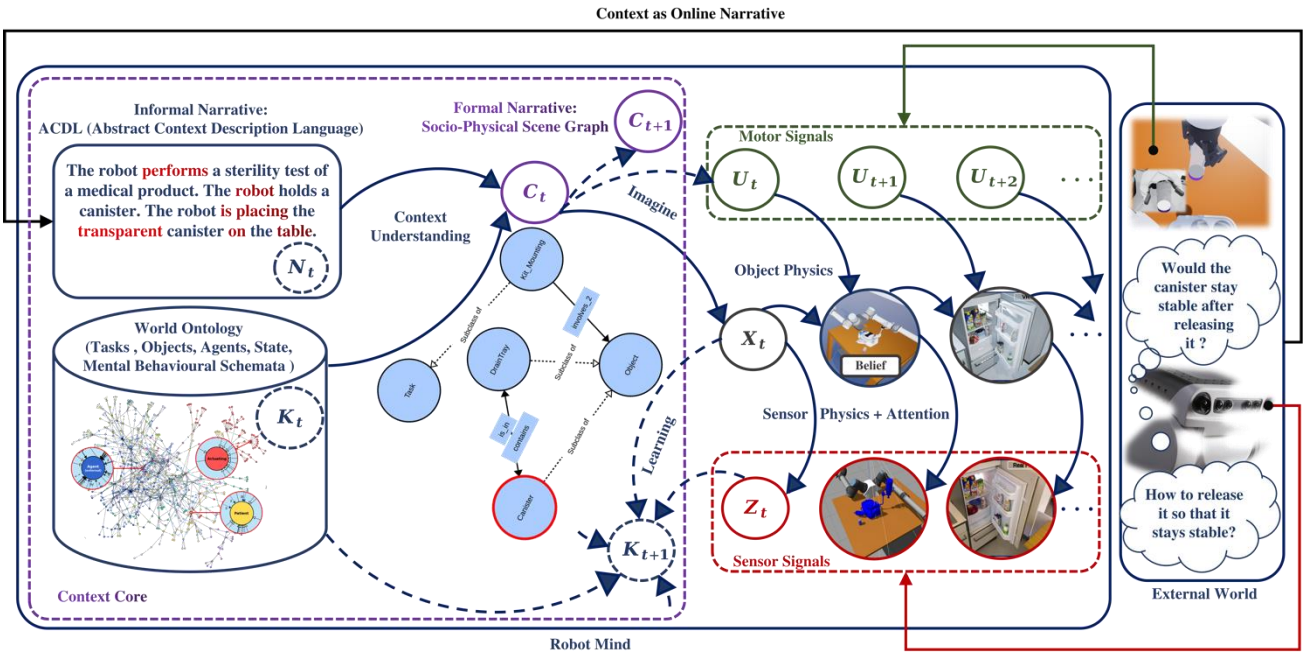


Figure 19 Sensors are severely limited in space, time and information quantity. Uncertainty about state of scene entities is increasing with interactions among these entities. This results in a lack of anticipation, poor learning from and therefore poor explanation of sensor data in complex worlds. Like humans, NaivPhys4RP (Naive Physics for Robot Perception) leverages commonsense to emulate how the world evolves in order to understand the world state under severe uncertainty. In this second iteration of NaivPhys4RP, after providing a complete first implementation of NaivPhys4RP, we demonstrate a learningless and safe recognition and 6D-pose estimation of objects from poor sensor data.

Reliable Simulation. It has been argued on how well emulation of robot actions as well as object interactions through physico-realistic simulation can significantly enhance reasoning in complex worlds such as in TraceBot. However, simulation is only reliable if it makes use of the correct physical parameters of the world such as surface friction, object masses, liquid density, which are most of the

Horizon 2020
TraceBot receives funding from the European Union's H2020-EU.2.1.1. INDUSTRIAL LEADERSHIP programme (grant agreement No 101017089)

time unknown and whose determination constitutes a more complex perception problem since there are no sensors that provide significant information for a straightforward computation of these physical quantities. This leads us then to a chicken-egg dilemma. NaivPhys4RP regards this issue as a perception problem and filters these physical quantities by not just realizing one simulation but rather a many parameterized simulations called simulation particles and promoting simulation particles that produce effects closer to actual real effects. The physical parameters of such simulation particles are considered as the ones of the real world.

Safe and Learningless Recognition and Pose Estimation of (Transparent) Objects. Though emulating the interactions of objects in the world through physico-realistic simulation enables the correction of objects' poses or the detection of wrong object classification, still it requires the classical perception to produce the first results and the overall system remains then bounded by the capability of the classification system, which itself suffers from high uncertainty from very lacking sensor data (poor learning due to high entropy training data). NaivPhys4RP addresses this issue by overcoming sensor data while generating and executing multiple emulations of very likely socio-physical (objects+interactions) scenes in simulation. This generation of world is transparent and causal and relies on capturing commonsense information that drives the organization of daily scenes such as intentions, preferences, object and event ontologies, and teleology (e.g., if the robot is serving milk, then there should probably be a coffee, a spoon and a milk bottle in the scene). Then, the most representative simulation particles are filtered over time as explained in the first point. We showed in a recently submitted paper [2] (ICRA 2024) how NaivPhys4RP can be used to safely recognize transparent objects and estimate their 6D-poses without data- and resource-intensive learning.

Fine-grained and Prospective Verification. Finally, we intensively discussed the crucial role of pre-conditions and post-conditions of actions in verifying those actions. However, these conditions have only so far been described symbolically, which remains coarse and not fine-grained enough to inform us about the actual physical effects if we were to perform those actions. For instance, a grasp action's pre-condition will require the gripper to be free and the canister object to be located, but it does not tell whether the object is reachable or which force range should the gripper apply on the canister for a stable grasp. On the other hand, symbolic descriptions of post-conditions will state that the canister should be in the robot gripper, but the canister can be effectively in the robot gripper but broken. Or, to refer to the previous example for serving milk, the milk bottle might be misplaced on the table causing the milk to spill while placing. NaivPhys4RP does not only engage in providing such fine-grained pre- and post-conditions but also allows the robot to envisioned in advance the consequences of a given set of conditions and only engages with those that produce satisfiable results: this is regarded as prospective verification (i.e., verifying in advance).

Note that in order to compute all these three problems, NaivPhys4RP fundamentally frames the world state as SPOHMP (Situated Partially-Observable Hidden Markov Process) that evolves within a given context and through the physics that scene entities undergo. Then, it formulates these scene understanding tasks as bayesian/markovian inference tasks that are solved through a commonsense-enabled particle filter, where particles are instances of a simulation. We proposed in this second iteration [2] (ICRA 2024), beyond text-based interfaces (c++/python function, config yaml files), a graphical user interface to interact with NaivPhys4RP, which is illustrated by Figure 20. In the final iteration i.e., last year of the project, the text-based interface will be completely and properly wrapped into KnowRob language specified in Deliverable 5.2.

Hyper–Parameterization: As shown by Figure 20, NaivPhys4RP's GUI is organized in tabs where the first tab enables the hyper-parameterization of the system. This first tab allows the users to enter

Horizon 2020

TraceBot receives funding from the European Union's H2020-EU.2.1.1. INDUSTRIAL LEADERSHIP programme (grant agreement No 101017089)

information about the target robot model and the streams to proprioceptive sensors (e.g., motion) of the robot, characterizing the system variable U_t. It also provides a way to enter information about the exteroceptive sensor models as well as streams to those sensors, characterizing the variable Z_t. Then, a possibility to enter information about the world ontology K_t as well as the language model for describing the robot intentions or activity in terms of narrative N_t is also provided. Finally, the GUI also enables the input of information about the physical model of the world X_t as well as the number of parallel worlds to maintain to represent uncertainty as it is the case in the quantum world.

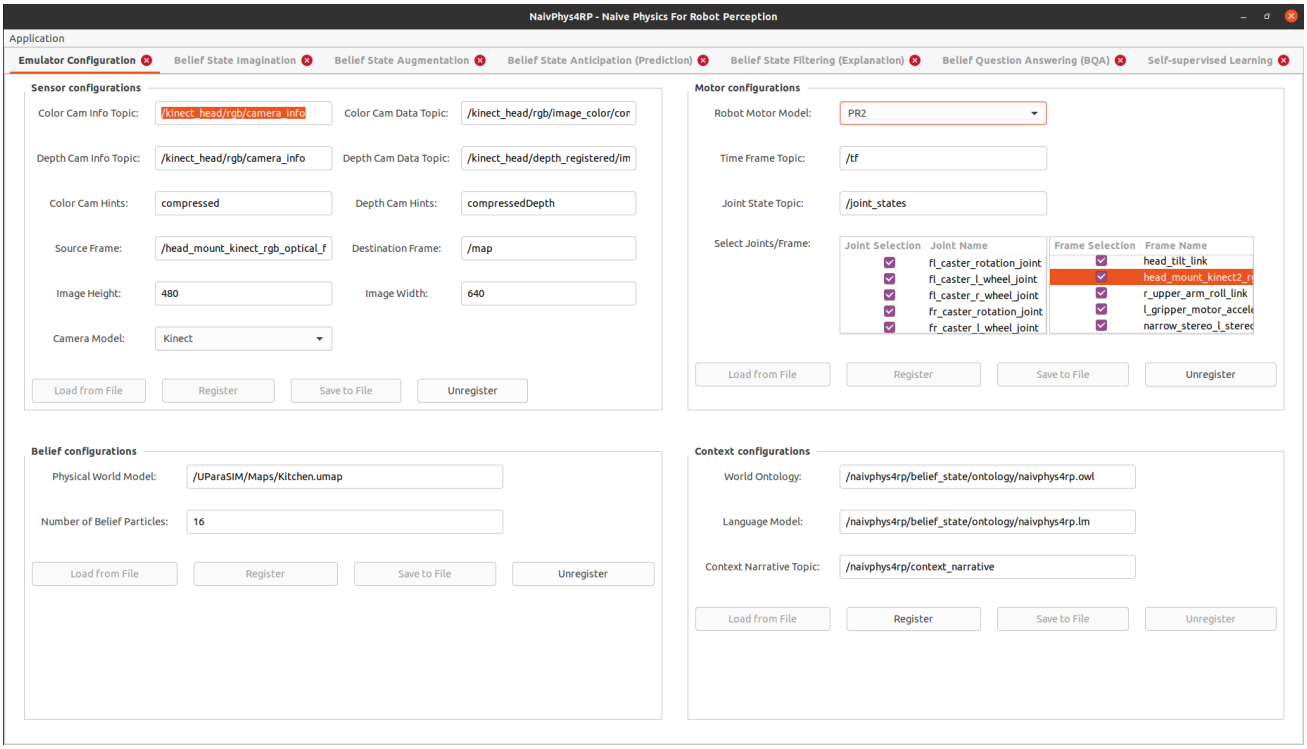


Figure 20 Graphical User Interface to hyper-parameterization of NaivPhys4RP

$P(C_t|N_t, K_t, C_{t-1})$: *Context_Understanding*(N_t, K_t, C_{t-1}). Once the system has been parameterized, this interface, as shown in Figure 20 and Figure 21, takes as input the world ontology $K_t$at a certain time *t*, the robot intentions or activity description in terms of narrative N_t (see the text field) and the previous socio-physical graph of the scene $C_{t-1}$to compute through commonsense-enabled sampling the next most likely and statistically sufficient socio-physical graph of the scene C_t. Figure 23 shows how NaivPhys4RP can handle nonsense discource and Figure 21 and Figure 22 show how NaivPhys4RP can generate a scene from a narrative of the robot activity or intention. Note that NaivPhys4RP does not only sample as many graphs as worlds, but also provides the sampling probability (low-probability graphs will die) and explanations of these generations (see points 1., 2., 3., and 4., of Figure 21).

$P(X_t, U_t|C_t)$: *SocioPhysical_Scene_Imagination*(C_t). Once the graphs have been generated, the corresponding scenes are straigthforwardly imagined from them in terms of object configurations and interactions (e.g., robot holds the bottle), since the graphs are sufficiently detailed (note the statistical sufficiency of these graphs mentioned earlier). This results in a portion of scene $X_t$and robot action $U_t$as shown at the bottom of Figure 21 and Figure 22. Figure 22 particularly highlights the generation

Horizon 2020

TraceBot receives funding from the European Union's H2020-EU.2.1.1. INDUSTRIAL LEADERSHIP programme (grant agreement No 101017089)

of robot motions from the generated socio-physical graph of the scene.

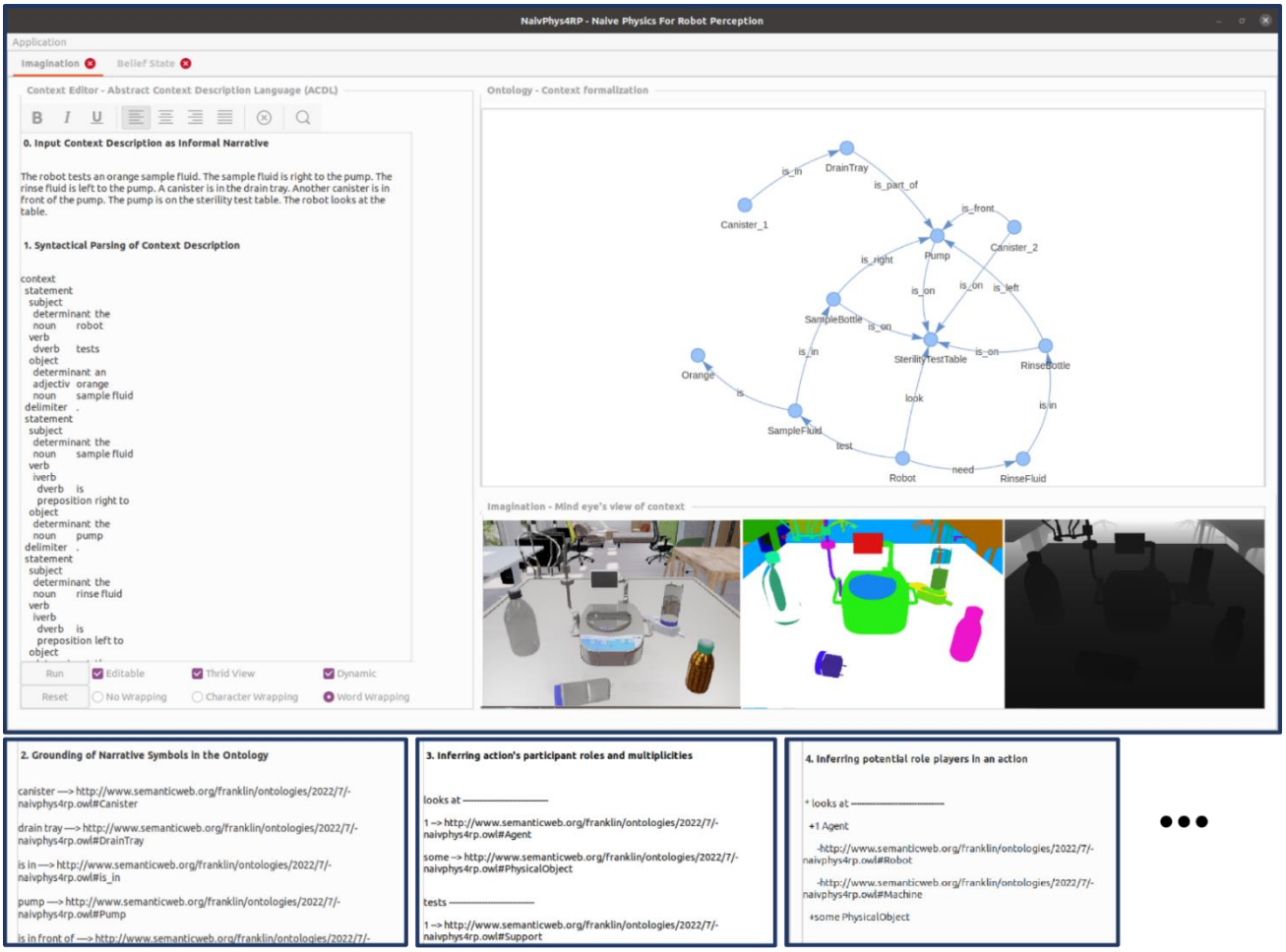


Figure 21 Relying on context-aware ontology to generate scenes in a transparent and causal manner from vague context descriptions and to support generative scene understanding

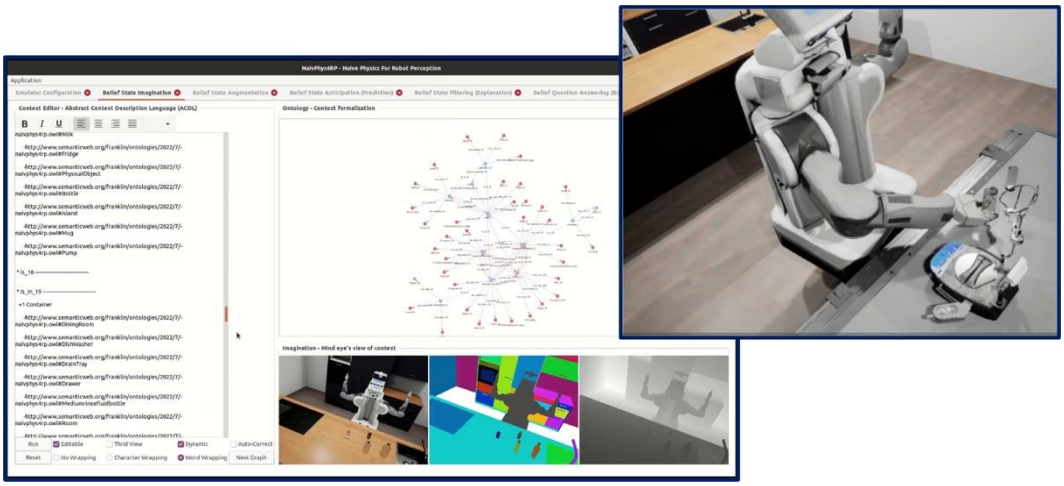


Figure 22 Generation of robot behaviours in NaivPhys4RP

Horizon 2020

TraceBot receives funding from the European Union's H2020-EU.2.1.1. INDUSTRIAL LEADERSHIP programme (grant agreement No 101017089)

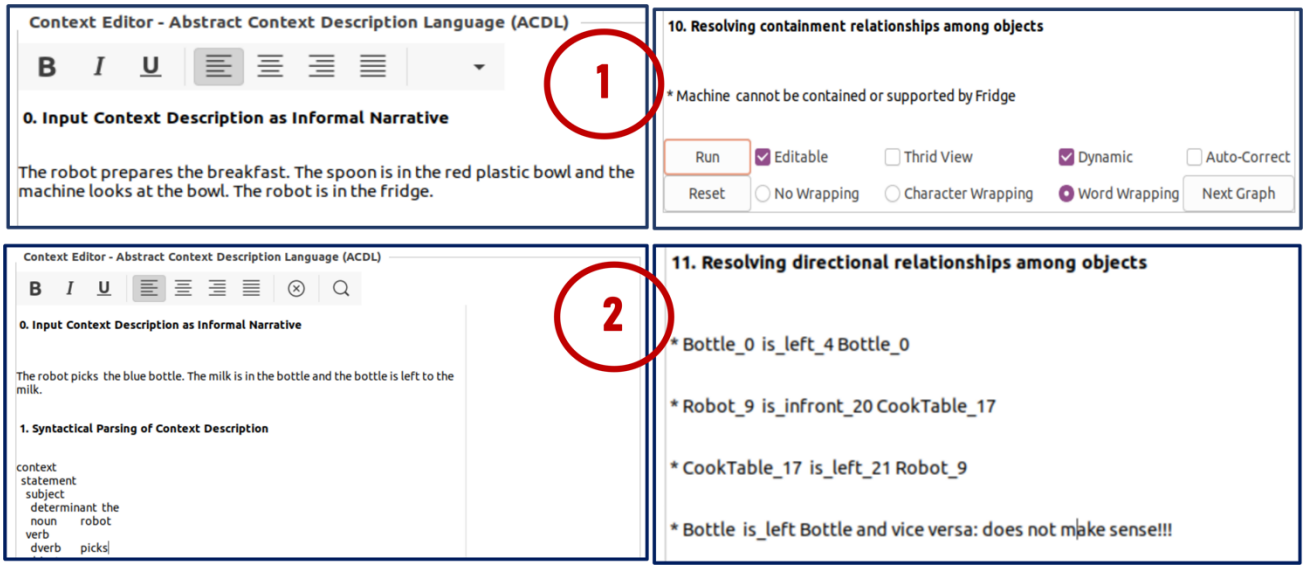


Figure 23 NaivPhys4RP handling nonsense discourse. (1) NaivPhys4RP claims that the robot cannot be kept in the fridge. (2) NaivPhys4RP explains that if the milk is in the bottle and the bottle is left to the milk, then that bottle is left to the container of the

$P(X_{t+1}|U_t, X_t, C_{t+1})$: *State_Anticipation*$(U_t, X_t, C_{t+1})$. Figure 24 illustrates how this NaivPhys4RP's interface continuously emulates the robot actions as well as object interactions to anticipate the most likely state of the world. Very important here is the illustration of this multiple worlds maintained by the system to represent the uncertainty the robot has of the world. This state anticipation results in a next state X_{t+1}.

$P(Z_{t+1}|X_{t+1})$: *Observation_Anticipation*(X_{t+1}). As also shown by Figure 24, by realistically rendering these maintained physico-realistic scenes, this interface can anticipates the robot observation of the world Z_t (e.g., how would the table look like if the bottle was to fall?).

$P(X_{t+1}|U_t, X_t, C_{t+1}, Z_{t+1})$: *Observation_Explanation*$(U_t, X_t, C_{t+1}, Z_{t+1})$. Once multiple worlds have been generated and realized, this interface filters or preserves the most likely ones based on how closer the effects resulted in are to the real ones. In order to compare these mental worlds with the real ones, we do not proceed in a straightforward way (e.g., very ineffective), but rather rely on Gestalt principles. Figure 25 illustrates how we recognize and estimate the pose of transparent objects.

$P(X_{t+1}, U_t|U_{t+1}, C_{t:t+2}, X_t, X_{t:t+2}, [Z_{t:t+2},])$: *State_Explanation*$(U_{t+1}, C_{t:t+2}, X_t, X_{t:t+2}, [Z_{t:t+2},])$. This is basically the global problem of maintaining awareness of the state of the world over time given all priors and evidences. We showed that it can be computed from the computation of the first five tasks.

$P(X_t|K_{0:t}, N_{0:t}, Z_{0:t}, U_{0:t-1})$: *State_Filtering*$(K_{0:t}, N_{0:t}, Z_{0:t}, U_{0:t-1})$. Fundamentally, this interface is concerned with what actions would cause a state (e.g., what grasp force range will cause stable grasp of canister?, which range will damage it?). Likewise, we showed that it can be computed from the computation of the first five tasks.

Horizon 2020

TraceBot receives funding from the European Union's H2020-EU.2.1.1. INDUSTRIAL LEADERSHIP programme (grant agreement No 101017089)

$P(K_{t+1}|K_t, N_t, Z_{t,}, X_t, U_{t-1}, C_t)$: *Learning* $(K_t, N_t, Z_{t,}, X_t, U_{t-1}, C_t)$. This interface is responsible for updating however carefully the abstract and stable structure of the world (i.e., ontology) K_{t+1}as the robot gains more and more experience. Actually, this update is mostly performed manually by observing the NEEMs and by providing templates of yaml configuration files to add knowledge into the ontology (e.g., the robot's preferences of locations of objects as it manipulates them). An automation of this interface will be targeted in the next development cycles.

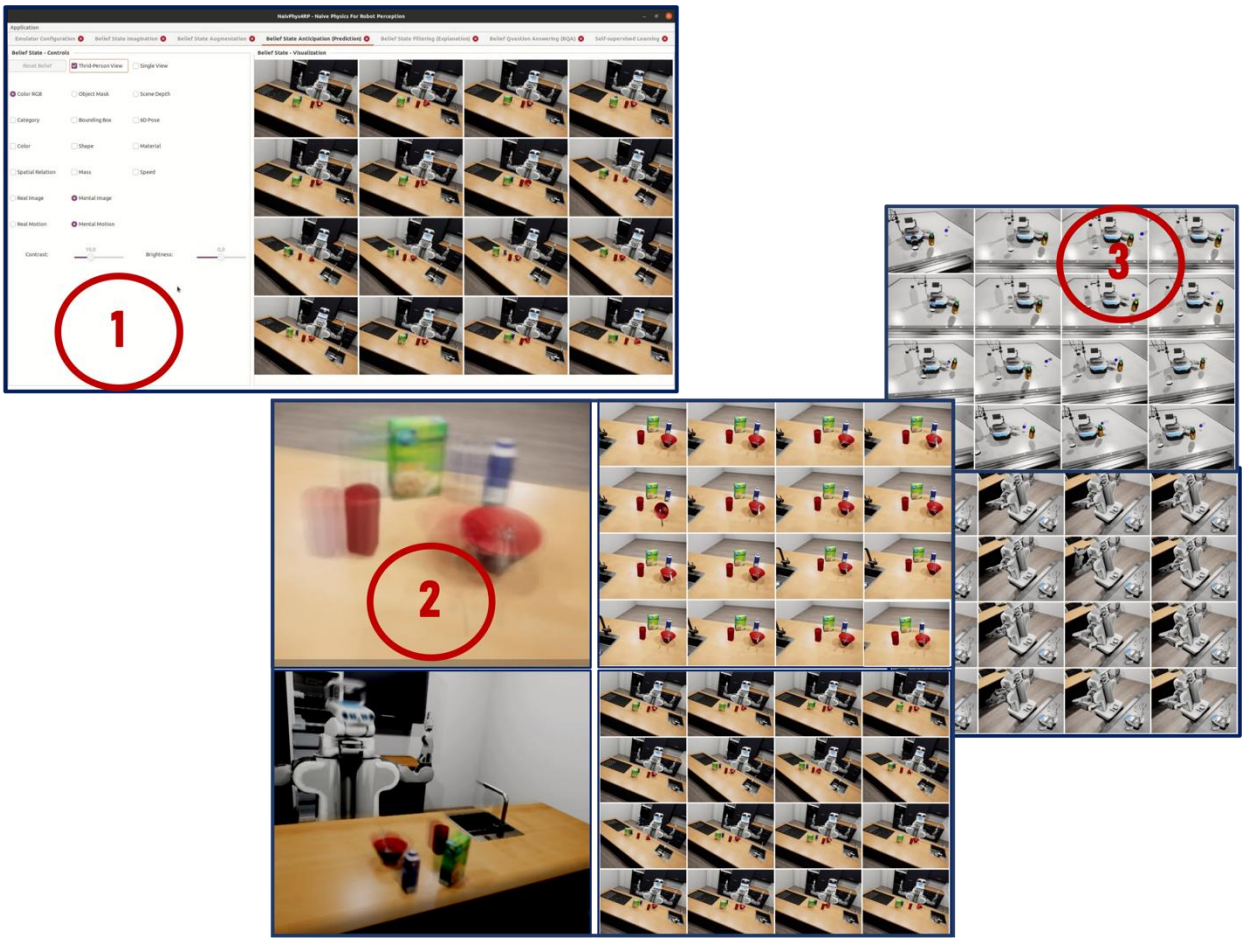


Figure 24 (1) Overview of the NaivPhys4RP's interface responsible for anticipating the world observation and state. Note multiple simulation particles to represent uncertainty about real world state. (2) Illustration of belief state in the kitchen domain from a third person view in single-display mode (bottom-left), multi-display mode (bottom-right), and from a first person view in single-display mode (top-left) and multi-display mode (top-right). (3) Illustration of the belief state in the medical laboratory domain.

Horizon 2020

TraceBot receives funding from the European Union's H2020-EU.2.1.1. INDUSTRIAL LEADERSHIP programme (grant agreement No 101017089)

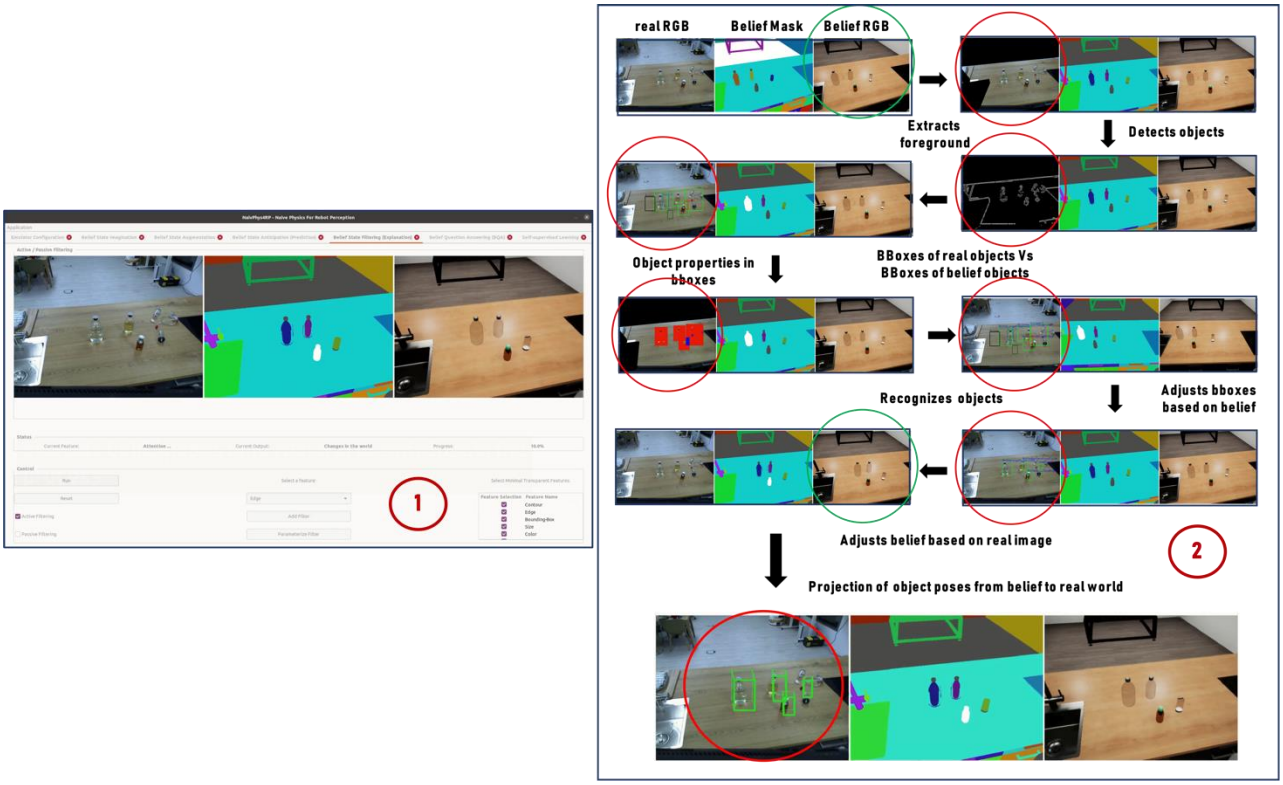


Figure 25 (1) Overview of NaivPhys4RP for observation explanation through comparison between imagination and reality. (2) Illustration of comparison between imagination and reality based on Gestalt principles for recognition and pose estimation of transparent objects.

3.5 Perception Executive

In this section of the deliverable we are presenting the changes in the perception executive. The presented interface from our last deliverable 5.2 is not changed. However, we are extending the definition of the underlying software model to provide a description on how the interface is mapped to an actual execution of perception-related tasks like for example imagistic reasoning or object detection.

One insight was that the perception tasks are changing over the course of the task execution but also the underlying mechanisms to analyze the sensor data against the estimated rendered world state requires different methods. The order in which they are executed needs to be adaptable and handle potential dependencies. Furthermore, it was necessary to develop a capable model for perception task execution that can adapt during runtime not only to the different tasks but also to different parameterizations of a perception process. A solution for this problem that we have developed are so-called Perception Pipeline Trees (PPTs). We introduced the concept in a publication [4] that is currently under review for ICRA 2024.

Horizon 2020

TraceBot receives funding from the European Union's H2020-EU.2.1.1. INDUSTRIAL LEADERSHIP programme (grant agreement No 101017089)

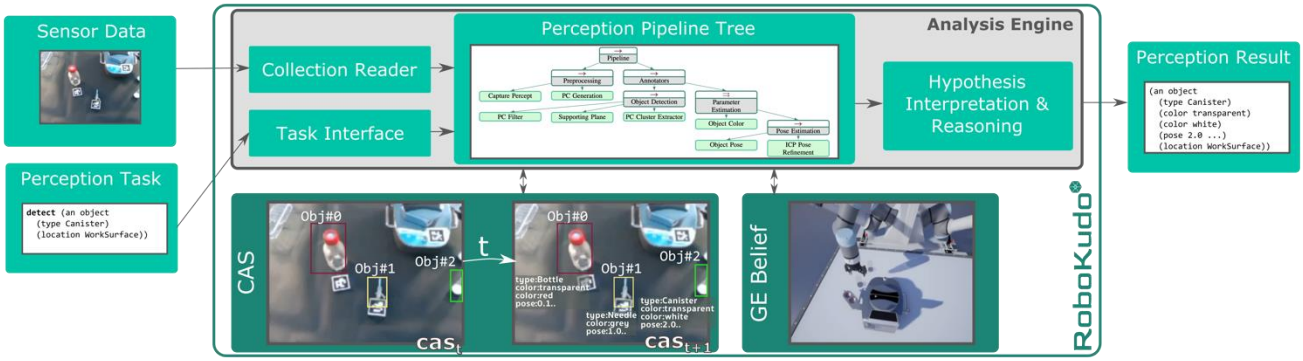


Figure 26 Architecture of our task-adaptable perception executive system. Perception tasks will be mapped into a task model called Perception Pipeline Tree, which is based on Behavior Trees. Gained information is annotated to a common, typed data structure which is used to infer the final result.

Consider the general architecture shown in Figure 26. When confronted with input sensor data and a query for a perception task, our proposed system is reasoning about a suitable perception process for the task at hand and adapts the perception process at runtime. One key component is an extension of Behavior Trees, that have been used in game development for reactive character control. In our framework, we extended Behavior Trees to PPTs which allow us to cover a wide variety of perception tasks and imagistic reasoning mechanisms. We have chosen Behavior Trees as our task model since they are a representation with concise semantics that allows to flexibly switch between tasks. BTs have received a lot of attention in robotics in the last years due to their flexibility, simplicity, generality and expressiveness with a small set of core elements. A key component inside of the architecture beside the PPT is the so-called Common Analysis Structure (CAS). The CAS is a central storage place which allows the different computer vision methods, that are used to analyze the images, to exchange their data during runtime. This allows for example to first detect objects in the image with the computer vision methods provided by TU Vienna and to afterwards apply to each detected object a color filter to detect for example if a needle cap is currently attached to a detected needle. Alternatively, one could also employ zero-shot vision models for a similar purpose.

A key aspect of the PPT representation is also the ability to have different perception processes that can be combined in one representation. Take for example the perception process of detecting an object and afterwards verifying if the object that has been detected looks similar to your rendered belief of it. In the first step one would model a standard object detection pipeline which perceives the sensor data, employs an object detection method and then asserts the detected object to the internal belief state. In the second step the system needs to synchronize the belief state with the digital twin and then perceive from the digital twin the rendered belief. Afterwards the rendered belief will be segmented to detect all the objects and their positions in the image frame of the rendered belief. Since we maintain a relation between each detected object in the real world and its rendered counterpart, we can now compare both images depicting the objects against each other on the pixel level. For visual comparison it is possible to use for example appearance-based features that can regress the object's appearance into a latent space representation, or you can also employ semantic knowledge. In the latter case, one could model a comparison method based on the expected properties of an object. For example, you expect to have a blue cap on the needle during the beginning of the process and compare the existence of this feature in both images.

In the following figures we will highlight two PPTs that are related to the imagistic reasoning tasks we are conducting in TraceBot. When describing the elements in the PPT it is often useful to be familiar

Horizon 2020

TraceBot receives funding from the European Union's H2020-EU.2.1.1. INDUSTRIAL LEADERSHIP programme (grant agreement No 101017089)

with the concept of Behavior Trees. We refer the reader to an excellent introduction in [1] to get an overview of the Behavior Tree concept.

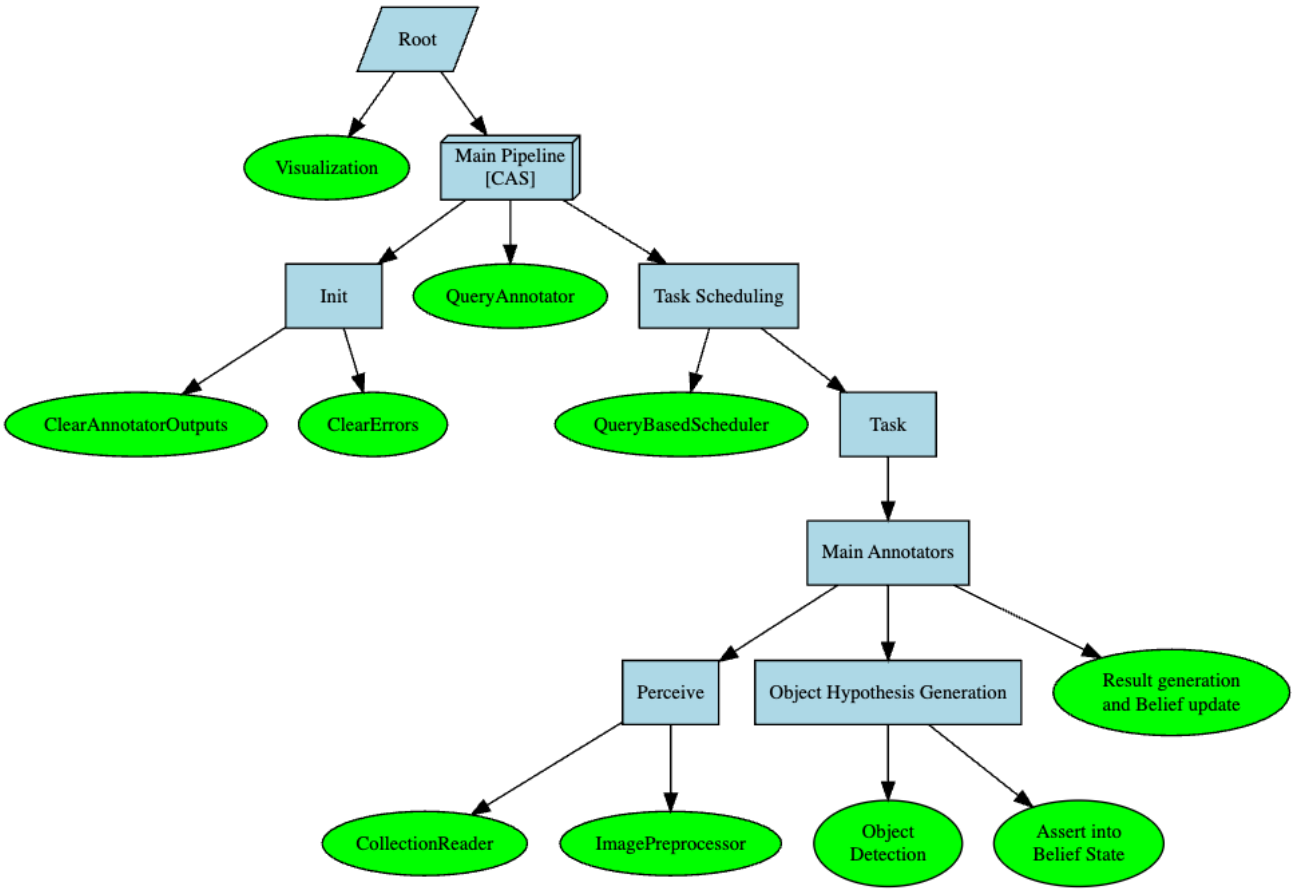


Figure 27 Perception Pipeline Tree for object detection. The top of the tree contains utility nodes for visualization and query analysis for the perception process instantiation and parametrization.

In the first tree (see Figure 27) you can see how object detection is modelled when the system is looking for objects initially. The top of the tree is common for the other perception processes as well and is handling the visualization capabilities of the system that are used by the developer during the creation of the model and the usage of the system for result analysis. The second main part is the related to the input query, which contains the perception task stated by the high-level of the system. The main perception process is then placed below the Task node. The object detection is a sequence of perceiving the actual data and the application of the object detection model that is also doing 6D pose estimation in the context of TraceBot. The result of this step is then asserted into the robots' belief state about the objects in the world. After the sequence is successfully completed, the tree is basically starting over and waiting for the next command. It also updates the visualization component to show the generated results of the different employed computer vision methods.

In the second tree depicted in Figure 28, we extend the object detection process by a second pipeline for imagistic reasoning. This new pipeline is first devoted to the update of our game engine-based belief state and secondly the comparison of the detected object and their rendered virtual counterparts. As you can see in Figure 28, it is possible to add another pipeline as a child of other

Horizon 2020

TraceBot receives funding from the European Union's H2020-EU.2.1.1. INDUSTRIAL LEADERSHIP programme (grant agreement No 101017089)

nodes in the perception pipeline tree. This allows semantically different subprocesses to be properly isolated and also Annotators to be observation-specific and task-centric during their analysis[2].

In the Main Annotators node, we have added the GE pipeline, where GE stands for Game Engine. In contrast to the pipeline which is devoted to the detection of objects, we first have to conduct an update step before the actual comparison can be done. In the beginning, the pipeline needs to initiate the synchronization of the belief state and the virtual world in the game engine. This results in either a) the addition of newly detected objects into the virtual world representation or b) in the update of the detected objects and their poses in the virtual world. After the virtual world in the game engine is set, we can finally perceive from it. Perceiving in this context is the retrieval of a rendered image of our belief state that is set up in the game engine. After this rendered data has been read in, we need to segment each of the objects shown in the rendered images to know exactly which region of this image belongs to a certain object.

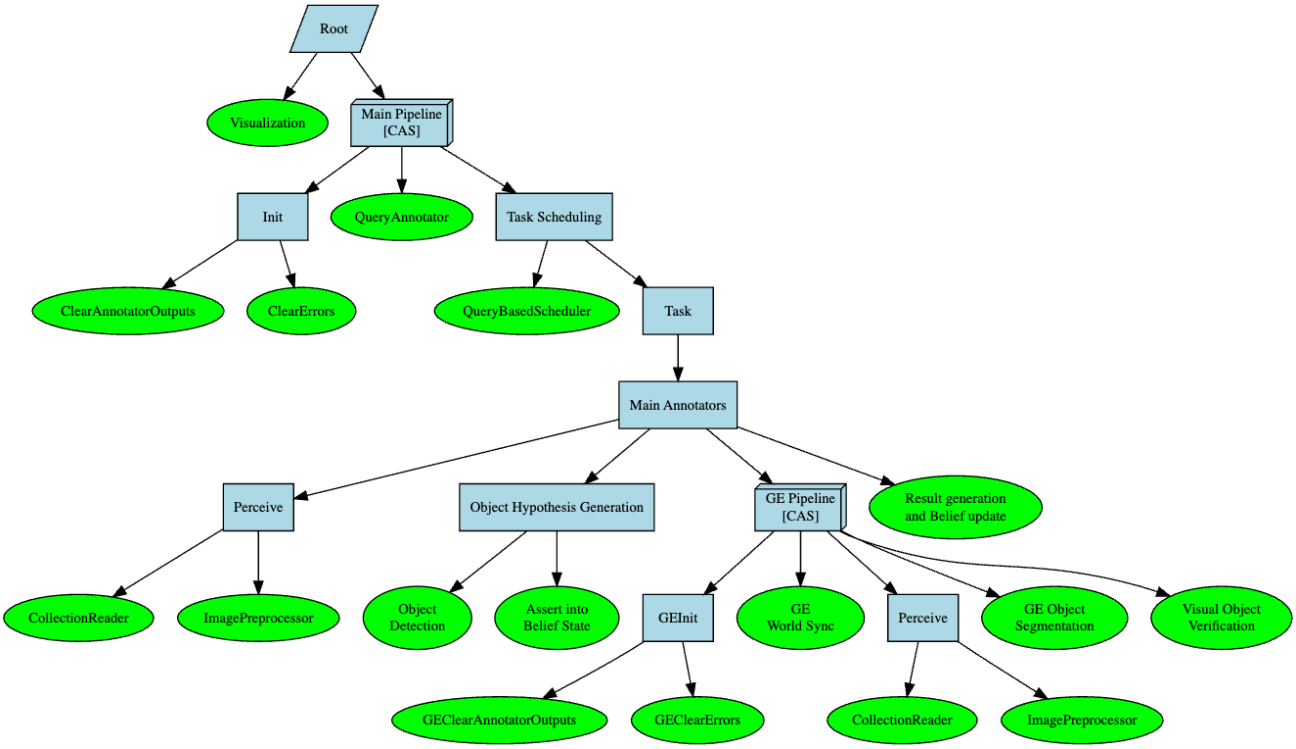


Figure 28 An extended Perception Pipeline Tree which contains a second pipeline for imagistic reasoning in the visual domain.

In the final step, we can now take the real world image on which the object detection has been done and then the rendered image from the game engine showing the estimated belief of the world. Since we have a location for both objects in image coordinates, we can now do a comparison on the pixel

[2] The details of these differences are outlined in the Paper mentioned above. The key idea is: Observation-specificity allows a Pipeline in the PPT to focus solely on the expected sensory inputs. For example, data from an RGB-only hand camera will be processed by its own Pipeline that will only employ algorithms on RGB data. A pipeline is task-centric if it explicitly models the subprocesses of a perception task for a specific semantic purpose.

Horizon 2020

TraceBot receives funding from the European Union's H2020-EU.2.1.1. INDUSTRIAL LEADERSHIP programme (grant agreement No 101017089)

level between both sub-images. This could, for example, be done with an apperance-based matching method.

In PPTs, it is possible that nodes return a failure and therefore abort the execution of the rest of the tree. If, for example, the object verification has detected a faulty object detection (e.g. wrong class), it can now generate a signal and tell the caller that the object detection is likely to have failed and that a re-perceive is necessary or abort the whole sterility testing process because manual intervention is required. If the object detection and the visual object verification has been done successfully, the system can now generate the result about the detected object and its relevant information for the Process Master and then update the belief of the robot as necessary. After this step has been done, the system is waiting for the next perception task to be sent by the Process Master.

Horizon 2020

TraceBot receives funding from the European Union's H2020-EU.2.1.1. INDUSTRIAL LEADERSHIP programme (grant agreement No 101017089)

4 Deviations from the workplan

No major deviation has been detected, and the document has been delivered on time.

Horizon 2020

TraceBot receives funding from the European Union's H2020-EU.2.1.1. INDUSTRIAL LEADERSHIP programme (grant agreement No 101017089)

5 Conclusion

This document represents a significant advancement in the TraceBot project, building upon the foundational work outlined in our previous deliverable 5.2. We have successfully addressed more complex use cases through key modifications and enhancements, particularly in the domains of knowledge representation, reasoning, and simulation within the Semantic Digital Twin (semDT) framework. Our focus on creating comprehensive interfaces for interaction among various components, especially those under Work Package 5, has been beneficial in improving the system's functionality and adaptability in our scenarios.

A notable development in this iteration is the introduction of audit trail interfaces as well as the interfaces to the verification framework, whose outputs are essential in the audit trail. These interfaces are crucial for ensuring transparency, traceability, and accountability in all operations within the TraceBot ecosystem. Furthermore, our approach to handling compound objects—such as a needle with a cap—demonstrates our system's enhanced capability to deal with complex, real world objects that require delicate manipulation and interaction. This ability is essential for tasks like sterility testing, where precise handling and tracking of objects are critical.

Our integration-driven approach has led to substantial improvements in the software models, particularly in the simulation aspect, enabling the updating of the semDT in real-time, irrespective of the robot platform used. These improvements are not just theoretical but have been validated through practical application in the system integration phase. The enhanced handling of compound objects and the implementation of a model that constrains one object to another highlight our commitment to modelling real world manipulation tasks and their respective challenges. We have also provided models for imagistic reasoning tasks that are capable of representing computer vision pipelines with additional components for visual verification based on our semDT technology.

In conclusion, the enhancements achieved provide a strong basis for our ongoing endeavours in the TraceBot project. By handling more robot configurations, manipulation types, object features and providing suitable representation and reasoning techniques, we are looking forward to the upcoming developments and challenges we are focussing on in the next period of the TraceBot project.

Horizon 2020

TraceBot receives funding from the European Union's H2020-EU.2.1.1. INDUSTRIAL LEADERSHIP programme (grant agreement No 101017089)

6 References

[1] Michele Colledanchise and Petter Ögren. Behavior trees in robotics and AI: An introduction. CRC Press, 2018.

[2] Franklin K. Kenghagho, Michael Neumann, Patrick Mania, and Michael Beetz. Perception through cognitive emulation: “a second iteration of naivphys4rp for learningless and safe recognition and 6d-pose estimation of (transparent) objects”. In 2024 IEEE International Conference on Robotics and Automation (ICRA) (Submitted).

[3] Franklin K. Kenghagho, Michael Neumann, Patrick Mania, Toni Tan, Feroz A. Siddiky, René Weller, Gabriel Zachmann, and Michael Beetz. Naivphys4rp - towards human-like robot perception “physical reasoning based on embodied probabilistic simulation”. In 2022 IEEE-RAS 21st International Conference on Humanoid Robots (Humanoids), pages 815–822, 2022.

[4] Patrick Mania, Simon Stelter, Gayane Kazhoyan, and Michael Beetz. An Open and Flexible Robot Perception Framework for Mobile Manipulation Tasks. In 2024 IEEE International Conference on Robotics and Automation (ICRA) (Submitted).

Horizon 2020

TraceBot receives funding from the European Union's H2020-EU.2.1.1. INDUSTRIAL LEADERSHIP programme (grant agreement No 101017089)