Project Acronym:	ACAT
Project Type:	STREP
Project Title:	Learning and Execution of Action Categories
Contract Number:	600578
Starting Date:	01-03-2013
Ending Date:	30-04-2016

Deliverable Number:	D4.4
Deliverable Title:	Action Execution
Type (Internal, Restricted, Public):	PU
Authors :	Barry Ridge, Andrej Gams, Aljaž Kramberger, Aleš Ude, Dimitris Chrysostomou, Minija Tamosiunaite, Florentin Wörgötter, Michael Beetz, Daniel Nyga, Gheorghe Lisca
Contributing Partners:	JSI, AAU, UGOE, SDU, UoB

Contractual Date of Delivery to the EC:	29-02-2016
Actual Date of Delivery to the EC:	01-03-2016

Contents

1 Executive Summary

This deliverable provides a report on the procedures, structures and the actual qualitative performance of action execution in ACAT. The first part of the deliverable after the introduction, Section 3, describes how action execution is performed by the execution engine in the IASSES scenario with the AAU system, where one of the main tasks is to deconstruct a linguistic instruction, typically for picking a rotor cap from a conveyor and placing it on a fixture, into its constituent action primitives and subsequent action chunks. The execution engine thus translates action primitives, such as "pick up" and "put down", from ADTs encoded by the linguistic compiler into a sequence of robot commands to be executed. Also described in Section 3 is how the JSI execution engine makes use of the AAU skill system to perform action execution in multiple ADTs pertaining to the "Peg In Hole (PiH)" task.

Section 4 of the deliverable focuses on the ChemLab scenario, where similar natural linguistic instructions as in the IASSES scenario are considered, and similar robot skills must be executed, though in a different context, i.e. a chemical laboratory with a robot conducting experiments, and making use of a different execution engine, in this case involving the PRAC probabilistic planner, the CRAM execution system, and the KnowRob knowledge processing system. A recently published paper describing this work [A1] has been attached.

Finally, in Section 5 of the deliverable, ADT execution under the UGOE system is described, where finite state machines are used to generate the semantic event chain state transitions as defined by ADTs in an object pick-and-place task. The outputs of the finite state machine are action primitives that are, similarly as in the IASSES scenario, executed sequentially using the UGOE robot control system in order to complete semantic event chain transitions. Another similarity shared between the scenarios is the linking of ROS bag recordings to the ADT descriptions to characterise the action primitives.

2 Introduction

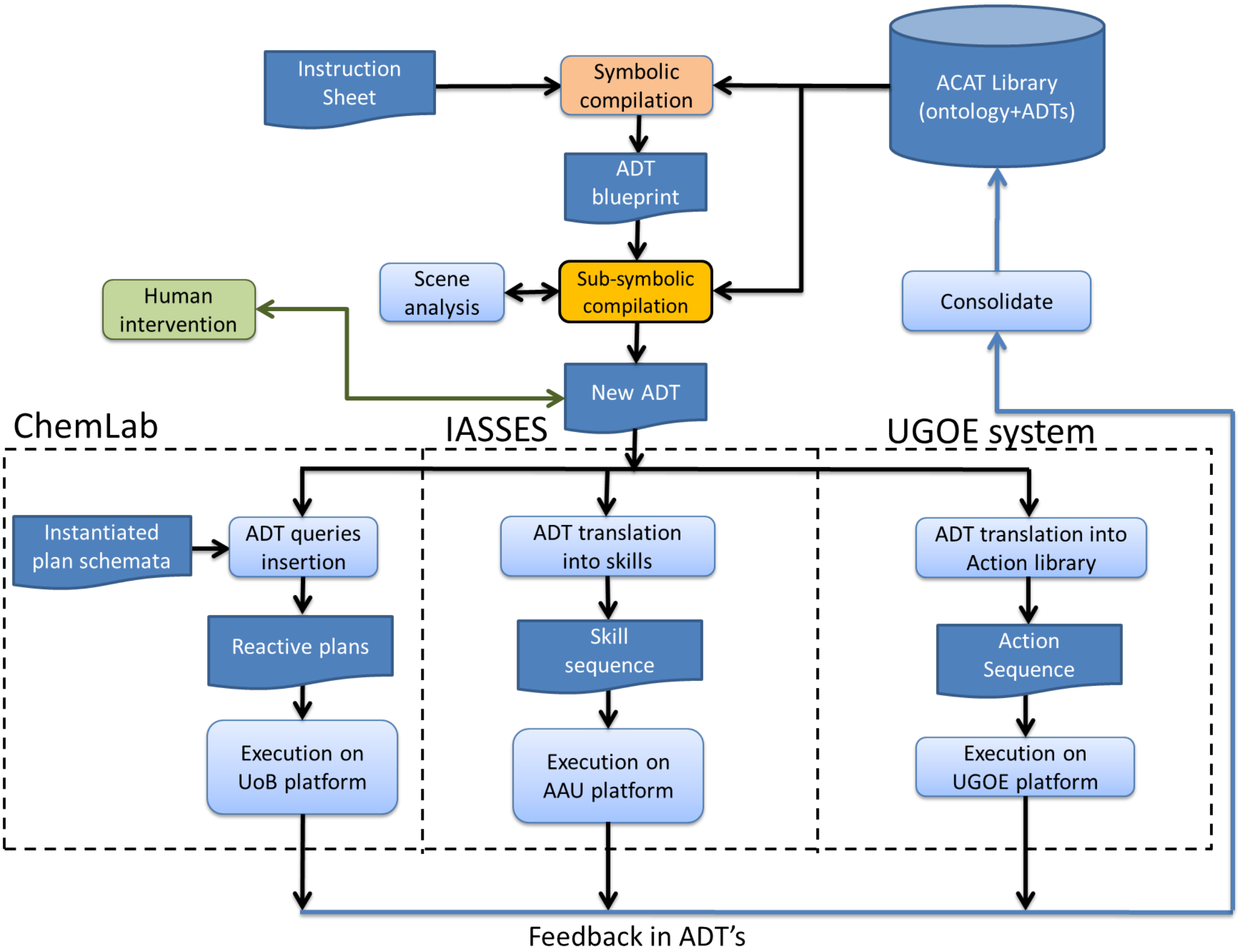


Figure 1: ADT-based integration between the different execution engines.

In the ACAT project, action data tables (ADTs) are the main data structures that provide the capacity for integration across the different execution engines of the three systems, which in turn drive the actual action executions on their respective hardware platforms. Figure 1 (taken from PPR2) illustrates how ADTs are fed to the execution engines in the Chemlab scenario, IASSES scenario and the UGOE system respectively, and how the ADTs are used differently in each case to perform the action executions. The remainder of this deliverable describes how the action execution thus proceeds in each of those cases, as well as how they make use of the information contained in the ADTs in more detail.

3 Action Execution in the IASSES scenario

In order to have a holistic approach towards the demonstration of the IASSES scenario, we developed an execution engine tailored for AAU's Little Helper Skill Based System. One of the main instructions under consideration in the IASSES scenario is *"Pick a rotor cap from conveyor and put it on a fixture"*. This instruction is divided into two action primitives namely "pick up" and "put down". As Figure 2 below illustrates four action chunks are produced from further analysis of the aforementioned action primitives.

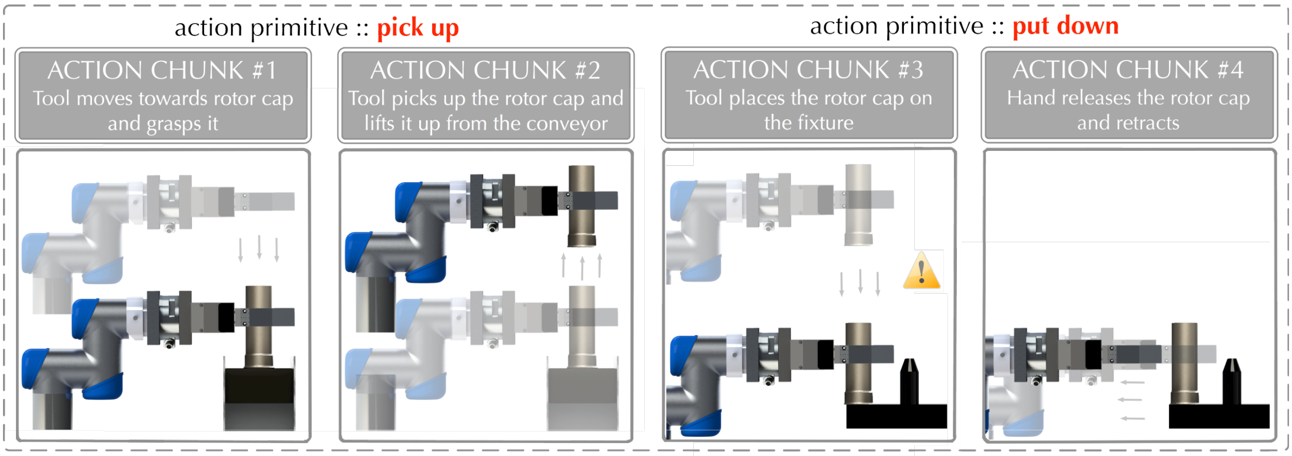


Figure 2: Action primitives and action chunks involved in the instruction *"pick up rotor cap from conveyor and put it on the fixture"*.

The main role of this execution engine is to interpret the ADT produced by the symbolic compilation from the linguistic compiler to a sequence of robot skills, assist the operator in teaching and executing the task and return the newly produced ADT back to the ADT database. The first direct correlation between ADTs and Skill Based System is the action primitives where "pick up" and "put down" are translated into a "Pick" and "Place" skill respectively. In the tables below we present the input parameters for the execution of the Pick and Place skills and, further on, we present their correspondence to the ADT parameters.

3.1 Translation of the Pick skill

A "Pick" skill is defined by several parameters in the Skill Based System (Table 1). As specified inside an ADT, the action chunks #1 *"Tool moves towards rotor cap and grasps it"* and #2 *"Tool picks up the rotor cap and lifts it up from the conveyor"* contain values for the parameters of the "Pick" skill. The correspondences between Skill Based System (SBS) and given ADT are shown in the Table 2.

3.2 Translation of the Place skill

In similar fashion, a "Place" skill is defined by several parameters (Table 3) and the action chunks #3 *"Tool places the rotor cap on the fixture"* and #4 *"Hand releases the rotor cap and retracts"* contain specifications for the parameters of the "place" skill. Table 4 shows how the values from the ADT are translated into input parameters for the skill.

Table 1: Input parameters for the execution of a "pick" skill

	Name	Type	User Input	Description
x1	MoveFrame	Frame	Teach	Target coordinate
x2	ApproachDirection	Char	Teach	Direction of approaching the target
x3	ApproachDistance	Float	Teach	Distance of approaching the target
x4	LeavingDistance	Float	Teach	Distance of leaving the target
x5	LeavingDirection	Char	Teach	Direction of leaving the target
x6	ObjectWidth	Float	Teach	Expected width of the object
x7	ObjectTolerance	Float	Specify	Tolerance of the object width
x8	GraspForce	Float	Specify	Grasping Force
x9	Velocity	Float	Specify	Velocity of the manipulator
x10	Stiffness	Int	Specify	Cartesian stiffness of the manipulator

Table 2: Skill based system - ADT correspondences for the "pick" skill

	Parameter's name	User input in SBS	Correspondence in ADT
x1	MoveFrame	Teach	*Action Chunk #1*: Required values for the execution recovered from analysis of start and end points for the wrist and main object given in the *Action Chunk #1*
x2	ApproachDirection	Teach	Analysis of the values in the position component of the wrist and the main object give the approach direction
x3	ApproachDistance	Teach	The absolute difference between the <end point> and <start point> of the wrist gives the ApproachDistance
x4	LeavingDistance	Teach	*Action Chunk #2*: As in x3, LeavingDistance is given by the absolute difference of the position values for the <end point> and the <start point> of the wrist
x5	LeavingDirection	Teach	Direction of leaving the target: Default
x6	ObjectWidth	Teach	Value derived from the parameters of the main object given in the ADT
x7	ObjectTolerance	Specify	Tolerance of the object width: Default
x8	GraspForce	Specify	Grasping Force: Default
x9	Velocity	Specify	Movement velocity derived from ADT
x10	Stiffness	Specify	Cartesian stiffness of the manipulator: Default

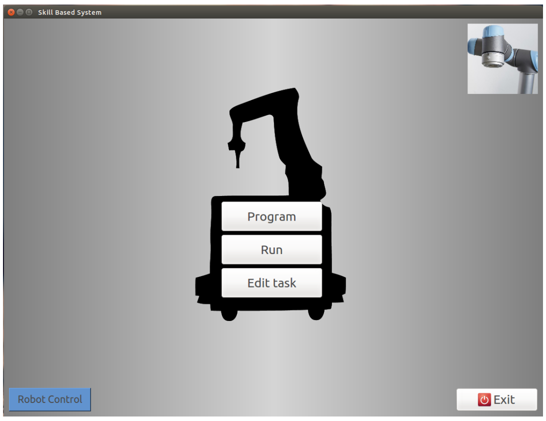
Table 3: Input parameters for the execution of a "place" skill

	Name	Type	User Input	Description
x1	MoveFrame	Frame	Teach	Target coordinate
x2	ApproachDirection	Char	Teach	Direction of approaching the target
x3	ApproachDistance	Float	Teach	Distance of approaching the target
x4	LeavingDistance	Float	Teach	Distance of leaving the target
x5	LeavingDirection	Char	Teach	Direction of leaving the target
x6	ObjectWidth	Float	Teach	Width of the grasped object
x7	Velocity	Float	Specify	Velocity of the manipulator
x8	Stiffness	Int	Specify	Cartesian stiffness of the manipulator

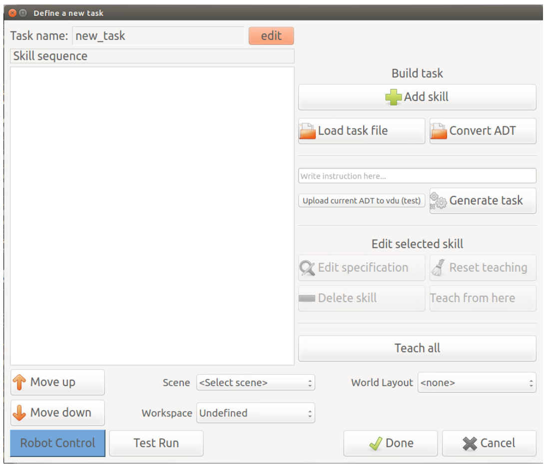
Table 4: Skill based system - ADT correspondences for the "place" skill

	Parameter's name	User input in SBS	Correspondence in ADT
x1	MoveFrame	Teach	*Action Chunk #3*: Analysis of start and end points for the wrist and main object give the coordinate frame
x2	ApproachDirection	Teach	Analysis of the values in the position component for the wrist and main object gives the approach direction
x3	ApproachDistance	Teach	The absolute difference between the <end point> of the wrist and <start point> of the main object gives the ApproachDistance
x4	LeavingDistance	Teach	*Action Chunk #4*: As in x3, LeavingDistance is given by the absolute difference of the position values for the <end point> and the <start point> of the main object
x5	LeavingDirection	Teach	Following the same analysis as in x2 but for action chunk #2
x6	ObjectWidth	Teach	Value derived from the parameters of the main object given in the ADT
x7	Velocity	Specify	Movement velocity derived from ADT
x8	Stiffness	Specify	Default value

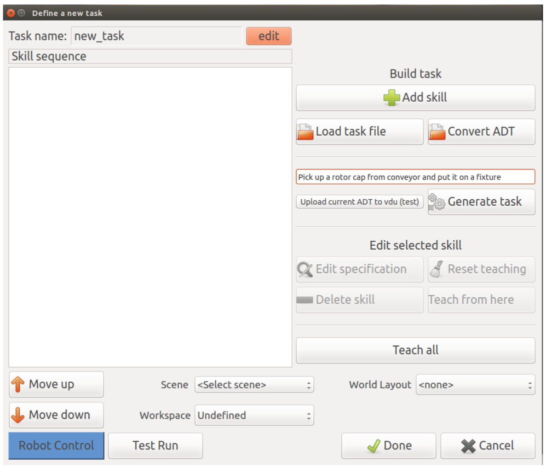
The scope of the IASSES execution engine is to cover the needs for fast task specification and execution. During the specification phase, the operator is able to specify the skills needed for the task either by only writing the instruction to the system or by using a blueprint ADT and a known similar ADT recorded previously during robot execution. In the former case (when similar ADTs don't exist) , the execution engine connects the Skill Based System with the online instruction compiler



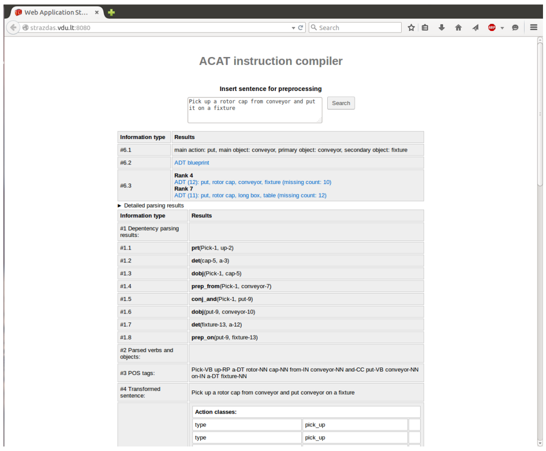
(a) Launch screen of SBS



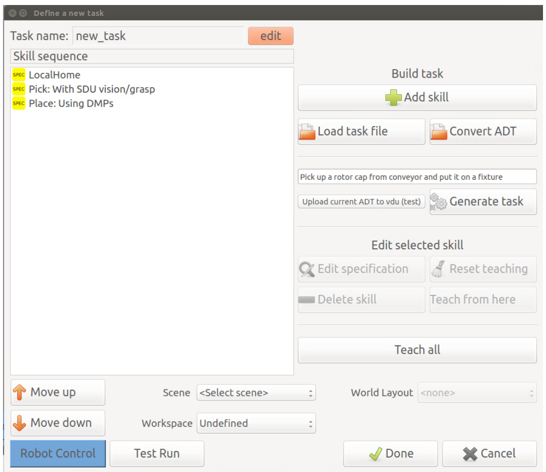
(b) System ready for task specification



(c) The operator gives the instruction

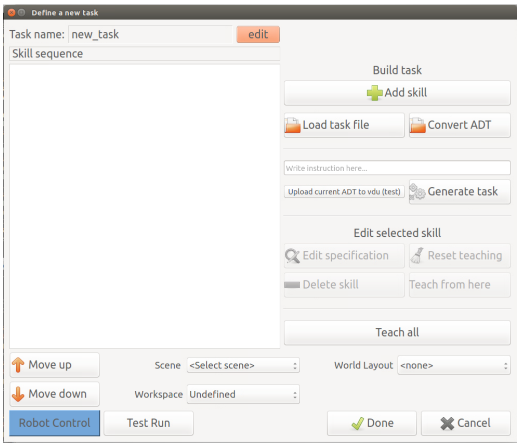


(d) The instruction is copied automatically to the online instruction compiler

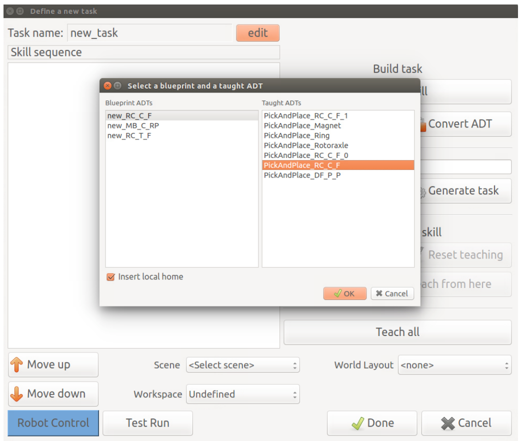


(e) The list of skills is being generated from the action chunks of the blueprint ADT

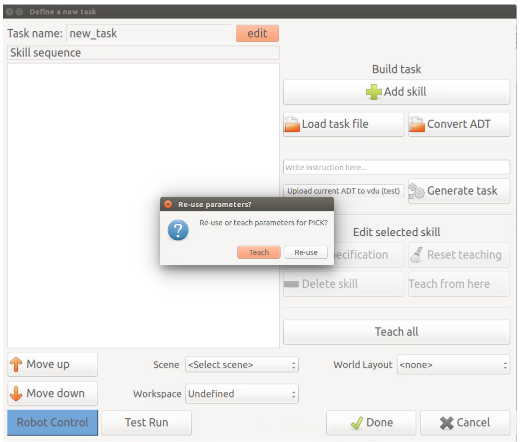
Figure 3: The operator uses an instruction and specifies a list of skills using a blueprint ADT



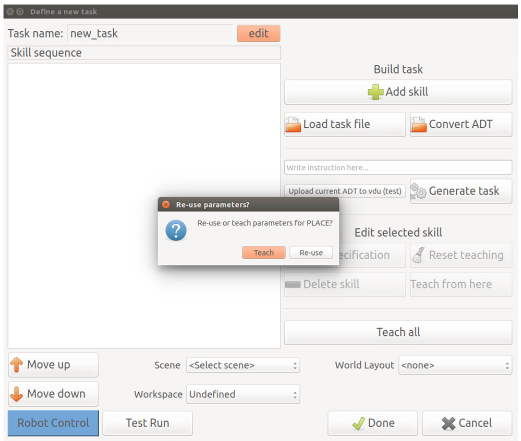
(a) System ready for task specification



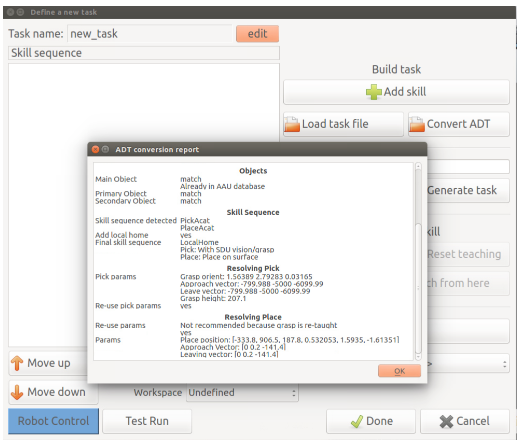
(b) The operator can pick a known ADT from the database



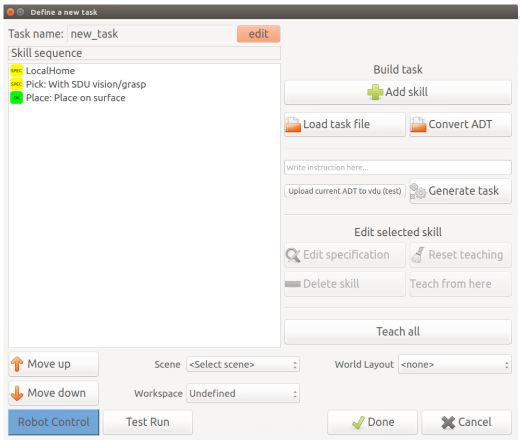
(c) Prompt to re-use or re-teach the parameters of the "pick" skill



(d) Prompt to re-use or re-teach the parameters of the "place" skill



(e) The system delivers a conversion report to the operator for verification



(f) The list of skills is being generated from the action chunks of the known ADT

Figure 4: The operator uses a known ADT and specifies a list of skills with known parameters

using an instruction (Fig. 3c), acquires a blueprint ADT (Fig. 3d), and interprets the action chunks of the blueprint ADT into a list of robot skills as it is described in the previous tables(Fig. 3e). The operator will have now to teach the delivered list of skills for successful execution of the task. Using this connection, the operator can quickly setup the correct order of skills by only giving one instruction to the system.

In the case when similar ADTs exist, the operator uses a known ADT with the highest ranking in the ADT database. This rank corresponds to a similarity outcome of the ACAT instruction complier with other existing ADTs in the database. The execution engine is capable to transfer the list of skills and all of the skills' parameters found in the highest ranked ADT to the Skill Based System, and prepare the robot platform for task execution. In order to keep the operator in the loop, the Skill Based System will ask him/her specific questions according to the confidence of the data delivered from the ADT. As Figure 4c and Figure 4d illustrate, the operator has the choice to re-teach or re-use specific action chunks that correspond to "pick" and "place" skills. As a result, the skills will be marked with a yellow symbol of "SPEC" for the skills that have to be taught in the next step and a green symbol of "OK" for the ones that are ready for execution as illustated in Figures 3e and 4f. In both cases and after a successful execution of an instruction the operator can upload the new taught ADT back to the ADT system interacting with the GUI and the respective button.

3.3 Execution of Complex Actions

In addition to pick and place tasks described in the section above, the IASSES assembly scenario presents several challenges where, in several cases, the "*Peg In Hole (PIH)*" task/skill needs to be applied to ensure a successful assembly of objects. This method is thus applied in different stages of the IASSES scenario and includes three out of the four parts: the rotor axle, all eight magnets, and the rotor cap.

The skill or action of peg in hole insertion is composed of both the motion trajectory and the associated forces and torques. These have to be adapted online during the insertion phase so that the execution is successful. The adaptation plays a key role during the execution, as it compensates for small discrepancies that are the outcome of imperfect grasping, vision miscalibration and uncertainties due to previously performed assembly tasks. The Peg In Hole task is composed of three segments:

- Approach phase (the robot tool approaches the hole). This can be executed with the AAU skill system (cf. Section 3) or with simple point to point movement. The key is to approach the start position of the action execution.

- PIH execution (robot inserts the object in the hole with force/torque adaptation of the learned trajectory). This is executed using dynamic movement primitives (DMPs), which allow stopping of the execution in order to minimize the error of force/torque with respect to the demonstrated data.

- Release and retract (the robot hand releases the part and retracts).

A more detailed illustration of the PIH primitive with action chunks as appearing in the ADT is given in the Fig. 5.

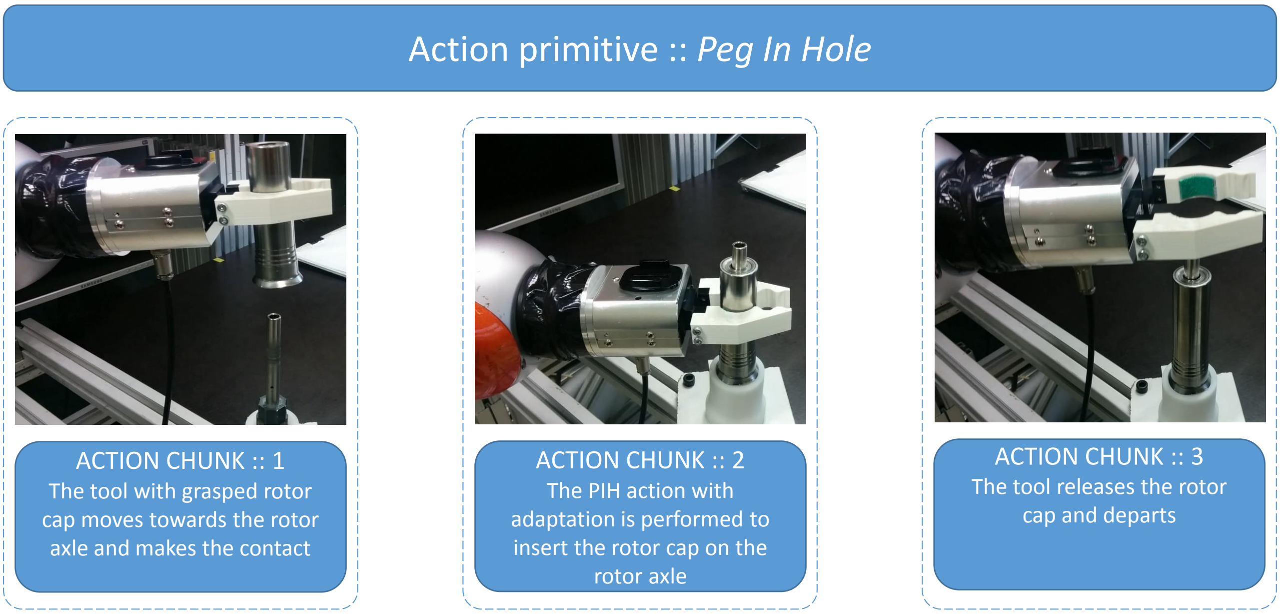


Figure 5: Action primitives described for an example: *"Place the rotor cap on the rotor axle"*.

3.3.1 PIH execution using DMPs

The execution of the DMPs was discussed in deliverable D4.3 [R4], which also describes statistical generalization between successful executions to adapt to a modified task parameter, such as the depth of the hole. The paper entitled "Generalization of Peg-in-Hole Skill for Intelligent Robotic Assembly" attached to Deliverable 4.3 [R4] provided details on the adaptation of the trajectory to minimize the difference between the desired force/torque profile and the measured one. In summary, the execution of the motion is stopped until the change in orientation reduces the aforementioned errors. Because this prolongs the motion, the corrected poses are stored for the next execution, which can be faster.

3.3.2 Object specific actions

ADTs store information on execution of specific actions with specific objects. Here we will explain the specificity of PIH actions for different objects in the IASSES scenario.

Rotor axle: The robot axle has to be inserted into the press, where the tolerances of the press are quite large and, technically, the PiH action is not required. But, the axle also has to be inserted into the ring, where the PiH skill is required. Therefore, at the end of the placement of the axle the motion has to be adapted.

Magnets: Eight magnets have to be inserted in total. After each insertion the axle is rotated. The insertion of magnets is different for each magnet, because with the increasing number of magnets, the force/torque profile is changed. Therefore, we recorded eight ADTs for magnet insertion, where the difference is only in the force profiles and the annotation of which magnet it is for. We acquired the eight profiles through first executing and optimizing one motion and then using this profile as the initial reference for optimization when an additional magnet is inserted. Thus we optimized the profiles for all eight magnets.

Rotor cap: The rotor cap is much wider than the axle, but at the top there is a chance it might get stuck on the top, where it is not shaped to guide the top opening onto the shaft. When we detect the contact of the top plate on the shaft (with the force sensor), we initiate a random search, which performs straight motions, but at an arbitrary angle of the motion. At the same time the cap is pushed down using admittance control. Once it "sits", it is only pushed down until the end.

3.3.3 Relation to ADTs

These actions are encoded directly in the ADTs, where the force/torque profiles are stored in accompanying ROS bags, with the timing data stored in the ADT XML table. The ADT XML tables are generated via the use of the *ADT XML Tool* and *ADT GUI Editor* from the *ACAT ADT Tools* suite [R1] previously described in deliverable D1.2 [R7]. In each of the eight separated ADTs that are created, the associated magnet is assigned as the main object and given a numerical label (e.g. "Magnet 1", "Magnet 2", etc.), with the primary object being the axle. The semantic event chain table, as well as the action chunks, that are described in each of the ADT XML files, associate start and end timestamps that index the ROS bags with each of the objects, indicating their respective involvements at each stage of the action execution.

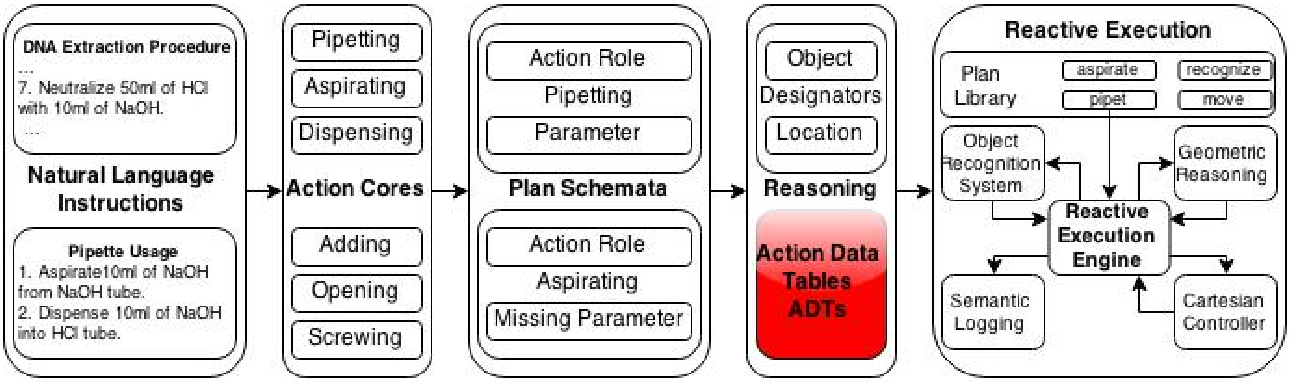


Figure 6: UoB Execution system in the context of *"Neutralize 10ml of hydrocloric acid."* command. Figure taken from PPR2.

4 Action Execution in the ChemLab Scenario

In the Chemlab scenario we consider natural instructions like "Neutralize 10ml of hydrochloric acid." [A1] to be executed. When the PR2 robot is tasked to perform this instruction the PRAC system [R5] probabilistically infers the plan which must run the symbolic and subsymbolic parameters of the plan. Once inferred, the plan and its symbolic and subsymbolic parameters are saved into the *Neutralization ADT* (cf. Table 5). Through a ROS service call, the CRAM [R3] system is triggered to execute the plan with the symbolic parameters probabilistically inferred.

Note, that UoB uses a very complex execution system, where many other components are participating in execution side-by-side with ADTs. At the execution time, when the CRAM system (cf. Figure 6) executes the probabilistically inferred plan it queries the *Neutralization ADT* through the KnowRob [R8] knowledge processing system. If the *Neutralization ADT* doesn't contain the requested parameters then CRAM requests these parameters from the learning-from-games module. This module asks the human users to demonstrate the neutralization action multiple times then it learns the parameters requested by the plan being in execution from the demonstrations collected so far.

After the successful execution of the plan the parameters learned from the users are saved through KnowRob into the *Neutralization ADT* from which they were initially missing.

4.1 The inference of the *Neutralization ADT* symbolic parameters

The PRAC system probabilistically infers that for the *Neutralization* action core (cf. Table 5) the wordnet [R9] concepts *hcl.n.01*, *naoh.n.01* have the roles *AcidSubstance* and *AlkalineSubstance* respectively. Furthermore the PRAC probabilistically refines the previously obtained results and identifies the *Adding* action core through which the *Neutralization* action core is achievable. Simultaneously the PRAC infers that the wordnet concepts *hcl.n.01* and *naoh.n.01* have the roles of *NewMember* respectively *Group* within this context. In the next phase of the probabilistically inference the PRAC infers that in order to perform the initial requested command the *Pipetting* action core is the most specialized. For the *Pipetting* action core the wordnet concepts *hcl.n.01* and *naoh.n.01* have the roles *Source* respectively *Destination* and that the *pipet* plan must be run with the *Source* and *Destination* as parameters (cf. Figure 7).

At this point the PRAC system triggers the execution of the *pipet* plan and forwards the inferred

Action Core	Action Role	Wordnet Concept	Description
Neutralization	AcidSubstance	hcl.n.01	the acid substance
Neutralization	AlkalineSubstance	naoh.n.01	the base substance
Neutralization	AchievedBy	addition.n.01	the next executable action core
Adding	NewMember	hcl.n.01	the acid substance
Adding	Group	naoh.n.01	the base substance
Adding	AchievedBy	pipetting.n.01	the next more refined and executable action core
Pipetting	Source	hcl.n.01	the acid substance
Pipetting	Destination	naoh.n.01	the base substance
Pipetting	Quantity	mililiter.n.01	the quantity

Table 5: The Neutralization ADT

parameters to the CRAM system which starts running the *pipet* plan.

4.2 The simulation-based execution of the *Neutralization ADT*

At execution time the *pipet* plan calls other sub-plans for *grasp*, *unscrew* Algorithm 2 *screw* the caps of different containers. Figure 8 depicts the PR2 robot ready to run the *pipet* plan in Gazebo simulation. The GzWeb client of Gazebo simulator is completely integrated in the PRAC system.

Algorithm 1 The *pipet* and *aspirate* plans

```
(def-plan pipet (source instrument amount destination)
  (seq
    (aspirate (source instrument amount))
    (dispense (instrument amount destination))))

(def-plan aspirate (source instrument amount)
  (seq
    (recognize source)
    (move (object-part effector instrument)
          (above source))
    (recognize (object-part button instrument))
    (press (object-part button instrument))
    (move (object-part effector instrument)
          (inside source))
    (release (object-part button instrument))
    (move (object-part effector instrument)
          (above source))))
```

The *pipet* plan (cf. Algorithm 1) represents a sequence of calls to *aspirate* and *dispense* plans. Each of the two sub-plans call other more specialized sub-plans like the *unscrew* plan in order to prepare the containers which hold the substances to be combined.

The *unscrew* plan (cf. Algorithm 2) sequentially calls the *grasp* plan twice then queries the *unscrew* ADT for the *unscrew-trajectory*. If the *unscrew ADT* (cf. Figure 9) contains the *unscrew-trajectory* then this trajectory is returned and immediately executed. If the *unscrew ADT* doesn't contain an *unscrew-trajectory* then the learning from games module is queried through *compute-*

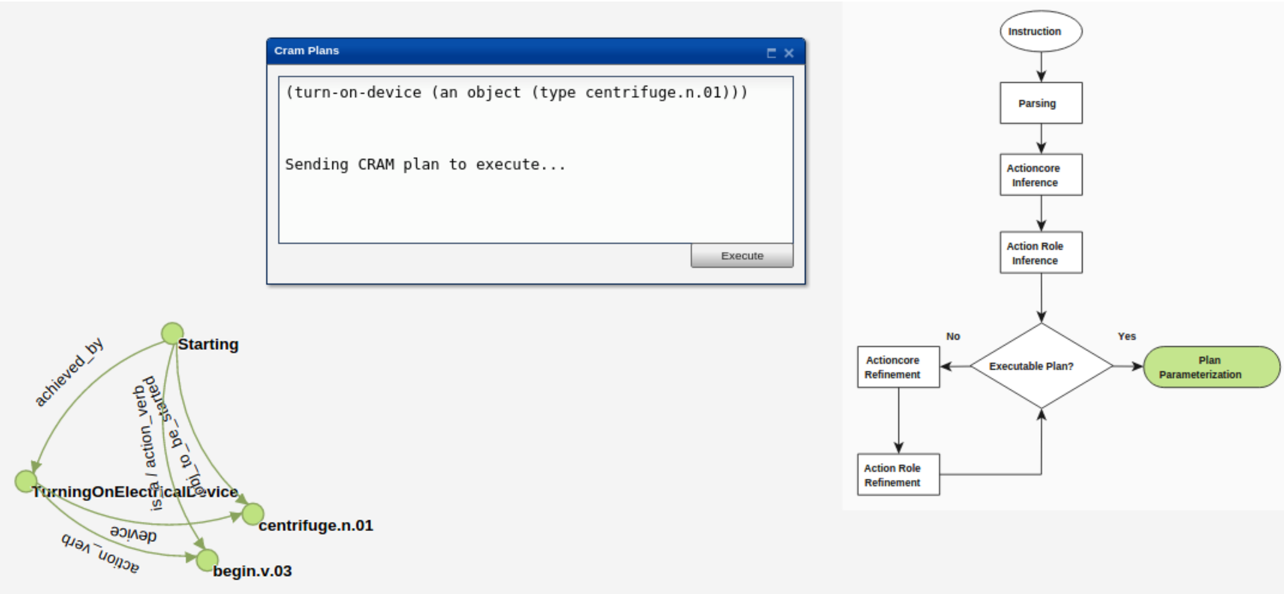


Figure 7: Probabilistic inference in the PRAC

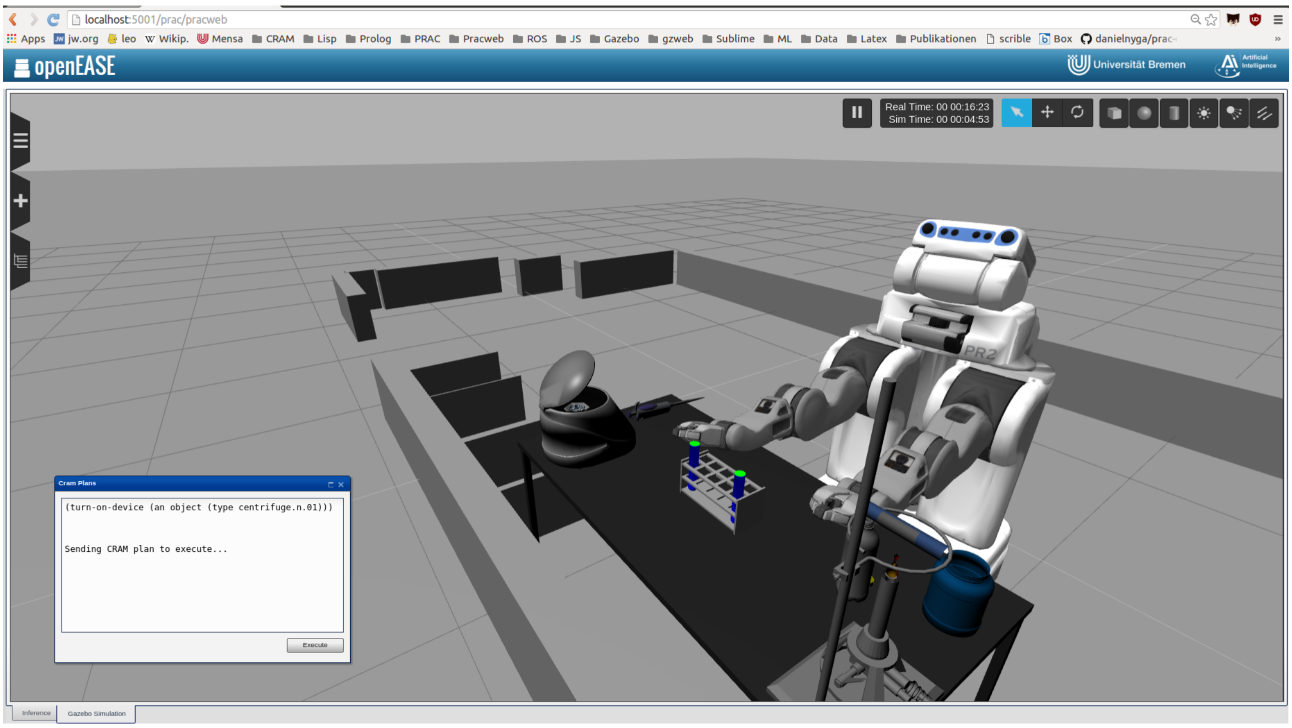


Figure 8: The simulation of the neutralization command in Gazebo simulator.

Algorithm 2 The *unscrew* plan

```
(def-plan unscrew (object-part object-body)
  (seq
    (grasp object-body)
    (grasp object-part)
    (let (unscrew-trajectory (querry-adt "unscrew" object-part object-body))
      (if unscrew-trajectory
        (execute unscrew-trajectory)
        (progn
          (let ((computed-trajectory (compute-unscrew-trajectory object-part object-body)))
            (execute computed-trajectory)
            (push-add "unscrew" computed-trajectory)))))))
```

unscrew-trajectory call and the *computed-trajectory* resulted is immediately executed. The *computed-trajectory* is saved also into the *unscrew* ADT.

4.3 The experience of *Neutralization ADT* as episodic memory available online to other robots

At the execution time of the *pipet* plan the CRAM system creates execution logs called *pipetting episodic memory*. The *pipetting episodic memory* is loaded into *openEASE* [R6] online knowledge base for autonomous robots (cf. Figure 10).

openEASE online knowledge base is able to answer a large variety of questions about the *pipetting episodic memory* like: *What was the trajectory of the right gripper while grasping the pipette?*, *On which objects the unscrewing action was performed?* Or even *where was the tip of the pipette at dispensing time?*.

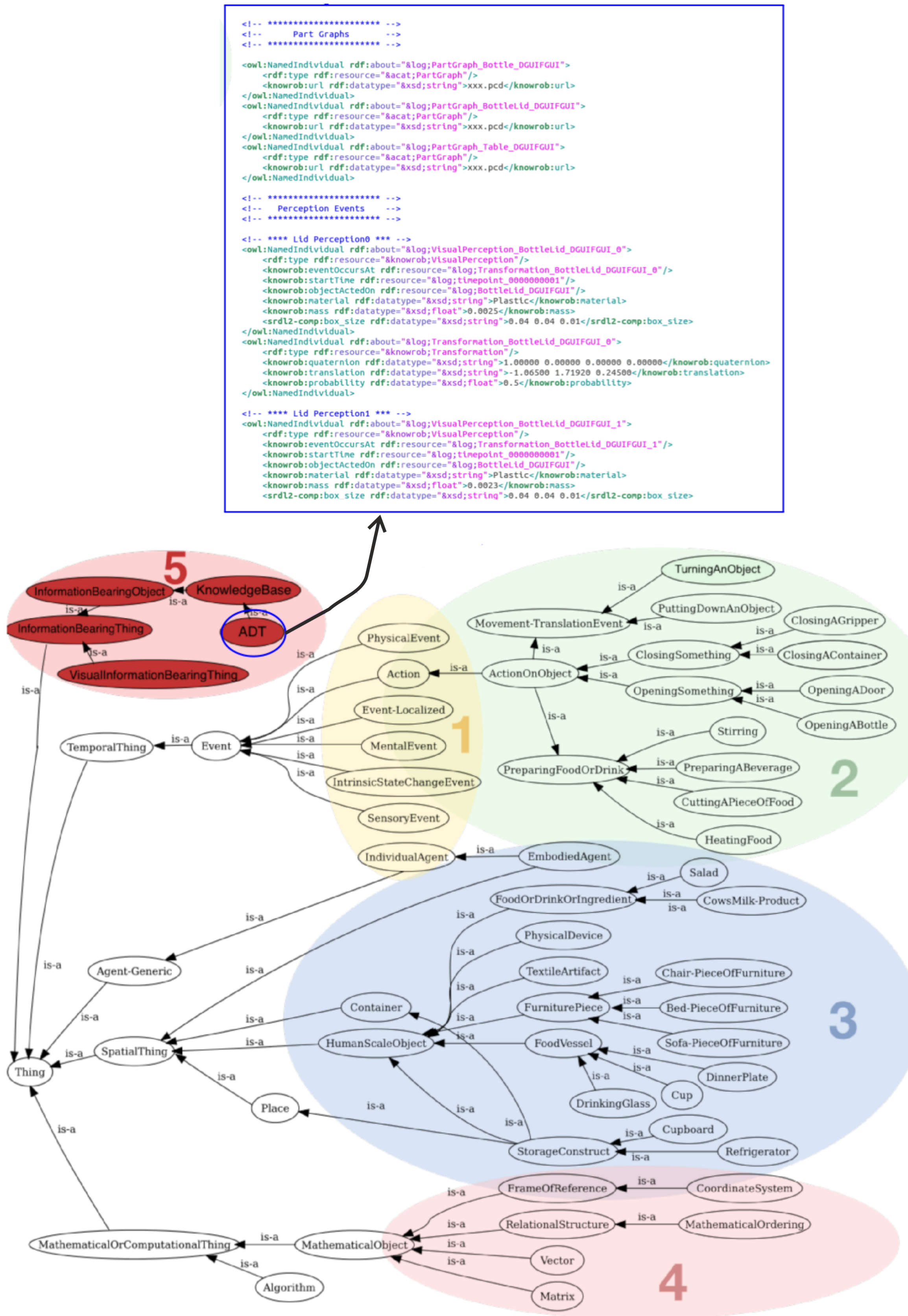


Figure 9: KnowRob's ontology extended with the ADT concepts.

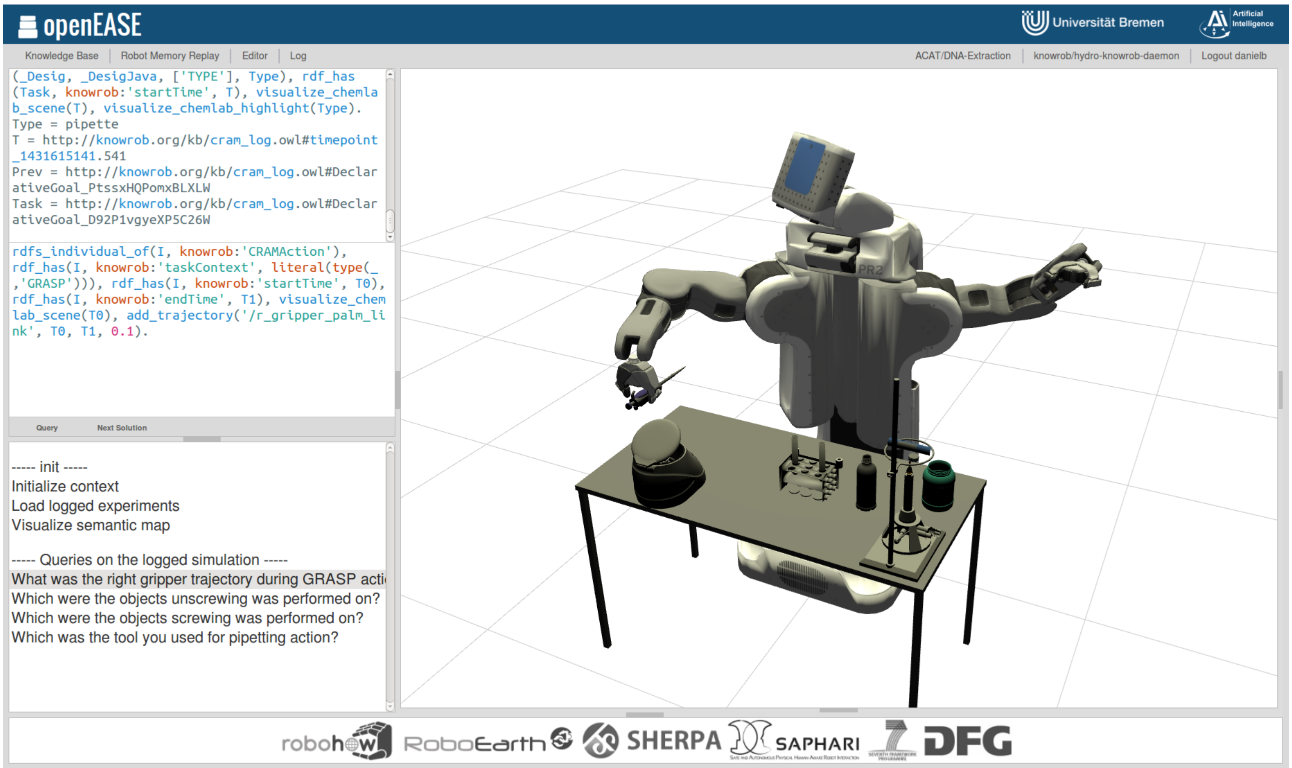


Figure 10: The episodic memory of the *Neutralization ADT* in *openEASE* online knowledge base for robots.

5 Action Execution in the UGOE system

The UGOE system is not specifically attributed to any of the two ACAT scenarios but is used for sub-symbolic compilation experiments (e.g. see D3.2 and D3.3) in both scenarios.

The UGOE system uses the movement primitive sequence as given by ADTs for execution and uses the action chunk sequence from the ADTs, to follow the progress of the action.

In the UGOE system the unit of execution is an "action" which co-insides in granularity with ADT-action (see the Glossary of Terms in PPR2). An ADT-action starts with the manipulator approaching an object and ends when the manipulator retracts having released the object.

In an ADT we use the following action roles, as defined in the Glossary of terms in PPR2:

- Manipulator: The doer of the action, the object that causes the changes in the relations of objects. This could be a human or robot hand.
- Main object: The first object which is touched by the manipulator. This is the main target of the action.
- Primary object: The first object, with which the relation of main object changes.
- Secondary object: The second object with which the relation of main objects changes.

Each pair of these objects provides a relation, which, as defined by the SEC framework in the ADT, can be: (changing) A-to-T (absent to touching), N-to-T (not touching to touching), T-to-A (touching to absent), T-to-N (touching to non-touching) or unchanging N-to-N (non-touching), T-to-T (touching), A-to-A (absent). UGOE uses execution based on finite state machines (FSM, [R2]) to create these Semantic Event Chain transitions. The SEC-based transitions (action chunks in the ATD) cannot normally be achieved by just a single movement. (E.g. we have reach (arm movement) and grasp (hand movement) included until the manipulator first touches the object for picking up). Hence, execution is performed based on movement primitives which are the sub-segments of the action chunks as defined by the Semantic Event Chain. Thus, the outputs of the FSM are the primitives which should be executed sequentially until the SEC-transition (i.e. action chunk in the ADT) completes.

The movement primitives are the simple functions that the control system of UGOE hardware (robot arm and hand) can perform. These primitives, when combined together, create meaningful manipulation actions. In the UGOE system, we use the following movement primitives:

- $arm\_move(goal\_pose)$: Move the robot arm to the pose defined by $goal\_pose$
- $arm\_move\_periodic(V, \omega)$: Move the robot arm periodically along the direction of V with frequency ω.
- $exert(f)$: Exert the force equal to f at the robot end effector.
- $hand\_move(goal\_angles)$: Move the robot hand to the configuration specified by joint angles denoted by $goal\_angles$
- $grasp()$: Grasp the object which is inside the fingers of the hand
- $ungrasp()$: Ungrasp the previously grasped object.

Note, that similar sets of movement primitives are used for controlling in many robotic systems consisting of arm and hand. Thus, in the last version of the ADT format primitives are indicated for each action chunk, which makes execution of the ADTs at the UGOE system straightforward: implementing the movement primitive sequence given in each ADT action chunk in order to achieve the SEC-transition by which the action chunk is characterized.

Figure 11 indicates snapshots of action execution at UGOE representing action chunk and action primitive execution.

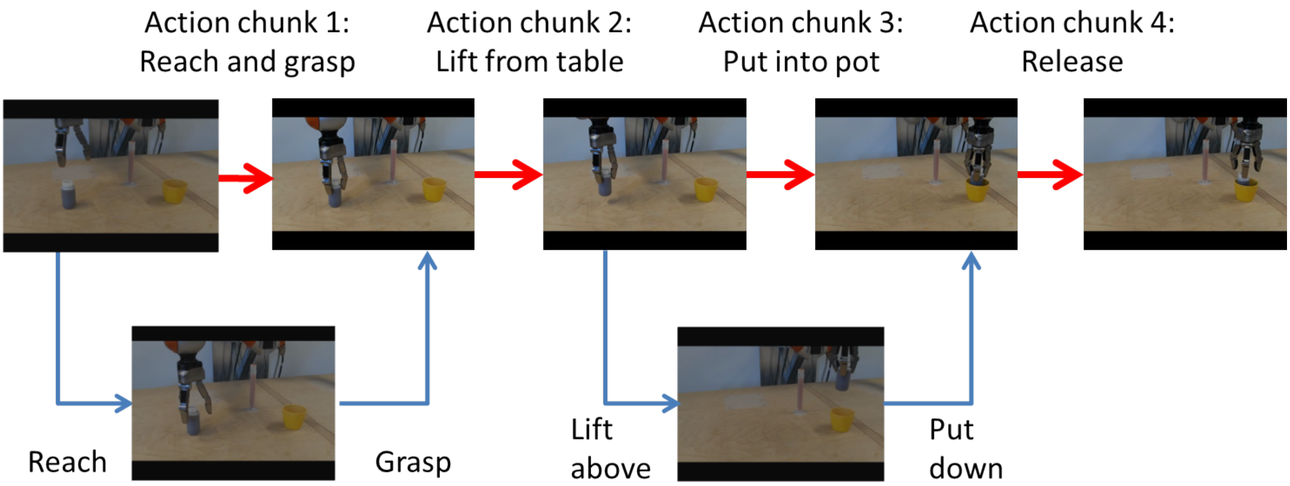


Figure 11: Execution of ADT-action "Pick and Place" at the UGOE system. The ADT-action consists of four action chunks. Stop frames from the video are shown and the action chunks are the transitions between those frames indicated by red arrows. For implementing robot control in action chunks 1 and 3 two movement primitives are needed to achieve the required state at the end of the chunk. Those primitives are indicated by blue pointers. In the remaining action chunks (3 and 4) only one movement primitive is required per chunk and those are not indicated separately.

In order to generalize based on previously executed ADTs we associate the obtained $goal\_poses$ for the arm_move primitive to the objects in the action (main, primary and secondary) based on the distance and the direction of robot movement, to determine which object has been the target of the robot at that movement primitive. We calculate relative poses of the arm to those objects. When the ADT has to be re-used (generalized) in a different scene configuration, we then recalculate movements of the manipulator so that the same relatives poses to specific objects (main, primary and secondary) and in-between the objects are achieved. In case that no object could be associated to a $goal\_pose$ in the original ADT, the argument of the $arm\_move()$ movement primitive is set to void, indicating an arbitrary pose. The exact value of the void poses can be set during the action.

The $hand\_move$ primitives are taken from rosbag recordings representing the corresponding grasps. In the rosbag the configuration sequence of the concrete gripper is recorded. As grasps required in manipulation are very much gripper-dependent we did not attempt to develop a platform-independent representation of the grasps. Thus, we use rosbag entries directly for implementing $hand\_move$ primitives, while into the ADT we lift only very general grasp information, like grasp type and grasp force, which in principle could indeed be transferred between different grippers. However, this is not fully informative about required hand movements and requires either recordings in the configuration space for the same gripper, as we use at UGOE, or grasp planning, which would be a prevalent approach e.g. at partner SDU.

6 Conclusions

The purpose of this deliverable was to describe how action execution proceeds in the ACAT project under the three different execution engines in the IASSES scenario, the ChemLab scenario, and the UGOE system respectively. The deliverable demonstrated, in each case, how ADTs are variously interpreted at the symbolic level and translated into robot actions at the subsymbolic level. Examples in the IASSES scenario included the translation of the "pick skill" and the "place skill", as well as peg-in-hole execution. The ChemLab scenario description provided an example of the simulation-based execution of the acid neutralization ADT, with a recent publication attached [A1] describing the work in further detail. Finally, the description of the UGOE system provided an example of "pick and place" ADT execution where finite state machines are used to create semantic event chain transitions.

References

[R1] *ACAT ADT Tools*. `https://github.com/barryridge/acat-adt-tools`. 2016.

[R2] M. J. Aein, E. E. Aksoy, M. Tamosiunaite, J. Papon, A. Ude, and F. Wörgötter. "Toward a library of manipulation actions based on Semantic Object-Action Relations". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2013.

[R3] M. Beetz, L. Mösenlechner, and M. Tenorth. "CRAM – A Cognitive Robot Abstract Machine for Everyday Manipulation in Human Environments". In: *IROS*. Taipei, Taiwan, 2010, pp. 1012–1017.

[R4] A. Gams et al. *Deliverable 4.3: Scene analysis and movement description - Update of D4.1*. Tech. rep. EU FP7 STREP Project ACAT, 2015.

[R5] D. Nyga and M. Beetz. "Everything Robots Always Wanted to Know about Housework (But were afraid to ask)". In: *IROS*. 2012.

[R6] *openEASE*. `open-ease.org`. 2015.

[R7] B. Ridge et al. *Deliverable 1.2: Sensori-motor and text data-fusion*. Tech. rep. EU FP7 STREP Project ACAT, 2015.

[R8] M. Tenorth and M. Beetz. "KnowRob – A Knowledge Processing Infrastructure for Cognition-enabled Robots". In: *International Journal of Robotics Research (IJRR)* 32.5 (2013), pp. 566 –590.

[R9] *WordNet*. `wordnet.princeton.edu`. 2008.

Attached papers

[A1] G. Lisca, D. Nyga, F. Bálint-Benczédi, H. Langer, and M. Beetz. "Towards Robots Conducting Chemical Experiments". In: *IROS*. Hamburg, Germany, 2015.

Towards Robots Conducting Chemical Experiments

Gheorghe Lisca, Daniel Nyga, Ferenc Bálint-Benczédi, Hagen Langer and Michael Beetz

Abstract— **Autonomous mobile robots are employed to perform increasingly complex tasks which require appropriate task descriptions, accurate object recognition, and dexterous object manipulation. In this paper we will address three key questions: *How to obtain appropriate task descriptions from natural language (NL) instructions, how to choose the control program to perform a task description, and how to recognize and manipulate the objects referred by a task description?* We describe an evaluated robotic agent which takes a natural language instruction stating a step of DNA extraction procedure as a starting point. The system is able to transform the textual instruction into an abstract symbolic plan representation. It can reason about the representation and answer queries about what, how, and why it is done. The robot selects the most appropriate control programs and robustly coordinates all manipulations required by the task description. The execution is based on a perception sub-system which is able to locate and recognize the objects and instruments needed in the DNA extraction procedure.**

I. Introduction

As the area of autonomous robot manipulation gets more mature it is also getting more important that we better understand the nature of the underlying information processing mechanism by building complete systems that perform human-scale manipulation tasks. The importance of research concerning the building of complete robotic agents cannot be overestimated. We have made impressive progress in component technologies such as navigation, grasping, and perception but so far it is not clear how the individual components have to be pieced together to produce competent autonomous activity.

Consider, for example, the control of robot motions. We see many systems that produce and often even learn to produce very sophisticated motion patterns such as flipping a pancake or catching a ball in a cup. However, these systems have no idea of what they are doing. You cannot ask them about the desired and undesired effects of actions, how the course of action could be changed in order to avoid some unwanted side effect, and so on. For example, the result of pouring a chemical substance into a container might cause an explosion.

The reason for this situation is that in order to learn or generate sophisticated motions you have to completely formulate the problem in a mathematical model that is then solved in order to generate a control law that constitutes a desirable mathematical solution. The problems of how the mathematical models and computational problems can be generated by a robot tasked with a NL instruction and looking at a particular scene has not received sufficient attention. The same holds for the problem of enabling robots to answer questions about what they are doing, how, why, what could possibly happen, and so on.

Institute for Artificial Intelligence, Department for Computer Science, University of Bremen, Germany. {lisca, nyga, balintbe, hlanger, beetz}@cs.uni-bremen.de

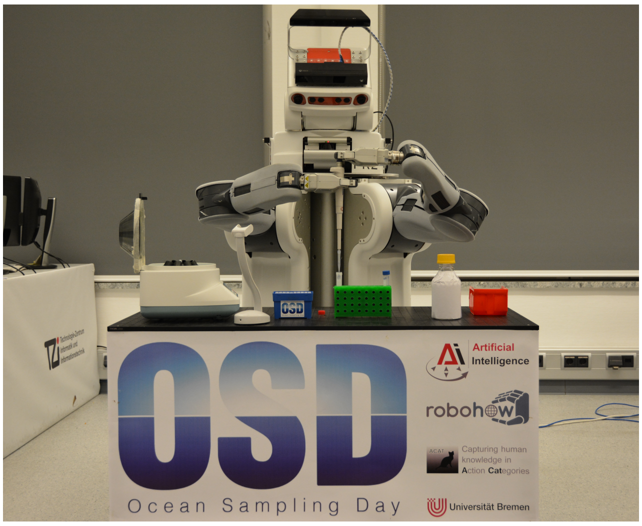


Fig. 1: Uni-Bremen's PR2 pipetting.

In this paper we describe a robotic agent that is capable of autonomously conducting chemical experiments with ordinary laboratory equipment based on NL instructions for these experiments. The actions that the robotic agent is to perform include taking tubes, opening and closing them, putting them into a rack, mixing chemical substances through pipetting, and operating a centrifuge by opening and closing it, loading and unloading it, and pushing the start button.

The application is interesting because it requires the robot to perform only a small set of manipulation actions but by combining these actions in different ways and performing them with different substances and quantities the robot can potentially perform thousands of different chemical experiments by reading and executing instructions for experiments. In addition, large knowledge bases about chemistry that are available in standardized and machine readable form in the semantic web enable us to realize knowledgeable robots with comparatively little effort.

The main contribution of this paper is the realization of a complete robot agent that can autonomously conduct (carefully selected) chemical experiments. In this context the main technical contributions are:

1) The generation of abstractly parameterized plans from NL instructions, which means that a language instruction such as "neutralize 250ml hydrochloric acid" is translated into an abstractly parameterized action description like the one in Algorithm 1. This description

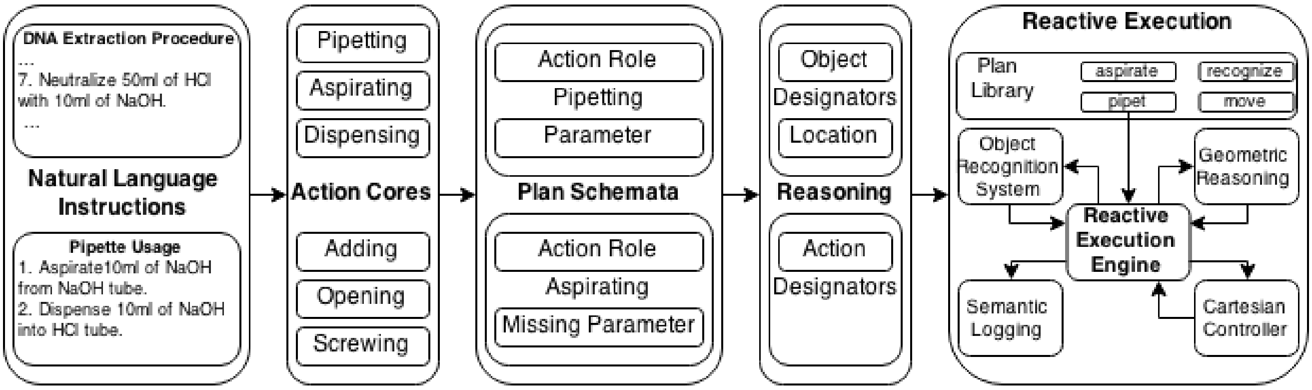


Fig. 2: Agent's Conceptual Architecture

Algorithm 1 Abstractly Parametrized Action Description

```
(perform
  (an action
    (type pipetting)
    (object-acted-on ...)
    (source ...)
    (destination ...)))
```

names the plan to be called, namely pipetting, and assigns each of the formal parameters of the pipetting plan an abstract symbolic parameter description. To deal with the incompleteness and ambiguities of NL instructions the robotic agent employs first-order probabilistic reasoning to carry out this interpretation step.

2) A knowledge-enabled perception component that is able to recognize symbolically described objects such as "the pipette containing the acid substance" or "the lid of the tube in the rack" and localize them accurately enough to allow for high precision manipulation tasks such as putting a tip on the pipette.
3) The perceptually grounded execution of abstractly parameterized plans that takes abstract descriptions of objects, locations, and actions and translates them into specific numeric parameters such as the 6D pose of the pipette for releasing the content of the pipette.
4) The acquisition and the reasoning about episodic memories of chemical experiment activities that enable the robotic agent to answer queries about what it did in the episode, how, why, what happened, etc.

The robotic agent was shown in a public demonstration (see youtube video[1]), in which it participated in the Ocean Sampling Day.

The remainder of this paper is structured in the following way: Section II will present an overview of our system. In Section III we will describe the NL understanding component. Section V will explain the first symbolic representation of an instruction. In Section V-A the reasoning mechanism which separates the symbolic task descriptions in more specific symbolic descriptions for action, object and location, will be presented. In Section V-B we will explain how the specific descriptions are used at runtime. The experiments and drawn conclusions will be summarized in the sections VI and VIII, respectively.

[1]https://www.youtube.com/watch?v=sB7_xEARquM

II. Conceptual Architecture of the Robotic Agent

From describing the *DNA Extraction Procedure* and *Pipette Usage* through NL instructions to having the robot reactively pipetting: *which are the key steps an intelligent robot has to go trough in order to parametrize its control programs from NL?* The first step is to understand the NL instructions which task him (cf. Figure 2).

The two sets of NL instructions for neutralization and pipette usage are parsed using the Stanford parser [1], and the identified syntactic roles are stored in a probabilistic first order relational database. *WordNet* [2] is used for identifying word meanings. Based on the meanings and syntactic roles of instruction's words, the action cores for *pipetting*, *aspirating* and *dispensing* which match the best given instructions are identified. The matching process assigns action roles to the words in the instruction. The roles of action cores which don't have an instruction word associated with them, will be used to infer instruction's implicit words which are missing from instruction's text. *aspirating* and *dispensing* involve the instrument *pipette* which doesn't explicitly appear in the *Pipette Usage* instructions' text.

Each action core has a *Plan Schema*, detailed in Section V, associated with it. A plan schema groups into a tuple the action verb and the action roles from the same action core. The tuple can be regarded as an abstract description of an action. Defined in this way, a plan schema is fully parameterizable by its associated action core. A fully parametrized plan schema is a plan schema for which all its action roles were replaced by instruction-specific entities.

In the first phase of the third step, from previously obtained fully parametrized plan schema, the *Reasoning Mechanism*, detailed in Section V-A, extracts the symbolic descriptions of objects, locations and actions. We call these symbolic descriptions: *designators*. In the second phase of this step, from the freshly extracted action designator, the reasoning mechanism infers which control program is the most competent one for performing the manipulations required by the action description. We call the control program simply *plan* and the entire collection of control programs *Plan Library*.

In the fourth step, from the plan library, the *Reactive*

Execution Engine, detailed in Section V-B, retrieves the plan inferred by the reasoning mechanism. The plan gets the previously extracted object and location designators as parameters and runs as a normal program. At plan's runtime the reactive execution engine triggers the *Semantic Logging* [3] module to log plan's context, goals events and the sensor data which influenced robot's decisions. *OpenEASE* [4] is the web-based knowledge service which collects the data about robot's runtime experiences and makes it available to other robots.

III. Generating Abstractly Parameterized Plans from NL Instructions

Robotic agents acting in human environments must be capable of proficiently performing complete jobs in open environments that they have not been preprogrammed for. A promising direction towards this skill, which has gained a lot of attraction in the recent couple of years, is to equip robots with the capability to acquire new high-level skills from interpreting NL instructions, which can be found in abundance on the web. Instruction sheets provide a rough and sketchy sequence of actions that needs to be executed in order to accomplish a task.

However, these instructions typically are written by humans and are intended for human use, so they lack massive amounts of information about how particular action steps are to be executed, on which objects they are to be performed, which utensils to be used and so on. In addition, a specific action can be achieved in different ways or even must be achieved in a very particular way, depending on the current context the action takes place in. As an example, consider an action like 'add hydrochloric acid', which might be taken from an instruction sheet describing a chemical experiment. It is neither specified explicitly where to add the acid to, how much of it, or how to add it. If the amount that is to be transferred is very small and accurately specified, such as '5 drops', one may want to choose a pipette for doing the addition. Conversely, if 100 ml should be transferred, one should use a measuring cup and pour directly from the container where the acid is located.

Thus, instructions stated in NL are severely vaguely formulated, they are ambiguous and underspecified, and proficiently performing instructions requires a robotic agent to *interpret what is meant* by an instruction by *understanding what is given* and *inferring what is necessary*.

Probabilistic Action Cores (PRAC) [5] are action-specific first-order probabilistic knowledge bases that are able to interpret instructions formulated in NL and infer the most probable *completion* of an action with respect to its abstract, symbolic parameterizations. More specifically, action cores can be regarded as abstract patterns of actions and events, which have a set of formal parameters attached that must all be known in order to parameterize a robot plan appropriately. As an example, consider the NL instruction "neutralize 10 ml of hydrochloric acid." In this example, a 'Neutralization' (in a chemical sense) represents an action core, which has attached to it two *action roles*, namely an *AcidSubstance*

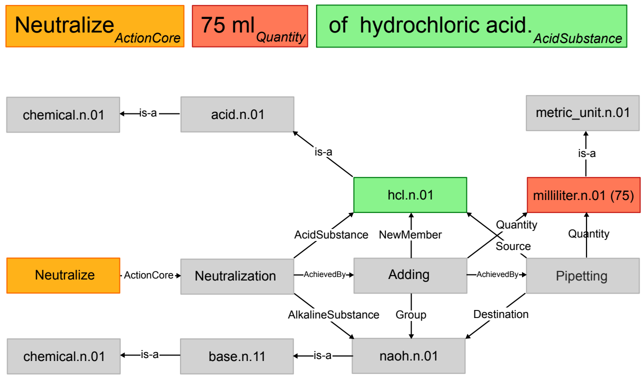


Fig. 4: Exemplary instance of action cores and their action roles for the 'neutralization' example. The colored nodes are given as evidence, whereas the gray nodes and the role assignments need to be inferred.

and an *AlkalineSubstance*, which both must be known in order to perform the neutralization. However, in the original instruction, the alkali counterpart is not specified. From a probabilistic point of view, one can query for the *most likely role assignment* given what is explicitly stated in the instruction:

$$\arg\max_c P\left(\textit{is-a}(s,c) \;\middle|\; \begin{array}{c} \textit{action-core}(a, \textit{Neutralization}) \\ \textit{AcidSubstance}(a, \textit{hcl}) \\ \textit{is-a(hcl, hcl.n.01)} \\ \textit{AlkalineSubstance}(a,\ s) \end{array} \right),$$

i.e. we are querying for the most probable type c of an entity s that fills the *AlkalineSubstance* role, given the action core *Neutralization* and the type *hcl.n.01* of the *AcidSubstance* role. A graphical representation of this action core is given in Figure 4.

In many cases, it is not sufficient to consider the action verb as it is stated in an instruction. In our example, the neutralization is not a directly executable action. It rather denotes a chemical process that needs to be triggered. A robot thus needs to be equipped with reasoning capabilities that allow to infer *how* a particular action can be achieved. The neutralization, for instance, can be achieved by *adding* the alkaline substance to the acid that is to be neutralized, and, since the amount of 10 ml is small an accurately specified, the adding action can be achieved by pipetting one substance to the other. PRAC uses a dedicated action role *AchievedBy*, which enables to reason about which action can be achieved by some other action, given its abstract parameterization.

PRACs are implemented as Markov logic networks [6], a powerful knowledge representation formalism combining first-order logic and probability theory. A key concept in PRAC is heavy exploitation of taxonomic knowledge, which enables to learn PRACs from very sparse data. By exploiting the relational structure of concepts in a real-world taxonomy like the WordNet lexical database, PRAC can perform reasoning about concepts that it has not been trained with.

Action cores can be regarded as conceptualizations of actions that can have an abstract plan schemata attached to them. In these cases, action roles interface the formal

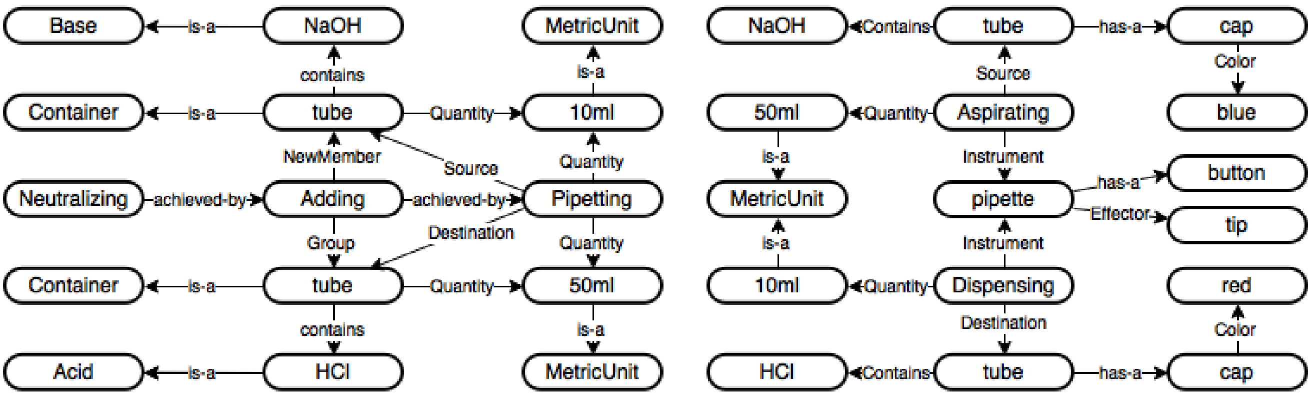


Fig. 3: Action Cores: Neutralizing, Pipetting, Aspirating and Dispensing

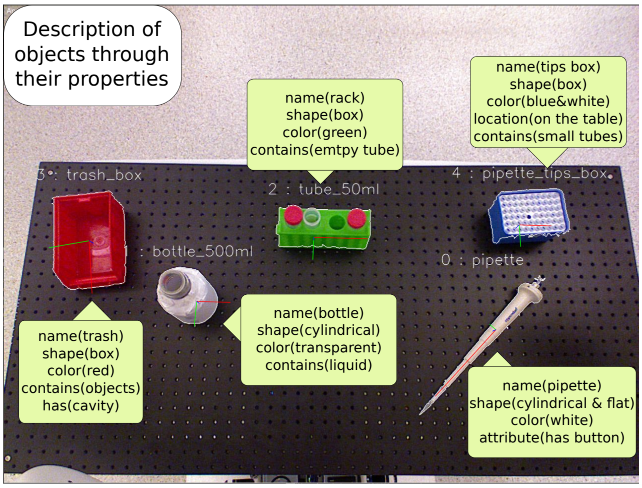


Fig. 5: Description of the perceived objects

parameters of the plan. For a more detailed discussion of the PRAC system, we refer to [7].

IV. Knowledge-enabled Perception of Experiment Setups

Detecting the necessary objects for executing the experiment becomes challenging, given the nature of the tasks which are needed to executed and the noisy input data. Furthermore it is not enough to detect the labels of each object, but identifying parts of them is also necessary (e.g. opening of a bottle or a tube). To address these challenges we use a knowledge-driven approach, where the perception system can reason about the objects it perceives and infer the correct processing step for detecting the parts of the objects to be manipulated[8].

This is done through a two step process. First the objects, their corresponding class lables, visual properties and their initial pose are detected. Since the objects are represented in our knowledge-base, based on their class labels we have access to information that can help further examining them. In Figure 5 for example the objects *rack* and *bottle* have the property *contains*, from which we can infer the next processing step necessary to find the openings of the bottle or detect if the tubes are closed or open.

We define Prolog rules which are able to deduce parame-

Algorithm 2 Prolog rule for deducing the radius of a cylindircal container to be detected.

```
fitCircle(Object, Radius) :-
  category(Object, 'container'),
  object-part(Object, Opening),
  geo_primitive(Object, 'cylindrical'),
  radius(Opening, Radius).
```

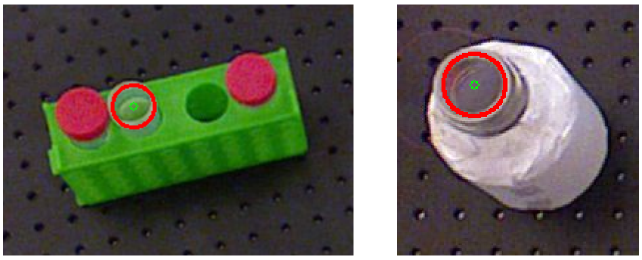


Fig. 6: The fitted circles on containers' openings

terizations for more general perception algorithms, in order to detect the necessary parts of the objects. For example the predicate from Algorithm 2 deduces the radius of a circle that needs to be fit to an object that is a cylindrical container. The results of the perception system after executing this query are shown in Figure 6.

V. Perceptually Grounded Execution of Abstractly Parametrized Plans

As we introduced it in Section II, a plan schema is a template defined over an action verb and the set of action roles defined within an action core.

$$(\langle Action\ Verb\rangle\ (\langle Action\ Role_0\rangle\ \ldots \langle Action\ Role_n\rangle))$$

A fully parameterized plan schema guides the reasoning mechanism in inferring the most adequate plan which has to run in order for robot to execute the instructions with which is tasked. *pipetting* plan schema, Code Excerpt 3, states that the pipetting action firstly needs a *source* which *contains* a specific chemical and is of *type* container and secondly it needs a *destination* which *contains* another specific chemical and is of *type* container too. *pipetting* plan schema starts to capture *what* has to be done for the *pipetting* action. Once the pipetting action schema is fully parameterized it specifies exactly with which objects the pipetting action has to be performed. *pipetting* fully parameterized plan schema

Algorithm 3 Plan Schemata

```
(pipetting
  (from ((source Pipetting)
         (chemical (contains (source Pipetting)))
         (type (is-a (source Pipetting)))))
  (into ((destination Pipetting)
         (chemical (contains (destination Pipetting)))
         (type (is-a (destination Pipetting))))))

(aspirating
  (from ((source Aspirating)
         (chemical (contains (source Aspirating)))))
  (amount (quantity Aspirating))
  (into ((instrument Aspirating))))

(screwing
  (the (mobile-object Screwing))
  (on (fixed-object Screwing))
  (using (tool Screwing)))
```

Algorithm 4 Action Designators

```
(pipetting
  (from Container)
  (with Instrument)
  (into Container))

(aspirating
  (from Container)
  (amount Quantity)
  (into Instrument))
```

doesn't contain *how* pipetting has to be done. *How* an action will be done only the control programs know. *screwing* fully parameterized plan schema cannot specify *how pressing* and *rotating* motions must happen. Instead the *screw* plan *knows* it must simultaneously run the *press* and *rotate* plans.

A. Reasoning On Fully Parametrized Plan Schemata

From a fully parametrized plan schema our Prolog-based reasoning mechanism extracts designators for: actions, objects, and locations. In particular for the fully parametrized pipetting plan schema the reasoning mechanism extracts the action designator for pipetting action, the object designators for pipette and containers and the location designators relative to them. In Code Excerpts 5 - 6. Designators are symbolic descriptions. Syntactically they have the form of a set of attribute-value pairs. Semantically they start existing

$$((\langle attr_0\rangle\ \langle val_0\rangle)\ \ldots(\langle attr_n\rangle\ \langle val_n\rangle))$$

as underspecified descriptions for each entity involved by NL instruction and needs a representation. The semantics of a designator gives the designator's type. While the control system is running those symbolic representations grow complex incorporating more details about the entities they are referring to.

The Pipetting action designator from Code Excerpt 4 states *what* pipetting action needs in terms of classes of entities - specifically it needs two entities of type *Container* and an entity of type *Instrument*. The difference between the pipetting plan schema and the pipetting action designator resides on their different domains of definition. The first of them is defined over the set of action roles and the second is defined over the set of symbolic features.

Test tube designator Code Excerpt 5 states that it is of *type Container*, having a *size* of *500ml*, contains *NaOH*, and

Algorithm 5 Object Designators

```
(test-tube
  (type Container)
  (size 500ml)
  (contains NaOH)
  (has-a
    (cover
      (type cap)
      (color blue))))

(pipette
  (type instrument)
  (capacity 10ml)
  (has-a button-designator)
  (has-a effector-designator))
```

Algorithm 6 Location Designators

```
(above
  test-tube-designator)

(inside
  bottle-NaOH-designator)
```

has-a cover of *type cap* and *color blue*. The robot's *Object Recognition System* detailed in Section IV accepts the vague and symbolic object designators of test tube and returns it enriched with more perceived details like for example test tube's 6D pose.

Location designators are defined relative to object designators. They behave like space quantifiers and refer to different regions around objects. *Above* and *inside* are two location designators. They are defined relatively to at least one object. In Code Excerpt 6 they refer to the spatial region above the test tube and to the spatial region inside the bottle containing the chemical compound NaOH.

B. Plan Execution

The reasoning mechanism infers from pipetting action designator that the *pipet* plan, depicted in Code Excerpt 7, is the most competent to perform it. The pipet plan takes as arguments four designators which symbolically describe the *source* holding the liquid from which the *amount* must be transfered into the *destination* by using the *instrument*. Inside its body, the *pipet* plan coordinates sequentially another two plans which *aspirate* a specific *amount* of liquid into the *instrument* and *dispense* it into the *destination*. At its turn the *aspirate* plan coordinates other simpler plans which *move* an object, *press* an object part respectively *release* an object part. In order to move *instrument*'s *effector* (pipette's tip) the *object-part* quantifier is used to cast the effector as an object and give it as actual parameter to the *move* plan call. Internally the *move* plan figures out the relation between object's frame to be moved and object's grasping points. Taking into account this relation the plan is appropriately parametrizing the controller to perform the right motions.

1) Plan Language: For coding the *pipet* plan we used CRAM Plan Language (CPL) [9] which reimplements and extends RPL [10]. CPL's control structures are designed to allow reasoning about the plan and revising it in case a failure is detected. Plans implemented in CPL can be more than a sequence of atomic actions. They can run concurrently, in loops, they can be synchronized and they benefit of failure handling mechanism. Reasoning on plans can be done

Algorithm 7 Pipetting Plan

```
(def-plan pipet (source instrument amount destination)
  (seq
    (aspirate (source instrument amount))
    (dispense (instrument amount destination))))

(def-plan aspirate (source instrument amount)
  (seq
    (recognize source)
    (move (object-part effector instrument)
          (above source))
    (recognize (object-part button instrument))
    (press (object-part button instrument))
    (move (object-part effector instrument)
          (inside source))
    (release (object-part button instrument))
    (move (object-part effector instrument)
          (above source))))
```

without a complete understanding of a whole plan because CPL's control structures support annotation.

2) Plan Library: Top-down the plan library contains task abstract but action specific plans. Bottom-up it contains hardware specific plans which communicate with robot's object recognition system and controllers via ROS [11] middleware. *pipet*, *aspirate* or *screw* are just few action specific plans. *recognize*, *move* or *rotate* are other few hardware specific plans. Action specific plans build on top of hardware specific plans.

3) Reactive Execution Engine: At execution time the *pipet* plan is run as a normal control program. In the first phase the reactive execution engine queries the object recognition system, detailed in the next section by sending vague object designators and receiving them enriched with more details about recognized objects. In the second phase, before triggering robot's controllers, the reactive execution engine asks the geometric reasoning module [12] to check if the intended manipulations are feasible. The geometric reasoning module temporal projects the requested manipulations and analyzes them. If an issue is detected then the plan gets the chance to fix it. If the geometric reasoning doesn't return any issue then in the fourth phase the cartesian controller is employed to move robot's arms and perform the motions requested. For future experiments we plan to employ either a motion planner either more flexible controllers [13].

4) Spatial Reasoning: At the plan's runtime within robot's specific context, all symbolic location designators must be converted into numerical values understandable by robot's controllers. The geometric reasoning mechanism associates a three dimensional probability distribution to each location designator and draws a sample out of it. For moving pipette's tip inside bottle which contains sodium hydroxide, the geometric reasoning mechanism draws a sample from the probability distribution describing the volume inside the bottle. Based on this sample the *move* plan will parametrize robot's arm controllers such that they move pipette's tip in the sampled three dimensional value. Besides converting symbolic descriptions to numerical values the geometric reasoning mechanism has other more powerful functionalities like asserting if the current manipulation of an object will obstruct future manipulations involving other objects or asserting if the current manipulation will leave the environment into a stable state.

5) Cartesian Controller: We focus our experiments on observing how robots can perform NL instructions, more precisely on bridging NL understanding with robot's control programs. In order to move robot's arms, for a moment, we chose the simplest approach of using a inverse kinematics on top of a joint controller. For future experiments we are integrating a more flexible controller which uses defined constraints over a set of features.

6) Semantic Logging: When running a given plan the reactive execution engine signals a multitude of execution context characteristics like: plan's goals, the relations between the plan being run and the other sub-plans called by it or pieces of sensor data which influenced robot's decisions [3]. All descriptions are synchronized based on a time stamp.

7) OpenEASE: [14] collects all descriptions generated by the semantic logging module and makes them available to other robots [4]. OpenEASE is equiped with inference tools which allow reasoning on this data and answering queries regarding to what did the robot see, why, how, did the robot behaved.

VI. Evaluation

Our robot took part in *Ocean Sampling Day* [15] an event organized with the aim of indexing all DNAs from planetary ocean. Ideally it should have performed the entire procedure of DNA extraction on the samples collected within this event, but we had to limit our experiments to just few of them due to their big number. For testing the pipeline proposed in Section II, from *DNA Extraction Procedure* we selected the neutralization instruction which according to the amount of involved substances requires and *Adding* action or a *Pipetting* action. *PRAC* successfully attached an action role to each instruction word and inferred that the pipetting plan schema is the most appropriate to be parametrized with the specific details coming from instructions' words. The Prolog-based reasoning mechanism successfully extracted the symbolic descriptions for the pipetting action, the containers involved and the necessarily instrument and inferred that the *pipet* plan is the most competent to perform the pipetting action. When running the reasoning mechanism on the extracted pipetting action designator only the pipetting plan is identified as the most appropriate for performing the pipetting action. In future experiments we want to test whether the reasoning mechanism is able to infer an ensemble of plans which combined will should perform given action designator, be it pipetting. When executing the pipetting plan, object recognition system successfully recognized all objects involved based only on their symbolic description Figure 5. For representing the type of knowledge the robot needs in order to press pipette's button such that right amount of liquid is released we use the *KnowRob* [16] knowledge processing system for our future experiments. The cartesian controller behaved well for simple manipulations but we expect it to be overtaken in our future experiments. When manipulating the necessary objects, we assumed that the robot doesn't have to navigate and each object had a virtual grasping pose attached to it. Currently we are extending robot's navigation plan library and its grasping capabilities by deriving grasps from objects' CAD models.

VII. Related Work

The system proposed in [17] probabilistically maps NL instructions into a set of robot primary actions and obtains the sequence of manipulations from planning in this set. The system [18] turns NL commands into a more structured representation and learns a probabilistic graphical model to associate the structured representation to a plan inferred from the set of groundings: objects, locations, actions. For training the probabilistic model people are shown a task happening inside a simulator and are asked to state in NL commands which correspond to task's requirements. The system described in [19] obtained very promising results by building, at learning time, a conditional random field over the set of NL commands and using it at runtime for minimizing an energy function over new commands. So far these systems skipped the problem of *understanding* NL instructions and focused more on correctly *associating* NL instructions to robotic primitive actions.

This approach is similar to [20] where the two robots read instructions from web and collaborated in order to perform them. The current approach tackles activities which require more accurate manipulations. Pipetting action requires the robot to use its both arms and perform accurate motions.

Adam the robot scientist is an laboratory automation system [21] which obtained remarkable results while trying to prove that the experimentation cycle can be automated. While Adam requires special deployment and minimizes the required manipulation, our robot is able to use humans' equipment and conduct experiments into a normal laboratory.

We believe that autonomous mobile robots are able to benefit of *Chemical Semantic Web*, access experiments recorded by Electronic Laboratory Notebooks [22] and perform and record their new results.

VIII. Conclusions

In this paper we present the control system of an intelligent autonomous robot which is able to understand NL instructions and infer and run the most competent control program for performing them. For each instruction the PRAC system successfully inferred the implicit knowledge and assembled a reach instruction representation. From this representation the reasoning mechanism extracted symbolic descriptions for action, objects, locations and the most competent control program to coordinate robot's required motions. When the inferred control program was run, the robot's reactive execution engine competently coordinated the object recognition system, the geometric reasoning system and the robot's controllers in order to accurately recognize the necessary objects and competently manipulating them. The control programs contained in the plan library proved to be very flexible and highly parametrizable. Overall the entire proposed control architecture turned out to be very scalable. The used symbolic mechanism is compatible with newly developed semantic web tools for chemistry. The results of our future experiments will report how the chemistry semantic web can be made available to intelligent robots. The semantic logging mechanism recorded all robot experiences and openEASE, the web-based knowledge base for robots makes them available to other robots.

Acknowledgements

This work is supported by the EU FP7 Projects *RoboHow* (grant number 288533) and *ACAT* (grant number 600578).

References

[1] M.-C. De Marneffe, B. MacCartney, C. D. Manning, *et al.*, "Generating typed dependency parses from phrase structure parses," in *Proceedings of LREC*, vol. 6, 2006, pp. 449–454.

[2] "WordNet," 2008, `wordnet.princeton.edu`.

[3] J. Winkler, M. Tenorth, A. K. Bozcuoglu, and M. Beetz, "CRAMm – memories for robots performing everyday manipulation activities," *Advances in Cognitive Systems*, vol. 3, pp. 47–66, 2014.

[4] M. Beetz, M. Tenorth, and J. Winkler, "Open-EASE – a knowledge processing service for robots and robotics/ai researchers," in *ICRA*, Seattle, Washington, USA, 2015.

[5] D. Nyga and M. Beetz, "Everything robots always wanted to know about housework (but were afraid to ask)," in *IROS*, Vilamoura, Portugal, 2012.

[6] M. Richardson and P. Domingos, "Markov Logic Networks," *Machine Learning*, vol. 62, no. 1-2, pp. 107–136, 2006.

[7] D. Nyga and M. Beetz, "Cloud-based Probabilistic Knowledge Services for Instruction Interpretation," in *ISRR*, Genoa, Italy, 2015, accepted for publication.

[8] M. Beetz, F. Balint-Benczedi, N. Blodow, D. Nyga, T. Wiedemeyer, and Z.-C. Marton, "RoboSherlock: Unstructured Information Processing for Robot Perception," in *ICRA*, Seattle, Washington, USA, 2015.

[9] M. Beetz, L. Mösenlechner, and M. Tenorth, "CRAM – A Cognitive Robot Abstract Machine for Everyday Manipulation in Human Environments," in *IROS*, Taipei, Taiwan, 2010, pp. 1012–1017.

[10] D. McDermott, "A Reactive Plan Language," Yale University," Research Report YALEU/DCS/RR-864, 1991.

[11] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "ROS: an open-source Robot Operating System," in *ICRA*, Kobe, Japan, 2009.

[12] L. Mösenlechner and M. Beetz, "Fast temporal projection using accurate physics-based geometric reasoning," in *ICRA*, Karlsruhe, Germany, 2013, pp. 1821–1827.

[13] G. Bartels, I. Kresse, and M. Beetz, "Constraint-based movement representation grounded in geometric features," in *ICHR*, Atlanta, Georgia, USA, 2013.

[14] M. Tenorth, J. Winkler, D. Beßler, and M. Beetz, "Open-ease – a cloud-based knowledge service for autonomous learning," *KI – Künstliche Intelligenz*, 2015, accepted for publication.

[15] Micro B3. (2014) Ocean sampling day. [Online]. Available: www.microb3.eu/osd

[16] M. Tenorth and M. Beetz, "KnowRob – A Knowledge Processing Infrastructure for Cognition-enabled Robots," *International Journal of Robotics Research (IJRR)*, vol. 32, no. 5, pp. 566 – 590, April 2013.

[17] Mario Bollini, Jennifer Barry, and Daniela Rus, "BakeBot: Baking Cookies with the PR2," in *The PR2 Workshop, from International Conference on Intelligent Robots and Systems (IROS)*, 2011.

[18] S. Tellex, T. Kollar, S. Dickerson, M. R. Walter, A. G. Banerjee, S. J. Teller, and N. Roy, "Understanding natural language commands for robotic navigation and mobile manipulation." in *AAAI*, 2011.

[19] Dipendra K Misra, Jaeyong Sung, Kevin Lee, Ashutosh Saxena, "Tell Me Dave: Context-Sensitive Grounding of Natural Language to Mobile Manipulation Instructions," in *RSS*, 2014.

[20] M. Beetz, U. Klank, I. Kresse, A. Maldonado, L. Mösenlechner, D. Pangercic, T. Rühr, and M. Tenorth, "Robotic Roommates Making Pancakes," in *ICHR*, Bled, Slovenia, 2011.

[21] R. D. King, "The automation of science," *Science*, vol. 324, no. 5923, pp. 85–89, April 3, 2009.

[22] C. L. Bird, C. Willoughby, and J. G. Frey, "Laboratory notebooks in the digital era: the role of elns in record keeping for chemistry and other sciences," *Chem. Soc. Rev.*, vol. 42, pp. 8157–8175, 2013.