Deliverable number: D3.1 (update)

Deliverable Title: Textual completion of instruction sheets (**update**)

Type (Internal, Restricted, Public): PU

Authors: Daiva Vitkute-Adzgauskiene, Irena Markievicz, Jurgita Kapociute-Dzikiene, Tomas Krilavicius, Minija Tamosiunaite, Leon Bodenhagen, Dimitris Chrysostomou, Jimmy Alison Rytz, Hagen Langer, Aleksandar Vorotnjak

Contributing Partners: VMU, UGOE, AAU, SDU, UoB

Project acronym: ACAT

Project Type: STREP

Project Title: Learning and Execution of Action Categories

Contract Number: 600578

Starting Date: 01-03-2013

Ending Date: 28-02-2016

Contractual Date of Delivery to the EC: N/A – this is an update
Actual Date of Delivery to the EC: 27-12-2014

Content

1. Executive summary

This document is the updated version of deliverable D3.1, providing exhaustive analysis of textual completion for CHEMLAB and IASSES instruction sheets.

In the ACAT project we are considering to perform human readable instruction compilation into a robot executable instruction sequence by transforming the human readable instruction into a sequence of Action Data Tables (ADTs, the format introduced in the deliverable D2.1).

The processes we are implementing for compilation in the ACAT project consist of the following steps:

1) Robot executable action sequence definition (i.e., ADT sequence definition);
2) ADT filling with symbolic information from textual resources and ACAT action ontology;
3) ADT filling with sub-symbolic (control level) information based on previous robot experience (previously filled ADTs);
4) Human validation and error correction of the automatically filled ADTs as well as entering of the missing information into the ADTs.

Here we present the designed algorithms and related tools for the filling-in of the missing **symbolic information** (step 2 from the list above) into ADTs. Other aspects (the other steps) will be covered by deliverables D3.2 as well as in system demonstrators (D5.4 and D5.7).

The algorithms presented here cover both, parsing of the available instructions and extraction of missing (or more specific) symbolic information for each action and its corresponding environment by querying the ACAT action ontology. The information is filled into an empty ADT template thus creating an ADT blueprint required for further compilation steps.

2. Introduction

The process of textual completion of instruction sheets is aimed at creating an instruction representation that provides textual (symbolic) information required for the planning and execution process on a robot. Specifically, we are considering here the process of converting natural language instruction into a sequence of Action Data Tables (ADTs, data structure for robot action data recording and execution in ACAT project, see D2.1 for details). In the textual completion phase the ADTs are pre-filled with symbolic information (action and object names, symbolic object properties). Filling of the signal level information into ADTs is not discussed here, but will be discussed in deliverable D3.2 (Compilation of instructions into action sequencing protocols.) at month 24.

A simple example of an instruction where textual completion is needed is "take a rotor cap and place *it* on the robot platform". Here one needs to replace the pronoun "it" with the object "rotor cap" such that it is clear that the object which needs placing on the robot platform is the rotor cap. Such simple re-substitutions might be addressed using "pure" text analytics techniques (specifics of robotics does not need to be taken into account) and are performed in the "pre-processing" step of the textual completion procedure discussed in this deliverable.

The other issues on textual completion discussed in this deliverable are tightly related to robotics. Next we will briefly discuss the link. Not every action word (verb) [1] in the instruction sheet is directly linked to robotic actions. E.g. consider the verbs "neutralize" or "harvest" (of E.coli), which cannot be directly linked to robotic actions, against "pick up" or "screw", which can be directly linked to robotic actions. Let us analyze an instruction from the CHEMLAB scenario: "Harvest the E.Coli by centrifugation at 6000g for 10 minutes at 4°C". Ignoring the details like speed or duration of centrifugation for now, first we need to tell which executable robot action sequence would correspond to the verbal instruction "harvest E.Coli by centrifugation", e.g.: locate centrifuge, open centrifuge, pick up test tube with E.Coli, place it into centrifuge, etc. It is obvious that the executable robot action sequence in such a complicated case cannot be extracted without additional knowledge (e.g. human-provided dictionary for translation of the instructions with verbs that cannot be directly linked to robotic actions into a sequence of instructions that *can* be directly linked to robotic actions). Instead, in this deliverable we will concentrate on instructions, which are already given at the level of robotic actions (See Appendices A.2-A.3 for actual instruction sets for the two scenarios which were processed in this deliverable). Systematic handling of the instructions that are not directly linked to robotic actions will be addressed in year 3 of the project.

[1] We use the notion of „action word“ instead of „verb“, to emphasize that action description in the sentence can be more complex than just an isolated verb: e.g. verbs with prepositions (e.g. „put into“), or more complicated constructs, like „harvest by centrifugation“. Still, we sometimes use „verb“ in exchange to make the text less confusing for the reader.

To distinguish at which level (directly linked to robotics action or not) the instruction is given, we define the action ontology such that action words (verbs) are supplied with appropriate features: “robotic” and “non-robotic” (introduced into the ontology in a semi-automated way). Thus, textual completion of instruction sheets presented here takes a sequence of instructions in natural language, given the action words (verbs) are “robotic”, and transforms it into a sequence of ADTs, where each ADT, in turn, is a sequence of primitive actions and action chunks; however, these finer sequencing issues will be discussed in the next deliverable D3.2.

For the purpose of textual completion of instruction sheets, the following actions are repeated for every instruction in the instruction sheet:

- textual analysis (parsing) of the instruction is done in order to structure the information in the instruction and to identify the action words (verbs),
- queries are made to ACAT ontology and action word features are identified (robotic or non-robotics),
- finally detailed object information for each action is filled into the instruction from the parse data and from the action ontology.

The goal of this document is to describe in detail the main algorithms and corresponding tools for textual instruction sheet completion as well as the results obtained for the instruction sheets from two ACAT scenarios: IASSES and CHEMLAB. In the Appendixes we are providing the Documentation of the ACAT instruction completion software (instruction compiler) as well as the instruction sheets for the two scenarios that have been used as test-data for this deliverable.

3. Overview of the textual completion procedure

An algorithm for textual completion of instruction sheets and insertion of relevant background information has been developed. The algorithm is based on the following techniques and resources:

1. Pre-processing and parsing of instruction texts in order to identify action verbs and related background structure elements.
    a. Semi-manually built restricted dictionary for a topical domain is used for better parsing quality.
2. SPARQL queries to the corresponding action ontology in order to extract the action background structure for a specific action, identified in the process of parsing the instruction sheets.
    a. Action ontologies are built for specific topical domains (namely, IASSES and CHEMLAB) focused on the instruction sheets provided in D5.1 and filled in with information from Wordnet as well as from domain-specific corpus texts, accumulated in the ACAT Project (see D1.1).

3. For each action, identified in the instruction sheet matching of the action structure, extracted from the action ontology, to the instruction parse-tree in order to:
    a. Assign semantic roles to the objects, identified in instruction parse-trees;
    b. Determine required objects and properties, missing from the instruction text.

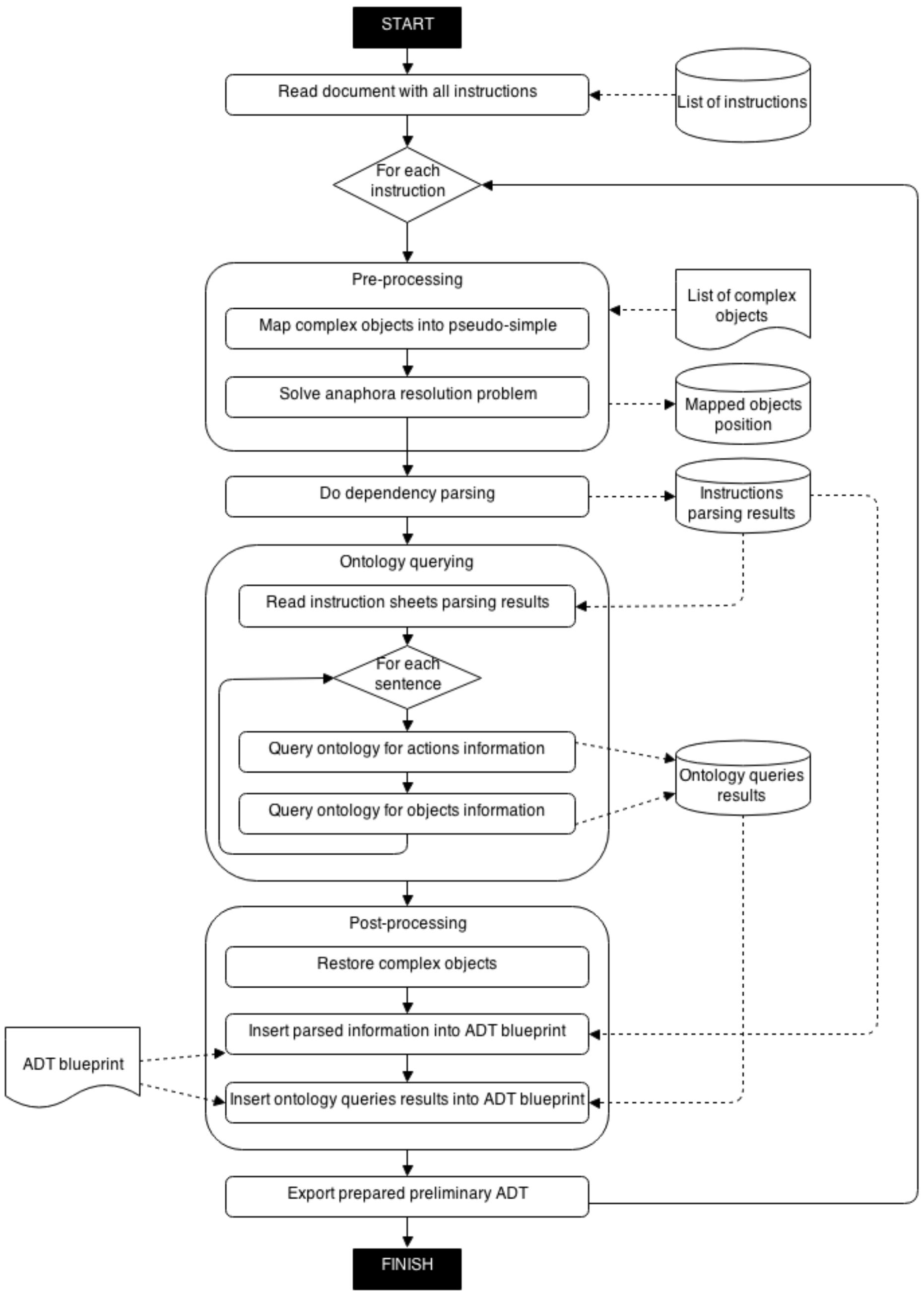

START
Read document with all instructions
List of instructions
For each instruction
Pre-processing
Map complex objects into pseudo-simple
Solve anaphora resolution problem
List of complex objects
Mapped objects position
Do dependency parsing
Instructions parsing results
Ontology querying
Read instruction sheets parsing results
For each sentence
Query ontology for actions information
Query ontology for objects information
Ontology queries results
Post-processing
Restore complex objects
Insert parsed information into ADT blueprint
Insert ontology queries results into ADT blueprint
ADT blueprint
Export prepared preliminary ADT
FINISH

Fig.1. Algorithm for filling in the missing information in instruction sheets

4. SPARQL queries to the corresponding action ontology in order to extract possible candidates for filling of the missing information (missing action background elements) in the instruction sheet.

5. The resulting action structure, built by combining instruction parsing and action ontology querying, is stored as an instance in the action ontology of the corresponding domain.

Fig.1 presents a more detailed view of the above described algorithm.

4. Detailed method description

This chapter is dedicated to a detailed description of the methods applied in the main two steps of textual instruction sheet completion:

1. Preprocessing and dependency parsing (section 4.1).
2. Filling-in missing instruction information with knowledge from ontology (section 4.4).

In addition we are reporting about the project work on advanced instruction text analysis (extension of Markov Logic Network-based reasoning and causal relation extraction from texts, section 4.2) as well as presenting properties of the ACAT ontology required for the filling-in of missing instruction information (section 4.3).

4.1. Preprocessing and dependency parsing

Parsing (synonymously: syntactic analysis) is the process of analyzing a string of symbols, either in natural language or in computer languages, according to the rules of a formal grammar. In our case the formal grammar is a *dependency grammar* (Kübler et al, 2009), where all dependency relations between syntactic units (words) are either directly or indirectly dependent on a verb as the structural center (core) of each clause.

Indeed an interpretation of the dependency parsing relations serves as the first step in the clarification and formalization process of an instruction written in a human language. The recognized core verb itself matches the action, which a robot has to perform and different types of dependency relations reveal how objects (and even their features as color, shape, etc.) are involved into that particular action. E.g. *locate* is the main verb in a "*locate a rotor cap on the robot platform"* clause; the direct object (***dobj*** dependency type**)** relation between *locate* and *rotor cap* indicates "what to locate"; the prepositional modifier "on" (***prep_on*** dependency type) between *locate* and *robot platform* – "where to locate".

The instruction parsing system, we are describing in this chapter, was implemented using the Apache UIMA software framework (Apache UIMA Development Community, 2009). Next we will give a brief description of each block of this system. The generalized schema for this is presented in Fig.2.

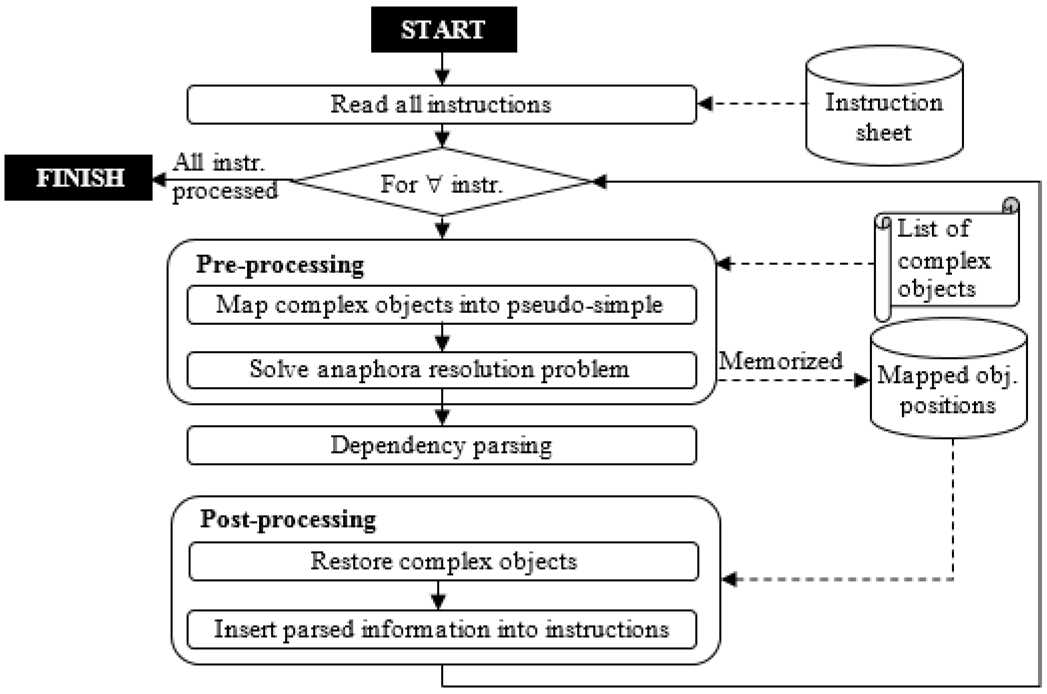


Fig.2. Generalized block schema of the instruction parsing system

- **Read all instructions.** Instruction parsing is an iterative process, performed instruction after instruction. Assuming that each instruction corresponds only to one sentence, we used the embedded Apache UIMA sentence tokenizer to split all the data into the sentence units in a given instruction sheet.

```
<words>
    <word>rotor cap</word>
    <word>robot platform</word>
    <word>bacterial pellet</word>
    <word>bacterial cells</word>
    <word>lysis buffer</word>
    <word>room temperature</word>
    <word>precipitation buffer</word>
    <word>gigaprep lysate filtration cartridge</word>
    <word>equilibration buffer</word>
    <word>DNA binding cartridge</word>
    <word>wash buffer</word>
    <word>elution buffer</word>
    <word>centrifuge bottle</word>
    <word>DNA pellet</word>
    <word>TE buffer</word>
    <word>plasmid DNA</word>
    ...
```

Fig.3. Snippet from the XML file containing complex objects

- **Pre-processing**. To avoid some dependency parsing errors, which may be crucial in the further system compilation steps (when linking with the ontology information, creating sequences of action categories, etc.), dependency parsing was complemented with the following capabilities:
  - **Complex object mapping to pseudo simple**. In order to treat complex objects (such as *rotor cap*, *robot platform* or *DNA binding cartridge*) as indivisible units, they were replaced with the appropriate pseudo simple objects, i.e. leaving only the last word instead of the entire collocation (e.g. *rotor cap* $\rightarrow$ *cap*, *DNA binding cartridge* $\rightarrow$ *cartridge*, etc.). This replacement protects sentences from redundant and often erroneous dependency relations. An XML file (dictionary, see the snippet in Fig.3) helped in recognizing complex objects in the text. The dictionary was built semi-manually extracting all complex objects from predefined instruction sheets for the two scenarios of ACAT. We kept track both of all mapped complex words and their positions in the sentence to avoid possible ambiguity between equal pseudo simple and simple words (e.g. replaced *rotor cap* $\rightarrow$ *cap*). E.g. *take a rotor cap and place it on the robot platform* was replaced with *take a cap and place it on platform*, memorizing that the 3rd word *cap* is actually *rotor cap* and the 8th word *platform* is *robot platform*.
  - **Anaphora resolution problem solving**. The anaphora resolution block is responsible for coping with the pronouns – i.e. indirectly expressed objects or subjects. If the part-of-speech of a word indicated a personal/possessive pronoun/wh-pronoun[2], then it had to be replaced with the appropriate noun. The noun was determined by searching back in the sentence for the first dependent with the dependency label indicating direct/indirect/of preposition object or (passive) nominal/clausal subject. The part-of-speech tags and the dependency relations were determined with the Stanford parser (Marneffe and Manning, 2008), incorporated into our instruction parsing system. If anaphora resolution problem solving involved the previously replaced complex objects. Thus, information about those replacements was taken into account. E.g. *take a cap and place it on platform* was replaced with *take a cap and place cap on platform*. Since *it* refers to the complex object *rotor cap* (replaced by pseudo *cap* in the previous block), this mapping was memorized as well: i.e. 6th word is *rotor cap*.

[2] Wh-pronoun – a pronoun, which is spelt with an initial wh: how, what, which, where, when, who, whom, whose, however, whatever, etc. Wh-pronouns are either interrogative pronouns or relative pronouns. More information: http://www.phon.ucl.ac.uk/home/dick/enc2010/frames/frameset.htm

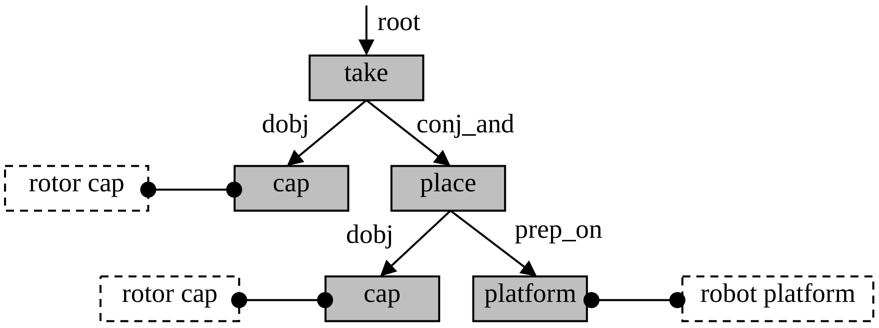


Fig.4. Grey blocks indicate words in the parsed sentence (determiners and punctuation marks are ignored); arrows – dependency relations and their labels; dashed blocks – mapped information. E.g. *place* is node's *take* dependent, but node's *platform* governor; core verb *take* is a *ROOT* node dependent.

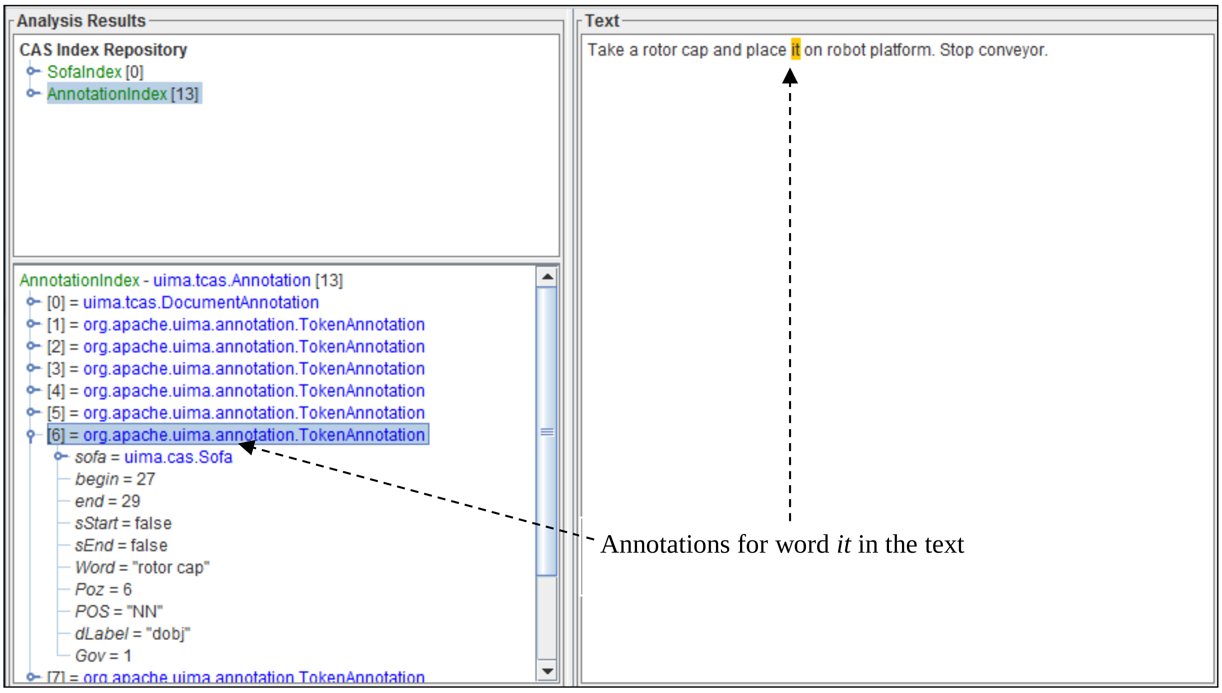
```
<?xml version="1.0" encoding="UTF-8"?>
<xmi:XMI xmlns:cas="http:///uima/cas.ecore" xmlns:xmi="http://www.omg.org/XMI" xmlns:tcas="http:///uima/tcas.ecore"
xmlns:annotation="http:///org/apache/uima/annotation.ecore" xmi:version="2.0"><cas:NULL xmi:id="0"/>
<cas:Sofa xmi:id="1" sofaNum="1" sofaID="_InitialView" mimeType="text" sofaString="Take a rotor cap and place it on robot platform. Stop conveyor."/>
<tcas:DocumentAnnotation xmi:id="8" sofa="1" begin="0" end="63" language="en"/>
   <annotation:TokenAnnotation xmi:id="13" begin="0" end="4" sStart="true" sEnd="false" Word="Take" Poz="1" POS="VB" dLabel="root" Gov="0"/>
    <annotation:TokenAnnotation xmi:id="24" begin="5" end="6" sStart="false" sEnd="false" Word="a" Poz="2" POS="DT" dLabel="det" Gov="6"/>
    <annotation:TokenAnnotation xmi:id="35" begin="7" end="16" sStart="false" sEnd="false" Word="rotor cap" Poz="3" POS="NN" dLabel="nn"
Gov="6"/>
    <annotation:TokenAnnotation xmi:id="46" begin="17" end="20" sStart="false" sEnd="false" Word="and" Poz="4" POS="CC" dLabel="" Gov="-1"/>
    <annotation:TokenAnnotation xmi:id="57" begin="21" end="26" sStart="false" sEnd="false" Word="place" Poz="5" POS="NN" dLabel="conj_and"
Gov="3"/>
    <annotation:TokenAnnotation xmi:id="68" begin="27" end="29" sStart="false" sEnd="false" Word="rotor cap" Poz="6" POS="NN" dLabel="dobj"
Gov="1"/>
    <annotation:TokenAnnotation xmi:id="79" begin="30" end="32" sStart="false" sEnd="false" Word="on" Poz="7" POS="IN" dLabel="" Gov="-1"/>
    <annotation:TokenAnnotation xmi:id="90" begin="33" end="47" sStart="false" sEnd="false" Word="robot platform" Poz="8" POS="NN"
dLabel="prep_on" Gov="6"/>
    <annotation:TokenAnnotation xmi:id="101" begin="47" end="48" sStart="false" sEnd="true" Word="." Poz="9" POS="." dLabel="" Gov="-1"/>
    <annotation:TokenAnnotation xmi:id="112" begin="49" end="53" sStart="true" sEnd="false" Word="Stop" Poz="1" POS="VB" dLabel="root" Gov="0"/>
    <annotation:TokenAnnotation xmi:id="123" begin="54" end="62" sStart="false" sEnd="false" Word="conveyor" Poz="2" POS="NN" dLabel="dobj"
Gov="1"/>
    <annotation:TokenAnnotation xmi:id="134" begin="62" end="63" sStart="false" sEnd="true" Word="." Poz="3" POS="." dLabel="" Gov="-1"/>
<cas:View sofa="1" members="8 13 24 35 46 57 68 79 90 101 112 123 134"/>
</xmi:XMI>
```

<annotation:TokenAnnotation xmi:id="13"
    begin="0" ← symbol in the text indicating the beginning of this word
    end="4" ← symbol in the text indicating the end of this word
    sStart="true" ← identifier indicating the beginning of the sentence
    sEnd="false" ← identifier indicating the end of the sentence
    Word="Take" ← analyzed word itself
    Poz="1" ← word position in the sentence
    POS="VB" ← part-of-speech tag
    dLabel="root" ← dependency label
Gov="0" ← governor node position in the text
/>

Fig.5. XML file (above) and explanations for the inserted annotations for the line: <annotation:TokenAnnotation xmi:id="13" begin="0" end="4" sStart="true" sEnd="false" Word="Take" Poz="1" POS="VB" dLabel="root" Gov="0"/> (below).

- **Dependency parsing**. Dependency parsing was done using the Stanford parser, which may determine 52 fine grained dependencies, referring to different relations between the words. We used a collapsed structure which ignores punctuation but involves prepositions and conjunctions into dependency labels. The example of the parsed sentence *"take a rotor cap and place it on robot platform"* is presented in Fig.4.

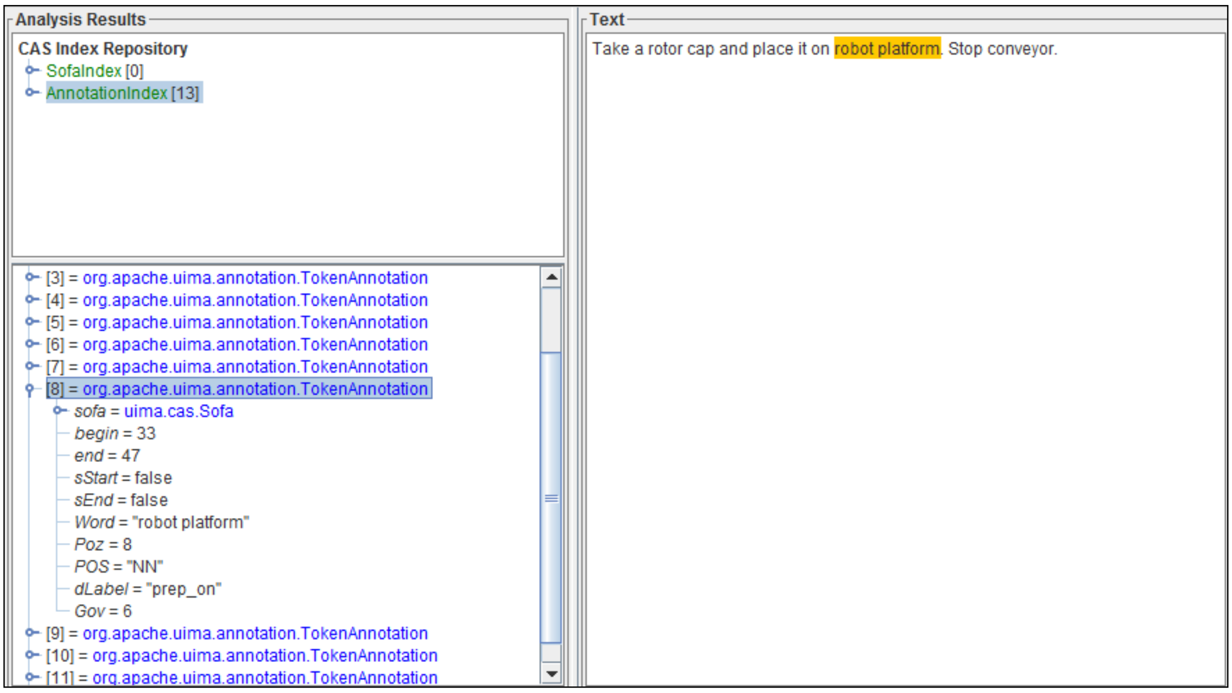
Post-processing. During this phase information about the mapped objects is restored (considering their replacements and positions in the sentences) and inserted into the text. Annotations are stored in an XML file (see the example in Figure 5); a few PrintScreens from CAS Visual Debugger are presented in Fig.6 – Fig.8 as well.


Analysis Results
CAS Index Repository
SofaIndex [0]
AnnotationIndex [13]

```
AnnotationIndex - uima.tcas.Annotation [13]
[0] = uima.tcas.DocumentAnnotation
[1] = org.apache.uima.annotation.TokenAnnotation
[2] = org.apache.uima.annotation.TokenAnnotation
[3] = org.apache.uima.annotation.TokenAnnotation
[4] = org.apache.uima.annotation.TokenAnnotation
[5] = org.apache.uima.annotation.TokenAnnotation
[6] = org.apache.uima.annotation.TokenAnnotation
    sofa = uima.cas.Sofa
    begin = 27
    end = 29
    sStart = false
    sEnd = false
    Word = "rotor cap"
    Poz = 6
    POS = "NN"
    dLabel = "dobj"
    Gov = 1
[7] = org.apache.uima.annotation.TokenAnnotation
```

Text
Take a rotor cap and place it on robot platform. Stop conveyor.

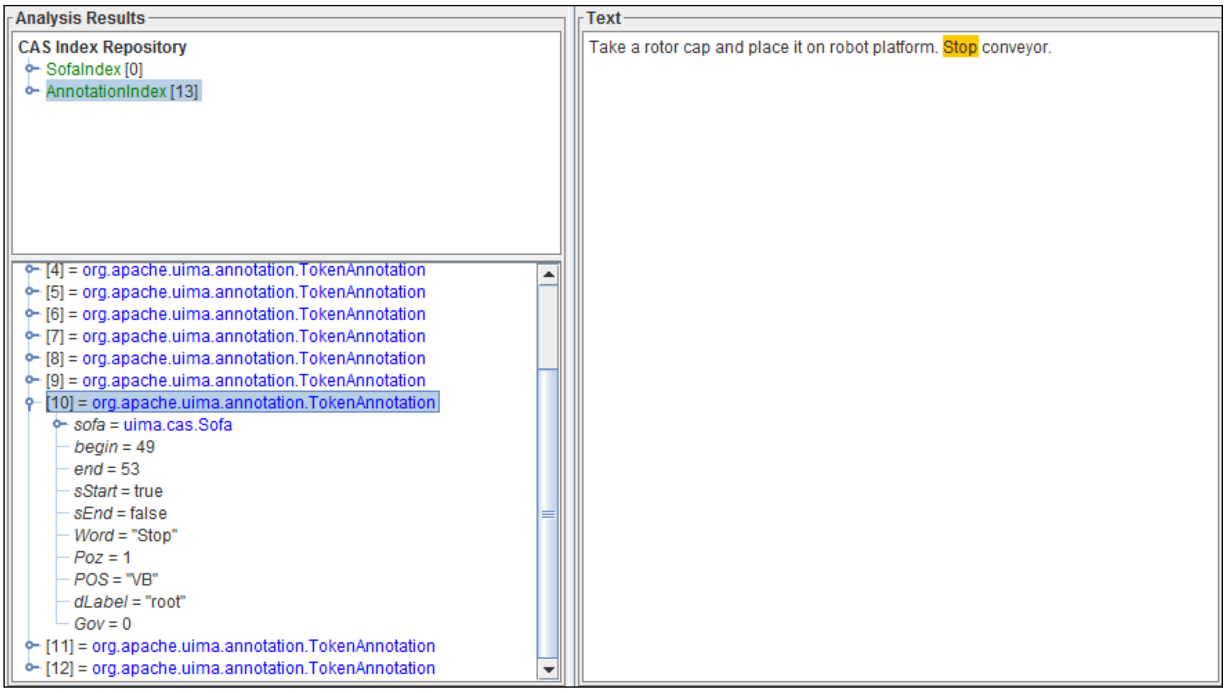
Fig.6. Detailed annotations for *it*



```
Analysis Results
CAS Index Repository
  SofaIndex [0]
  AnnotationIndex [13]

  [3] = org.apache.uima.annotation.TokenAnnotation
  [4] = org.apache.uima.annotation.TokenAnnotation
  [5] = org.apache.uima.annotation.TokenAnnotation
  [6] = org.apache.uima.annotation.TokenAnnotation
  [7] = org.apache.uima.annotation.TokenAnnotation
  [8] = org.apache.uima.annotation.TokenAnnotation
      sofa = uima.cas.Sofa
      begin = 33
      end = 47
      sStart = false
      sEnd = false
      Word = "robot platform"
      Poz = 8
      POS = "NN"
      dLabel = "prep_on"
      Gov = 6
  [9] = org.apache.uima.annotation.TokenAnnotation
  [10] = org.apache.uima.annotation.TokenAnnotation
  [11] = org.apache.uima.annotation.TokenAnnotation
```

```
Text
Take a rotor cap and place it on robot platform. Stop conveyor.
```

Fig.7. Detailed annotations for *robot platform*



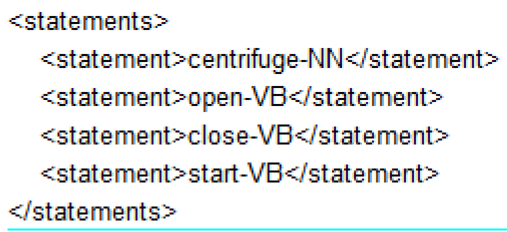
```
Analysis Results
CAS Index Repository
  SofaIndex [0]
  AnnotationIndex [13]

  [4] = org.apache.uima.annotation.TokenAnnotation
  [5] = org.apache.uima.annotation.TokenAnnotation
  [6] = org.apache.uima.annotation.TokenAnnotation
  [7] = org.apache.uima.annotation.TokenAnnotation
  [8] = org.apache.uima.annotation.TokenAnnotation
  [9] = org.apache.uima.annotation.TokenAnnotation
  [10] = org.apache.uima.annotation.TokenAnnotation
      sofa = uima.cas.Sofa
      begin = 49
      end = 53
      sStart = true
      sEnd = false
      Word = "Stop"
      Poz = 1
      POS = "VB"
      dLabel = "root"
      Gov = 0
  [11] = org.apache.uima.annotation.TokenAnnotation
  [12] = org.apache.uima.annotation.TokenAnnotation
```

```
Text
Take a rotor cap and place it on robot platform. Stop conveyor.
```

Fig.8. Detailed annotations for *Stop*

There is always a possibility, that POS and dependency parser will return incorrect annotation results due to ambiguities in the sentence. This can happen when instructions are domain specific (e.g., chemical or mechanical texts) or have unusual grammatical and syntactic sentence structure (e.g., imperative mood used for verbs in instructions, which is many times incorrectly interpreted by the standard Stanford parser). For example, the instruction “*Open centrifuge”* is annotated as *“Open-NNP centrifuge-VBP”* (POS annotation) and relation “*nsubj(centrifuge-2, Open-1)”* (dependency parsing results). In this case, “*centrifuge”* is interpreted as verb form and “*open*” as a noun form; the interpretation of the entire sentence by a parser is similar to “*Object “Open” is being centrifuged”.*

Wrong parser results affect the overall quality of the compiler. In order to reduce the probability of parser errors, an additional dictionary in the form of an XML file with “always true” statements based on instructions in the ACAT instruction sheets has been introduced (see the snippet in Fig.9). For example, “*<statement>centrifuge-NN</statement>”* from the dictionary means, that the word “*centrifuge”* should always be interpreted as a noun, as “*centrifuge”* always takes the role of an object in the instructions we were analyzing. Though this is not a universal solution, it allows making a shortcut for obtaining bigger percentages of correct parsing results which, in turn, allows better testing of the other more robotics specific functionalities of the textual instruction completion sub-system. Alternative solutions are discussed in the “Discussion” section.



```
<statements>
   <statement>centrifuge-NN</statement>
   <statement>open-VB</statement>
   <statement>close-VB</statement>
   <statement>start-VB</statement>
</statements>
```

Fig.9. Snippet from the XML file containing “always true” statements

4.2. Advanced instruction text analysis

In [Nyga and Beetz 2012] we have proposed PRAC (Probabilistic Robot Action Cores), a novel approach to the problem of action-specific knowledge processing, representation and acquisition by autonomous robots performing everyday activities. PRAC is a probabilistic first-order knowledge base which can be acquired from annotated natural language text. In [Nyga and Beetz 2012] we have also discussed how to address the problems of incompleteness, under-specification and ambiguity of naturalistic action specifications and how PRAC models can tackle those. In our recent research PRAC and its underlying formal framework (Markov Logic Networks, MLN) have been extended into two different directions:

1. In [Nyga and Beetz 2015, submitted] composite likelihood learning (CLL) is described. CLL is a method for parameter estimation in Markov logic networks, which is a generalization of both likelihood and pseudo-likelihood learning allowing for a tradeoff between computational costs and learning accuracy.

2. In [Nyga and Beetz 2014, submitted] we propose an extension of MLNs for enabling efficient incorporation of concept taxonomies in probabilistic relational models. By allowing evidence ground atoms taking real-valued degrees of truth, the proposed knowledge representation

formalism is capable of compactly covering semantic similarity of concepts given by a class taxonomy and, hence, allows efficient reasoning about concepts not contained in the model. While in classical Markov logic, clique potentials in the ground Markov random field take discrete binary values that are determined by first-order logical formulas, our approach employs fuzzy logic semantics for each of the feature functions. This makes the proposed method indeed a generalization of classical Markov logic being fully compatible with its original semantics.

Another focus of our work in the area of text analysis, annotation, and completion was on the automated detection and classification of causal relations in texts from the CHEMLAB domain. We use lexico-syntactic templates for relation extraction, which are in the tradition of Hearst patterns [Hearst 1992]. The input text is preprocessed and has both standard POS tags and domain-specific labels (e.g., a label for chemical substances). It also includes syntactic labels for phrase types like NP etc. For the purpose of linguistic preprocessing (tokenization, POS tagging, and syntactic analysis) we used components of the Stanford CoreNLP framework [Toutanova 2013]. For some technical terms of the chemistry domain, especially complicated names of chemical substances (e.g., rac-N-[(3-methylamino-1-phenyl)propyl]-5-(dimethylamino)-1-naphthale nesulfonamide), it turned out that the error rate of the preprocessing pipeline could be reduced significantly by employing a domain-specific semantic annotation tool, the OSCAR4 tagger [Jessop 2011].

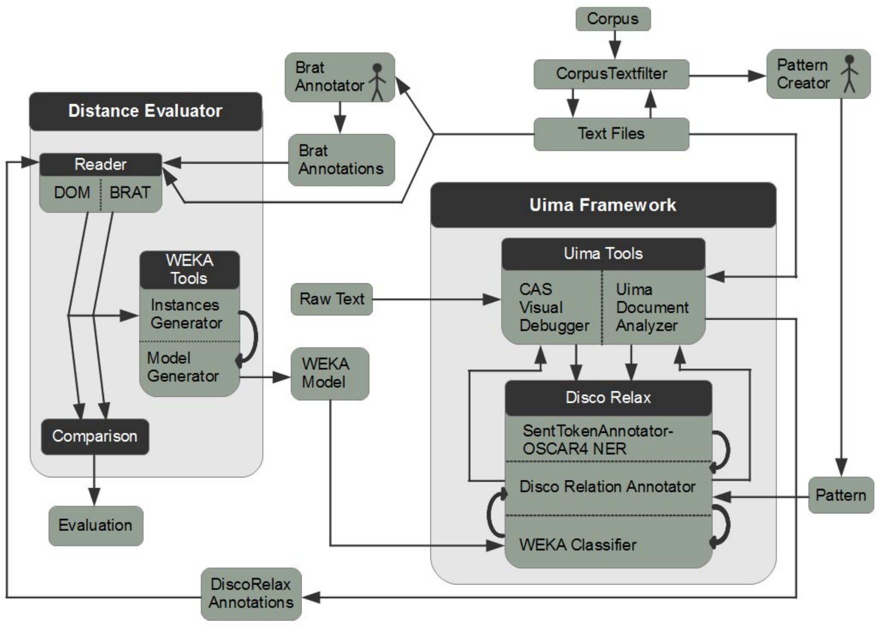


Fig. 10. Architecture overview of the causal relation extraction engine.

The basic structure of an annotation pattern is:

<type> /arg1 ... /span ... /arg2 ...

where "type" is the label of the specific causal relation (e.g., CAUSAL_INSTRUMENT_UsedFor}) "/arg1" and "/arg1" are the arguments of the relation, and "/span" is the connective. Our implementation of the pattern interpreter supports the usual operators for the specification of regular expressions, e.g. the Kleene star, optionality of subpatterns, and XOR.

The following example pattern matches passive clauses which include the lexical pattern {\em used as} either with or without an optional adverb:

The nitrogen was used as carrier gas.

The nitrogen was permanently used as carrier gas.

Pattern: [NP] [VBD] [RB]? used as [NP]

Our extraction engine DiscoRelax has been implemented within the UIMA framework [Ferrucci 2004] which has, amongst others, the advantage that it can be easily integrated with other annotators. The overall architecture is given in Fig 10.

4.3. Action ontology

As instruction textual completion critically depends on the used ontology, here we will briefly discuss the changes introduced into ACAT ontology as compared to the structures presented in D2.1 and the Year 1 Project Report.

Initial textual instruction completion experiments were made with the ACAT ontology developed in the Year 1 of the project (built from texts and based on the structure presented in deliverable D2.1). After analyzing first experiment results for instruction completion, the following problems were identified:

- Action ontology is fragmented, information on some typical actions and objects is missing, for different action words the level of detailed-ness differs.
- Not sufficient space for possible language interpretations when analyzing new instructions (e.g. using synonyms, etc.).
- Insufficient hierarchical information: e.g. not enough hypernyms and hyponyms, which could allow ontology class substitution in case of new actions.
- Structural information on object interdependence, e.g. an object being a part of another object (meronym and holonym information) is rather scarce.
- Ambiguous meaning of some ontology objects and actions.

It was decided to focus the ontology and increase its quality in the following way:

1) Select a focused set of instructions (in written or video format; in the latter case manual transcriptions of the videos in instruction form are used), which would represent the area of experiments done both in CHEMLAB and IASSES scenarios.

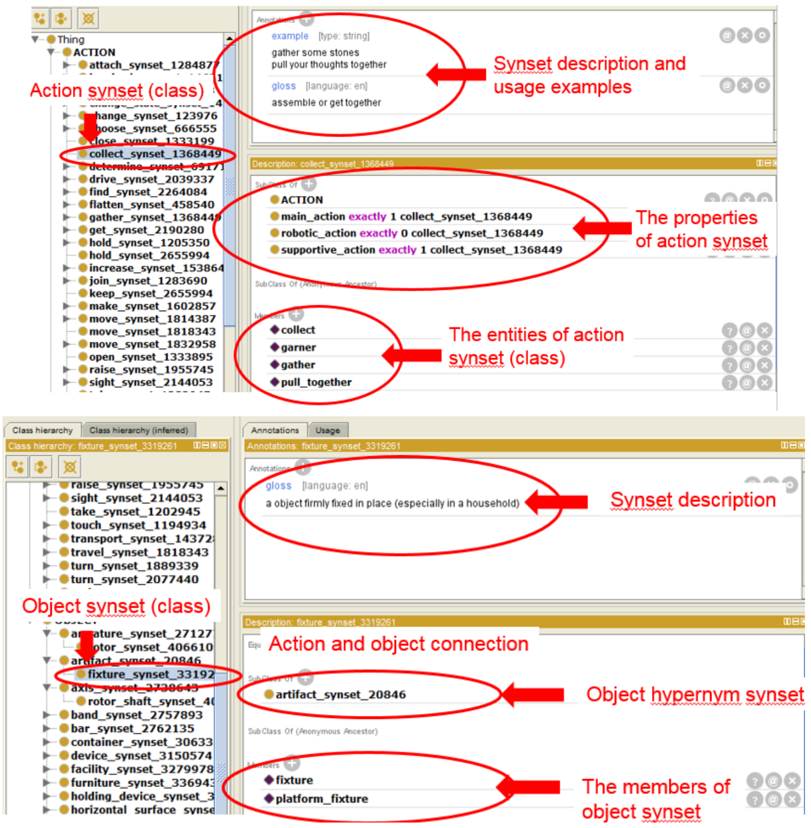
2) Construct core ontologies from main actions and objects used in those actions from the selected instruction sets for both CHEMLAB and IASSES scenarios.
3) Expand the core ontologies with relevant information from Wordnet lexical ontology, adding synset, hypernym and hyponym information.
4) Expand the core ontologies with relevant expert information about “part of relations”.
5) Integrate the core ontologies with earlier built corpus text-based ontologies.

For this reason, the following structure adjustments were made to the action ontology:

1) Classification into pre-defined action classes (e.g. tool action, tool with mover action etc., see D2.1) is skipped in order to avoid the possible limitations on the number of such classes. This classification is intended to be brought back later by applying automated classification procedures to action environment information in the ontology.
2) General classes (ACTION and OBJECT) previously introduced into the ACAT ontology remain as they were.
3) Corresponding classes (ACTION and OBJECT) and relations from Wordnet lexical ontology are used.
4) Actions entities are displayed as individuals of actions synsets (class members).
5) Each action class is described by properties: main action, robotic action and supportive action.
6) Action individuals are connected with object individuals by relationships: “with main object”, “with primary object”, “with secondary object”, objects holonyms/meronyms - by property: “part of”.

Extracted actions and objects from focused text instructions and the transcribed focused videos are used as basic data for focused action ontology for IASSES and CHEMLAB scenarios.

Some illustrations on the obtained ontology structure are given in the figures next. The leftmost frames in Fig. 11 show the hierarchical structure of action and object classes (synsets). (A synset is a set of entities with the same meaning, either an action, or an object.) Actions entities are displayed as individuals of actions synsets (class members) and each synset class is described by gloss and examples from Wordnet.



gather some stones
pull your thoughts together

assemble or get together

ACTION
main_action exactly 1 collect_synset_1368449
robotic_action exactly 0 collect_synset_1368449
supportive_action exactly 1 collect_synset_1368449

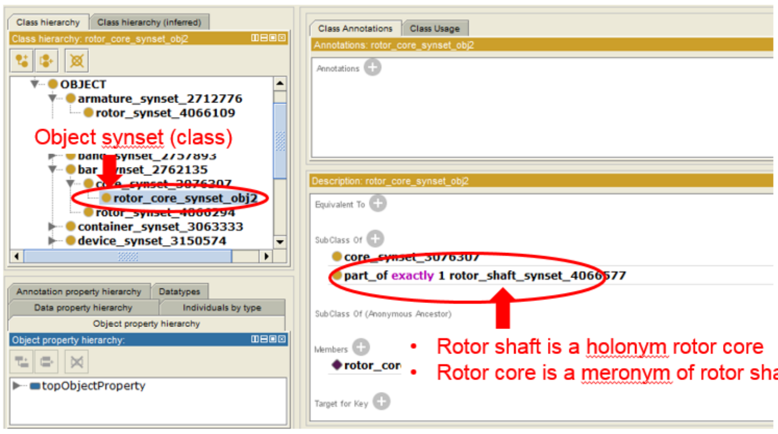
collect
garner
gather
pull_together

a object firmly fixed in place (especially in a household)

artifact_synset_20846

fixture
platform_fixture

Fig. 11. ACAT ontology structure: action and object synsets (IASSES scenario)



core_synset_3076307
part_of exactly 1 rotor_shaft_synset_4066577

rotor_cor

- Rotor shaft is a holonym rotor core
- Rotor core is a meronym of rotor shaft

Fig. 12. ACAT ontology structure: part relations (IASSES scenario)

Each action class is described by the following properties: “main action”, “robotic action” and “supportive action” (variables: main_action, robotic_action and supportive_action in the ontology, upper table in Fig. 11). The property “robotic action” (1 or 0) allows us to distinguish at which level (directly associated to robotic action or not) an instruction is given. For the second year of the project we are analyzing only instruction directly associated to robotics actions (like pick&place, open, close or shake). Interpretation of instructions not directly associated to robotic actions (like neutralize or harvest) is left for the last year of the project, as already was indicated in the introduction.

With the help of properties “main action” and “supportive action” we solve the question of mapping instructions into ADTs. Note, we want to create one ADT for each so called “ADT-action”, which starts with the hand approaching and grasping an object and ends with the hand releasing an object (and retracting). The property “main action” is given to the “central” action word (verb) to which ADT is to be associated. The property “supportive” action is given to action words which shall be mapped to parts of ADT. For example, if one has an instruction in the instruction sheet “Pick up the test tube and shake it”, we want an ADT associated only to the verb “shake”, where picking up of the test tube will be just initial part of the ADT “shake”. Another part of the same ADT will indicate putting down of the test tube after shaking, even though the putting down was not mentioned in the example instruction explicitly. Linguistically, supportive actions can either be mentioned in the instruction or not, but the main action will always be mentioned. Thus, the properties “main action” and “supportive action” were introduced to dis-entangle action-word-to-ADT mapping which is not straightforward, due to natural omissions in language.

Further, action individuals are connected with object individuals by the following relationships: with main object, with primary object, with secondary object. Objects are connected with holonyms/meronyms by the “part of” relation.

Synsets are organized in a hierarchical structure, where a synset can have superordinate and subordinate synsets. E.g., the “*container”* synset is the hypernym for the “*box”* synset.

Fig. 12 shows the members of a synset with the holonym/meronym relations, showing which objects are parts of other objects.

4.4. Filling in missing information with knowledge from ACAT action ontology

In the process of instruction completion, the ACAT ontology is queried for the following purposes:

- to extract action structure for known actions,
- to get detailed information for objects participating in an action.

By the structured conceptual view of stored data, the ACAT ontology is used in describing the class hierarchy (taxonomy) of the concepts, concept properties and allows filling missing information (e.g. which object can be used with a given action) by using known relationships.

Fig. 13 presents the conceptual model of knowledge extraction by querying the ACAT action ontology. Each query makes use of the instruction parsing results – recognized verbs (actions that the robot has to perform) and objects associated to the action.

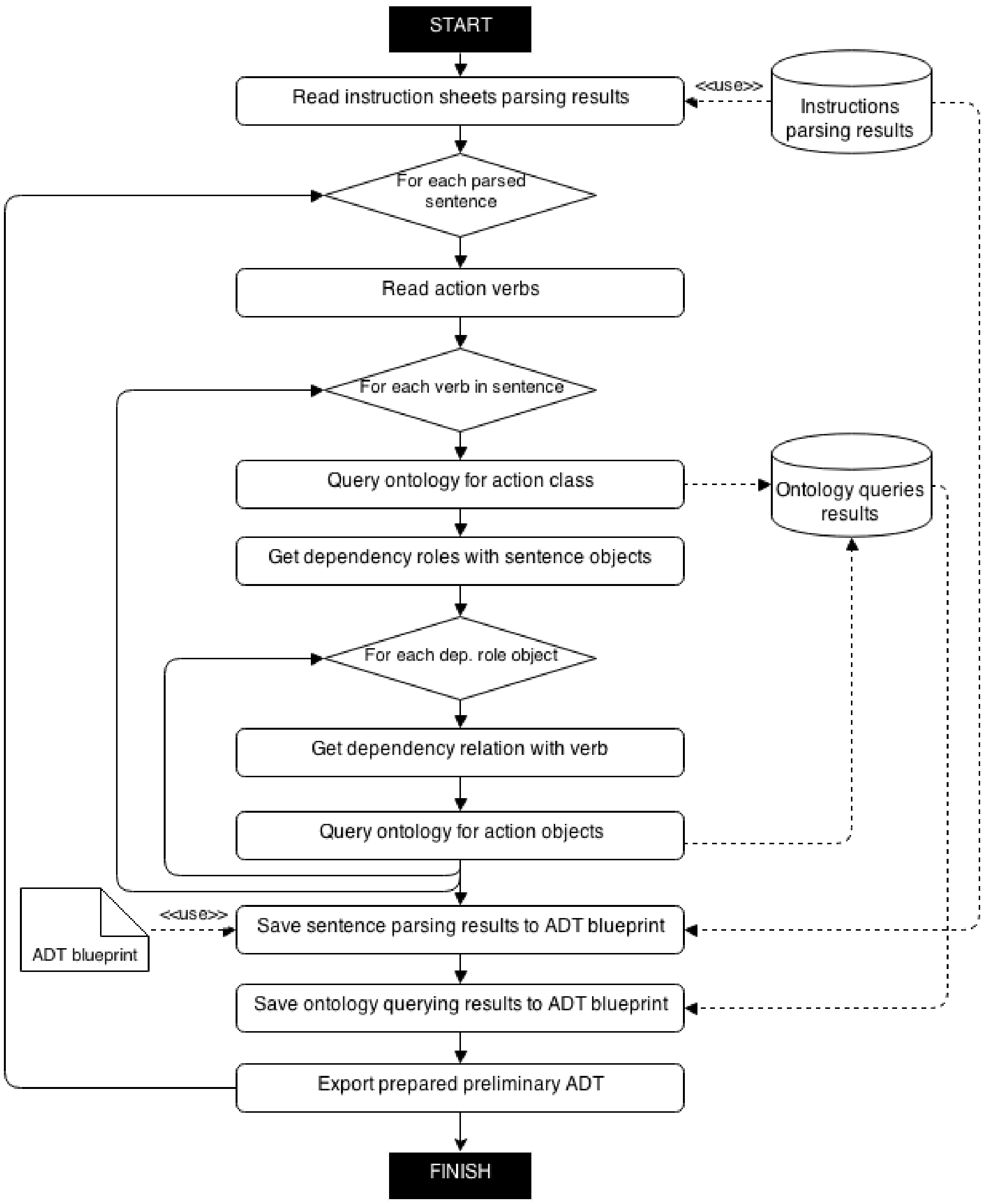

START
Read instruction sheets parsing results
<<use>>
Instructions parsing results
For each parsed sentence
Read action verbs
For each verb in sentence
Query ontology for action class
Ontology queries results
Get dependency roles with sentence objects
For each dep. role object
Get dependency relation with verb
Query ontology for action objects
ADT blueprint
<<use>>
Save sentence parsing results to ADT blueprint
Save ontology querying results to ADT blueprint
Export prepared preliminary ADT
FINISH

Fig. 13. Knowledge extraction by querying the ACAT ontology

Ontology querying starts after gathering the results of syntax dependency and POS parsing (described in section 4.1). Verbs are extracted from each parsed sentence and verbs with the property “main_action=1” are identified. Each of those verbs corresponds to one ADT that has to be created.

By querying the ontology using SPARQL queries we can define the action class, which in turn defines the structure (e.g. how many objects and in which roles are involved) for the action. An example of a SPARQL query for defining the action class for an action "harvest" is given in Fig. 14.

Query:
```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX corpus_ontology: <http://www.semanticweb.org/user/ontologies/2013/11/robot_ontology#>

SELECT DISTINCT ?action ?property ?object
 WHERE {?class rdfs:subClassOf* corpus_ontology:ACTION.
        ?action ?property ?object.
        FILTER(?action=corpus_ontology:harvest)
       }
```

Fig. 14. Example: SPARQL query for action information extraction

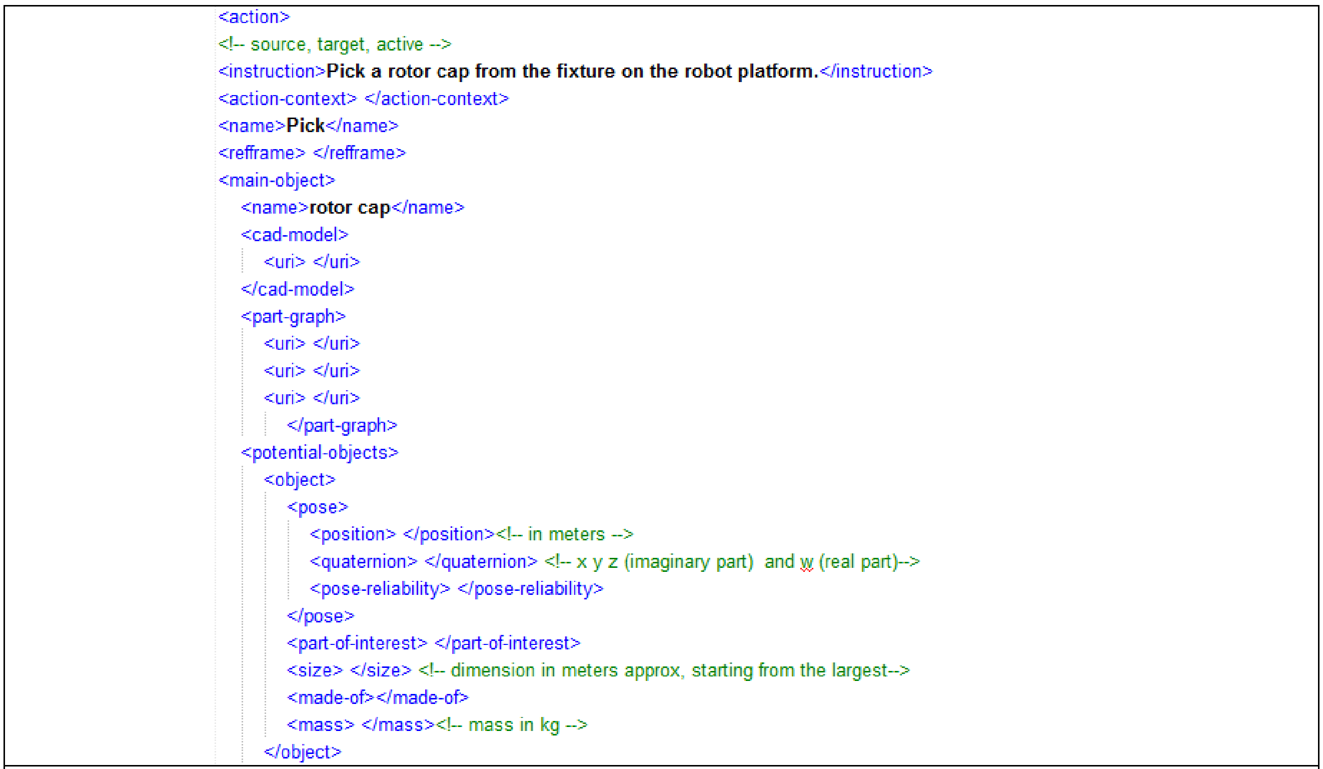
Knowing the structure of the action, the next step of instruction sheet knowledge processing is to fill-in information for each object involved in the action. Objects and their relations to main action (roles) need to be defined. Dependency parsing data is used for describing the syntactic roles of the objects. Additional action object properties, which are not mentioned explicitly in the instruction sheets, can be obtained by querying the ACAT ontology using SPARQL queries (Fig. 15). If an action background object is not mentioned in the instruction sheet, there is the possibility to query the most probable action object from the list of objects, included in ACAT ontology.

Query:
```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX corpus_ontology: <http://www.semanticweb.org/user/ontologies/2013/11/robot_ontology#>

SELECT DISTINCT ?action_object ?property ?object
 WHERE {?action_object ?property ?object.
        FILTER(?action_object=corpus_ontology:cell)
       }
```

Fig. 15. SPARQL query for describing properties of an action object.

Main action as well as object roles (main object, primary object, secondary object, tool, see Appendix A.4 for object role definitions) are defined by analyzing the sentence structure and dependency parsing results. Syntactic relations between words in the sentence are useful for object role identification. Relation *"dobj"* identifies direct (main) object of an action word (verb) in the sentence. A corresponding noun phrase is taken as the object. (E.g. in a sentence "*Put rotor cap on conveyor",* the "*rotor cap*" is the main object of the action verb "*put*"). Additional rules are needed to identify primary and secondary objects. (E.g. in the mentioned sentence "*Put rotor cap on conveyor",* the "*conveyor*" in ACAT notation is the secondary object of the action verb "*put*"). Syntactic patterns with prepositional modifiers (e.g. *prep_in, prep_on, prep_for*) are used in the recognition of primary and secondary objects. List of all syntactic rules can be found in Stanford parser typed dependencies manual[3].

A preliminary action data table (ADT), filled out with data from instruction sheets and ACAT ontology queries, is constructed by filling in general instruction information: instruction description (e.g. *Pick a rotor cap from the fixture and put it on the robot platform*), action name (e.g. *pick*), main object (e.g. *rotor cap*), primary object (e.g. *fixture*) and secondary object (e.g. *robot platform*) (Fig. 16). This preliminary ADT can later be updated with signal level data which is the topic of D3.2.



```
<action>
<!-- source, target, active -->
<instruction>Pick a rotor cap from the fixture on the robot platform.</instruction>
<action-context> </action-context>
<name>Pick</name>
<refframe> </refframe>
<main-object>
  <name>rotor cap</name>
  <cad-model>
    <uri> </uri>
  </cad-model>
  <part-graph>
    <uri> </uri>
    <uri> </uri>
    <uri> </uri>
      </part-graph>
  <potential-objects>
    <object>
      <pose>
        <position> </position><!-- in meters -->
        <quaternion> </quaternion> <!-- x y z (imaginary part)  and w (real part)-->
        <pose-reliability> </pose-reliability>
      </pose>
      <part-of-interest> </part-of-interest>
      <size> </size> <!-- dimension in meters approx, starting from the largest-->
      <made-of></made-of>
      <mass> </mass><!-- mass in kg -->
    </object>
```

Fig. 16. Example of a preliminary ADT structure

5. Textual instruction completion experiments for ACAT project scenarios

All the results presented in this chapter were obtained using the “ACAT Instruction Compiler” for which the algorithmic details were provided in Chapter 4 and the documentation is provided in Appendix A.1.

[3] http://nlp.stanford.edu/software/dependencies_manual.pdf

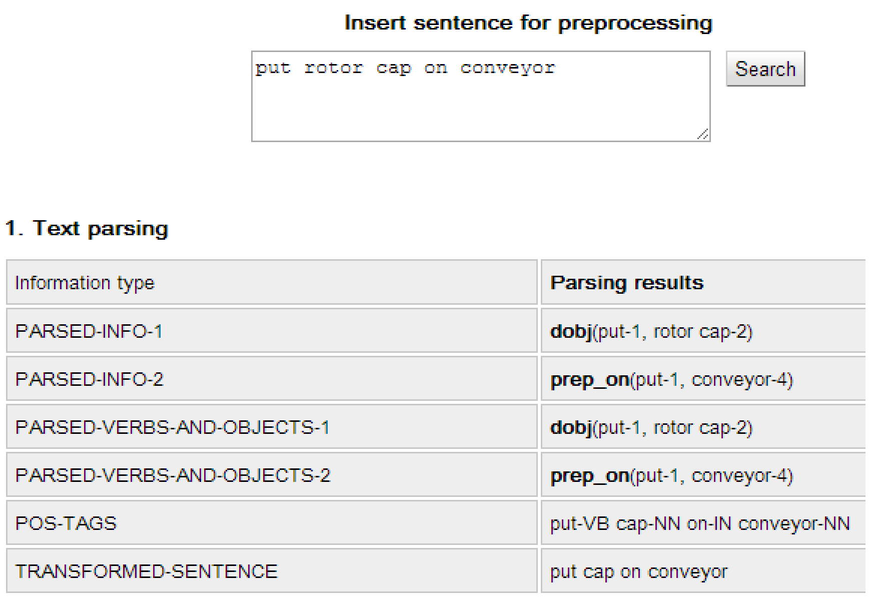
For both scenarios, IASSES and CHEMLAB, the ACAT instruction completion application gives results of the same structure:

- syntactic analysis results with action words (verbs) and their dependency with action objects (nouns);
- ACAT ontology query results with identified action synsets, action synset description and possible action objects;
- ACAT ontology query results with identified action, objects and their properties;
- preliminary ADT's for each action step.

Instruction parsing algorithms are the same for both scenarios. The difference is in the knowledge base, which is used for extracting information needed for instruction completion. Each scenario has a different ontology, built from domain-specific texts and focused, using instruction sets from specific area of planned experiments.

5.1. IASSES scenario

First, the ACAT instruction parsing and ontology quering in IASSES scenario will be demonstrated "step by step" for an instruction: *"Put rotor cap on conveyor"* (Fig. 17). Later we will give more condensed analysis results for the entire instruction sheet of the IASSES scenario as presented in the Appendix A.2.


Insert sentence for preprocessing

```
put rotor cap on conveyor
```

1. Text parsing

Information type	**Parsing results**
PARSED-INFO-1	**dobj**(put-1, rotor cap-2)
PARSED-INFO-2	**prep_on**(put-1, conveyor-4)
PARSED-VERBS-AND-OBJECTS-1	**dobj**(put-1, rotor cap-2)
PARSED-VERBS-AND-OBJECTS-2	**prep_on**(put-1, conveyor-4)
POS-TAGS	put-VB cap-NN on-IN conveyor-NN
TRANSFORMED-SENTENCE	put cap on conveyor

Fig. 17. Example of text parsing results for IASSES scenario

Instruction parsing starts from complex object mapping into simple ones. In this case, object *rotor cap* is mapped to simple *cap* (Fig. 17 – "*Transformed sentence"* row information).

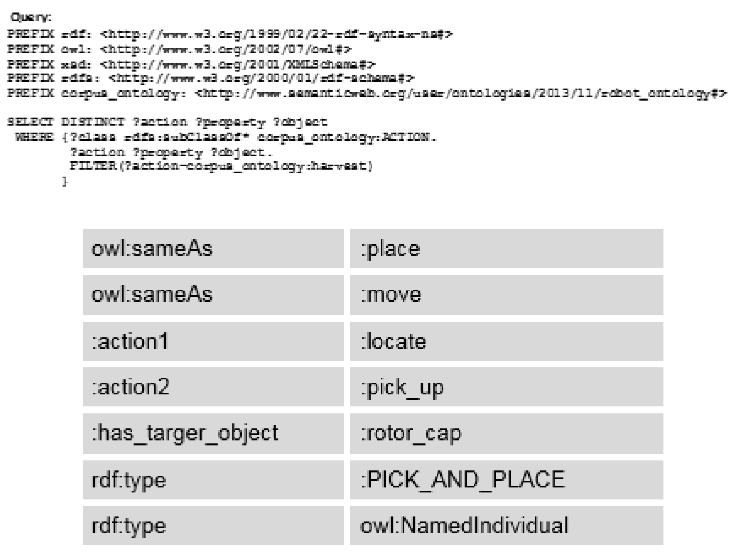
In the next step, POS and dependency parsing is executed for the instruction sentence. Each word in the instruction sentence is described by its part of speech and also dependencies between the words are defined (Fig. 17).

Sentence *"Put rotor cap on conveyor"* POS annotation (*"POS tags"* row information in Fig. 17) results: put – verb, base form (VB), cap – noun, singular (NN), on – preposition (IN), conveyor – noun, singular (NN).[4]

Dependency parsing annotation results are presented in the *"Parsed info"* rows (Fig 17): ***dobj****(put-1, rotor cap-2)* means, that the *rotor cap* is the direct object of the action *put*, ***prep_on****(put-1, conveyor-4)* – the target object (place) of the action *put* is the *conveyor.* We used a collapsed dependencies model, which attaches all preposition to a basic prepositional modifier role.[5]

In case the analyzed sentence is more complex and contains information not only about actions and objects, then verb and noun dependency analysis results are filtered (*"Parsed verbs and objects"* rows) – this information is used in the ontology querying step.

After saving the parsing results to an application internal database, the ontology querying phase begins. For convenience, the ontology querying results can be grouped into ACTION-INFORMATION and OBJECT-INFORMATION logical groups (Fig 18).



```
Query:
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX corpus_ontology: <http://www.semanticweb.org/user/ontologies/2013/11/robot_ontology#>

SELECT DISTINCT ?action ?property ?object
 WHERE {?class rdfs:subClassOf* corpus_ontology:ACTION.
        ?action ?property ?object.
        FILTER(?action=corpus_ontology:harvest)
       }
```

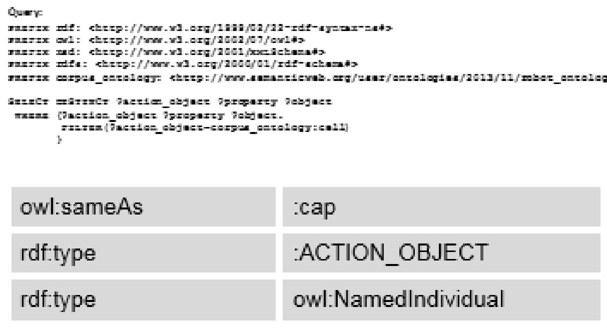
owl:sameAs	:place
owl:sameAs	:move
:action1	:locate
:action2	:pick_up
:has_targer_object	:rotor_cap
rdf:type	:PICK_AND_PLACE
rdf:type	owl:NamedIndividual

Fig. 18. Sample ontology querying results for IASSES scenario

For each action verb in a sentence, an ontology SPARQL query is built. All corresponding verb instances in the ontology, belonging to the ACTION category, are selected. Query results also

[4] Full list of Penn Treebank POS tags, used in Stanford NLP project: https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html
[5] Full list of dependency types, used in Stanford NLP project: http://nlp.stanford.edu/software/dependencies_manual.pdf

describe that the action *put* is similar to actions *place* and *move* (owl:sameAs – *move, place*). All these actions belong to the same synset. Ontology querying results also include synonyms, both for actions and for action background elements (objects), wherever possible. In the example (Fig. 19), *rotor cap* is defined as an object similar (owl:sameAs) to *cap*. It is defined as ontology entity (owl:NamedIndividual).



Query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX corpus_ontology: <http://www.semanticweb.org/user/ontologies/2013/11/robot_ontology#>

SELECT DISTINCT ?action_object ?property ?object
 WHERE {?action_object ?property ?object.
        FILTER(?action_object=corpus_ontology:cell)
       }
```

owl:sameAs	:cap
rdf:type	:ACTION_OBJECT
rdf:type	owl:NamedIndividual

Fig. 19. Sample ontology querying results for IASSES scenario

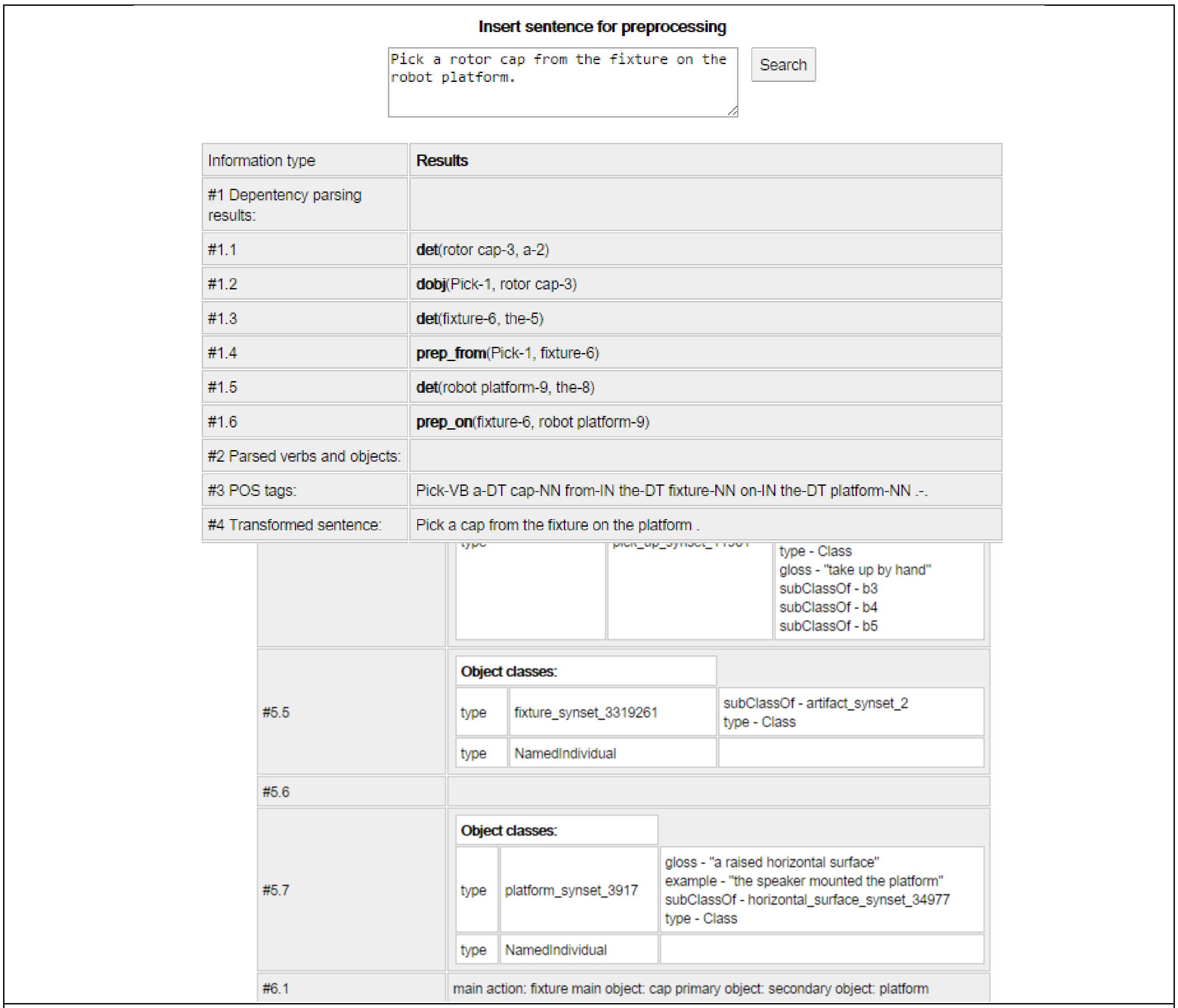
Further, the instruction completion experiment, using the ACAT Instruction Compiler was run for the set of IASSES instructions taken from the IASSES instruction sheet (Appendix A.2). Instructions 1 to 7 from the sheet were processed. Fig. 20 presents Compiler screenshots with the examples of dependency parsing, ontology querying and reasoning results.

The results of the instruction completion experiment for the IASSES scenario are presented in Table 1. The table includes the sentence, which was parsed, dependency parsing results and ontology querying results. It also includes analysis results on recognized actions and objects, which are included in the examples of preliminary ADTs.

Table 1 is to be read as follows:

The column “*Command”* presents an instruction in natural language the way it is written in the instruction sheet, e.g. “Take one metal ring from the top of ring stack and put it into the ring dispenser”.

The column “*Dependency and POS parsing results”* presents part of speech parsing results, as well as dependencies between extracted entities; e.g. for part of speech: “Take-VB” denotes that “Take” is a verb; “one-CD” denotes that “one” is a cardinal number and “metal-NN” denotes that “metal” is a noun, etc.; for dependencies e.g.: “num(ring-4, one-2)” indicates that “one” shows number of “rings”; “dobj(Take-1, ring-4)” indicates that the “ring” is the object of the verb “take”, etc.



Insert sentence for preprocessing

```
Pick a rotor cap from the fixture on the
robot platform.
```

Information type	**Results**
#1 Depentency parsing results:	
#1.1	**det**(rotor cap-3, a-2)
#1.2	**dobj**(Pick-1, rotor cap-3)
#1.3	**det**(fixture-6, the-5)
#1.4	**prep_from**(Pick-1, fixture-6)
#1.5	**det**(robot platform-9, the-8)
#1.6	**prep_on**(fixture-6, robot platform-9)
#2 Parsed verbs and objects:	
#3 POS tags:	Pick-VB a-DT cap-NN from-IN the-DT fixture-NN on-IN the-DT platform-NN .-.
#4 Transformed sentence:	Pick a cap from the fixture on the platform .

	[illegible]	[illegible]	type - Class gloss - "take up by hand" subClassOf - b3 subClassOf - b4 subClassOf - b5
#5.5	**Object classes:**		
	type	fixture_synset_3319261	subClassOf - artifact_synset_2 type - Class
	type	NamedIndividual	
#5.6			
#5.7	**Object classes:**		
	type	platform_synset_3917	gloss - "a raised horizontal surface" example - "the speaker mounted the platform" subClassOf - horizontal_surface_synset_34977 type - Class
	type	NamedIndividual	
#6.1	main action: fixture main object: cap primary object: secondary object: platform		

Fig. 20. Compiler screenshot examples (IASSES scenario)

The column *"Ontology quering results"* shows information extracted from the ontology for action and objects indicated in the instruction. All synsets associated with the appropriate instance of the verb or the noun in the sentence, including Wordnet synset ID (when available) are indicated (e.g. put_synset_1481373) as well as detailed synset description, including Wordnet gloss (e.g. "put into a certain place or abstract location") and Wordnet class hierarchy (e.g. subClassOf - move_synset_1832958).

The column *"Recognized action and objects"* show actual results of the instruction textual completion procedure. These results are later filled into the ADT. These include main action of the instruction, main object, primary object and secondary object parsed from an instruction sentence.

The column *“ADT blueprint”* has links to actual ADTs pre-filled with symbolic information for the corresponding instructions, which are downloadable from the document in computer format. For the printed material we provide an excerpt of an example of the ADT blueprint in Fig. 21. Note, that the ADT at this stage contains information only about analyzed instruction and its main action, main object, primary object and secondary object.

Table 1. Results of textual instruction completion experiment (IASSES scenario)

Command	**Dependency and POS parsing results**	**Ontology querying results**		Recognized actions and objects	**ADT blueprint**

Take one metal ring from the top of ring stack and put it into the ring dispenser.	num(ring-4, one-2)
	dobj(Take-1, ring-4)
	det(top-7, the-6)
	prep_from(Take-1, top-7)
	prep_of(top-7, stack-10)
	conj_and(Take-1, put-12)
	dep(put-12, stack-13)
	det(dispenser-17, the-15)
	prep_into(stack-13, dispenser-17)
	Take-VB one-CD metal-NN ring-NN from-IN the-DT top-NN of-IN ring-NN stack-NNS and-CC put-VB stack-VB into-IN the-DT ring-NN dispenser-NN
		take_synset_17	example - string
			subClassOf - ACTION
			type – Class
				• main action: put
				• main object: ring
				• primary object: stack
				• secondary object: dispenser
					ADT_blueprint_put_ring.xml

		take_synset_12	example - string subClassOf - ACTION type – Class		
		remove_synset_17	subClassOf - ACTION type - Class example - string		
		choose_synset_666555	subClassOf - ACTION type – Class		
		pull_synset_1338932	example - string subClassOf - remove_synset_17 type – Class		
		hold_synset_12	subClassOf - ACTION type – Class		
		put_synset_1481373	gloss - "put into a certain place or abstract location" example - string subClassOf - move_synset_1832958 type – Class		
		put_synset_148	example - "That song put me in awful good humor" subClassOf - change_synset_123976 type - Class gloss - "put into a certain place or abstract location" example - string subClassOf - move_synset_1832958		
		pose_synset_1481373	gloss - "put into a certain place or abstract location" example - string subClassOf - move_synset_1832958 type – Class		
		place_synset_1481373	gloss - "put into a certain place or abstract location" example - string subClassOf - move_synset_1832958 type – Class		

		ring_synset_3495342	gloss - "a rigid circular band of metal or wood or other material used for holding or fastening or hanging or pulling" example - "there was still a rusty iron hoop for tying a horse" subClassOf - band_synset_2757893 type – Class		
		discard_synset_22	gloss - "Throw or cast away" example - string subClassOf - get_rid_of_synset_22 type – Class		

Grasp the ring from the side at the jaw of the ring dispenser and drop it into a cylindrical holder standing on the station.	det(ring-3, the-2)
	dobj(drop-15, dispenser-16)
	det(standing-21, a-18)
	amod(standing-21, cylindrical-19)
	prep_into(drop-15, standing-21)
	det(station-24, the-23)
	prep_on(standing-21, station-24)
	dobj(Grasp-1, ring-3)
	det(side-6, the-5)
	prep_from(Grasp-1, side-6)
	det(jaw-9, the-8)
	prep_at(side-6, jaw-9)
	det(dispenser-13, the-11)
	prep_of(jaw-9, dispenser-13)
	conj_and(Grasp-1, drop-15)
	Grasp-VB the-DT ring-NN from-IN the-DT side-NN at-IN the-DT jaw-NN of-IN the-DT ring-NN dispenser-NN and-CC drop-VB dispenser-NN into-IN a-DT cylindrical-JJ holder-NN standing-NN on-IN the-DT station-NN
		grasp_synset_1204684	Gloss – “Hold firmly”
			Example – string
			subClassOf - hold_synset_1205350
			type – Class
				• main action: drop
				• main object: ring
				• primary object: dispenser
				• secondary object: holder
					ADT_blueprint_drop_ring.xml

		ring_synset_3495342	gloss - "a rigid circular band of metal or wood or other material used for holding or fastening or hanging or pulling" example - "there was still a rusty iron hoop for tying a horse" subClassOf - band_synset_2757893 type – Class		
		take_synset_2	gloss - "take something or somebody with oneself somewhere" example - string subClassOf - transport_synset_1437285 type – Class		
		bring_synset_1421	subClassOf - ACTION type – Class		
		holder_synset_3487214	gloss - "a holding device" example - "a towel holder" subClassOf - holding_device_synset_3487525 type – Class		

Take the rotor shaft from the fixture and insert it into the cylindrical holder.	det(shaft-4, the-2)
	dobj(Take-1, shaft-4)
	det(fixture-7, the-6)
	prep_from(Take-1, fixture-7)
	conj_and(Take-1, insert-9)
	dobj(insert-9, fixture-10)
	det(holder-14, the-12)
	amod(holder-14, cylindrical-13)
	prep_into(insert-9, holder-14)
	Take-VB the-DT rotor-NN shaft-NN from-IN the-DT fixture-NN and-CC insert-VB fixture-NN into-IN the-DT cylindrical-JJ holder-NN
		holder_synset_3487214	gloss - "a holding device"
			example - "a towel holder"
			subClassOf - holding_device_synset_3487525
			type – Class
				• main action: insert
				• main object: shaft
				• primary object: fixture
				• secondary object: holder
					ADT_blueprint_put_shaft.xml

		take_synset_2	gloss - "take something or somebody with oneself somewhere" example - string subClassOf - transport_synset_1437285 type – Class		
		shaft_synset_4134	subClassOf - rod_synset_4 type - Class gloss - "a revolving rod that transmits power or motion" gloss - "the axis around which the major rotor of a helicopter turns" subClassOf - b subClassOf - axis_synset_2738643		
		fixture_synset_3319261	subClassOf - artifact_synset_2 type – Class		
		insert_synset_184835	gloss - "introduce" example - "Insert your ticket here" subClassOf - put_synset_1481373 type – Class		

Turn the rotor shaft so that the magnet hole is in front of the robot.	det(shaft-4, the-2)
	dobj(Turn-1, shaft-4)
	advmod(is-10, so-5)
	mark(is-10, that-6)
	det(hole-9, the-7)
	nsubj(is-10, hole-9)
	advcl(Turn-1, is-10)
	det(robot-15, the-14)
	prep_in_front_of(is-10, robot-15)
	Turn-VB the-DT rotor-NN shaft-NN so-RB that-IN the-DT magnet-NN hole-NN is-VBZ in-IN front-NN of-IN the-DT robot-NN
		turn_synset_2	subClassOf - ACTION
			type – Class
				main action: turn
				main object: shaft
				primary object:
				secondary object:
					ADT_blueprint_turn_shaft.xml

		change_state_synset_143724	subClassOf - ACTION type – Class		
		shaft_synset_4	subClassOf - rod_synset_4 type - Class gloss - "a revolving rod that transmits power or motion" gloss - "the axis around which the major rotor of a helicopter turns" subClassOf - axis_synset_2738643		

Pick a magnet from the magnet dispenser and insert it	det(magnet-3, a-2)
	rcmod(hole-14, is-16)
	advmod(is-16, directly-17)
	det(robot-22, the-21)
		pick_synset_668416	gloss - "select carefully from a group"
			subClassOf - choose_synset_666555
			type – Class
				main action: insert
				main object: magnet
				primary object: dispenser
				secondary
					ADT_blueprint_insert_magnet.xml

		magnet_synset_3664443	gloss - "a device that attracts iron and produces a magnetic field"		

into the magnet hole that is directly in front of the robot.	prep_in_front_of(is-16, robot-22)
	dobj(Pick-1, magnet-3)
	det(dispenser-7, the-5)
	prep_from(Pick-1, dispenser-7)
	conj_and(Pick-1, insert-9)
	dobj(insert-9, dispenser-10)
	det(hole-14, the-12)
	prep_into(insert-9, hole-14)
	nsubj(is-16, hole-14)
	Pick-VB a-DT magnet-NN from-IN the-DT magnet-NN dispenser-NN and-CC insert-VB dispenser-NN into-IN the-DT magnet-NN hole-NN that-WDT is-VBZ directly-RB in-IN front-NN of-IN the-DT robot-NN
			subClassOf - device_synset_315
			type – Class
				object: hole	

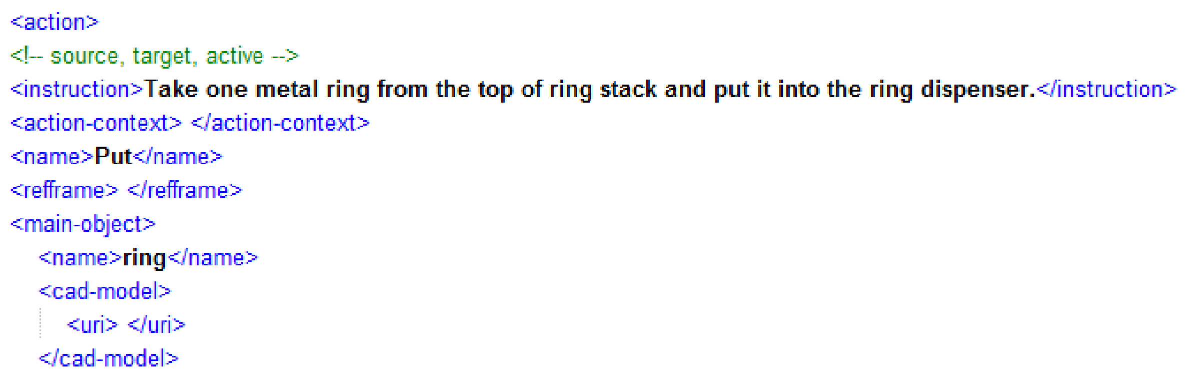
		dispenser_synset_31775	gloss - "a container so designed that the contents can be used in prescribed amounts" subClassOf - container_synset_3 type – Class		
		insert_synset_184835	gloss - "introduce" example - "Insert your ticket here" subClassOf - put_synset_1481373 type – Class		

Repeat steps 4 and 5 for 7 times.	nsubj(steps-2, Repeat-1)
	dobj(steps-2, 4-3)
	conj_and(4-3, 5-5)
	number(times-8, 7-7)
	prep_for(steps-2, times-8)
	Repeat-NN steps-VBZ 4-CD and-CC 5-CD for-IN 7-CD times-NNS
		No data	No data	main action: steps
				main object:
				primary object: steps
				secondary object:	

Pick a rotor cap from the fixture on the robot platform.	det(rotor cap-3, a-2)
	dobj(Pick-1, rotor cap-3)
	det(fixture-6, the-5)
	prep_from(Pick-1, fixture-6)
	det(robot platform-9, the-8)
	prep_on(fixture-6, robot platform-9)
	Pick-VB a-DT cap-NN from-IN the-DT fixture-NN on-IN the-DT platform-NN
		pick_synset_668416	gloss - "select carefully from a group"
			subClassOf - choose_synset_666555
			type – Class
				main action: pick
				main object: cap
				primary object: fixture
				secondary object:
					ADT_blueprint_pick_rotor_cap.xml

		cap_synset_2926697	gloss - "something serving as a cover or protection" subClassOf - protective_covering_synset_3969138 type – Class		
		platform_synset_3917	gloss - "a raised horizontal surface" example - "the speaker mounted the platform" subClassOf - horizontal_surface_synset_34977 type – Class		

		fixture_synset_331926 1	subClassOf - artifact_synset_2 type – Class		



```
<action>
<!-- source, target, active -->
<instruction>Take one metal ring from the top of ring stack and put it into the ring dispenser.</instruction>
<action-context> </action-context>
<name>Put</name>
<refframe> </refframe>
<main-object>
   <name>ring</name>
   <cad-model>
      <uri> </uri>
   </cad-model>
```

Fig. 21. Excerpt of a prefilled ADT file for instruction command: ***Take one metal ring from the top of ring stack and put it into the ring dispenser***

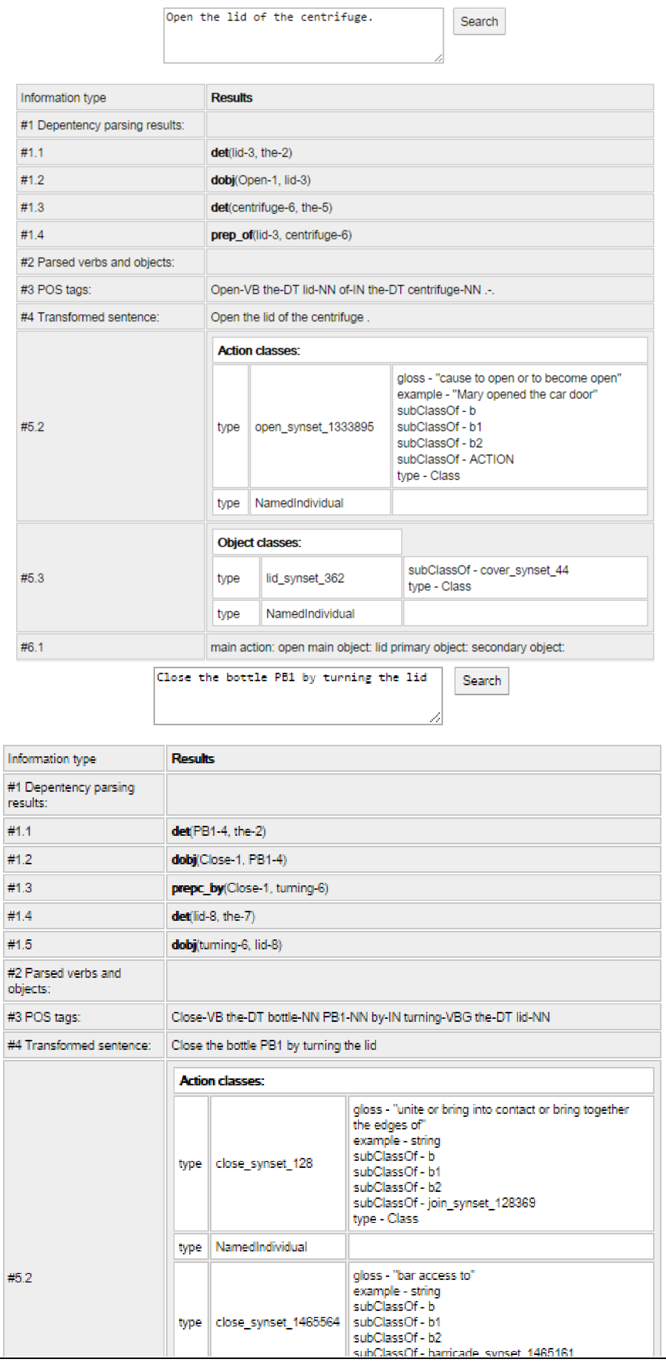
Table 2 presents the analysis of the above presented instruction completion experiment, pointing to inaccuracies, explaining them and planning actions for improvement.

Table 2. Error analysis for textual instruction completion experiment (IASSES scenario)

Instruction	Result of completion	Explanation	Possible actions
Turn the rotor shaft so that the magnet hole is in front of the robot.	main action: turn main object: shaft primary object: - secondary object: -	First part of the parser results is correct – main action and main object were recognized. However, the information about detailed action post conditions (i.e. "hole is in front of the robot") can not be passed to the ADT (a slot for the detailed post-condition description is not included into the ADT).	Adding the more detailed post-condition information to ADT would allow defining action more accurately.
Pick a magnet from the magnet dispenser and insert it into the magnet hole that is directly in front of the robot.	main action: insert main object: magnet primary object: dispenser secondary object: hole	First part of the results of the parser is correct – main action and main, primary and secondary objects were recognized. However, the information about magnet hole position can not be passed to the ADT.	Adding the more detailed post-condition information to ADT would allow defining action more accurately.
Repeat steps 4 and 5 for 7 times.	main action: steps main object: primary object: steps secondary object:	All parsed information is incorrect. This happened, because of wrong parser annotation: *nsubj(steps-2, Repeat-1), dobj(steps-2, 4-3), conj_and(4-3, 5-5), number(times-8, 7-7), prep_for(steps-2, times-8), Repeat-NN steps-VBZ 4-CD and-CC 5-CD for-IN 7-CD times-NNS.* Repeat action is recognized as an object and steps – as an action.	Though here a parsing error has occurred, instruction "repeat" shall be in general treated as a special case. This is a very specific instruction, telling what to do with the instruction steps, but not with actual objects.

5.2. CHEMLAB scenario

The instruction textual completion experiment with the ACAT Instruction Compiler was run for the set of instructions presented in Appendix A.3. Approximately half of the instructions were processed, where repeating instructions were omitted (e.g. in the instruction sheet opening and closing centrifuge, opening and closing bottles, pouring liquid into cartridges was happening multiple times). Fig. 22 presents Compiler screenshots, with the examples of dependency parsing, ontology querying and reasoning results.



```
Open the lid of the centrifuge.
```

Information type	Results		
#1 Depentency parsing results:			
#1.1	**det**(lid-3, the-2)		
#1.2	**dobj**(Open-1, lid-3)		
#1.3	**det**(centrifuge-6, the-5)		
#1.4	**prep_of**(lid-3, centrifuge-6)		
#2 Parsed verbs and objects:			
#3 POS tags:	Open-VB the-DT lid-NN of-IN the-DT centrifuge-NN .-.		
#4 Transformed sentence:	Open the lid of the centrifuge .		
#5.2	**Action classes:**		
	type	open_synset_1333895	gloss - "cause to open or to become open" example - "Mary opened the car door" subClassOf - b subClassOf - b1 subClassOf - b2 subClassOf - ACTION type - Class
	type	NamedIndividual	
#5.3	**Object classes:**		
	type	lid_synset_362	subClassOf - cover_synset_44 type - Class
	type	NamedIndividual	
#6.1	main action: open main object: lid primary object: secondary object:		

```
Close the bottle PB1 by turning the lid
```

Information type	Results		
#1 Depentency parsing results:			
#1.1	**det**(PB1-4, the-2)		
#1.2	**dobj**(Close-1, PB1-4)		
#1.3	**prepc_by**(Close-1, turning-6)		
#1.4	**det**(lid-8, the-7)		
#1.5	**dobj**(turning-6, lid-8)		
#2 Parsed verbs and objects:			
#3 POS tags:	Close-VB the-DT bottle-NN PB1-NN by-IN turning-VBG the-DT lid-NN		
#4 Transformed sentence:	Close the bottle PB1 by turning the lid		
#5.2	**Action classes:**		
	type	close_synset_128	gloss - "unite or bring into contact or bring together the edges of" example - string subClassOf - b subClassOf - b1 subClassOf - b2 subClassOf - join_synset_128369 type - Class
	type	NamedIndividual	
	type	close_synset_1465564	gloss - "bar access to" example - string subClassOf - b subClassOf - b1 subClassOf - b2 subClassOf - barricade_synset_1465161

Fig. 22. Compiler screenshot examples (CHEMLAB scenario)

The results of the instruction completion experiment for the CHEMLAB scenario are presented in Table 3. The table includes sentence, which was parsed, dependency parsing results and ontology querying results. It also includes analysis results on recognized actions and objects, which are included in the examples of preliminary ADTs.

The structure of the Table 3 is analogous to the Table 1 presented for IASSES scenario (see section 5.1 for detailed Table 1 column description).

Table 3. Results of the textual instruction completion experiment (CHEMLAB scenario)

Command	**Dependency parsing results**	**Ontology querying results**		Recognized actions and objects	**ADT blueprint**
Open centrifuge.	nsubj(centrifuge-2, Open-1) Open-NNP centrifuge-VBP	open_synset_1333895	gloss - "cause to open or to become open" example - "Mary opened the car door" subClassOf - ACTION type - Class	main action: open main object: centrifuge primary object: - secondary object: -	ADT_blueprint_open_centrifuge.xml
		centrifuge_synset_2966875	gloss - "an apparatus that uses centrifugal force to separate particles from a suspension" subClassOf - apparatus_synset_27 type - Class		
		centrifuge_synset_2	gloss - "an apparatus that uses centrifugal force to separate particles from a suspension" subClassOf - apparatus_synset_27 type - Class gloss - "rotate at very high speed in order to separate the liquids from the solids" subClassOf - spin_synset_2 example - string		

Pour suspension of e-coli from a flask into a plastic bottle PB1	dobj(pour-3, suspension-4)
	prep_of(suspension-4, e-coli-6)
	det(flask-9, a-8)
	prep_from(e-coli-6, flask-9)
	det(PB1-14, a-11)
	amod(PB1-14, plastic-12)
	prep_into(pour-3, PB1-14)
	Pour-VB suspension-NN of-IN e-coli-NN from-IN a-DT flask-NN into-IN a-DT plastic-JJ bottle-NN PB1-NN
		suspension_synset_14397489	gloss - "a mixture in which fine particles are suspended in a fluid where they are supported by buoyancy"
			subClassOf - mixture_synset_14392656
			type - Class
				main action: pour
				main object: suspension
				primary object: flask
				secondary object: bottle pb1
					ADT_blueprint_pour_suspension.xml

Pick a bottle PB1	det(PB1-4, a-2) dobj(Pick-1, PB1-	pick_synset_668416	gloss - "select carefully from a group"	main action: pick	

with e-coli and put it into centrifuge	4) prep_with(Pick-1, e-coli-6) conj_and(Pick-1, put-8) dobj(put-8, e-coli-9) prep_into(put-8, centrifuge-11) Pick-VB a-DT bottle-NN PB1-NN with-IN e-coli-NN and-CC put-VB e-coli-NN into-IN centrifuge-NN		example - string subClassOf - choose_synset_666555 type - Class	main object: bottle PB1 primary object: - secondary object: centrifuge	
		pick_synset_17541	gloss - "remove in small bits" example - "pick meat from a bone" subClassOf - remove_synset_17 type - Class		
		pick_synset_1369876	gloss - "look for and gather" example - string subClassOf - gather_synset_1368449 type - Class		
		choose_synset_666555	subClassOf - ACTION type - Class		
		pick_up_synset_1957772	gloss - "take and lift upward" subClassOf - raise_synset_1955745 type - Class		
		pick_up_synset_11961	gloss - "take up by hand" example - "He picked up the book and started to read" subClassOf - touch_synset_1194934 type - Class		
		bottle_synset_2848798	gloss – "a glass or plastic vessel used for storing drinks or other liquids typically cylindrical without handles and with a narrow neck that can be plugged or capped" subClassOf – vessel_synset_4476617		
		put_synset_1481373	gloss - "put into a certain place or abstract location" example - string subClassOf - move_synset_1832958 type - Class		
		put_synset_148	example - "That song put me in awful good humor" subClassOf - change_synset_123976 type - Class gloss - "put into a certain place or abstract location" example - string subClassOf - move_synset_1832958		
		pose_synset_1481373	gloss - "put into a certain place or abstract location" example - string subClassOf - move_synset_1832958		

			type – Class		
		place_synset_1481373	gloss - "put into a certain place or abstract location" example - string subClassOf - move_synset_1832958 type - Class		
		discard_synset_22	gloss - "Throw or cast away" example - string subClassOf - get_rid_of_synset_22 type - Class		

Close the lid of centrifuge.	det(lid-3, the-2)
	dobj(Close-1, lid-3)
	prep_of(lid-3, centrifuge-5)
	Close-VB the-DT lid-NN of-IN centrifuge-NN
		close_synset_128	gloss - "unite or bring into contact or bring together the edges of"
			example - string
			subClassOf - join_synset_128369
			type - Class
				main action: close
				main object: lid
				primary object: -
				secondary object: -
					ADT_blueprint_close_lid.xml

		close_synset_1465564	gloss - "bar access to" example - string arricade_synset_1465161 type - Class		
		close_synset_1333199	example - string subClassOf - ACTION type - Class		
		lid_synset_362	subClassOf - cover_synset_44 type - Class		

Start centrifuge by pressing a button But1.	dep(centrifuge-2, Start-1)
	prepc_by(centrifuge-2, pressing-4)
	det(But1-7, a-5)
	dobj(pressing-4, But1-7)
	Start-NNP centrifuge-JJ by-IN pressing-VBG a-DT button-NN But1-NNS
		centrifuge_synset_2966875	gloss - "an apparatus that uses centrifugal force to separate particles from a suspension"
			subClassOf - apparatus_synset_27
			type - Class
				main action: pressing
				main object: but
				primary object: centrifuge
				secondary object: -
					ADT_blueprint_press_button.x

		centrifuge_synset_2	gloss - "an apparatus that uses centrifugal force to separate particles from a suspension" subClassOf - apparatus_synset_27 type - Class gloss - "rotate at very high speed in order to separate the liquids from the solids" subClassOf - spin_synset_2 example - string		

Wait for 10 minutes.	num(minutes-4, 10-3)
	prep_for(Wait-1, minutes-4)
	Wait-VB for-IN 10-CD minutes-NNS
				main action: wait
				main object: -
				primary object: -
				secondary object:-
					ADT_blueprint_wait_.xml

Stop centrifuge by pressing button But1.	dep(centrifuge-2, Stop-1)
	prepc_by(centrifuge-2, pressing-4)
	dobj(pressing-4,
		barricade_synset_1465161	subClassOf - ACTION
			type - Class
				main action: press
				main object: button but1
				primary
					ADT_blueprint_press_button_1.xml

		centrifuge_synset_2966875	gloss - "an apparatus that uses centrifugal force to separate		

	But1-6) Stop-VB centrifuge-JJ by-IN pressing-VBG button-NN But1-NNS		particles from a suspension" subClassOf - apparatus_synset_27 type - Class	object: centrifuge secondary object: -	
		centrifuge_synset_2	loss - "an apparatus that uses centrifugal force to separate particles from a suspension" subClassOf - apparatus_synset_27 type - Class gloss - "rotate at very high speed in order to separate the liquids from the solids" subClassOf - spin_synset_2 example - string		

Open the lid of the centrifuge.	det(lid-3, the-2)
	dobj(Open-1, lid-3)
	det(centrifuge-6, the-5)
	prep_of(lid-3, centrifuge-6)
	Open-VB the-DT lid-NN of-IN the-DT centrifuge-NN .-.
		open_synset_1333895	gloss - "cause to open or to become open"
			example - "Mary opened the car door"
			subClassOf - ACTION
			type - Class
				main action: open main object: lid
				primary object: -
				secondary object:-
					ADT_blueprint_open_lid.xml

		lid_synset_362	subClassOf - cover_synset_44 type - Class		

Take out the bottle PB1 of centrifuge and put it on fixature	prt(Take-1, out-2)
	det(PB1-5, the-3)
	dobj(Take-1, PB1-5)
	prep_of(PB1-5, centrifuge-7)
	conj_and(Take-1, put-9)
	dobj(put-9, centrifuge-10)
	prep_on(put-9, fixature-12)
	
	Take-VB out-RP the-DT bottle-NN PB1-NN of-IN centrifuge-NN and-CC put-VB centrifuge-NN on-IN fixature-NN
		take_synset_17	example - string
			subClassOf - ACTION
			type - Class
				main action: put
				main object: bottle pb1
				primary object: centrifuge
				secondary object: fixature
					ADT_blueprint_put_bottle.xml

		take_synset_12	example - string subClassOf - ACTION type - Class		
		remove_synset_17	subClassOf - ACTION type - Class example - string		
		choose_synset_666555	subClassOf - ACTION type - Class		
		bottle_synset_2848798	subClassOf - vessel_synset_4476617 type - Class		
		centrifuge_synset_2966875	gloss - "an apparatus that uses centrifugal force to separate particles from a suspension" subClassOf - apparatus_synset_27 type - Class		
		pull_synset_1338932	example - string subClassOf - remove_synset_17 type - Class		
		hold_synset_12	subClassOf - ACTION type - Class		
		put_synset_1481373	gloss - "put into a certain place or abstract location" example - string subClassOf - move_synset_1832958		

			type - Class		
		put_synset_148	example - "That song put me in awful good humor" subClassOf - change_synset_123976 type - Class gloss - "put into a certain place or abstract location" example - string subClassOf - move_synset_1832958		
		pose_synset_1481373	gloss - "put into a certain place or abstract location" example - string subClassOf - move_synset_1832958 type - Class		
		place_synset_1481373	gloss - "put into a certain place or abstract location" example - string subClassOf - move_synset_1832958 type - Class		
		discard_synset_22	gloss - "Throw or cast away" example - string subClassOf - get_rid_of_synset_22 type - Class		
Open the plastic bottle PB1	det(PB1-5, the-2) amod(PB1-5, plastic-3) dobj(Open-1, PB1-5) Open-VB the-DT plastic-JJ bottle-NN PB1-NN	open_synset_1333895	gloss - "cause to open or to become open" example - "Mary opened the car door" subClassOf - ACTION type - Class	main action: open main object: bottle primary object: - secondary object:-	ADT_blueprint_open_bottle.xml
		bottle_synset_2848798	subClassOf - vessel_synset_4476617 type - Class		
Pour the liquid out of the bottle PB1 into a flask	det(liquid-3, the-2) dobj(Pour-1, liquid-3) det(PB1-8, the-6) prep_out_of(liquid-3, PB1-8) det(flask-11, a-10) prep_into(Pour-1, flask-11) Pour-VB the-DT liquid-NN out-RB of-IN the-DT bottle-NN PB1-NN into-IN a-DT flask-NN	liquid_synset_14743667	subClassOf - MATERIAL type - Class	main action: pour main object: liquid primary object: bottle PB1 secondary object: flask	ADT_blueprint_pour_liquid.xml
		liquid_synset_14743353	gloss - "a substance in the fluid state of matter having no fixed shape but a fixed volume" subClassOf - fluid_synset_14742729 type - Class		
		bottle_synset_2848798	subClassOf - vessel_synset_4476617 type - Class		
Pipette 15 ml of buffer R3 into bottle PB1	num(ml-3, 15-2) dobj(Pipette-1, ml-3) prep_of(ml-3, R3-6) prep_of(ml-3, buffer-5) prep_into(Pipette-1, PB1-9) prep_into(Pipette-1, bottle-7)	bottle_synset_2848798	subClassOf - vessel_synset_4476617 type - Class	main action: pipette main object: buffer primary object: - secondary object: bottle	ADT_blueprint_pipette_buffer.xml

	Pipette-VB 15-CD ml-NN of-IN buffer-NN into-IN bottle-NN				
Close the bottle PB1 again	det(PB1-4, the-2) dobj(Close-1, bottle-3) advmod(Close-1, again-4) Close-VB the-DT bottle-NN again-RB	close_synset_128	gloss - "unite or bring into contact or bring together the edges of" example - string subClassOf - join_synset_128369 type - Class	main action: close main object: bottle primary object: secondary object:	ADT_blueprint_close_bottle.xml
		close_synset_1465564	gloss - "bar access to" example - string arricade_synset_1465161 type - Class		
		bottle_synset_2848798	subClassOf - vessel_synset_4476617 type - Class		

Shake the bottle PB1 by holding against the shaking device	det(bottle-3, the-2)
	dobj(Shake-1, bottle-3)
	prepc_by(Shake-1, holding-5)
	det(device-9, the-7)
	amod(device-9, shaking-8)
	prep_against(holding-5, device-9)
	Shake-VB the-DT bottle-NN by-IN holding-VBG against-IN the-DT shaking-VBG device-NN
		shake_synset_1872	gloss - "move or cause to move back and forth"
			example - string
			subClassOf - move_synset_1814387
			type - Class
				main action: shake
				main object: bottle
				primary object:
				secondary object:
					ADT_blueprint_shake_bottle.xml

		bottle_synset_2848798	subClassOf - vessel_synset_4476617 type - Class		
Open the bottle PB1	det(bottle-3, the-2) dobj(Open-1, bottle-3) Open-VB the-DT bottle-NN	open_synset_1333895	gloss - "cause to open or to become open" example - "Mary opened the car door" subClassOf - ACTION type - Class	main action: open main object: bottle primary object: secondary object:	ADT_blueprint_open_bottle_1.xml
		bottle_synset_2848798	subClassOf - vessel_synset_4476617 type - Class		

Open the bottle with lysis buffer PB2	det(bottle-3, the-2)
	dobj(Open-1, bottle-3)
	prep_with(Open-1, lysis buffer-5)
	Open-VB the-DT bottle-NN with-IN buffer-NN
		open_synset_1333895	gloss - "cause to open or to become open"
			example - "Mary opened the car door"
			subClassOf - ACTION
			type - Class
				main action: open
				main object: bottle
				primary object: lysis buffer
				secondary object:
					ADT_blueprint_open_bottle_2.xml

		bottle_synset_2848798	subClassOf - vessel_synset_4476617 type - Class		

Pipette 75 ml of lysis buffer into the bottle PB1	num(ml-3, 75-2)
	dobj(Pipette-1, ml-3)
	prep_of(ml-3, lysis buffer-5)
	det(PB1-9, the-7)
	prep_into(Pipette
		buffer_synset_14591449	gloss - "an ionic compound that resists changes in its pH"
			subClassOf - compound_synset_14622879
			type - Class
				main action: pipette
				main object: buffer
				primary object:
				secondary
					ADT_blueprint_pipette_buffer_1.xml

	-1, PB1-9) Pipette-VB 75-CD ml-NN of-IN buffer-NN into-IN the-DT bottle-NN PB1-NN	bottle_synset_2848798	subClassOf - vessel_synset_4476617 type - Class	object: bottle	
Close the bottle PB1 by turning the lid	det(bottle-3, the-2) dobj(Close-1, bottle-3) prepc_by(Close-1, turning-5) det(lid-7, the-6) dobj(turning-5, lid-7) Close-VB the-DT bottle-NN by-IN turning-VBG the-DT lid-NN	close_synset_128	gloss - "unite or bring into contact or bring together the edges of" example - string subClassOf - join_synset_128369 type - Class	main action: turn main object: lid primary object: secondary object:	ADT_blueprint_turn_lid.xml
		close_synset_1465564	gloss - "bar access to" example - string arricade_synset_1465161 type - Class		
		bottle_synset_2848798	subClassOf - vessel_synset_4476617 type - Class		
		lid_synset_362	subClassOf - cover_synset_44 type - Class		
Mix the contents of the bottle PB1 gently by inverting 4-6 times	det(contents-3, the-2) dobj(Mix-1, contents-3) det(bottle-6, the-5) prep_of(contents-3, bottle-6) advmod(Mix-1, gently-7) prepc_by(Mix-1, inverting-9) num(times-11, 4-6-10) dobj(inverting-9, times-11) Mix-VB the-DT contents-NNS of-IN the-DT bottle-NN gently-RB by-IN inverting-VBG 4-6-CD times-NNS	mix_synset_181426	type - Class gloss - "add as an additional element or part" example - short subClassOf - add_synset_179714	main action: invert main object: contents primary object: secondary object:	ADT_blueprint_invert_contents.xml
		bottle_synset_2848798	subClassOf - vessel_synset_4476617 type - Class		
Pour 200 ml of equilibration buffer into the DNA binding cartridge	num(ml-3, 200-2) dobj(Pour-1, ml-3) prep_of(ml-3, equilibration buffer-5) det(DNA binding cartridge-8, the-7) prep_into(Pour-1, DNA binding cartridge-8) Pour-VB 200-CD ml-NN of-IN buffer-NN into-IN the-DT cartridge-NN	cartridge_synset_2943728	gloss - "a module designed to be inserted into a larger piece of equipment" example - "he loaded a cartridge of fresh tape into the tape deck" subClassOf - module_synset_3737 type - Class	main action: pour main object: buffer primary object: secondary object: cartridge	ADT_blueprint_pour_buffer.xml
		buffer_synset_14591449	gloss - "an ionic compound that resists changes in its pH" subClassOf - compound_synset_14622879 type - Class		

Table 4 presents the analysis of instruction completion experiment, pointing to inaccuracies, explaining them and planning actions for improvement.

Table 4. Error analysis for textual instruction completion experiment (CHEMLAB scenario)

Instruction	Result of completion	Explanation	Possible actions
Open centrifuge	main action: open main object: centrifuge primary object: - secondary object: -	Parser annotation was incorrect - nsubj(centrifuge-2, Open-1) Open-NNP centrifuge-VBP. However, we used additional "always true" statement dictionary to eliminate as much as possible such errors. Finally, result of ADT completion was correct.	Extending "always true" statement dictionary will partly eliminate wrong annotation problems.
Pick a bottle PB1 with e-coli and put it into centrifuge	main action: pick main object: bottle PB1 primary object: - secondary object: centrifuge	Pick action was defined as the main action, when the "put" should have been indicated as the main action.	This problem can be fixed by adjusting property definition in the ontology: main action or supportive action.
Mix the contents of the bottle PB1 gently by inverting 4-6 times	main action: invert main object: contents primary object: secondary object:	The main object of this action in robotic context shall be "bottle", but the "contents" was indicated as the main object instead. Here the object is complex, but was recognized as a simple object. In this case, the instruction on its own does not contain sufficient information.	The problem with complex objects can be fixed by extracting the main object with POS pattern *NN IN-of NN*. In this case, the ontology must be queried with both components of complex object. Also, adding example information from a set of manually annotated ADTs would help to solve the problem.

6. Discussion

In this deliverable textual completion of instructions from the CHEMLAB and IASSES instructions sheets has been analyzed. Here we considered "instruction textual completion" as extraction of symbolic information (from instruction sheets with the support of the ACAT ontology) that is required for the path towards instruction execution. Specifically, we were considering how to fill in action name, main object name, primary object name and secondary object name into the Action Data Tables (ADTs, execution counter-part of the ACAT system).

We were performing textual completion of the instruction sheets transcribed at the level where most of the action words (verbs) were given at the level where a direct association between the verb and robotic action is possible. E.g. instructions addressing actions like pick&place (or "place" in short), open, close, pour have been analyzed. At this level we can obtain a direct association between the verb in the instruction and the actual robotic action in ACAT project described with the help of ADTs. Thus we were using not the high level instructions presented in D5.1 (populated with action words like, harvest, lyse or neutralize), but transcription of the corresponding scenarios in the previously mentioned lower level terms. We plan to build the ACAT system in such a way, that first the high level instructions are transcribed in low level terms and ADTs are associated only to the lower level terms (where the algorithms presented in this deliverables are operational). The issue of systematic handling of the high to low level transcriptions is planned for the Year 3 of the project.

The presented instruction completion algorithms show the following performace: from seven analyzed IASSES instructions, six were interpreted correctly, given the task to fill the existing ADT structure. Additional two instructions had information which was not possible to assign to the ADT slots due to the chosen ADT structure (not rich enough). However, we have to start investigation of the text-to-robotic-code compilation processes based on limited data structures, otherwise the complexity of the task is not manageable. The one mentioned “true” miss-interpretation was made for the instruction “repeat steps 4 and 5 for 7 times”. Technically, a parsing error has occurred. However, even if the parsing were correct, the instruction “repeat” obviously shall be analyzed as a special case. This is a special type of instruction used for the flow control of other instructions and can’t be treated the same way as regular instructions directly corresponding to robotic actions.

From the 20 investigated instructions from the CHEMLAB scenario instruction sheet two were interpreted in a wrong way. One was interpreted wrongly because of a complex object of a sentence, where the connection between the “main object” of the robotic action and the “object” of the sentence in the language syntax was not straightforward. Here both, re-interpretation of the sentence as well as correction of the object nomenclature in ADTs (re-interpretation from robotic action side), are possible. In addition, we have a plan to enrich purely syntax-based mapping of the object roles in the sentence to object roles in ADTs by reasoning based on hand annotated examples, employing the Markov Logic Networks based methods described in section 4.2.

The other error (wrong main action assigned) occurred due to inaccuracies in action parameter annotation. This is a technical error. Adjustment of action parameters in the ontology need be made to avoid such errors in future.

The third case, mentioned among the CHEMLAB scenario instruction interpretation inaccuracies, was already corrected by including extra rules in addition to the rules used by a standard Stanford parser. This case needs to be discussed a bit deeper, as parsing errors occurred in both scenarios. In our work we are using the Stanford parser6 – i.e. statistical dependency parser, which is reported as one of the fastest (1000 sentences per second) and the most accurate (92.2% unlabeled attachment score) currently known approaches for parsing English sentences (Chen & Manning, 2014). Due to the proposed novel technique (based on learning neural network classifier for use in greedy, transition-based dependency parser) it outperforms baselines of arc-eager and arc-standard parsers and achieves 2% improvement on both labeled and unlabeled attachment scores, while running about 20 times faster. Despite all these advantages and superiority over the other dependency parsing techniques, the Stanford parser still has shortcomings that emerge mostly because it is not adjusted to any specific domain. The Stanford parser is trained on the

[6] http://nlp.stanford.edu/software/lex-parser.shtml

English Penn Treebank[7] (Marcus et al., 1993), composed of ~40,000 sentences, taken from the Wall Street Journal. This newspaper domain is indeed very different from the robotics and chemistry domains that we are dealing in ACAT project. However, major accuracy problems (in both, indicating part-of-speech tags and dependency labels) are caused by the sentence structure itself: i.e. we need to process sentences written in imperative mood, but the Stanford parser is trained on the sentences in indicative.

The solution of this problem would be to obtain an English “Penn Treebank” data set, complemented by samples of labeled instruction sentences in imperative mood and do the training of the parser from scratch. Retraining of the Stanford parser[8] should boost the accuracy. However, how much the accuracy would increase it is hard to say in advance. Some part-of-speech tagging and dependency parsing problems would be remaining. We have chosen an alternative solution: to use Stanford parser in the primary step, but to implement the necessary automatic corrections afterwards. This is a faster and cheaper solution allowing us to attribute more time in the project to the problems, which are more tightly connected to robotics, as compared to purely linguistic work: linguistic instruction annotation and parser re-training.

7. Conclusions and future work

Here we have presented our textual instruction sheet completion framework, which reaches into the compilation process of the instruction sheets. We take a human readable instruction, add missing information and provide a preliminary Action Data Table (ADT) with filled-in symbolic level information. Thus, instruction compilation is currently only discussed at the symbolic information level and additional parts that adds signal (control level) information will be presented in the following deliverable D3.2.

The designed instruction completion schema and the corresponding ACAT instruction compilation processes are able to parse an instruction sheet instruction-by-instruction and define the corresponding sequence of robotic actions as well as roles of objects in those actions. The approach of combining parsed instruction sheet information with ontology query results seems to give adequate information for instruction completion. According to evaluation on IASSES and CHEMLAB scenario instructions, the errors rate is at the level of 10-20% and this is appropriate given the early research stage in the topic of human-readable instruction compilation for robotic applications. One cannot expect to perform the conversion of human readable instructions to an Action Data Table data in a totally error-free way, as this requires very far reaching reasoning processes, which current artificial reasoning systems cannot yet address. Thus we expect to also have some residual errors, which will be corrected in the phase of human validation in the ACAT system.

[7] http://www.cis.upenn.edu/~treebank/home.html
[8] See “6. Can I train the parser?” in http://nlp.stanford.edu/software/parser-faq.shtml#d

Knowledge of actions and objects participating in those actions can be further improved by more advanced querying the action ontology, including more details on action execution parameters (properties, size, quantity, location, etc.). There are possibilities for filling in missing action- or object information with the most probable action or object instances. However, these are different problems as compared to the one of instruction-to-ADT-symbolic-part conversion we were addressing in this deliverable.

In conclusion, the current instruction textual completion status allows us to proceed to the next step of instruction compilation, by filling the here initialized ADTs with sub-symbolic information, which would further lead to the execution on a robot.

8. References

Apache UIMA Development Community, UIMA Tutorial and Developers' Guides <http://uima.apache.org/downloads/releaseDocs/2.3.0-incubating/docs/pdf/tutorials_and_users_guides.pdf>, 2009.

Chen, D and Manning, C.D.: A Fast and Accurate Dependency Parser using Neural Networks. Proceedings of Empirical Methods in Natural Language Processing (EMNLP), 2014.

Ferrucci, David and Lally, Adam: UIMA: an architectural approach to unstructured information processing in the corporate research environment. Natural Language Engineering, Vol.12, 2004, No. 3-4, pages 327-348.

Hearst, Marti A.: Automatic acquisition of hyponyms from large text corpora. Proceedings of the 14th conference on Computational linguistics, 1992, pages 539-545.

Jessop, David M and Adams, Sam E and Willighagen, Egon L and Hawizy, Lezan and Murray-Rust, Peter: OSCAR4: a flexible architecture for chemical text-mining. Journal of Cheminformatics, Vol 3, 2011, No. 1, pages 1-12.

Kotsiantis S. B.: Supervised Machine Learning: A Review of Classification Techniques. Informatica, 2007, 31:249–268.

Kübler, S., McDonald, R. and Nivre, J. *Dependency Parsing*. Morgan and Claypool, 2009.

de Marneffe, Marie-Catherine and Manning, Christopher D. Stanford Dependencies manual, <http://nlp.stanford.edu/software/dependencies_manual.pdf>, 2008.

Markievicz, I., Kapočiūtė-Dzikienė, J., Tamošiūnaitė, M., Vitkutė-Adžgauskienė, D.: Action Classification in Action Ontology Building Using Robot-Specific Texts, Information Technology and Control, Kaunas, 2014 (in review).

Marcus, M.P., Marcinkiewicz, M.A. and Santorini, B.: Building a Large Annotated Corpus of English: The Penn Treebank. Computational Linguistics, 19 (2), 1993, pages 313-330.

Nyga, D. and Beetz, M. Everything robots always wanted to know about housework (but were afraid to ask. Intelligent Robots and Systems (IROS), 2012, pages 243-250.

Nyga, D. and Beetz, M. Composite Likelihood Learning for Markov Logic Networks. Submitted for AAAI 2015.

Nyga, D. and Beetz, M. Incorporating Class Taxonomies in Probabilistic Relational Models. Submitted for AAAI 2014.

Toutanova, K and Klein, D and Manning, C and others: Stanford Core NLP. 2013. http://nlp. stanford. edu/software/corenlp.

Appendices

A.1. Documentation of the ACAT instruction completion (ACAT Instruction Compiler) software

Introduction

The ACAT instruction compiler is designed as a Google Web Toolkit AJAX application. Technologies used: Google Web Toolkit (GWT), Jena Library, Stanford NLP parser. The main tool functions: ACAT instruction sheet parsing, ontology querying, and preparation of preliminary ADT structures.

Conceptual model

The ACAT instruction compiler is based on the MVP (Model – View – Presenter) architecture (Fig. A1). The selected GWT technology allows us to develop and maintain complex AJAX front-end applications in Java.

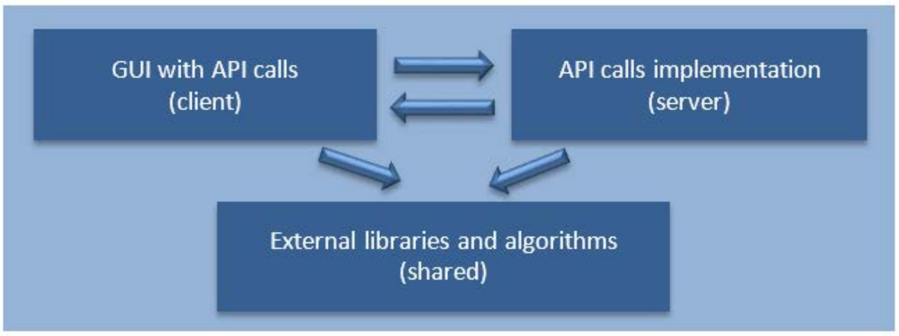


Fig. A1. The conceptual MVP model of the ACAT instruction compiler

All instruction sheets sentences are analyzed with asynchronous remote procedure calls from the *client* to the *server* side. Additional libraries (Jena ontology querying library, Stanford NLP parser, etc.) and algorithms implementation from *shared* package are used in both *client* and *server* sides. System class diagram is presented in two parts, for client and server packages (Fig. A2, Fig. A3).

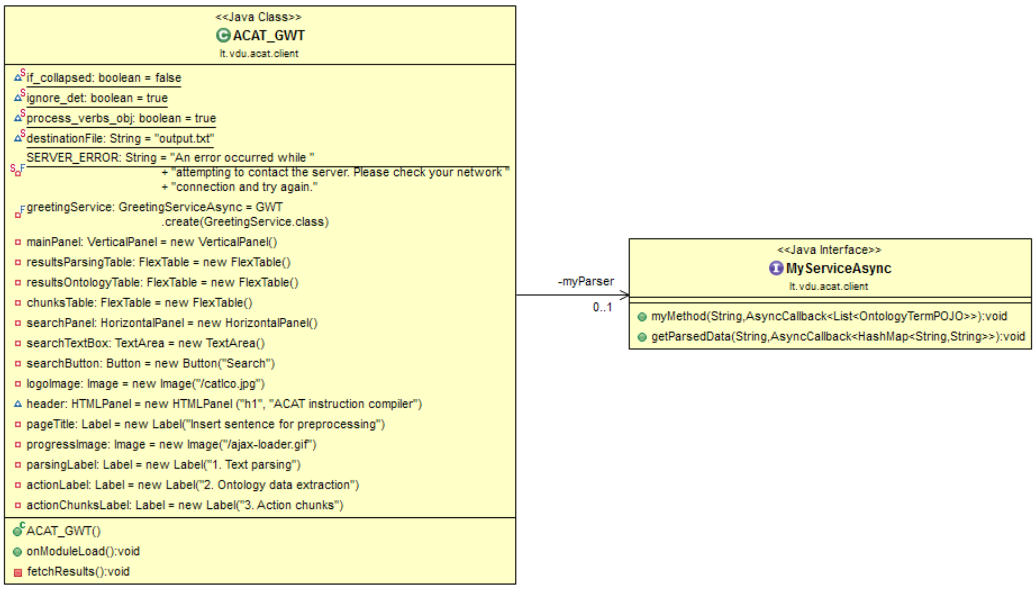

<<Java Class>>
ACAT_GWT
lt.vdu.acat.client
if_collapsed: boolean = false
ignore_det: boolean = true
process_verbs_obj: boolean = true
destinationFile: String = "output.txt"
SERVER_ERROR: String = "An error occurred while " + "attempting to contact the server. Please check your network " + "connection and try again."
greetingService: GreetingServiceAsync = GWT .create(GreetingService.class)
mainPanel: VerticalPanel = new VerticalPanel()
resultsParsingTable: FlexTable = new FlexTable()
resultsOntologyTable: FlexTable = new FlexTable()
chunksTable: FlexTable = new FlexTable()
searchPanel: HorizontalPanel = new HorizontalPanel()
searchTextBox: TextArea = new TextArea()
searchButton: Button = new Button("Search")
logoImage: Image = new Image("/catlco.jpg")
header: HTMLPanel = new HTMLPanel ("h1", "ACAT instruction compiler")
pageTitle: Label = new Label("Insert sentence for preprocessing")
progressImage: Image = new Image("/ajax-loader.gif")
parsingLabel: Label = new Label("1. Text parsing")
actionLabel: Label = new Label("2. Ontology data extraction")
actionChunksLabel: Label = new Label("3. Action chunks")
ACAT_GWT()
onModuleLoad():void
fetchResults():void
-myParser
0..1
<<Java Interface>>
MyServiceAsync
lt.vdu.acat.client
myMethod(String,AsyncCallback<List<OntologyTermPOJO>>):void
getParsedData(String,AsyncCallback<HashMap<String,String>>):void

Fig. A2. Class diagram for ACAT instruction compiler client side

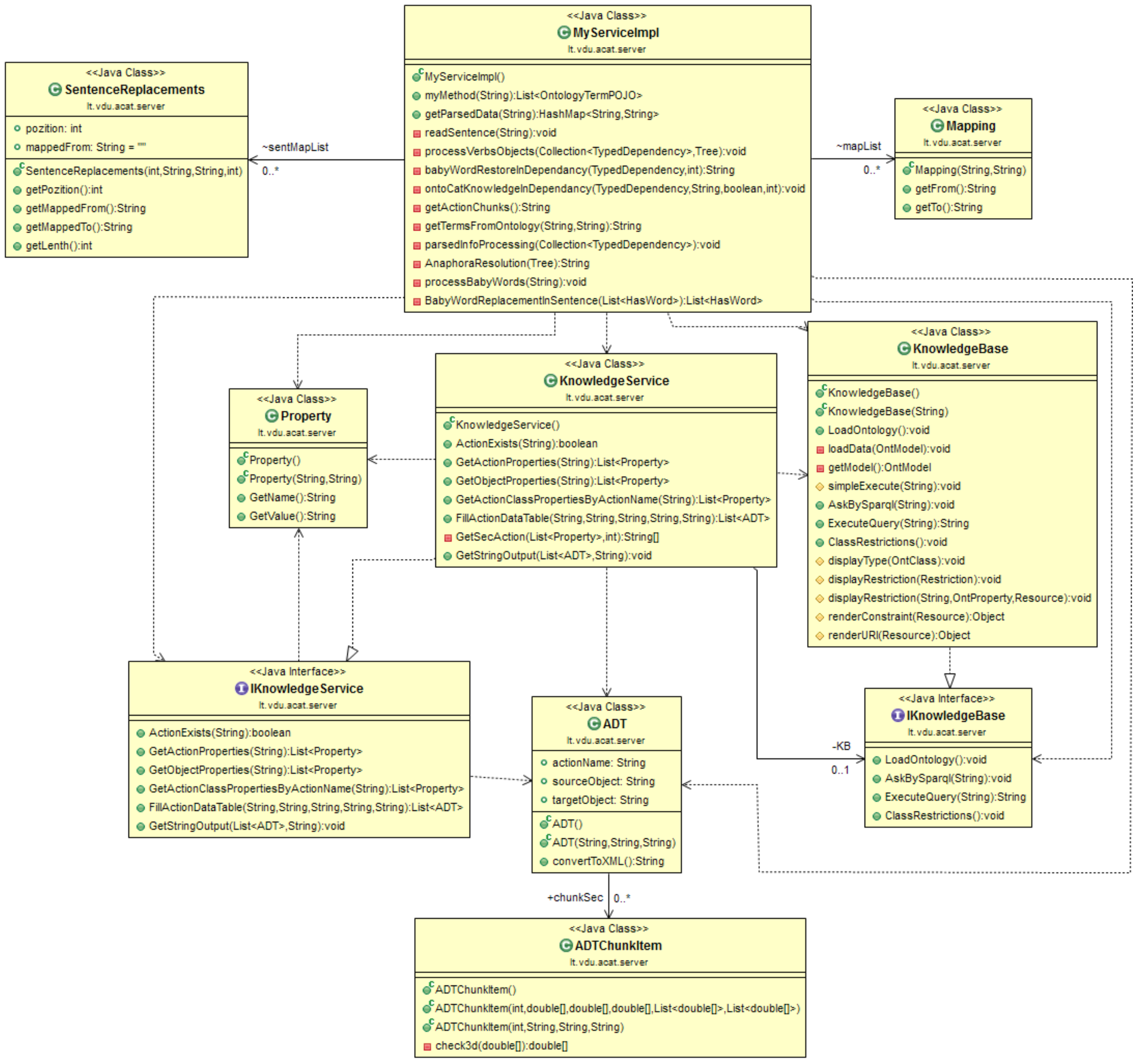

<<Java Class>>
MyServiceImpl
lt.vdu.acat.server
MyServiceImpl()
myMethod(String):List<OntologyTermPOJO>
getParsedData(String):HashMap<String,String>
readSentence(String):void
processVerbsObjects(Collection<TypedDependency>,Tree):void
babyWordRestoreInDependancy(TypedDependency,int):String
ontoCatKnowledgeInDependancy(TypedDependency,String,boolean,int):void
getActionChunks():String
getTermsFromOntology(String,String):String
parsedInfoProcessing(Collection<TypedDependency>):void
AnaphoraResolution(Tree):String
processBabyWords(String):void
BabyWordReplacementInSentence(List<HasWord>):List<HasWord>
<<Java Class>>
SentenceReplacements
lt.vdu.acat.server
pozition: int
mappedFrom: String = ""
SentenceReplacements(int,String,String,int)
getPozition():int
getMappedFrom():String
getMappedTo():String
getLenth():int
~sentMapList
0..*
~mapList
0..*
<<Java Class>>
Mapping
lt.vdu.acat.server
Mapping(String,String)
getFrom():String
getTo():String
<<Java Class>>
Property
lt.vdu.acat.server
Property()
Property(String,String)
GetName():String
GetValue():String
<<Java Class>>
KnowledgeService
lt.vdu.acat.server
KnowledgeService()
ActionExists(String):boolean
GetActionProperties(String):List<Property>
GetObjectProperties(String):List<Property>
GetActionClassPropertiesByActionName(String):List<Property>
FillActionDataTable(String,String,String,String,String):List<ADT>
GetSecAction(List<Property>,int):String[]
GetStringOutput(List<ADT>,String):void
<<Java Class>>
KnowledgeBase
lt.vdu.acat.server
KnowledgeBase()
KnowledgeBase(String)
LoadOntology():void
loadData(OntModel):void
getModel():OntModel
simpleExecute(String):void
AskBySparql(String):void
ExecuteQuery(String):String
ClassRestrictions():void
displayType(OntClass):void
displayRestriction(Restriction):void
displayRestriction(String,OntProperty,Resource):void
renderConstraint(Resource):Object
renderURI(Resource):Object
<<Java Interface>>
IKnowledgeService
lt.vdu.acat.server
ActionExists(String):boolean
GetActionProperties(String):List<Property>
GetObjectProperties(String):List<Property>
GetActionClassPropertiesByActionName(String):List<Property>
FillActionDataTable(String,String,String,String,String):List<ADT>
GetStringOutput(List<ADT>,String):void
<<Java Class>>
ADT
lt.vdu.acat.server
actionName: String
sourceObject: String
targetObject: String
ADT()
ADT(String,String,String)
convertToXML():String
-KB
0..1
<<Java Interface>>
IKnowledgeBase
lt.vdu.acat.server
LoadOntology():void
AskBySparql(String):void
ExecuteQuery(String):String
ClassRestrictions():void
+chunkSec
0..*
<<Java Class>>
ADTChunkItem
lt.vdu.acat.server
ADTChunkItem()
ADTChunkItem(int,double[],double[],double[],List<double[]>,List<double[]>)
ADTChunkItem(int,String,String,String)
check3d(double[]):double[]

Fig. A3. Class diagram for ACAT instruction compiler server side

User interface

The user interface of the ACAT instruction compiler is built using standard GWT UI widgets and panels. It consists of (Fig. A4): dialog window for entering the instruction to be analyzed and button, starting the analysis/compilation.

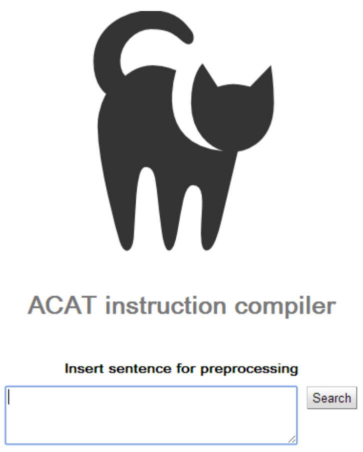
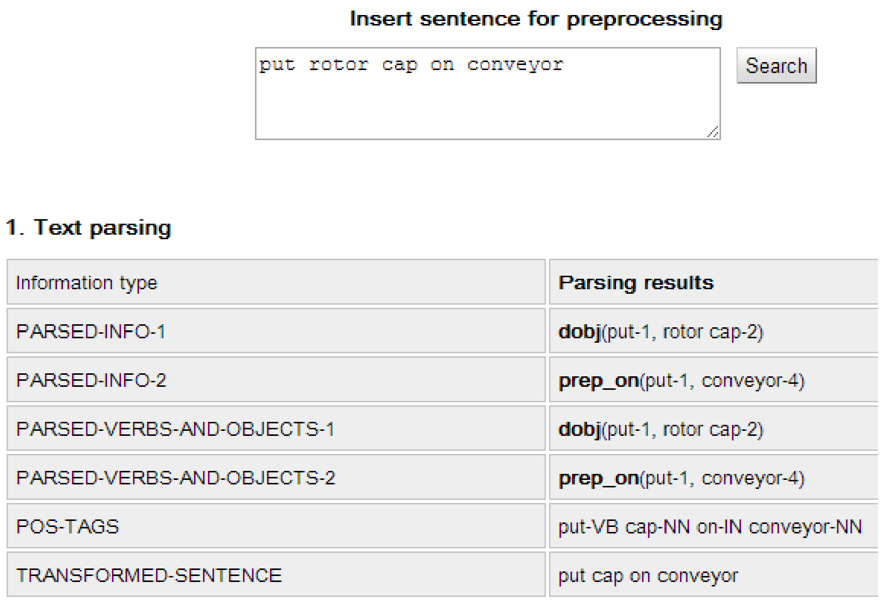


Fig. A4. User interface of the ACAT instruction compiler

Examples and screenshots

The ACAT action ontology, queried by the ACAT instruction compiler, is structured by identifying action and object descriptions from the action ontology. Main action, main object, primary object and secondary object are defined using syntactic structure of the sentence and dependency relations between root verb (action) and other sentence components. It also contains links to the Action Data Table (ADT) data, structured as defined in D2.1.

The compiler execution starts with entering the instruction in the compiler dialog window. The first step of the compiler execution results with the parse data for the instruction (Fig. A5).



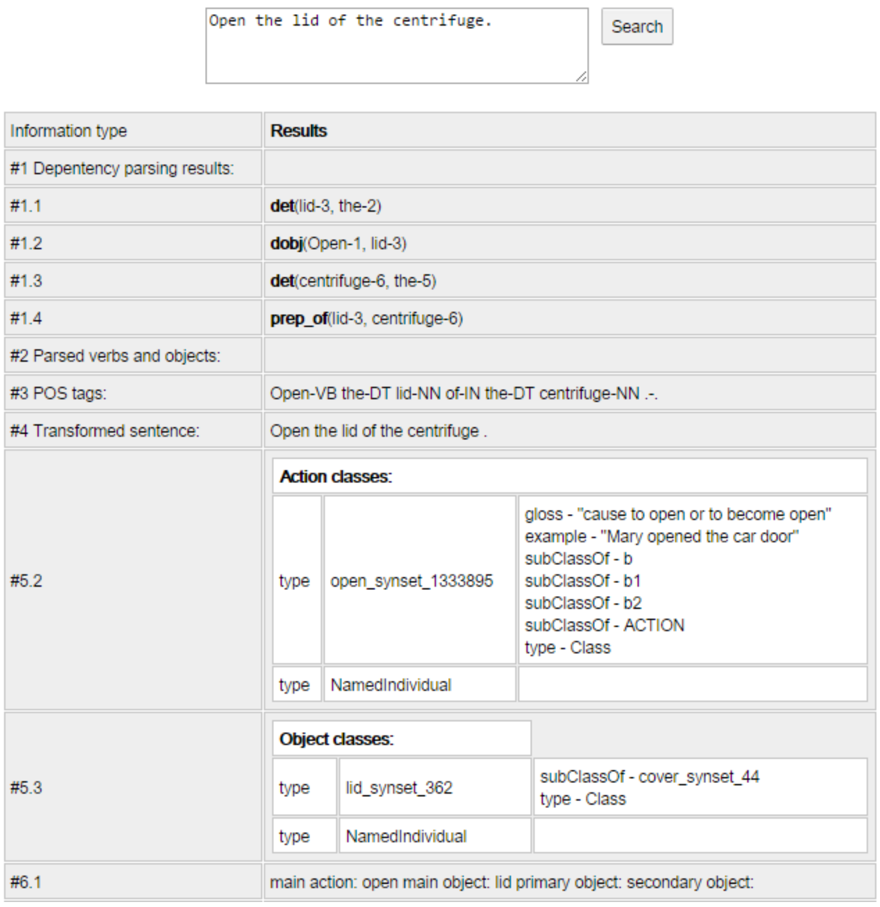
```
put rotor cap on conveyor
```

1. Text parsing

Information type	**Parsing results**
PARSED-INFO-1	**dobj**(put-1, rotor cap-2)
PARSED-INFO-2	**prep_on**(put-1, conveyor-4)
PARSED-VERBS-AND-OBJECTS-1	**dobj**(put-1, rotor cap-2)
PARSED-VERBS-AND-OBJECTS-2	**prep_on**(put-1, conveyor-4)
POS-TAGS	put-VB cap-NN on-IN conveyor-NN
TRANSFORMED-SENTENCE	put cap on conveyor

Fig. A5. Example of instruction parsing results

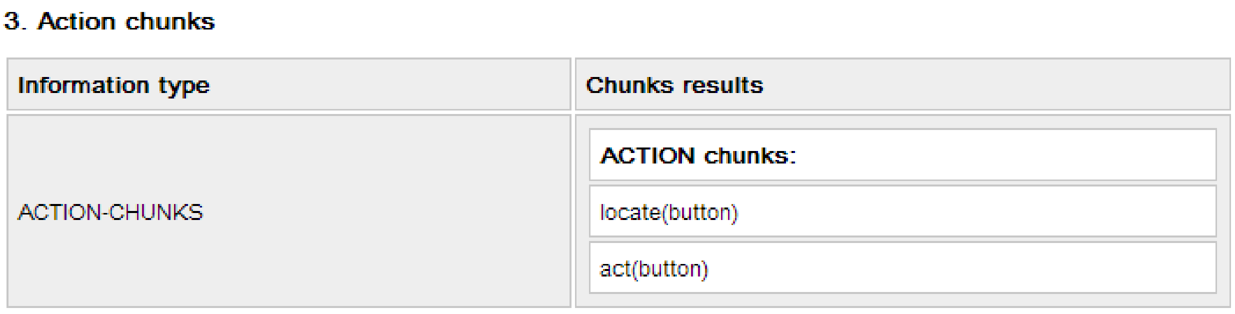
An example of ontology query results using the instruction compiler tool is presented in Fig. A6.



Information type	Results
#1 Depentency parsing results:	
#1.1	**det**(lid-3, the-2)
#1.2	**dobj**(Open-1, lid-3)
#1.3	**det**(centrifuge-6, the-5)
#1.4	**prep_of**(lid-3, centrifuge-6)
#2 Parsed verbs and objects:	
#3 POS tags:	Open-VB the-DT lid-NN of-IN the-DT centrifuge-NN .-.
#4 Transformed sentence:	Open the lid of the centrifuge .
#5.2	**Action classes:**
	type | open_synset_1333895 | gloss - "cause to open or to become open" example - "Mary opened the car door" subClassOf - b subClassOf - b1 subClassOf - b2 subClassOf - ACTION type - Class
	type | NamedIndividual
#5.3	**Object classes:**
	type | lid_synset_362 | subClassOf - cover_synset_44 type - Class
	type | NamedIndividual
#6.1	main action: open main object: lid primary object: secondary object:

Fig. A6. Excerpt of results of an Action ontology query

Wherever more detailed action data is present in the ontology (ADT data), detailed action sequence is returned by the compiler (Fig. A7).



3. Action chunks

Information type	Chunks results
ACTION-CHUNKS	**ACTION chunks:**
	locate(button)
	act(button)

Fig. A7. Results for a detailed action sequence

A.2. Set of instructions for instruction completion experiments for IASSES scenario

Instructions for “Rotor Assembly using the KUKA LWR” (https://www.youtube.com/watch?v=gcGez97HhGI&list=UUDUps4-IXGwAUYqZ46BVoew, manual transcription):

1. Take one metal ring from the top of ring stack and put it into the ring dispenser.
2. Grasp the ring from the side at the jaw of the ring dispenser and drop it into a cylindrical holder standing on the station.
3. Take the rotor shaft from the fixture and insert it into the cylindrical holder.
4. Turn the rotor shaft so that the magnet hole is in front of the robot
5. Pick a magnet from the magnet dispenser and insert it into the magnet hole that is directly in front of the robot.
6. Repeat steps 4 and 5 for 7 times.
7. Pick a rotor cap from the fixture on the robot platform.
8. Put the cap on the rotor shaft.
9. Adjust the cap so that it is aligned to the rotor axis.
10. Operate the press (not shown in the movie).
11. Take the rotor from the press, invert it and place it on the fixture on the robot platform.

A.3. Set of instructions for instruction completion experiments for CHEMLAB scenario

Instructions for “Plasmid DNA Extraction (Megaprep)” (https://www.youtube.com/watch?v=xcPlV-xMdVM, manual transcription):

1) Open centrifuge.

2) Pour suspension of e-coli from a flask into a plastic bottle PB1.

3) Turn the lid to cover the bottle PB1.

5) Pick a bottle PB1 with e-coli and put it into centrifuge.

6) Close the lid of centrifuge.

7) Start centrifuge by pressing a button But1.

8) Wait for 10 minutes.

9) Stop centrifuge by pressing button But1.

10) Open the lid of the centrifuge.

11) Take out the bottle PB1 of centrifuge (and put it on fixature).

12) Open the plastic bottle PB1 PB1.

13) Pour the liquid out of the bottle PB1 into a flask.

14) Pipette 15 ml of buffer R3 into bottle PB1.

15) Close the bottle PB1 again.

16) Shake the bottle PB1 by holding against the shaking device.

17) Open the bottle PB1.

18) Open the bottle with lysis buffer PB2.

19) Pipette 75 ml of lysis buffer into the bottle PB1.

20) Close the bottle PB1 by turning the lid.

21) Mix the contents of the bottle PB1 gently by inverting 4-6 times.

22) Open the bottle PB1.

23) Open the bottle PB3 with precipitation buffer N3.

23) pour 75 ml of the precipitation buffer N3 into plastic bottle PB1.

24) Close the bottle PB1.

25) Mix by inverting PB1 4-6 times.

26) Put the bottle PB1 into centrifuge.

27) Close the lid of the centrifuge.

28) Start centrifuge.

29) Wait for 15 minutes.

30) Stop centrifuge.

31) Open the lid.

32) Take out the bottle PB1 and put it on the table.

33) Open the bottle PB1.

34) Pour the contents of PB1 into lysate filtration cartridge attached on the glass bottle GB1 slowly.

35) Wait until the liquid has drained into the bottle GB1.

36) put DNA binding cartridge on the different bottle GB2.

37) open the bottle with equilibration buffer.

38) Pour 200 ml of equilibration buffer into the DNA binding cartridge.

39) Wait until the liquid has drained from the cartridge.

40) Take the lysate filtration cartridge off the bottle GB1.

41) Pour the contents of the bottle with lysate GB1 into the DNA binding cartridge.

42) Wait until the liquid has drained.

43) Open the bottle with wash buffer PB4.

44) Pour 550 wash buffer W8 into the DNA binding cartridge.

45) Wait until the liquid has drained.

46) Unscrew the DNA binding cartridge from the full bottle GB2 and screw it on an empty bottle GB3.

47) Pour the contents of the full bottle GB2 into a plastic bottle PB5.

47a) Cover the bottle PB5 with a lid.

48) Unscrew the bottle with elution buffer PB6.

49) Add 120 ml of elution buffer E4 to the cartridge.

50) Wait until it had drained into the bottle.

51) Unscrew the cartridge.

56) Unscrew the plastic bottle.

56a) Pour the contents of the bottle with the elution buffer into the plastic bottle PB5.

57) Pipette 85 ml isopropanol into the plastic bottle PB5.

58) Screw the lid on a plastic bottle PB5.

59) Mix by inverting bottle PB5 several times.

60) Put the bottle PB5 into centrifuge.

61) Close the lid of the centrifuge.

62) Start the centrifuge.

63) Wait for 30 min.

64) Stop the centrifuge.

65) Open the lid of the centrifuge.

66) Pick the bottle PB5 from the centrifuge and put it on the table.

67) Unscrew the bottle PB5.

68) Pour the liquid from the bottle PB5 into a glass vessel carefully.

69) Pipette 70% ethanol into the plastic bottle PB5 onto the DNA pallet attached to the bottom.

70) Open test tube TT1 standing on a holder (New class “hand only with grasping” plus pick and place)

71) Collect the solution of the plastic bottle PB5 with the pipette and put it into the test tube TT1.

72) Click on the lid on the test tube TT1.

73) Put the test tube TT1 into the centrifuge.

74) Close the centrifuge.

75) Start the centrifuge

76) Wait for 10 minutes

77) Stop the centrifuge.

78) Open the lid of the centrifuge.

79) Take out the test tube TT1 off the centrifuge.

80) Pour out the liquid off the test tube TT1 into a glass vessel.

81) Put a test tube TT1 onto a holder.

82) Cover the holder with the lid.

83) Wait for 10 minutes.

84) Open the test tube with TE buffer TT2.

85) Pipette 1 ml of TE buffer from TT2 into the test tube TT1.

86) Close the test tube TT1.

87) Put the test tube onto the fixature.

A.4. Objects role definition in Action Data Tables

Hand: The hand (always present in an action).

Tool: The entity grasped by the hand to perform an action instead of the hand (not always present).

Main Object (MO): The object which is first touched by *hand*/*too*l (always present).

Primary object=Source (PO): The object which is first touched/untouched by the *main object* (not always present).

Secondary object=Target (SO): The object which is second touched/untouched by the main object (not always present).