Deliverable number:	D3.1
Deliverable Title:	Textual completion of instruction sheets
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:	01-09-2014
Actual Date of Delivery to the EC:	15-09-2014

Content

1. Executive summary

This document is the first version of deliverable D3.1 and we are planning to provide an update in about 2 months to show more results on the two scenarios of the ACAT project.

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 by-now designed algorithms and related tools for the filling-in of the missing **symbolic information** (steps 1 and 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. As action sequence definition is our first step in the compilation process, to make the presentation consistent, we include action sequencing already into this deliverable, which normally would belong to the next deliverable D3.2.

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. The dividing line between D3.1 and D3.2 is hard to draw. Given the ACAT data structure based on ADTs we have chosen in D2.1, it was natural to include into D3.1 the entire process for symbolic filling of the ADTs which already includes action sequencing (a topic from D3.2). However, the time was too short to develop the project up to that point in month 18, thus we will re-submit an update of D3.1 in month 21. Here is the version presenting achievements of the project until month 18.

A simple example of an instruction where textual completion is needed is "take a rotor cap and place *it* on 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 (specificity 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 issue on textual completion that is much wider discussed in this deliverable is very tightly related to robotics. Not every verb in the instruction sheet is directly linked to robotic actions. E.g. consider the verbs "neutralize" or "harvest" (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 "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 what 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, and place it into centrifuge, etc. Linking of action verbs to action sequences executable by a robot in ACAT is implemented through action ontology. Action verbs in the ontology are divided into action classes where one class has the same sequence of so called action primitives (as described in D2.1 Chapter 4). Thus, textual completion of an instruction takes an instruction in natural language and transforms into a sequence of action primitives based on action classes.

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 verbs,
- corresponding action classes are identified in the ACAT ontology and the structure of the instructions is defined,
- finally detailed background 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 giving Documentation of the ACAT instruction completion (instruction compiler) software.

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.

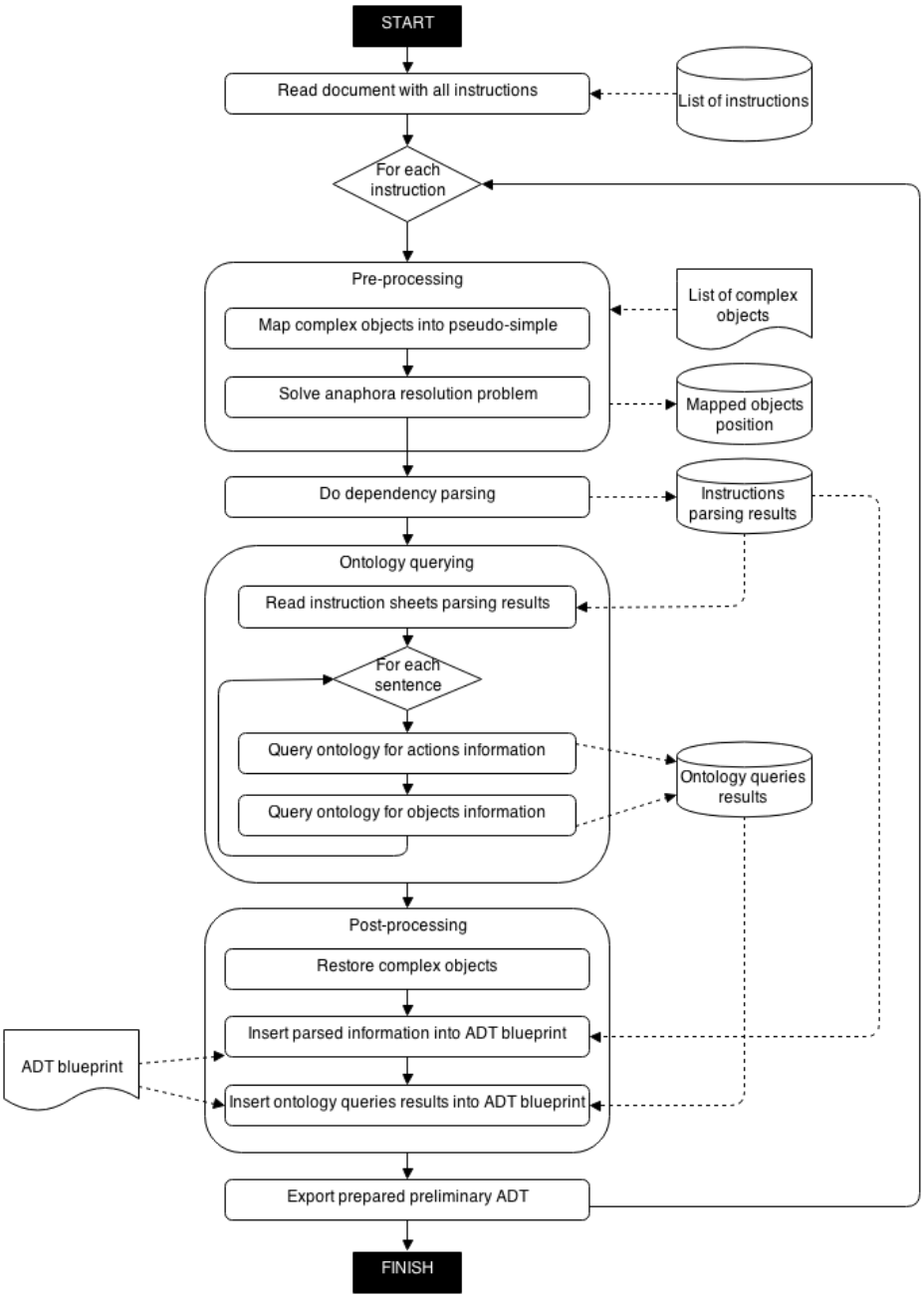


Fig.1. Algorithm for filling in the missing information in instruction sheets

2. SPARQL queries to the corresponding action ontology in order to extract 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) from domain-specific corpus texts, accumulated in the ACAT Project (see WP1, WP2).

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 action background elements, identified in instruction parse-trees;
    b. Determine required action background elements, missing from the instruction text.
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.
2. Filling-in missing instruction information with knowledge from ontology.

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.

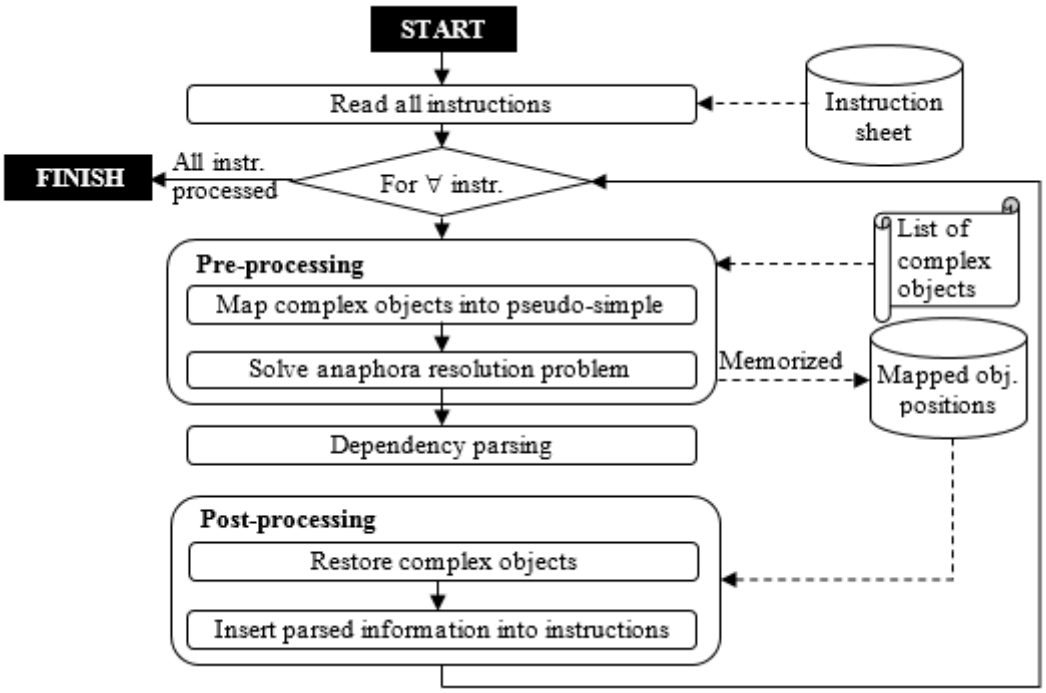


Fig.2. **Generalized block schema of the instruction parsing system**

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 *locate a rotor cap on robot platform* clause; direct object (***dobj*** dependency type**)** relation between *locate* and *rotor cap* indicates "what to locate"; 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 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:

- **Read all instructions**. Instruction parsing is an iterative process, performed instruction after instruction. Assuming that each instruction corresponds only to one sentence, we used 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 objects 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. 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 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**. 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[1], 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, 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*.

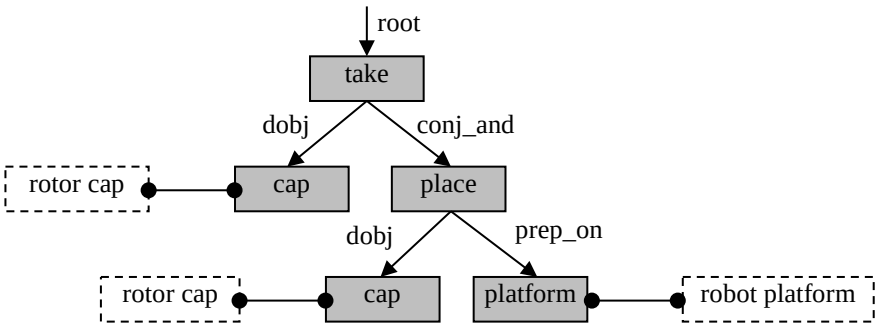


Fig.4. **Grey blocks indicate words in the parsed sentence (determiners and punctuation marks are ignored); arrows – dependency relations and their labels; dashed blocks – mapped information. E.g. *place* is node's *take* dependent, but node's *platform* governor; core verb *take* is a *ROOT* node dependent.**

- **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

[1] 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

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 XML file (see the example in Figure 5); few PrintScreens from CAS Visual Debugger are presented in Fig.6 – Fig.8 as well.

```
<?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).

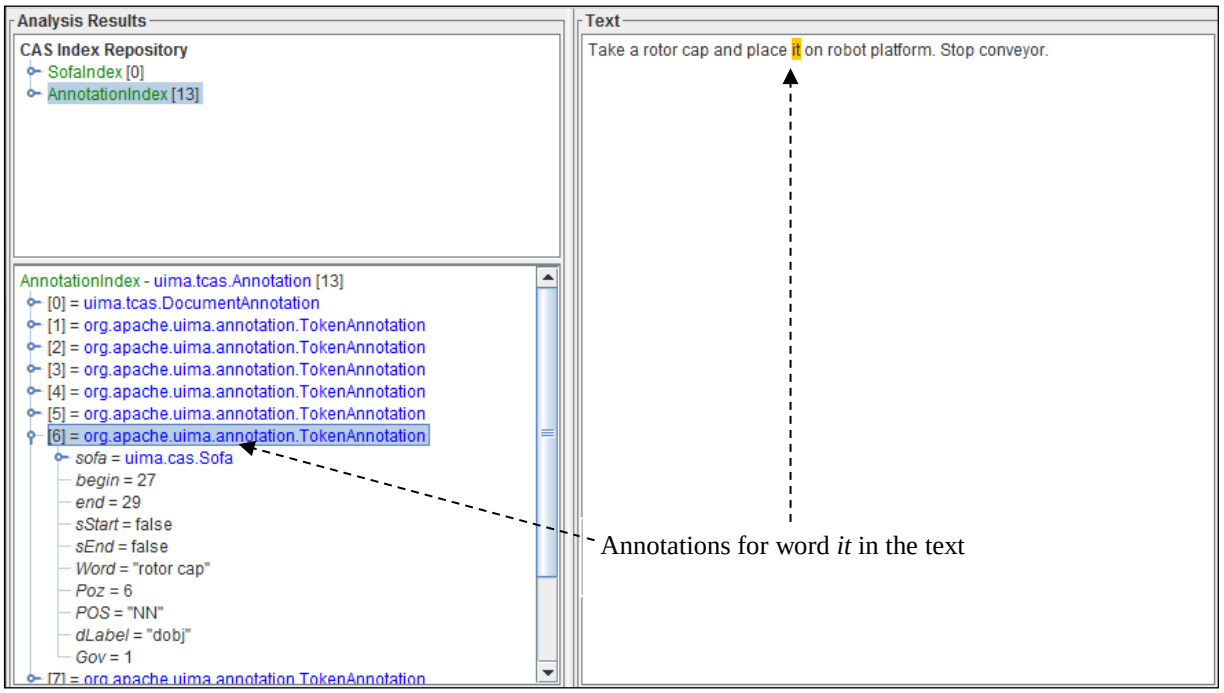


Fig.6. Detailed annotations for *it*

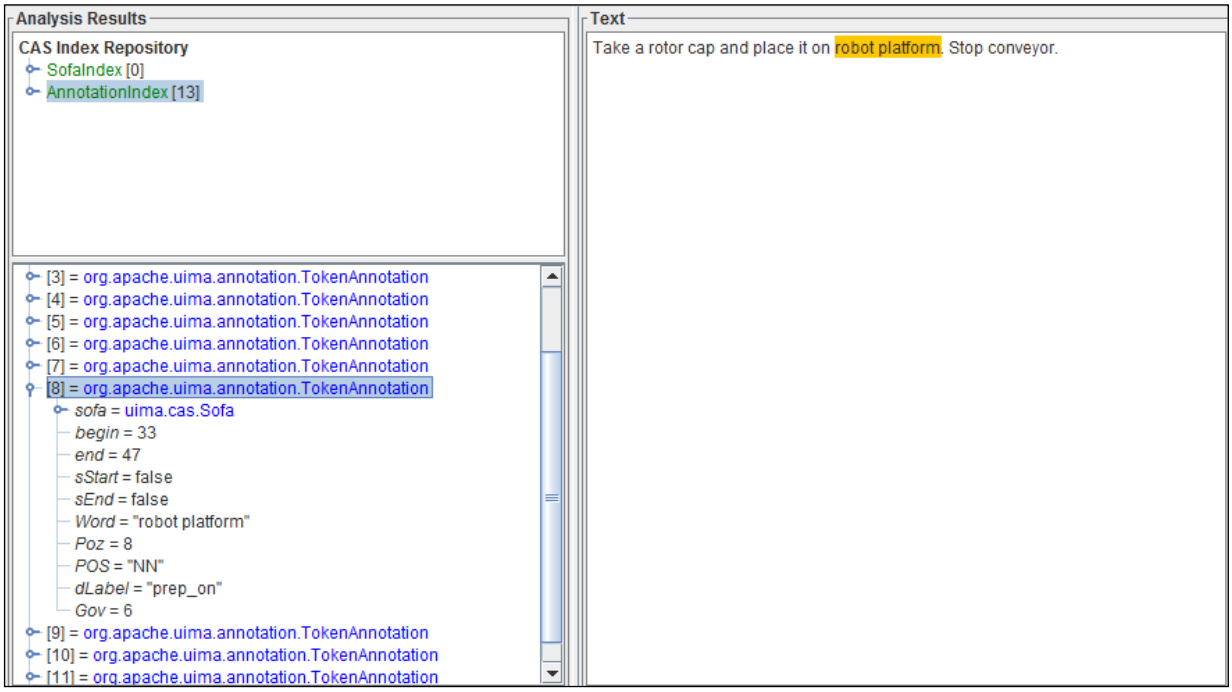


Fig.7. Detailed annotations for *robot platform*

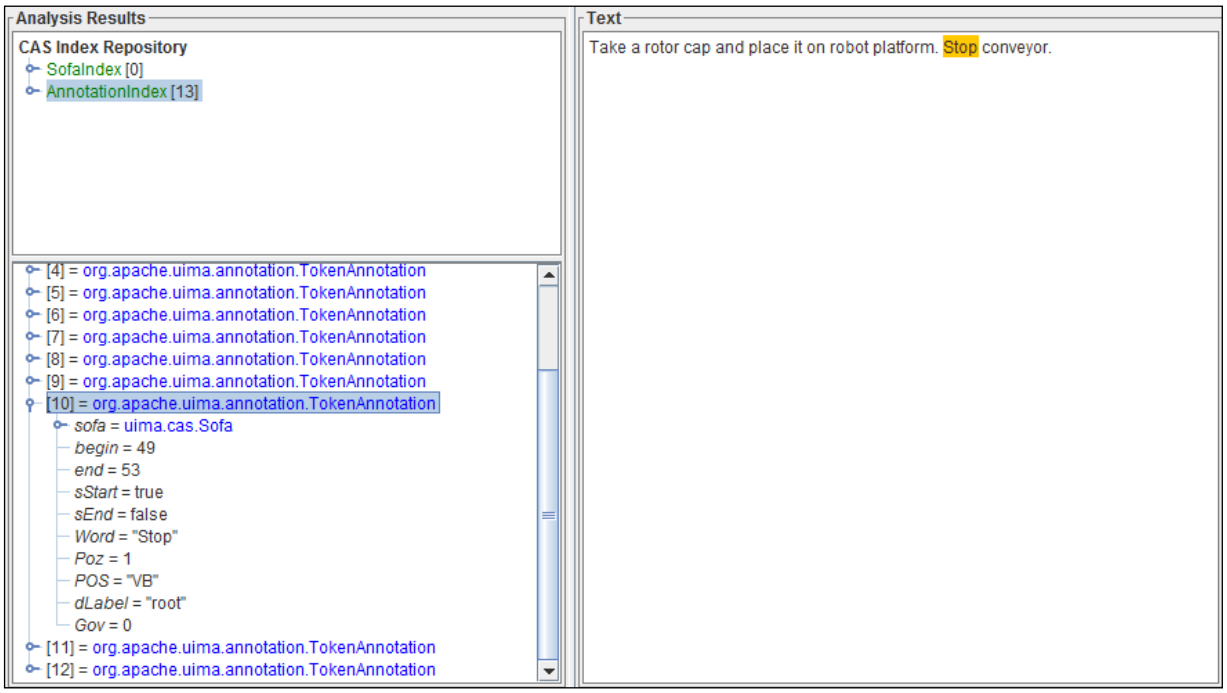


Fig.8. Detailed annotations for *Stop*

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, underspecification 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 a 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].

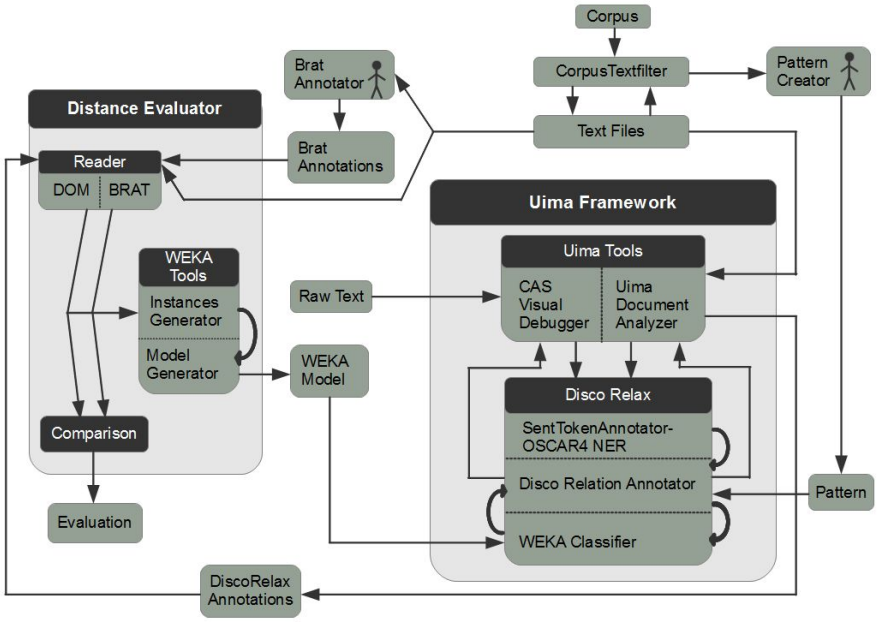


Fig. 9. Architecture overview of the causal relation extraction engine.

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 9.

4.3. 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 an action background.

By the structured conceptual view of stored data, ACAT ontology is used in describing the class hierarchy (also called a 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. ACAT ontology is queried to extract action structure for known actions and to get detailed information (as far as possible) for an action background.

Fig. 10 presents the conceptual model of knowledge extraction by querying ACAT action ontology. Each query makes use of the instruction parsing results – recognized core verbs (actions that the robot has to perform) and action background.

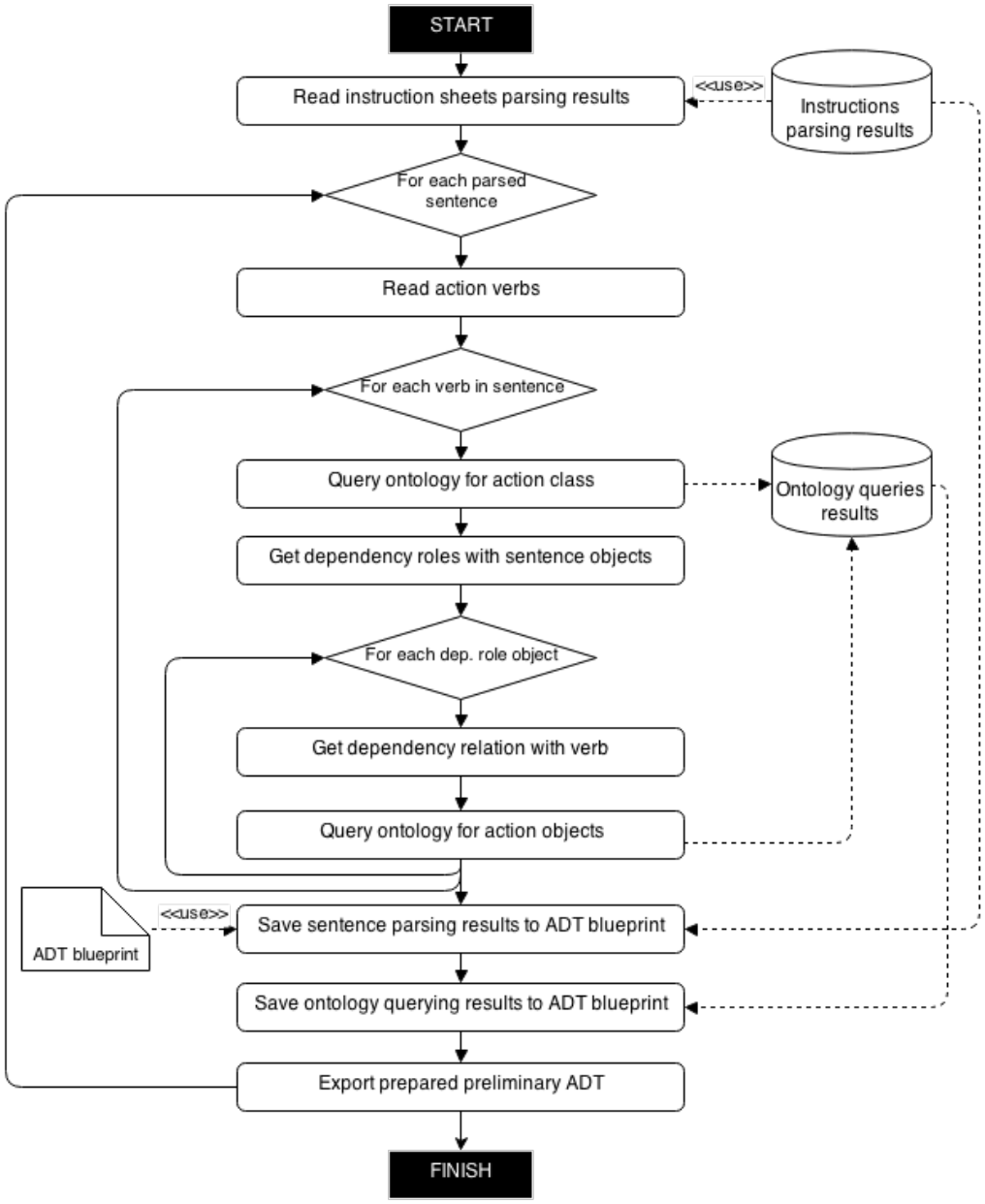


Fig. 10. Knowledge extraction by querying the ACAT ontology

Ontology querying starts from gathering the results of syntax dependency and POS parsing (See 4.1. Preprocessing and dependency parsing). Verbs are extracted from each parsed sentence. Each verb corresponds to an action, that robot has to perform. At the next step, it is necessary to classify each verb to a corresponding action class. A pre-defined set of action classes was described in the ACAT Deliverable D2.1: tool action, tool with movement action, homing action, null-action, hand-only action, handling action, pick and place action.

By querying the ontology using SPARQL queries we can define the action class for each action, and, furthermore, extract information on the action structure - action steps (action primitives).

An example of a SPARQL query for defining the action class for an action "harvest" is given in Fig. 11.

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. 11. Example: SPARQL query for action class extraction

If the ACAT ontology contains only one instance of the action verb, a simple SPARQL query for the action class extraction is used. However, if the action ontology contains multiple instances of a particular verb with diversified action class hierarchy, it is impossible to rely only on ontology queries.

In action class disambiguation tasks, action frequency information from a domain specific corpus is used. Along with common action properties, each action in the ACAT ontology contains the calculated action frequency for a given action category in its feature set. SPARQL query results are sorted in descending order by action instance frequency for separate action categories.

Action category disambiguation results can be also improved by applying action classification by its context (Markievicz et al, 2014). The classification by context approach is used in unrecognized (unclassified) action instance assignment to the appropriate action class. The problem is solved as the text classification task, where appropriate verb contexts are treated as classification instances. Action contexts are classified using Support Vector Machine (SVM) algorithm with the Naïve Bayes (NB) as the baseline (Kotsiantis , 2009).

After defining the action class and, thus, knowing the structure of the action background, the next step of instruction sheet knowledge processing is to fill-in information for each action background object. Action objects and their relations to main action (roles) need to be defined. Dependency parsing data is used for describing environment objects syntactic roles. 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. 12). 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. 12. SPARQL query for describing properties of action object.

All the data obtained from parsed instruction sheets and ACAT ontology queries is structured and included in the descriptions of separate actions steps (action primitives), predefined for each action class. For example, if an action PUT belongs to the PICK AND PLACE action class, the following action primitives are defined: 1 – locate object, 2 – pick up object from source object, 3 – put down object, 4 – put down on target object.

A preliminary action data table (ADT), filled out with data from instruction sheets and ACAT ontology queries, is constructed for each action primitive. Usually, it contains only the predefined action class and action background objects involved (Fig. 13). This preliminary ADT can later be updated with detailed experimental or observation data.

```
<instruction>Take a jar and put it in a pot</instruction>
<action-context>void</action-context>

<name>PickAndPlace</name>

<refframe>kuka_goe.baseframe </refframe>

<main-object>
    <name>Jar</name>
    <cad-model>
        <uri>Jar.pcd</uri>
    </cad-model>
    <part-graph>
        <uri>void</uri>
    </part-graph>
    <potential-objects>
        <object>
            <pose>
                <position>0.552 0.563 0.121</position>
                <quaternion>0.663 0.635 -0.301 -0.259</quaternion>
                <pose-reliability>0.5</pose-reliability>
            </pose>
            <part-of-interest>void</part-of-interest>
            <object-parts>void</object-parts>
            <size>0.13 0.08 0.08</size>
            <made-of>Plastic</made-of>
            <mass>0.75</mass>
        </object>
    </potential-objects>
</main-object>

<primary-object>
    <name>Table</name>
    <cad-model>void</cad-model>
    <part-graph>
        <uri>void</uri>
    </part-graph>
```

Fig. 13. Example of an ADT structure

5. Instruction completion application to ACAT project scenarios

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

- syntactic analysis results with action verbs and their dependency with action background objects;
- ACAT ontology query results with identified action categories and defined action steps (action primitives), predefined for each action category;
- ACAT ontology query results with identified action background objects and their properties;
- detailed instruction steps, filled-out with data, collected from instruction sheet parsing and ontology querying results;
- preliminary ADT's for each action step.

Instruction parsing and action step detailing 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.

5.1. Application to IASSES scenario

The ACAT instruction completion application execution in IASSES scenario is demonstrated for an instruction: *"Put rotor cap on conveyor"* (Fig. 14).

Insert sentence for preprocessing

put rotor cap on conveyor

Search

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. 14. 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. 14 – "*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. 14).

Sentence *"Put rotor cap on conveyor"* POS annotation (*"POS tags"* row information in Fig. 14) results: put – verb, base form (VB), cap – noun, singular (NN), on – preposition (IN), conveyor – noun, singular (NN).[2]

Dependency parsing annotation results are presented in *"Parsed info"* rows (Fig 14): ***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.[3]

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 ontology querying step.

After saving the parsing results to an application internal database, the ontology querying phase begins. Ontology querying results are grouped into ACTION-INFORMATION and OBJECT-INFORMATION logical groups (Fig 15).

```
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. 15. Sample ontology querying results for IASSES scenario

[2] 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

[3] Full list of dependency types, used in Stanford NLP project: http://nlp.stanford.edu/software/dependencies_manual.pdf

For each action verb in a sentence, an ontology SPARQL query is build. All corresponding verb instances in the ontology, belonging to the ACTION category, are selected. In this example, we locate an action, belonging to the *PICK_AND_PLACE* category (rdf:type – PICK_AND_PLACE). It is defined as ontology entity (owl:NamedIndividual). Query results also describe that the action *put* is similar to actions *place* and *move* (owl:sameAs – move, place). There are also predefined steps of the *PICK_AND_PLACE* action category: *first* (:action1) – *locate, second* - (:action1) *pick_up*, *third* (:action3) – *put_down.* The target object (:has_targer_object) of thr *put* action is the *rotor_cap.* Ontology querying results also include synonyms, both for actions and for action background elements (objects), wherever possible. In the example (Fig. 16), *rotor cap* is defined as an objects similar (owl:sameAs) to *cap*. It belongs to the ACTION_OBJECT category and is defined as ontology entity (owl:NamedIndividual).

Fig. 16. Sample ontology querying results for IASSES scenario

In the example, the following action steps (action primitives) are defined (Fig. 17): 1 – locate cap, 2 – pick up cap, 3 – put down cap on conveyor. The sequence of action steps is defined from the query results of the action information (Fig. 15). For each step, a corresponding preliminary Action Data Table (ADT) is filled-out, including information about action background objects, extracted both from parse data and ontology.

```
locate(cap)
pick_up(?,cap)
put_down(cap,conveyor)
```

Fig. 17. Defined action steps (action primitives) in the IASSES scenario example. If there is no possibility to define some action objects or their properties, the "?" mark is used

The instruction compiler for the IASSES scenario was tested in a demo-setup at AAU with the Little Helper 4 robot. Fig. 18 and Fig. 19 show screenshots from this demo setup for "Pick rotor-cap from conveyor" instruction.

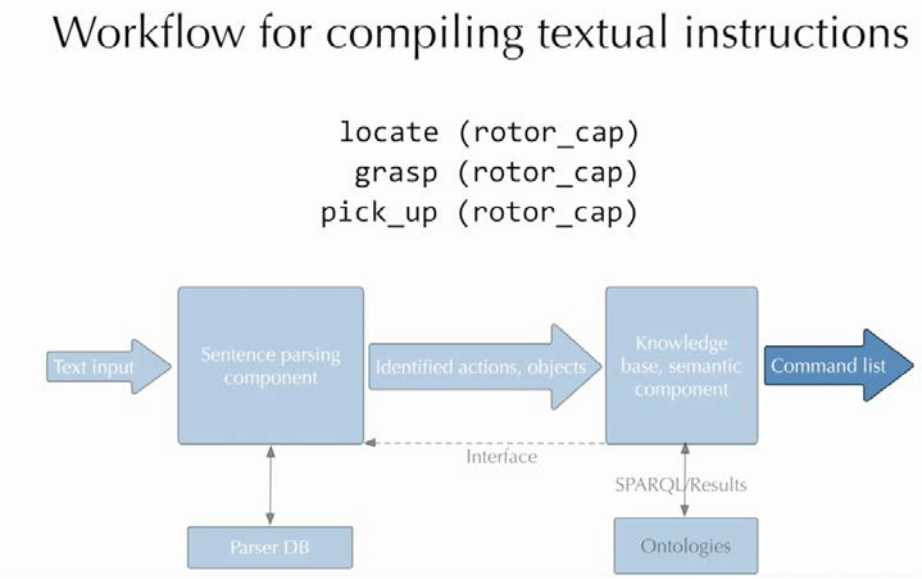


Fig. 18. The trials of the instruction compiler in IASSES demo setup with Little Helper 4 (1)

Fig. 19. The trials of the instruction compiler in IASSES demo setup with Little Helper 4 (2)

5.2. Application to CHEMLAB scenario

The instruction sentence "*Harvest the E.Coli by centrifugation at 6000g for 10 minutes at 4°C"* is used for the demonstration of the instruction completion algorithm for the CHEMLAB scenario. First, the words in the instruction sentence are annotated with dependency syntax (Fig 20). Dependency syntax results (*"Parsed Info" rows*): dep(harvest,E.Coli) – unrecognized role,

prep_by*(E.Coli, centrifugation)* – action object *E.Coli* is modified by *centrifugation* entity, ***prep_at****(centrifugation, 6000g)* – action object *E.Coli* is modified by *6000g* entity, ***num****(minutes, 10)* – defines quantity of minutes, ***prep_for****(6000g, minutes)* – entity *6000g* is modified by *minutes* entity, ***prep_at****(minutes, C)* – entity *minutes* is modified by *C* entity.

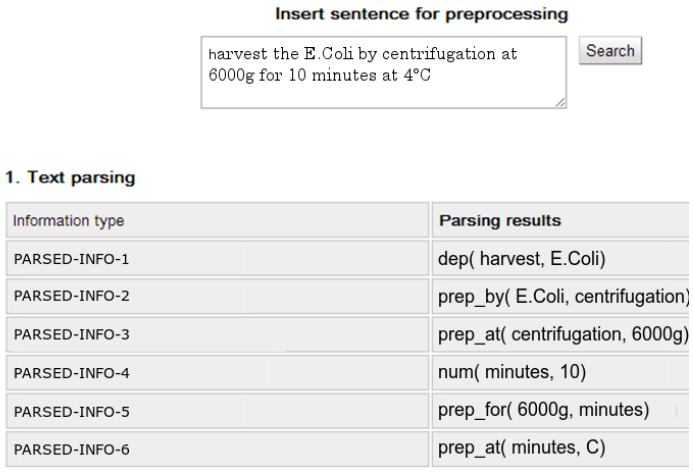


Fig. 20. Example of text parsing results for CHEMLAB scenario

In the next stage, each verb and noun from the parsed instruction sentence is used as an input for CHEMLAB ontology SPARQL queries (Fig 21).

In this example, the action *harvest* in the instruction is recognized as a *HAND-ONLY* action (rdf:type HAND-ONLY_ACTION). However, it includes a sub-action – *centrifugation* (:include_action centrifugation). It is also defined as entity (owl:NamedIndividual) and has target object *polymer* (:has_target_object polymer).

By executing another SPARQL query, it is possible to classify the *centrifugation* action as a *TOOL_ACTION* (rdf:type TOOL_ACTION), which requires tool *centrifuge* (:requires_tool centrifuge). It has also target object *cell* (:has_target_object cell). Also, information about action background objects is extracted – it is possible to recognize, that E.coli is a cell (owl:is_a cell) and bacteria (owl:is_a bacteria) – type of material (rdf:type TOOL_ACTION).

As in this case we have two action categories, identified for the instruction sentence: *HAND-ONLY* (for *harvest*) and *TOOL-WITH-MOVERS-ACTION* (for *centrifugation*), two action step sequences are defined (Fig. 22).

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)
       }
```

Results:

action	property	object
harvest_3	has_target_object	polymer
harvest_3	type	NamedIndividual
harvest_3	include_action	centrifuge_v
harvest_3	type	HAND-ONLY_ACTION

Query:

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

Results:

action	property	object
centrifuge_v	has_target_object	cell
centrifuge_v	type	NamedIndividual
centrifuge_v	type	TOOL_ACTION
centrifuge_v	requires_tool	centrifuge_n

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)
       }
```

Results:

action	property	object
cell	type	NamedIndividual
cell	type	ACTION_OBJECT
cell	type	MATERIAL

Query:

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

Results:

action	property	object
E_Coli	type	NamedIndividual
E_Coli	type	MATERIAL
E_Coli	is_a	bacteria
E_Coli	is_a	cell

Fig. 21. Example of ontology query results for CHEMLAB scenario

HARVEST action steps:	CENTRIFUGATION action steps:
locate(E.Coli)	locate(centrifuge)
act(E.Coli)	pick_up(?,centrifuge)
	locate(cell)
	act(cell)
	remove(?,centrifuge)
	put_down(centrifuge,?)

Fig. 22. Example of action steps in IASSES scenario.

Additional language level reasoning needs to be added to blend the two actions "harvest" and "centrifugation" into one robotic action sequence, as here obviously centrifugation is the actual action that needs be performed.

In both action sequences, there remains an unrecognized action step *act*. In addition, our currently defined action classes are obviously insufficient. The centrifugation was attributed to the closest class from the existing one (tool with movers action), however our action primitive sequence attributed to centrifugation is incorrect, as we shall open the centrifuge instead of picking it up, then we need to place the test-tubes into centrifuge, etc. These are all indicators that the action sequence needs more details. However currently this cannot be extracted, neither from instruction parse data, nor from the action ontology. In this case, more action classes, or a more detailed instruction sheet, including more detailed steps of centrifugation action, would be necessary. Or, alternatively, detailed ADTs, compiled from experiments, could be used.

6. 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 and convert it into action primitive sequence providing for each action primitive 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 part 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 application are in principle able to parse instruction sheet information and define the actions and syntactic roles for action background elements. The approach of combining parsed instruction sheet information with ontology query results seems to give adequate information for instruction completion. Knowledge of actions and action background objects can be further improved by 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 – in this case, an SVM classification algorithm is used.

Currently we are getting some mistakes in defining the action primitive sequence and are expecting to correct most of those mistakes until the month 21, when an update of this deliverable will be provided. One cannot expect to perform the conversion of human readable instructions to an ADT 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.

The action ontology used includes predefined action classes with a description of action primitives (action step sequences), where the action class set is not yet complete. This is the first source of mistakes in the current compilation procedure. The evaluation of compiler results for both scenarios should also consider the fact, that the size of the knowledge base, currently used for each task, is different. The IASSES ontology is built from a corpus containing over 3.5 million words, CHEMLAB form a corpus, containing more than 3 million words of common chemical laboratory descriptions and about 5.9 million words of biochemical texts. These ontologies are still being put together and extended by us.

It is impossible to predict – but to some degree expected - that an ontology, built from a larger domain-specific corpus gives better queries results. The results of instruction completion using ontology queries depend, in addition to size, on the focus of the ontology and the domain-specific corpus focus w.r.t the specific tasks. Future work on instruction completion should include the improvement of algorithm precision and accuracy by expanding the ontology from additional domain-specific corpus data, and also by including additional weight information, obtained from the feedback of ADT validation by the human and execution.

7. 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.

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, ppages 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).

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.

8. Appendices

8.1. Documentation of the ACAT instruction completion (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, 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.

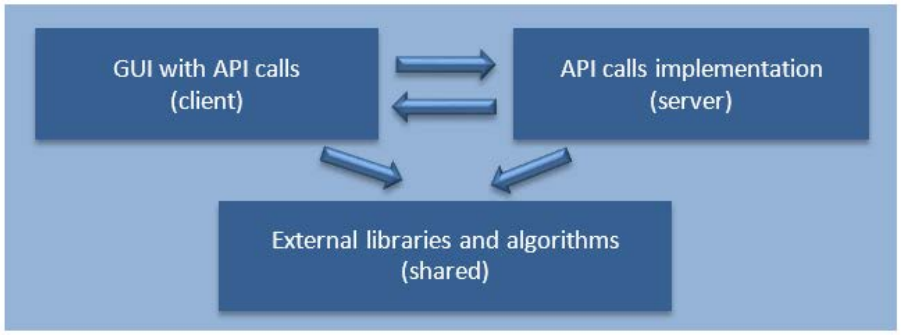


Fig. A1. The conceptual MVP model of the ACAT instruction compiler

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).

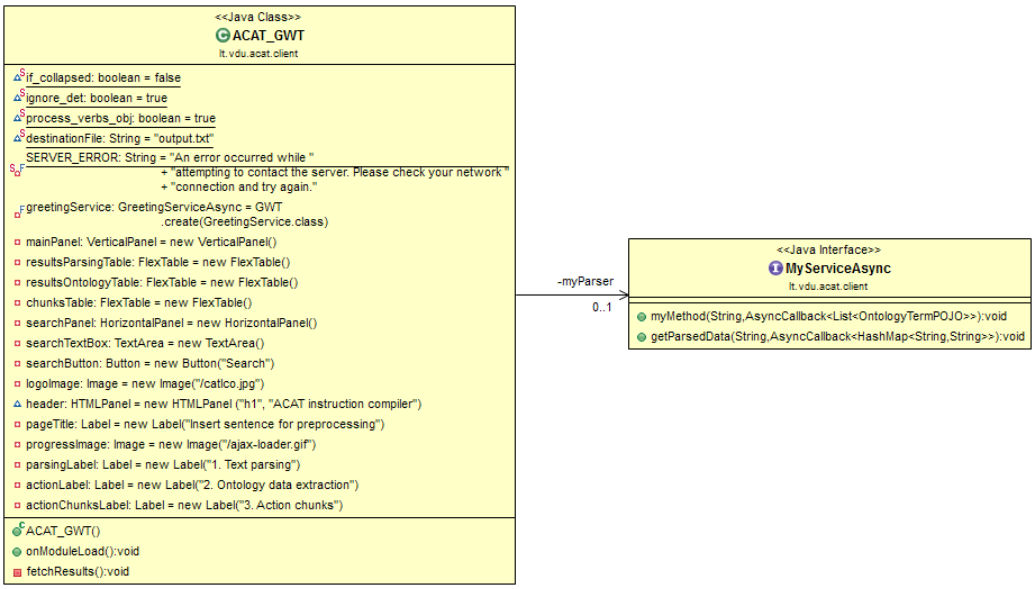


Fig. A2. Class diagram for ACAT instruction compiler client side

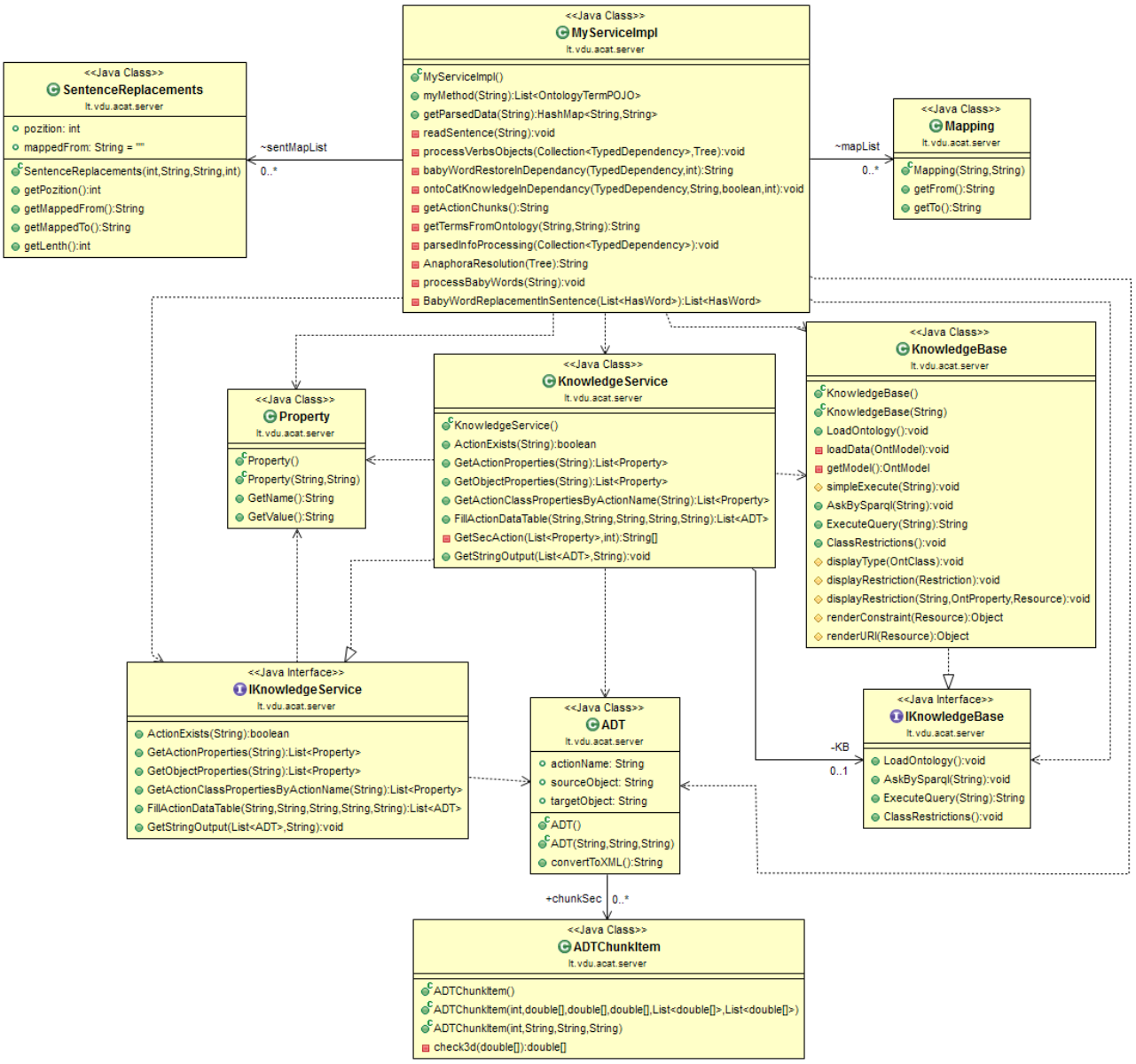


Fig. A3. Class diagram for ACAT instruction compiler server side

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.

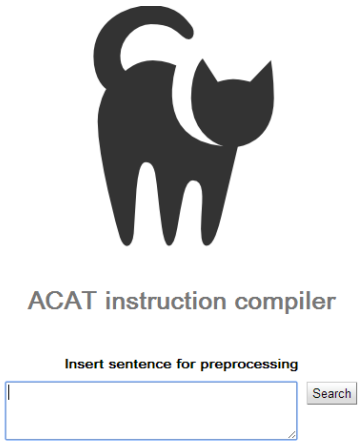
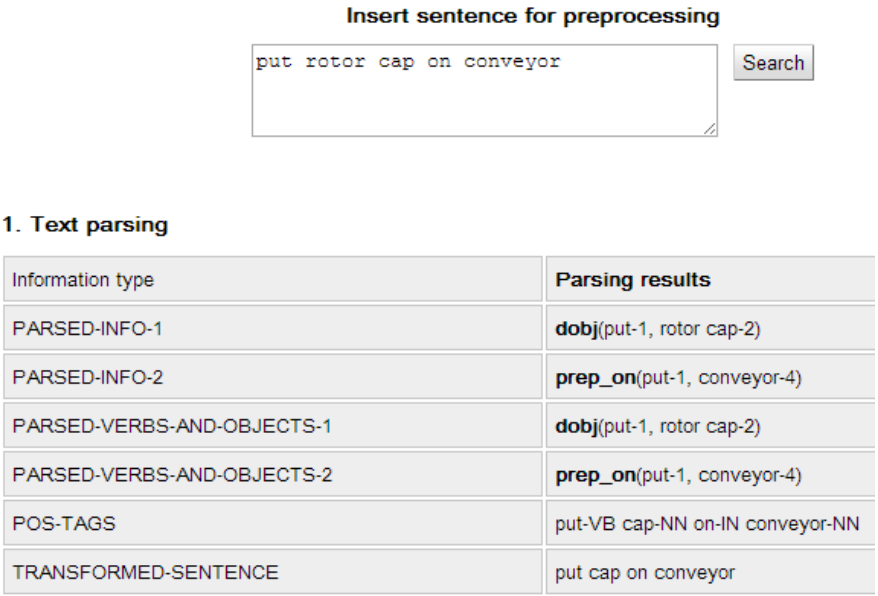


Fig. A4. User interface of the ACAT instruction compiler

Examples and screenshots

The ACAT action ontology, queried by the ACAT instruction compiler, is structured by classifying actions into action categories as described in the Deliverable D2.1. It also contains links to the Action Data Table (ADT) data, also 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).



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.

2. Ontology data extraction

Information type	Ontology results	
ONTOLOGY-INFO-ACTION-1	**Action classes:**	
	HAND-ONLY_ACTION	subClassOf
ONTOLOGY-INFO-ACTION-2	**Action classes:**	
	HAND-ONLY_ACTION	subClassOf
ONTOLOGY-INFO-OBJECT-2	**Object classes:**	
	subClassOf	PART_OF_ACTION_OBJECT
	label	"button"
	type	Class

Fig. A6. 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