

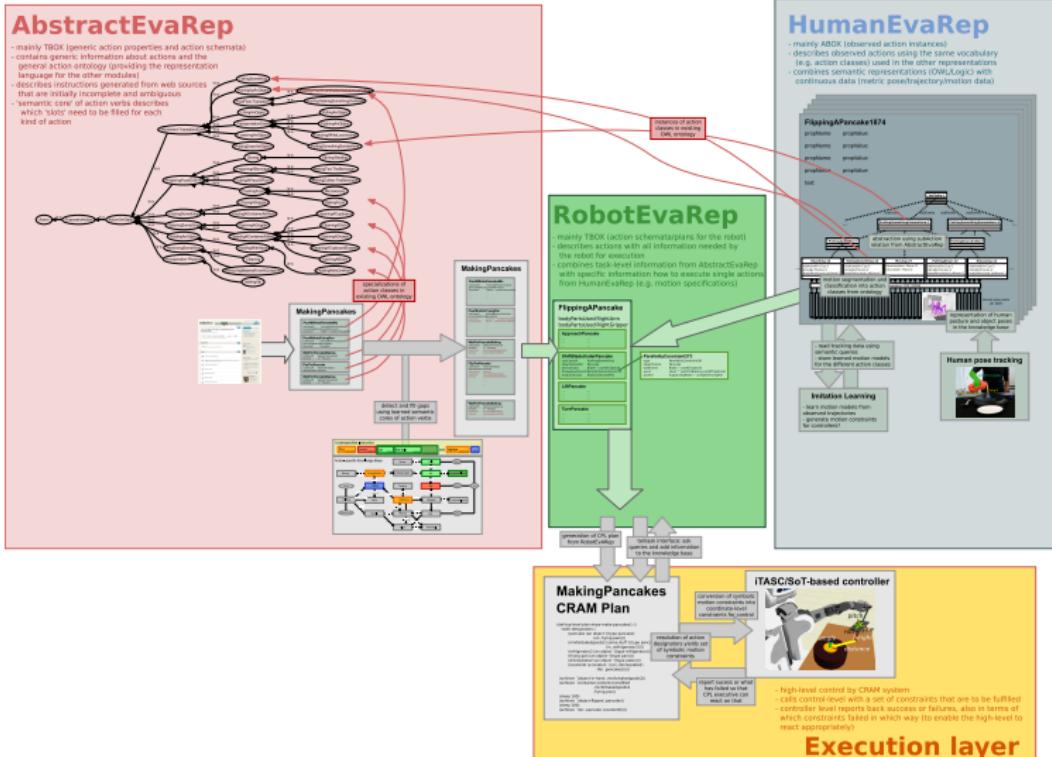
Tutorial: Knowledge Representation in RoboHow

Moritz Tenorth, Daniel Nyga and Asil Bozcuoğlu
Intelligent Autonomous Systems Group
Universität Bremen

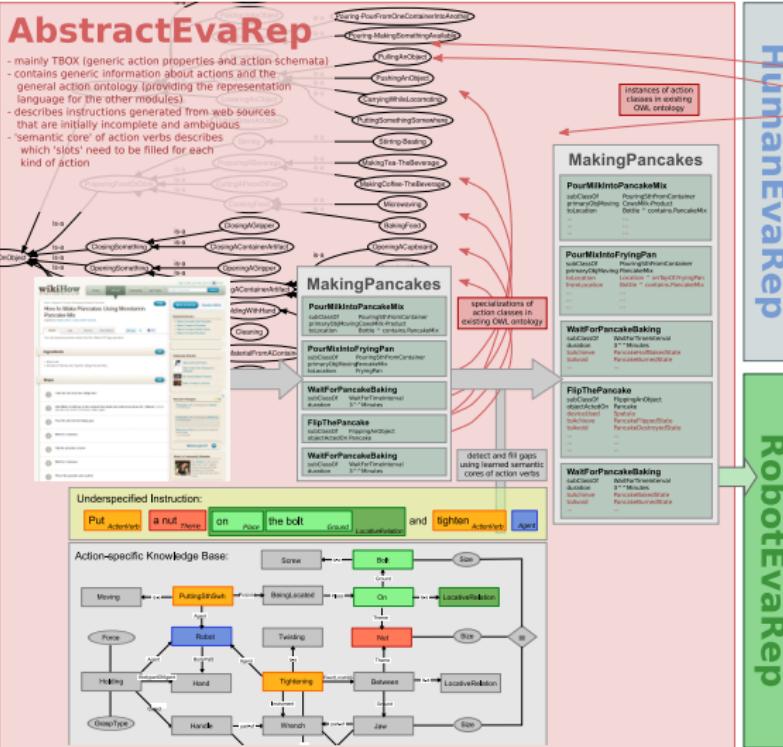
February 18, 2013



Representations in RoboHow



AbstractEvaRep



Tutorial: Knowledge Representation in RoboHow

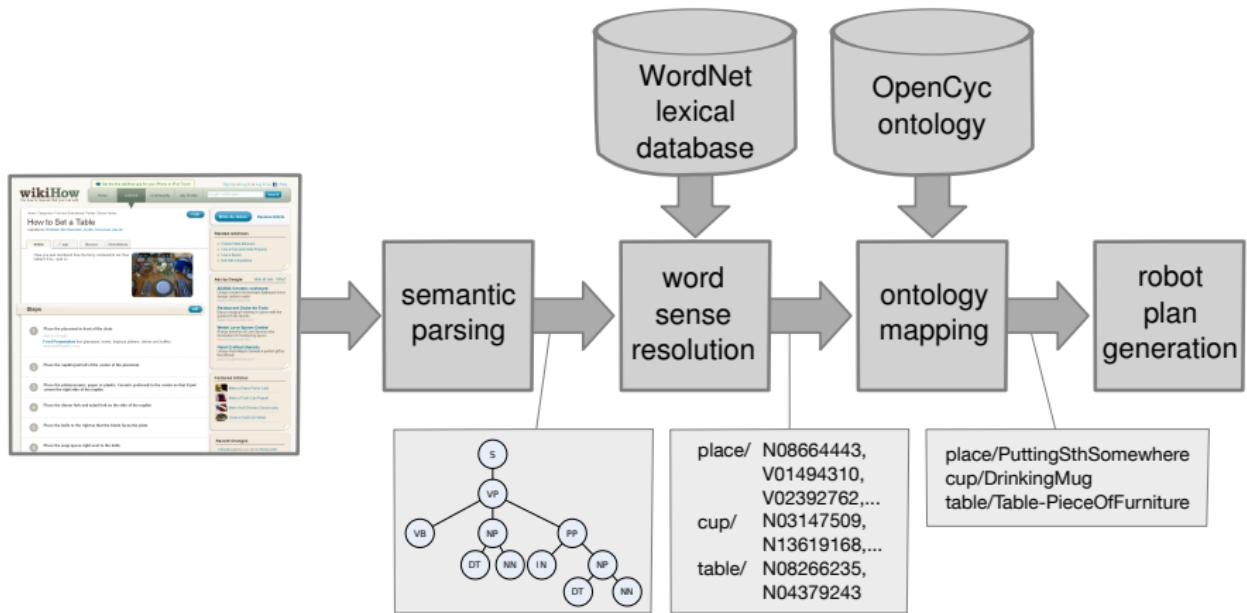
Moritz Tenorth, Daniel Nyga and Asil Bozcuoglu



AbstractEvaRep

- ▶ **Inputs:**
 - ▶ Instruction texts from the Web
 - ▶ Learned action-verb specific knowledge bases
- ▶ **Output:**
 - ▶ Formally described task specifications that contain information from instruction plus inferred statements
- ▶ **Consumers:**
 - ▶ **All:** Vocabulary for talking about actions
 - ▶ **HumanEvARep:** Abstract description of what is to be expected in the observations
 - ▶ **RobotEvARep:** Composition of tasks from actions

Task instructions from the WWW



Background knowledge about action classes

- ▶ Generic action class definition, incl. super-classes and sub-actions

```
Class : PuttingSomethingSomewhere
SubClassOf:
    Movement-TranslationEvent
    TransportationEvent
    subAction some PickingUpAnObject
    subAction some CarryingWhileLocomoting
    subAction some PuttingDownAnObject
    orderingConstraints value SubEventOrdering1
    orderingConstraints value SubEventOrdering2
```

- ▶ Specification of sub-actions and ordering constraints

```
Individual : SubEventOrdering1
Types:
    PartialOrdering - Strict
Facts:
    occursBeforeInOrdering  PickingUpAnObject
    occursAfterInOrdering   CarryingWhileLocomoting
```

Composing actions to tasks

Class: SetATable

Annotations: label "set a table"

SubClassOf: Action

EquivalentTo:

subAction some PutPlaceMatInFrontOfChair

subAction some PutPlateInCenterOfPlaceMat

subAction some PutKnifeRightOfPlate

subAction some [...]

orderingConstraints value [...]

Class: PutPlaceMatInFrontOfChair

EquivalentTo:

PuttingSomethingSomewhere

objectActedOn value PlaceMat1

toLocation some Place1

Class: Place1

EquivalentTo:

inFrontOf—Generally some Chair—PieceOfFurniture

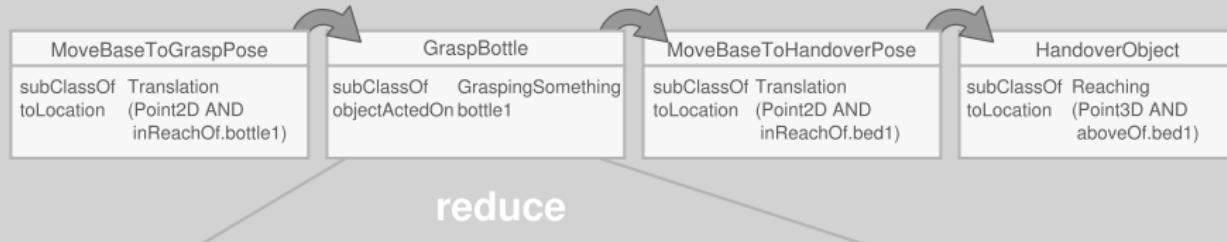
Individual: PlaceMat1

Types: PlaceMat

Task specification

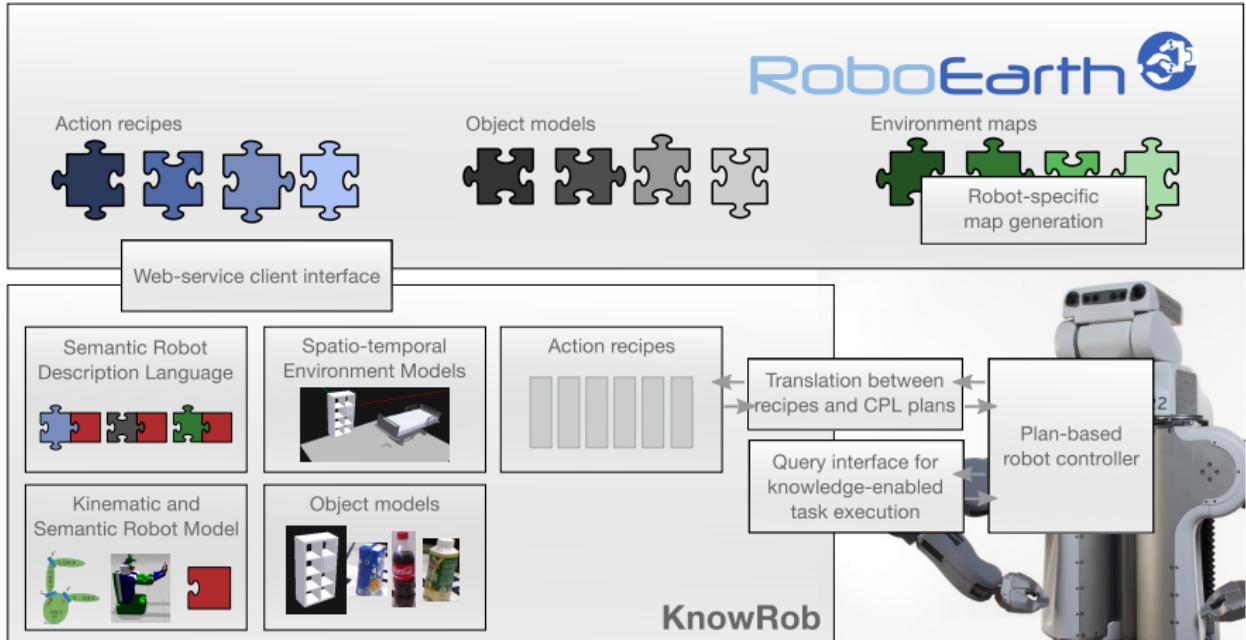
ServeADrink

dependsOnComponent ObjectRecognitionModel AND providesModelFor.Bottle
dependsOnComponent ObjectRecognitionModel AND providesModelFor.Bed
dependsOnComponent ObjectRecognitionModel AND providesModelFor.Cabinet



→ also used in RoboEarth as “action recipes”

RoboEarth: Towards a Wikipedia for robots



(see also http://www.knowrob.org/exchanging_information_via_RoboEarth)

Launching KnowRob

- ▶ KnowRob is composed of modules
- ▶ Dependencies are automatically loaded by rosprolog
- ▶ General syntax: `rosrun rosprolog rosprolog <pkg>`
 - ▶ starts a Prolog environment
 - ▶ checks for dependencies of `<pkg>`
 - ▶ locates them on the disk
 - ▶ loads the prolog/init.pl file for each of these packages

Prolog syntax

- ▶ Predicate names and constants start with a lowercase letter or are inside single quotes
- ▶ Variables start with an uppercase letter
- ▶ Formulas end with a full stop '.'
- ▶ Stepping through all results of a query with ';'
- ▶ Logical operators:

| Operator | Prolog equivalent | Example |
|----------|-------------------|------------------------------------|
| \wedge | , | type(A, 'Cup'), on(A, T). |
| \vee | ; | type(A, 'Food'); type(A, 'Drink'). |

Example

```
dairyproduct(milk).  
perishable(Stuff) :- storedIn(Stuff, Ref), refrigerator(Ref).
```

Tutorial: Import & visualize instructions

- ▶ Use the wikihow importer to translate instructions to OWL
- ▶ Input file: `comp_ehow/howtos/make_pancakes`

Terminal 1: Start Cyc knowledge base

```
$ rosmake comp_ehow  
$ rosrun opencyc opencyc.launch
```

Terminal 2: Import instructions

```
$ rosrun comp_ehow ehow_gui  
$ rosrun comp_ehow  
$ rosrun rosprolog rosprolog comp_ehow  
?- rdf_triple(knowrob:forCommand, A, 'make pancakes').  
A = 'http://ias.cs.tum.edu/kb/ehow_input.owl#MakePancakes'.
```

(see also http://www.ros.org/wiki/comp_ehow)

Tutorial: Import & visualize instructions

- ▶ Alternative: download resulting OWL file from
<http://www.knowrob.org/kb/make-pancakes.owl>

Terminal 2: Load instructions from file

```
$ roscd comp_ehow
$ rosrun rosprolog rosprolog comp_ehow
?- owl_parse('/path/to/make-pancakes.owl, false, false, true).
true.
```

- ▶ Visualize imported instructions

Terminal 2: Visualize imported plan

```
?- register_ros_package(mod_vis).
?- planvis_create(_).
?- planvis_load(ehow:'MakePancakes', _).
```

Tutorial: Query for instruction properties

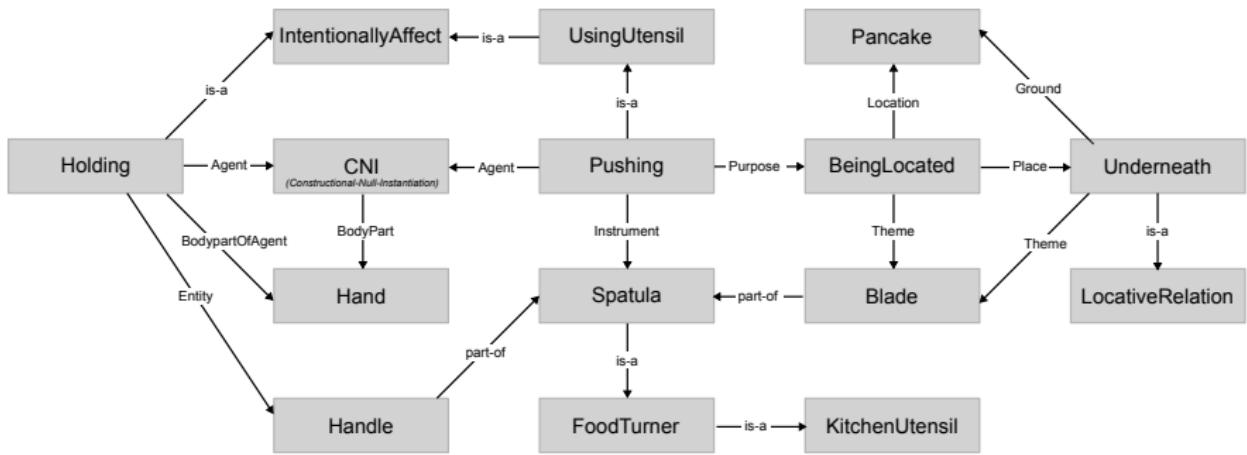
Queries for action properties

```
?- plan_subevents('ehow:MakePancakes', Sub).  
Sub = ['http://ias.cs.tum.edu/kb/ehow_input.owl#TakingSomething1',  
       'http://ias.cs.tum.edu/kb/ehow_input.owl#Incorporation-Physical2',  
       'http://ias.cs.tum.edu/kb/ehow_input.owl#FluidFlow-Translation3',  
       'http://ias.cs.tum.edu/kb/ehow_input.owl#Waiting4', ...].  
  
?- class_properties('ehow:TakingSomething1', P, O).  
P = 'http://ias.cs.tum.edu/kb/knowrob.owl#fromLocation',  
O = 'http://ias.cs.tum.edu/kb/knowrob.owl#Refrigerator' ;  
P = 'http://ias.cs.tum.edu/kb/knowrob.owl#objectActedOn',  
O = 'http://ias.cs.tum.edu/kb/knowrob.owl#MixForBakedGoods' ;  
  
?- rdf_has('ehow:TakingSomething1', rdfs:label, L).  
L = literal(type(xsd:string,'take the mix from the refrigerator')).
```

Action Intelligence

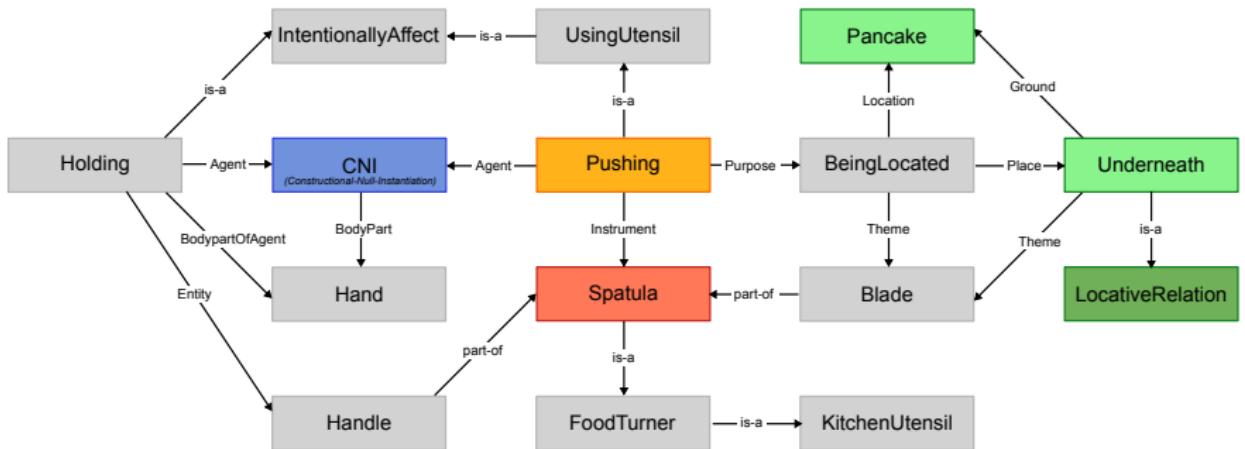
Action Specification

Action Core: Set of inter- and intraconceptual relations that constitute an abstract event type, assigning an **action role** to each entity that is affected by an action verb.



Action Intelligence

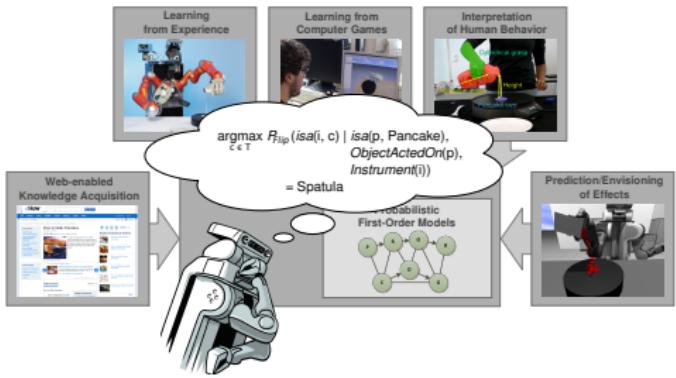
Action Core



PRAC – Probabilistic Robot Action Cores

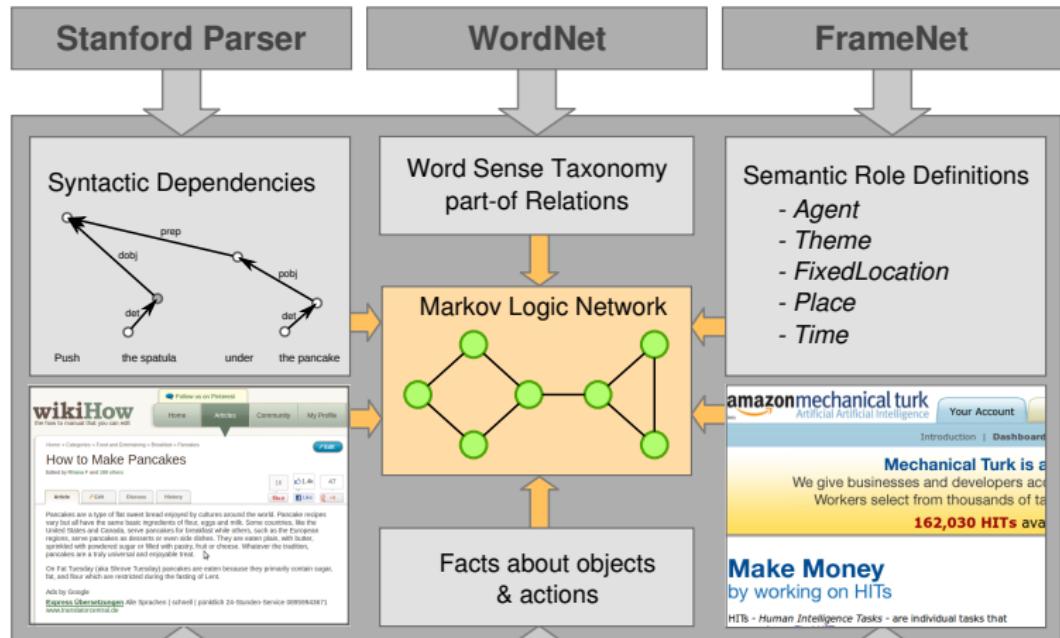
“Flip the pancake!”

- ▶ probabilistic first-order models of actions
- ▶ generic event patterns that recur frequently in everyday activities
- ▶ model joint distribution on action parameterizations on a symbolic level
 - ~~ perform **word sense disambiguation** and **role labeling** simultaneously
- ▶ enable to infer what is **needed** from what is **given** in an instruction
 - ~~ action completion



- ▶ incorporate class taxonomy for high generalization performance
- ▶ can be learned from **natural-language** recipes, **observations** of humans or **computer games**

Action-verb Knowledge Acquisition & Representation



Tutorial: Infer additional action properties

Terminal 1: Start PRAC service

```
$ rosrun rosprac rospracinfer.sh
```

Terminal 2: Queries to PRAC from KnowRob

```
?- register_ros_package(rosprac).  
?- infer_roles_for_actionclass(ehow:'FlippingAnObject5', Roles).  
Roles = [['Theme', 'pancake.n.01'],  
          ['ActionVerb', 'flip.v.08'],  
          ['Instrument', 'spatula.n.01']].
```

Terminal 1: Output of PRAC inference

Missing Roles:

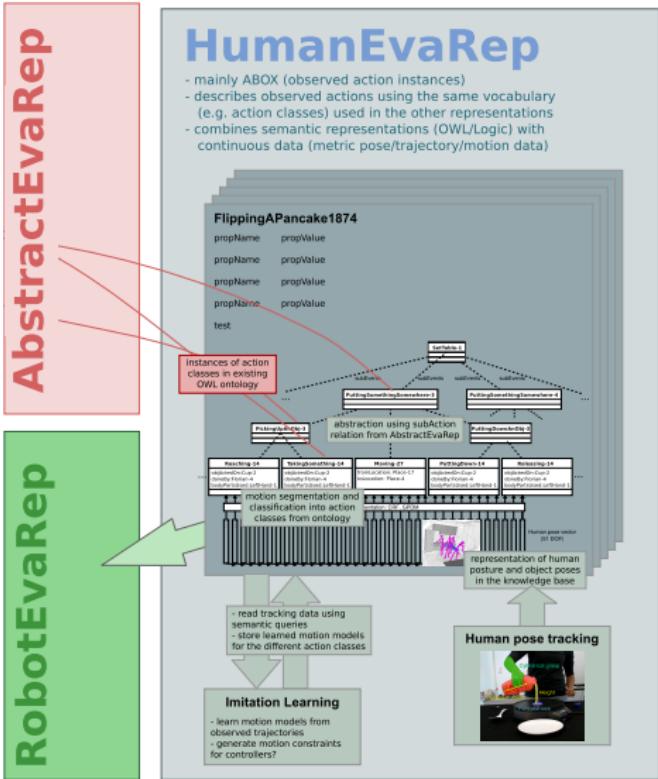
Instrument -- The tool that is used to perform the flipping action.

```
spatula.n.01 [||||||||||||||||||]  
tongs.n.01  [||||||||||||||||||]  
[...]  
pancake.n.01 [||||]
```

Discussion

- ▶ Abstract, formal action representation
- ▶ OWL descriptions as common 'interlingua' for RoboHow
- ▶ Can be exchanged via RoboEarth with other robots
- ▶ Some parts are still work in progress:
 - ▶ Natural-language sentences are analyzed twice:
→ integrate ehow-importer with PRAC
 - ▶ PRAC outputs WordNet concepts
→ Identifiers are to be integrated with KnowRob

HumanEvaRep



HumanEvARep

- ▶ **Input:** Tracked motions of hands and objects (FORTH)
- ▶ **Output:** Semantically annotated database of human observations
- ▶ **Consumers:**
 - ▶ Imitation learning: Read structured observations of activities as training data
 - ▶ Robot executive: Read examples of action parameters and their effects in observed situations

Semantic models of human activities

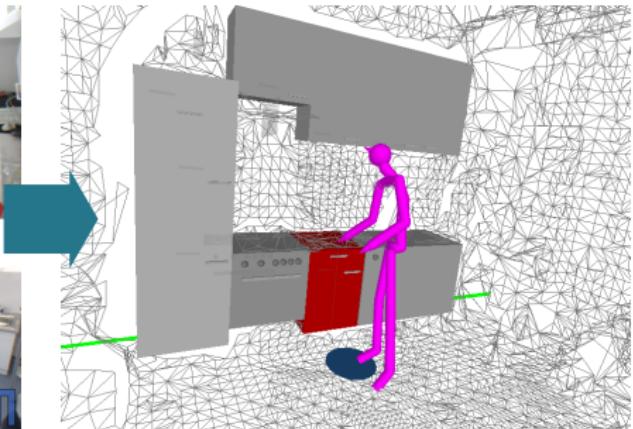
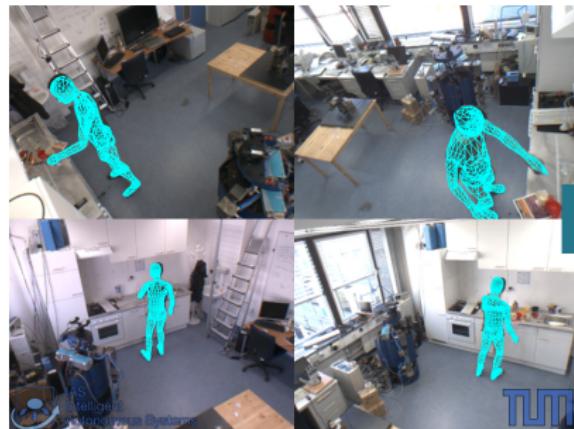
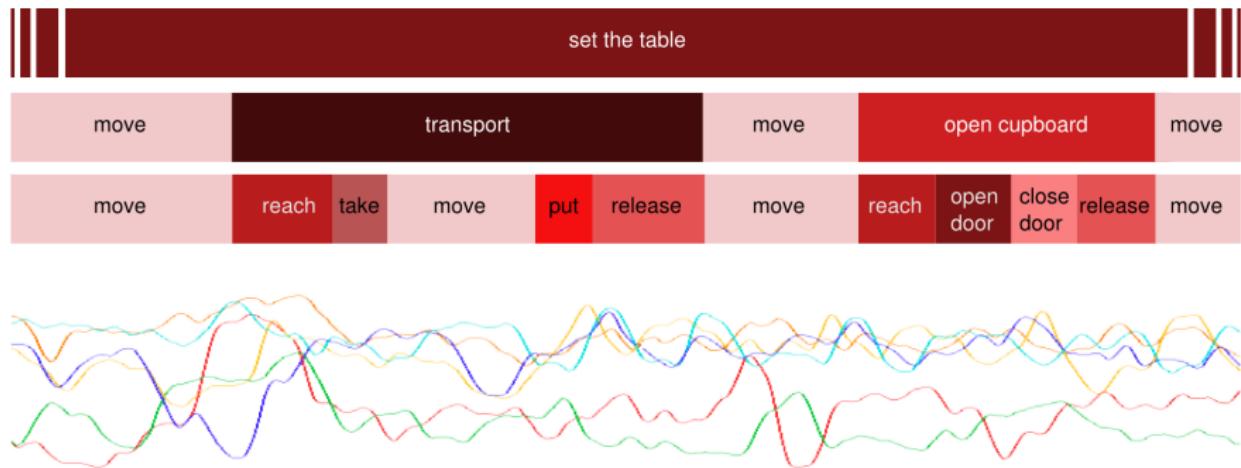


Image courtesy of Jan Bandouch

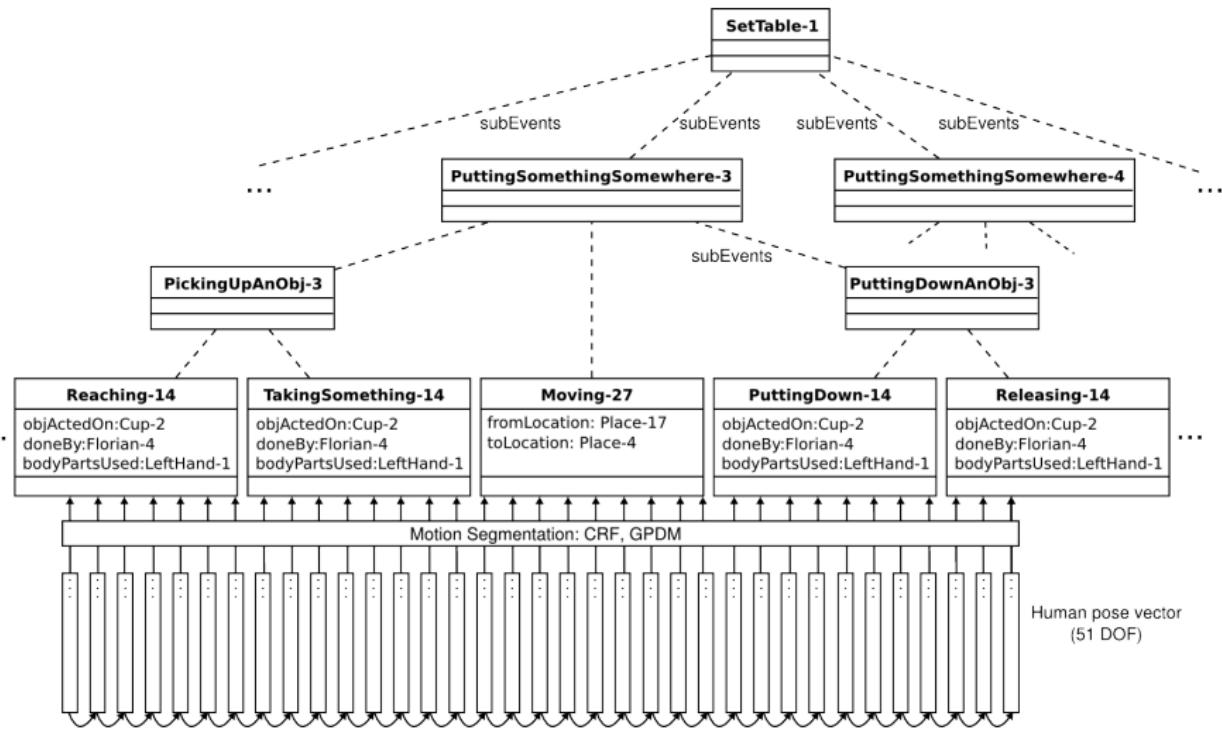
- ▶ Create knowledge bases from observations of human activities
- ▶ Segmentation, classification, formal representation

Segmenting and abstracting observations

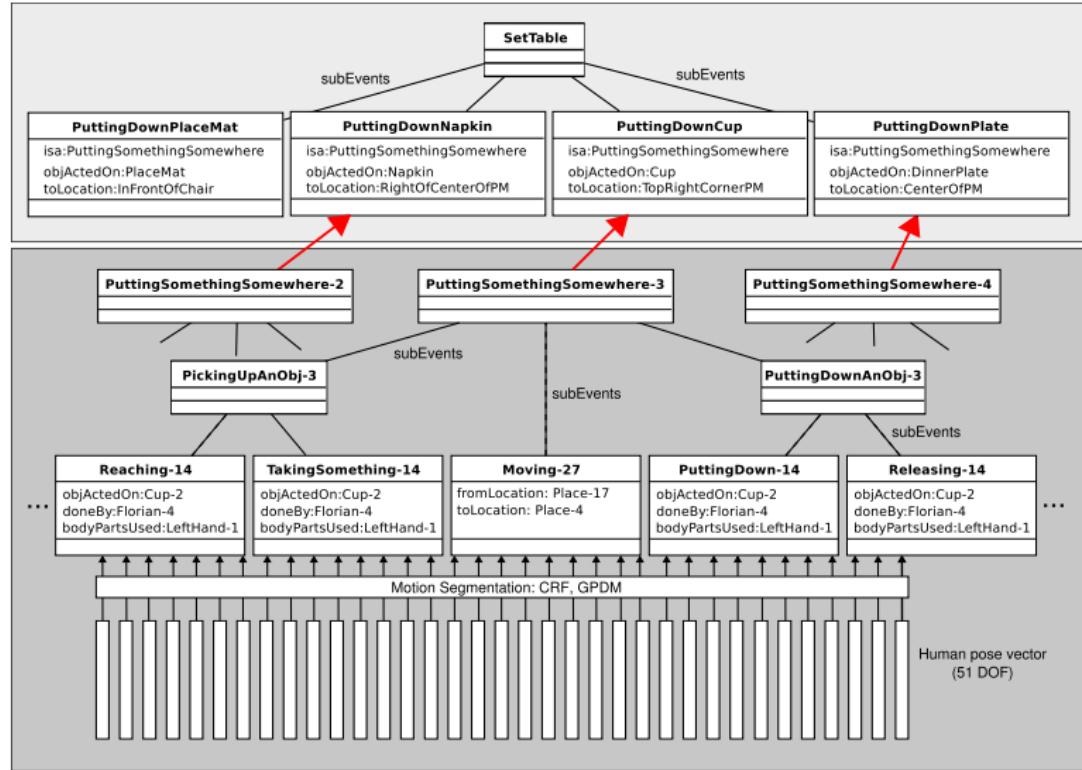


- ▶ Motion segmentation, e.g. based on Conditional Random Fields
- ▶ Representation of segments as instances of the resp. classes in KB
- ▶ Automatic abstraction using knowledge about action composition

Hierarchical semantic action models



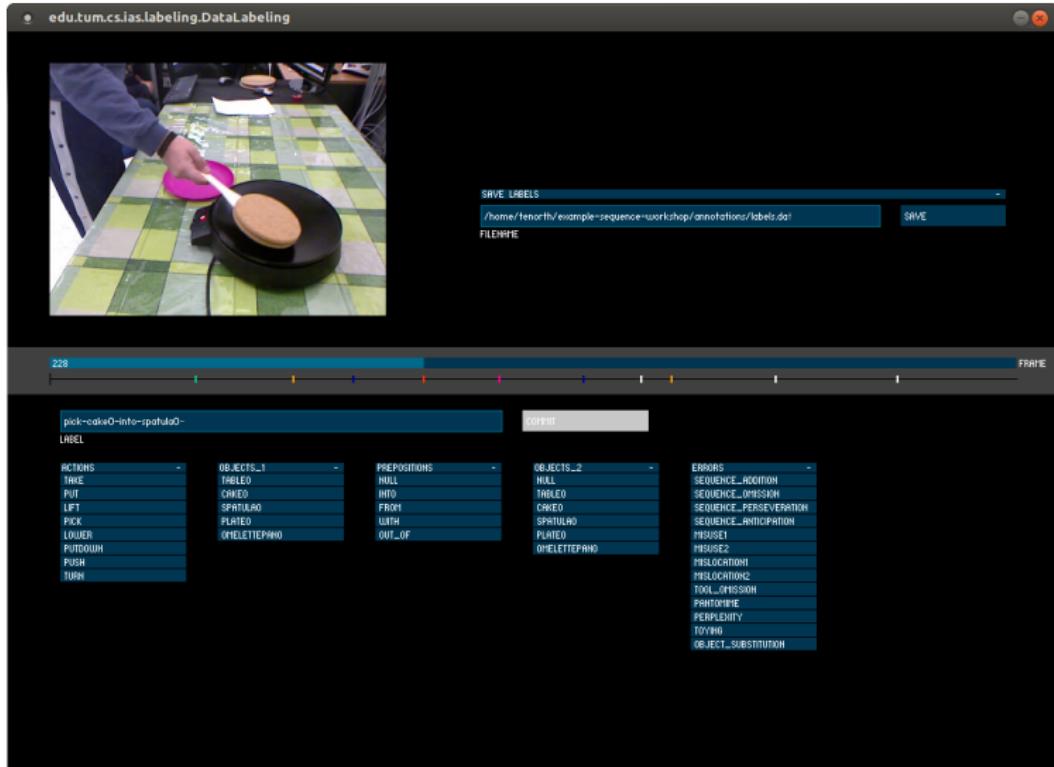
Aligning observations with instructions



Semantic action annotations

- ▶ Annotations of the observed motions required for interpretation:
 - ▶ Segmentation: start and end of motions
 - ▶ Semantic labels: class in OWL ontology
 - ▶ Action properties: objects, locations, tools
- ▶ Annotations represented as instances of action classes + properties in the knowledge base → exported as OWL file
- ▶ Class – instance relation to AbstractEvaRep
- ▶ For now: Manually created annotations using labeling tool

Labeling tool



Tutorial: Action annotation and OWL export

- ▶ Download and extract test data linked at
<http://robohow.eu/meetings/second-integration-workshop/knowrob-tutorial>
- ▶ Start the labeling tool and load [video/video5.mp4](#)
- ▶ Inspect the already created labels with PgUp/PgDn
- ▶ Add labels for the rest of the sequence by setting markers at the end of a segment
- ▶ Click on save button to export sequence to OWL + other formats

(see also http://ias.in.tum.de/kb/wiki/index.php/Labeling_tools)

Tutorial: Action annotation and OWL export

- ▶ Load OWL file into KnowRob and ask queries about the actions

Queries for action properties

```
example-seq/annotations$ rosrun rosprolog rosprolog knowrob_actions

?- owl_parse('labels.owl', false, false, true).

?- owl_individual_of(A, knowrob:'PushingAnObject').
A = 'http://ias.cs.tum.edu/kb/knowrob.owl#PushingAnObject_6oZ5GLYT' .
A = 'http://ias.cs.tum.edu/kb/knowrob.owl#PushingAnObject_xqam2N0j' ;

?- owl_has($A, knowrob:nextAction, Next).
Next = 'http://ias.cs.tum.edu/kb/knowrob.owl#PickingUpAnObject_soTvLL5h' ;
Next = 'http://ias.cs.tum.edu/kb/knowrob.owl#TurningAnObject_EmNJkK0t' ;
```

Tutorial: Action annotation and OWL export

Queries for action properties

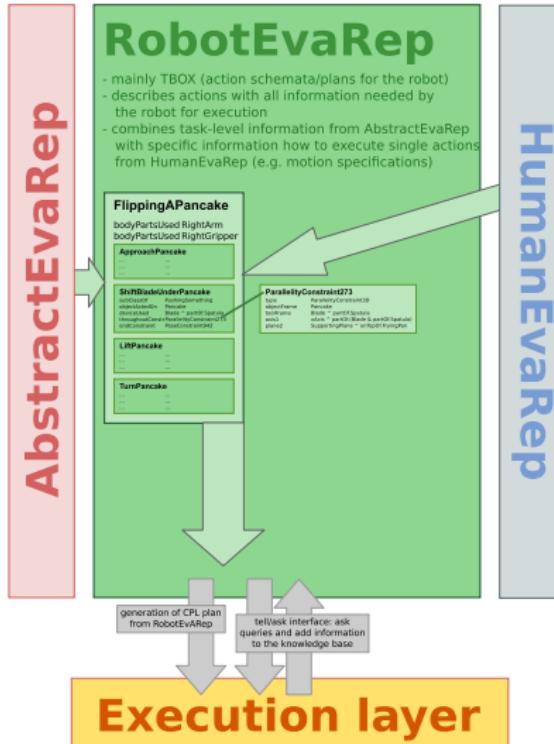
```
?- owl_has($A, P, O).
P = 'http://ias.cs.tum.edu/kb/knowrob.owl#primaryObjectMoving',
O = 'http://ias.cs.tum.edu/kb/knowrob.owl#spatula0' ;
P = 'http://ias.cs.tum.edu/kb/knowrob.owl#startTime',
O = 'http://ias.cs.tum.edu/kb/datalabel.owl#timepoint_150' ;
P = 'http://ias.cs.tum.edu/kb/datalabel.owl#objectActedOn',
O = 'http://ias.cs.tum.edu/kb/knowrob.owl#cake0' ;
P = 'http://ias.cs.tum.edu/kb/knowrob.owl#startTime',
O = 'http://ias.cs.tum.edu/kb/datalabel.owl#timepoint_150' ;

?- owl_has(A, knowrob:objectActedOn, O),
   owl_individual_of(O, knowrob:'Cake').
A = 'http://ias.cs.tum.edu/kb/knowrob.owl#PushingAnObject_6oZ5GLYT',
O = 'http://ias.cs.tum.edu/kb/knowrob.owl#cake0' ;
A = 'http://ias.cs.tum.edu/kb/knowrob.owl#PushingAnObject_6oZ5GLYT',
O = 'http://ias.cs.tum.edu/kb/knowrob.owl#cake0' ;
```

Discussion

- ▶ Formal representation of **observed** actions
- ▶ Episodic memory of observations as knowledge base for learning
- ▶ Instances of action classes in AbstractEvaRep → class–instance relation between representations
- ▶ Also here, parts are still work in progress:
 - ▶ Link formal models to tracked motions
 - ▶ Automatic segmentation and classification
 - ▶ More detailed annotations incl. objectActedOn, deviceUsed, primaryObjectMoving, fromLocation, toLocation
 - ▶ Representation of task context and multi-level annotations

RobotEvaRep



RobotEvARep

► **Inputs:**

- ▶ AbstractEvaRep gives the overall task structure
- ▶ HumanEvARep + imitation learning provide motion constraints

► **Output:**

- ▶ Executable plan in the CPL language:
bridge the gap between information in instructions and observations
and what is needed for execution
- ▶ Tell/Ask interface for action parameters

► **Consumers:**

- ▶ Action execution subsystem
(CRAM executive + constraint-/optimization-based controllers)

CRAM plan generation

```
(def-top-level-plan ehow-make-pancakes1 ()
  (with-designators (
    (pancake (an object '((type pancake)
                           (on ,frying-pan))))
    (mixforbakedgoods2 (some stuff '((type pancake-mix)
                                       (in ,refridgerator2))))
    (refridgerator2 (an object '((type refrigerator))))
    (frying-pan (an object '((type pan))))
    (dinnerplate2 (an object '((type plate))))
    (location0 (a location '((on ,dinnerplate2)
                             (for ,pancake2)))))

  (achieve '(object-in-hand ,mixforbakedgoods2))
  (achieve '(container-content-transfilled
             ,mixforbakedgoods2
             ,frying-pan))

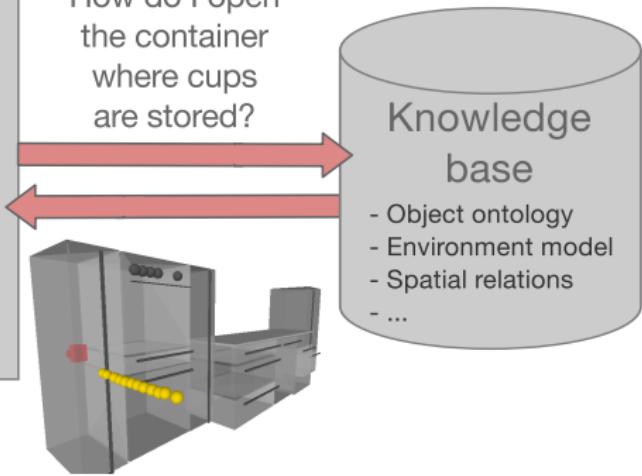
  (sleep 180)
  (achieve '(object-flipped ,pancake))
  (sleep 180)
  (achieve '(loc ,pancake ,location0)))))
```

Knowledge-based decision making

Robot control program

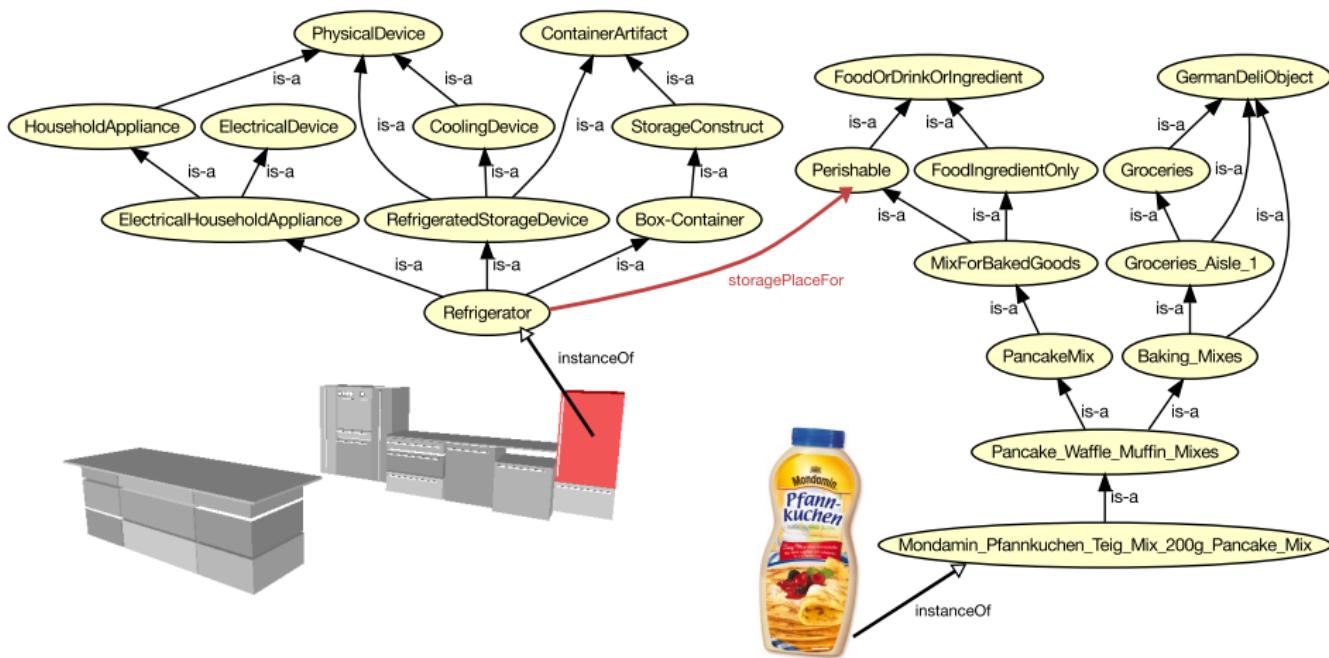
```
routine fetch (obj)
  loc ← likely-storage-loc(obj)
  if in-container(loc, container)
    then
      traj ← articulation(container)
      open-container(container, traj)
      pick-up(obj)
```

How do I open
the container
where cups
are stored?



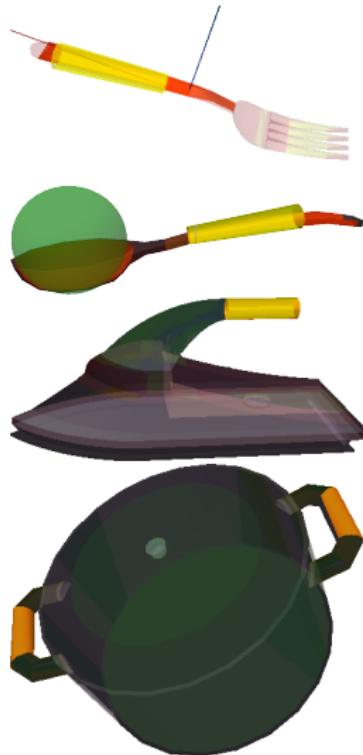
- ▶ Control decisions are formulated as inference tasks
- ▶ Separation of control flow and execution context
- ▶ Increased re-usability of robot plans

Inferring object locations in the environment



Semantic & geometric object models

- ▶ Translate abstract descriptions like “grasp the spatula at the handle” to geometric primitives
- ▶ Combined semantic and geometric object models:
 - ▶ **Geometric knowledge:** mesh segments, coordinates, motions
 - ▶ **Semantic knowledge:** types, properties, parts, functions
- ▶ Automatically extracted from CAD models by mesh segmentation



Querying KnowRob from your program

- ▶ Tell/ask interface via a ROS service
 - ▶ `tell`: add statements
 - ▶ `ask`: send a query
- ▶ Send queries as string like you would do from a console
- ▶ Result serialized using json
- ▶ Client libraries for C++, Python, Java, Lisp available

Example json_prolog launch file

```
<launch>
  <node name="knowrob" pkg="rosprolog" type="run_with_prolog_env"
    args="mod_vis $(find json_prolog)/bin/json_prolog" />
</launch>
```

(see also http://ros.org/wiki/json_prolog)

Querying KnowRob from Python

knowrob/json_prolog/examples/test_json_prolog.py

```
#usr/bin/env python

import roslib; roslib.load_manifest('json_prolog')
import rospy
import json_prolog

if __name__ == '__main__':

    rospy.init_node('test_json_prolog')
    prolog = json_prolog.Prolog()

    query = prolog.query("member(A, [1, 2, 3, 4]), B = ['x', A]")

    for sol in query.solutions():
        print 'Found solution.  A = %s, B = %s' % (sol['A'], sol['B'])

    query.finish()
```

Querying KnowRob from C++

knowrob/json_prolog/examples/test_json_prolog.cpp

```
#include <string>
#include <iostream>
#include <ros/ros.h>
#include <json_prolog/prolog.h>

using namespace std;
using namespace json_prolog;

int main(int argc, char *argv[]) {

    ros::init(argc, argv, "test_json_prolog");

    Prolog pl;

    PrologQueryProxy bdgs = pl.query("member(A, [1, 2, 3]), B = [x, A]");

    for(PrologQueryProxy::iterator it=bdgs.begin(); it != bdgs.end(); it++) {

        PrologBindings bdg = *it;
        cout << "Found solution: " << (bool)(it == bdgs.end()) << endl;
        cout << "A = " << bdg["A"] << endl;
        cout << "B = " << bdg["B"] << endl;
    }
    return 0;
}
```

Discussion

- ▶ Robot-specific action representation
- ▶ Class-level representation building upon AbstractEvaRep
- ▶ Adds information required for execution on the robot, like object recognition models or environment maps
- ▶ Current and future work:
 - ▶ Representation of motion constraints
 - ▶ Merge information from AbstractEvaRep and HumanEvARep into coherent robot plans
 - ▶ Enable the robot to autonomously identify and fill knowledge gaps in the instructions

Tutorial: Tell/Ask interface for action parameters

Depending on the remaining time and your interests, try out some of the following tutorials:

- ▶ http://www.knowrob.org/reasoning_about_objects
- ▶ http://www.knowrob.org/reasoning_about_actions
- ▶ http://www.knowrob.org/create_and_load_a_semantic_map
- ▶ [http://www.knowrob.org/
write_an_interface_to_your_perception_system](http://www.knowrob.org/write_an_interface_to_your_perception_system)

Thank you for your attention!

<http://www.knowrob.org>