# Controlling Robots with ASP and ROSoClingo

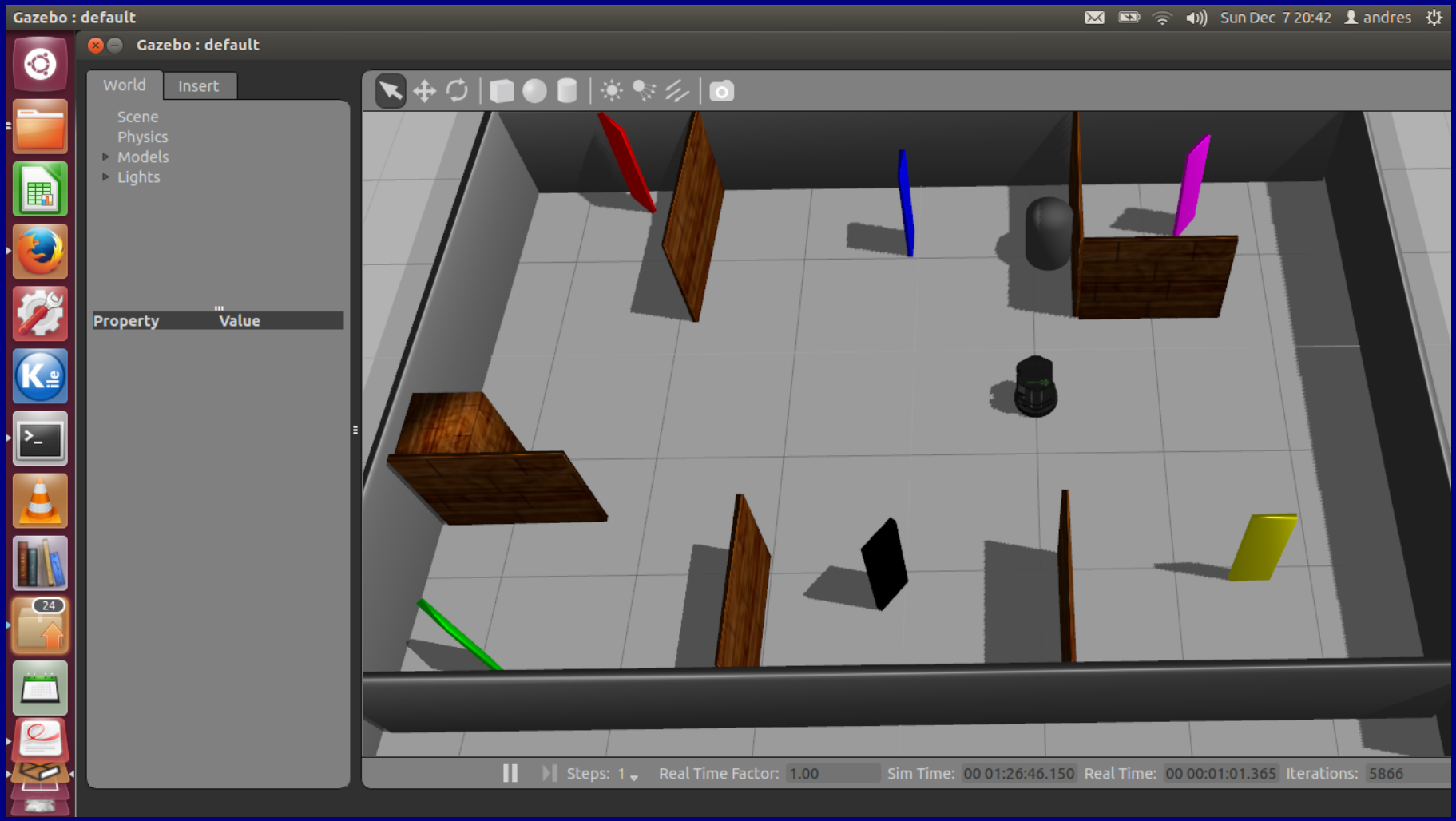
Benjamin Andres

University of Potsdam, Germany

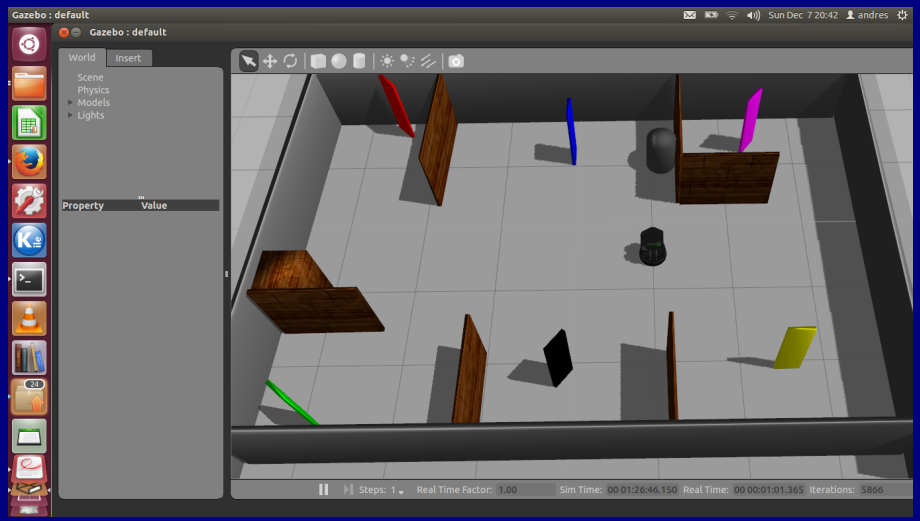
# Outline

- Problem Description
- Planning with ASP
- ROSoClingo
- Improvements
- Files and Running the System

# Problem Description

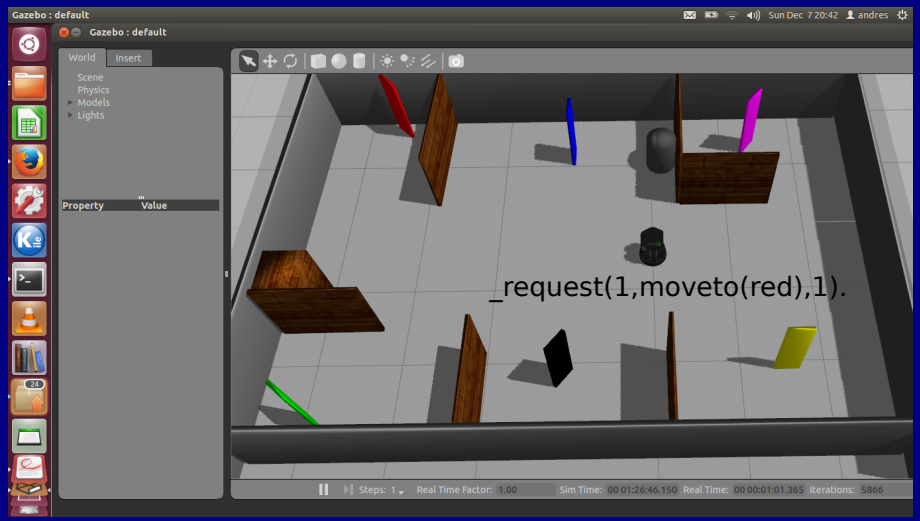


# Problem Description



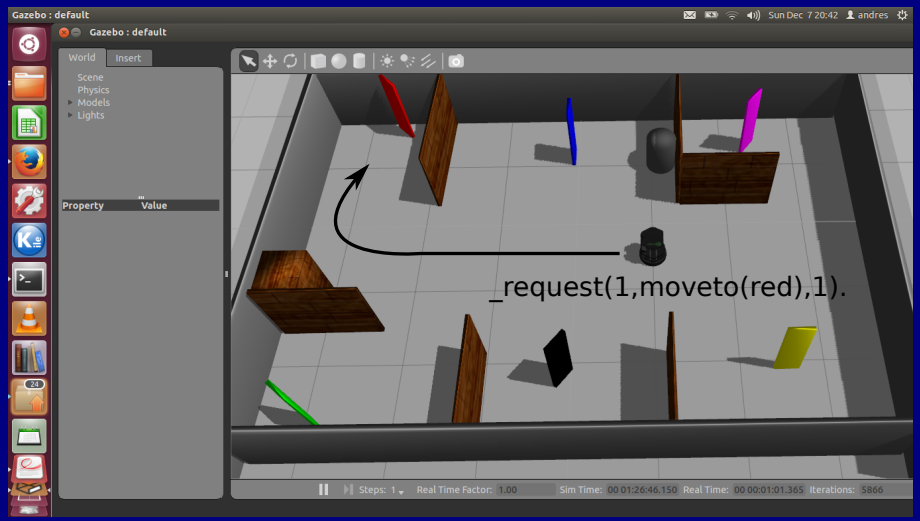

- receiving requests online
  move to requested content
  and give a signal
- requests may be canceled
- content of some QR-Codes is not known

# Problem Description



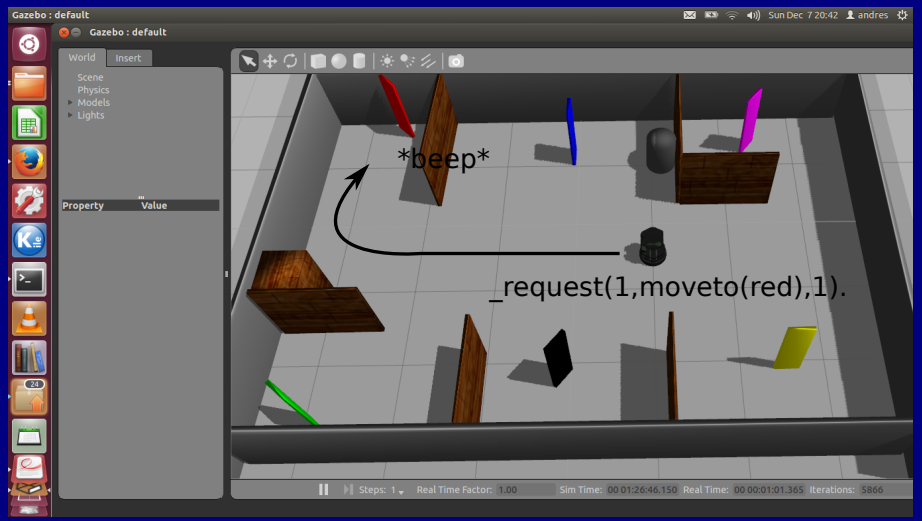

- receiving requests online
  - move to requested content
  - and give a signal
- requests may be canceled
- content of some QR-Codes is not known

# Problem Description



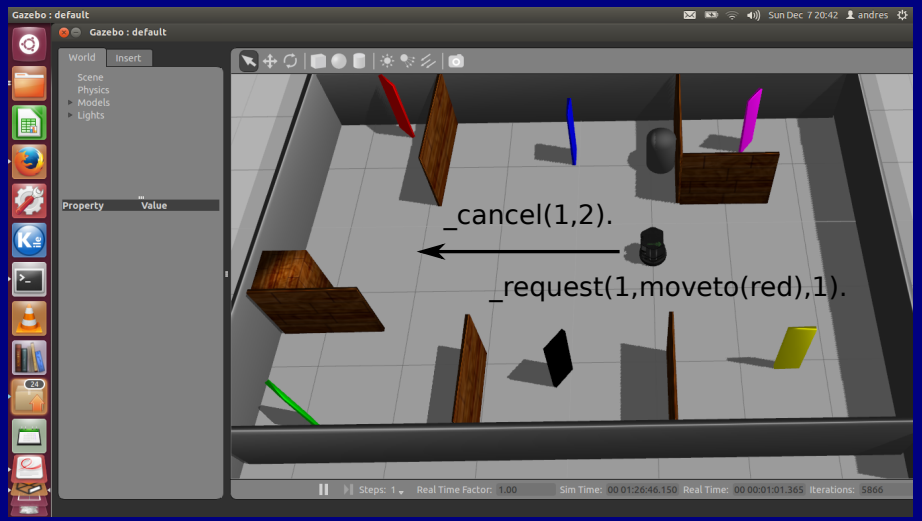

- receiving requests online
  move to requested content
  and give a signal
- requests may be canceled
- content of some QR-Codes is not known

# Problem Description



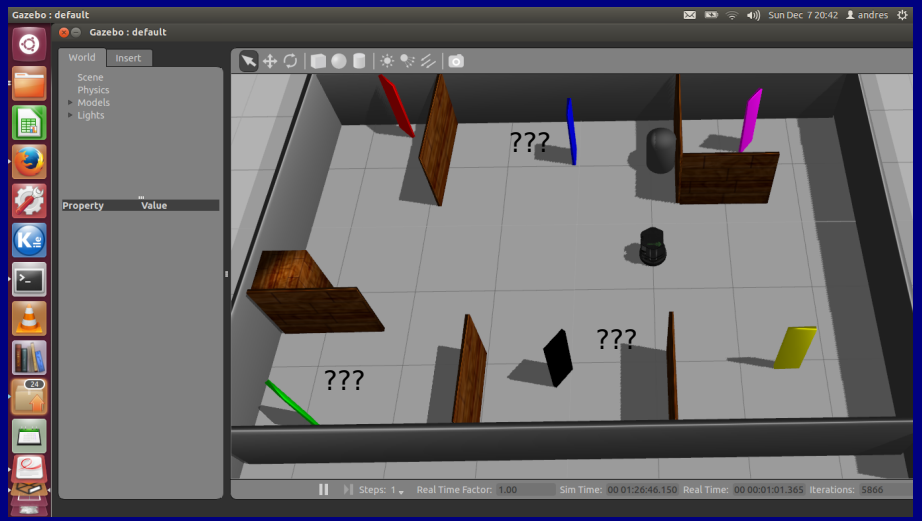

- receiving requests online
  move to requested content
  and give a signal
- requests may be canceled
- content of some QR-Codes is not known

# Problem Description




- receiving requests online
  move to requested content
  and give a signal
- requests may be canceled
- content of some QR-Codes is not known

# Problem Description




- receiving requests online
  move to requested content
  and give a signal
- requests may be canceled
- content of some QR-Codes is not known

# Problem Description

- use Answer Set Programming (ASP) to find a task plan
- use ROSoCLingo as an interface between ASP solver and ROS
- an ASP encoding is provided
- not very effective, e.g. no scanning
- do some improvements on your own...

# Planning with ASP

- the world is modelled as a transition system
- actions change the state of the world

```
_action(Robot,Action,t)
occurs((Robot,Acation),t)
```

- fluents describe the state of the world and change over time
- external events describes information from the outside

# Planning with ASP

- the world is modelled as a transition system
- actions change the state of the world

```
_action(Robot,Action,t)
occurs((Robot,Acation),t)
```

- fluents describe the state of the world and change over time
- external events describes information from the outside

# Planning with ASP

- the world is modelled as a transition system
- actions change the state of the world
- fluents describe the state of the world and change over time

  ```
  at(Robot,Location)
  holds(at(Robot,Location),t)
  ```

- external events describes information from the outside

# Planning with ASP

- the world is modelled as a transition system
- actions change the state of the world
- fluents describe the state of the world and change over time

```
at(Robot,Location)
holds(at(Robot,Location),t)
```

- external events describes information from the outside

# Planning with ASP

- the world is modelled as a transition system
- actions change the state of the world
- fluents describe the state of the world and change over time
- external events describes information from the outside

```
_request(ID,moveto(Content),t)
event(request,(ID,bring(O,P)),t)
```

# Planning with ASP

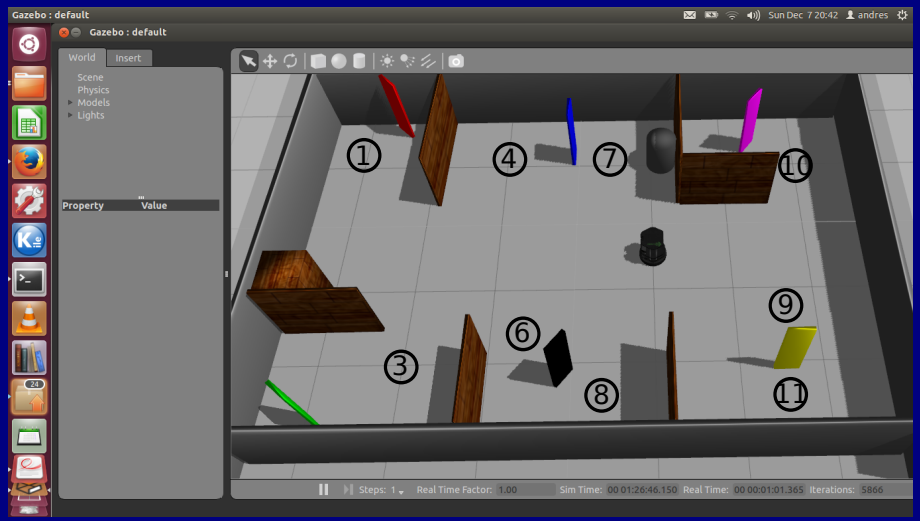
- the world is modelled as a transition system
- actions change the state of the world
- fluents describe the state of the world and change over time
- external events describes information from the outside

```
_request(ID,moveto(Content),t)
event(request,(ID,bring(O,P)),t)
```

# Planning with ASP

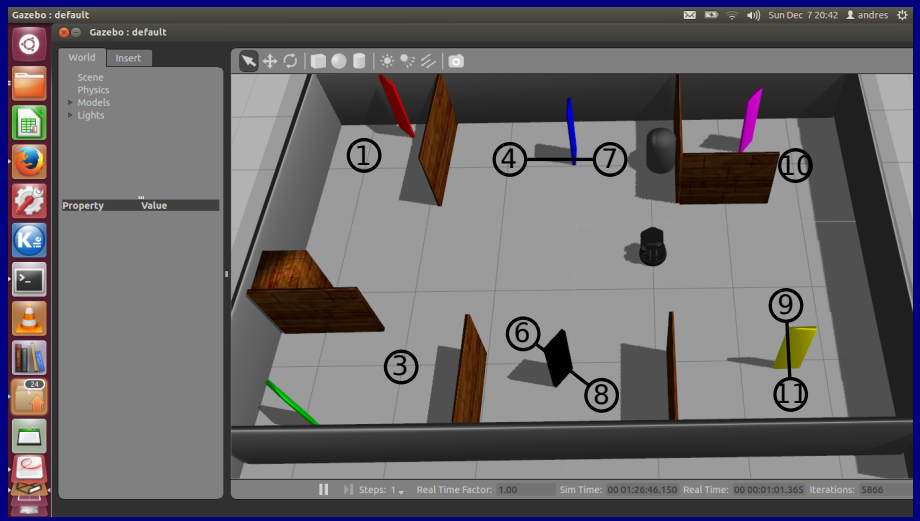
- the world is modelled as a transition system
- actions change the state of the world
- fluents describe the state of the world and change over time
- external events describes information from the outside
- static domain knowledge: environment.lp
- state transition: winterschool.lp

# Static Domain Knowledge



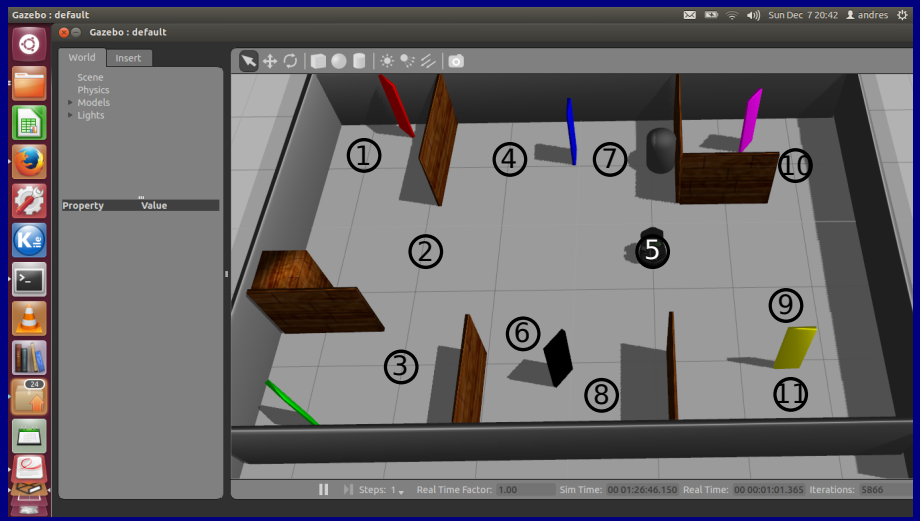

```
qr(1). ... qr(11). opposite(4,7).
opposite(6,8). opposite(9,11).
opposite(X,Y) :- opposite(Y,X).
location(1..11). connection( 1, 2).
... connection(9,11). connection(X,Y)
:- connection(Y,X). robot(turtlebot_1).
potential(move(L)) :- location(L).
potential(scan). potential(beep).
```

# Static Domain Knowledge



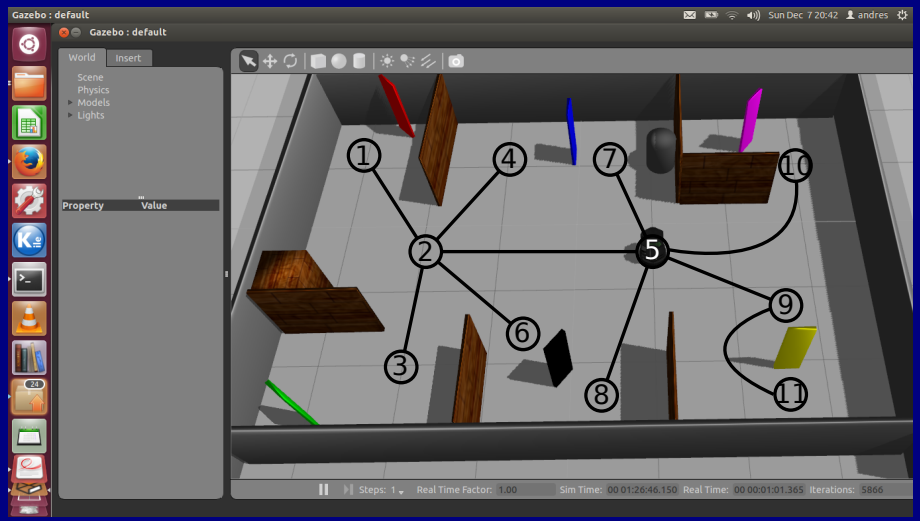

```
qr(1). ... qr(11). opposite(4,7).
opposite(6,8). opposite(9,11).
opposite(X,Y) :- opposite(Y,X).
location(1..11). connection( 1, 2).
... connection(9,11). connection(X,Y)
:- connection(Y,X). robot(turtlebot_1).
potential(move(L)) :- location(L).
potential(scan). potential(beep).
```

# Static Domain Knowledge



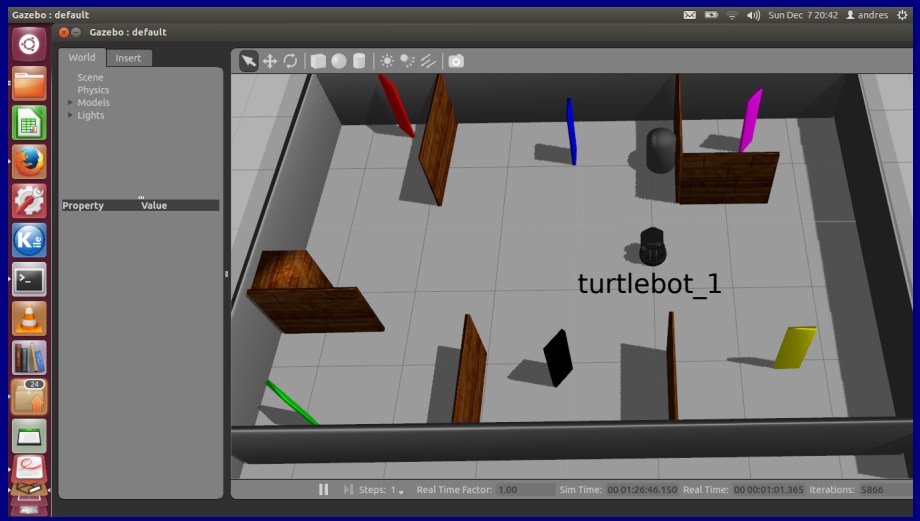

```
qr(1). ... qr(11). opposite(4,7).
opposite(6,8). opposite(9,11).
opposite(X,Y) :- opposite(Y,X).
location(1..11). connection( 1, 2).
... connection(9,11). connection(X,Y)
:- connection(Y,X). robot(turtlebot_1).
potential(move(L)) :- location(L).
potential(scan). potential(beep).
```

# Static Domain Knowledge



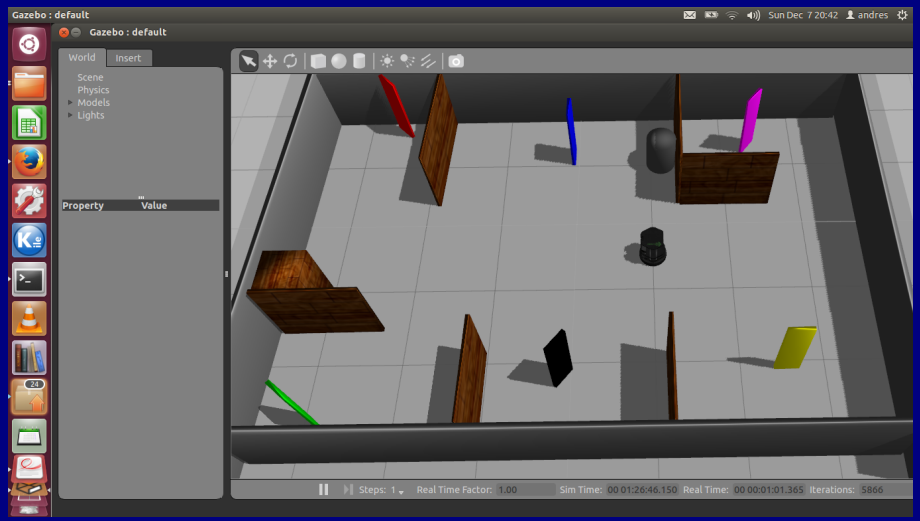

```
qr(1). ... qr(11). opposite(4,7).
opposite(6,8). opposite(9,11).
opposite(X,Y) :- opposite(Y,X).
location(1..11). connection( 1, 2).
... connection(9,11). connection(X,Y)
:- connection(Y,X). robot(turtlebot_1).
potential(move(L)) :- location(L).
potential(scan). potential(beep).
```

# Static Domain Knowledge




```
qr(1). ... qr(11). opposite(4,7).
opposite(6,8). opposite(9,11).
opposite(X,Y) :- opposite(Y,X).
location(1..11). connection( 1, 2).
... connection(9,11). connection(X,Y)
:- connection(Y,X). robot(turtlebot_1).
potential(move(L)) :- location(L).
potential(scan). potential(beep).
```

# Static Domain Knowledge




```
content(-1..6).
id(1..10).
```

# Initial Situation

- turtlebot_1 is at location 5 — `init(at(turtlebot_1,5)).`
- content of QR-Code  1 is 1 — `init(content( 1,1)).`
- content of QR-Code  3 is 2 — `init(content( 3,2)).`
- content of QR-Code 10 is 5 — `init(content(10,5)).`
- content of QR-Code 11 is 6 — `init(content(11,6)).`

# Actions

- actions change the state of the world
- time points are needed to order actions in the task plan
- each robot may execute one potential action per time point
- actions have preconditions that must be met

```
{_action(R,Action,t):potential(Action)}1 :- robot(R).
:- _action(R,Action,t), not possible(R,Action,t).
```

# Actions

- actions change the state of the world
- time points are needed to order actions in the task plan
- each robot may execute one potential action per time point
- actions have preconditions that must be met

```
{_action(R,Action,t):potential(Action)}1 :- robot(R).
:- _action(R,Action,t), not possible(R,Action,t).
```

# Actions

- actions change the state of the world
- time points are needed to order actions in the task plan
- each robot may execute one potential action per time point
- actions have preconditions that must be met

```
{_action(R,Action,t):potential(Action)}1 :- robot(R).
:- _action(R,Action,t), not possible(R,Action,t).
```

# Preconditions of Actions

- possible describes that the preconditions of an action are met
- WHENEVER a robot has just been at a location
  AND there is a connection to an other location
  THEN the robot may move to the other location
- WHENEVER holds(at(Robot,From),t-1)
  AND connection(From,To)
  THEN possible(Robot,move(To),t)
- ```
  possible(Robot,move(To),t) :-
    holds(at(Robot,From),t-1),
    connection(From,To).
  ```
  ```

# Preconditions of Actions

- possible describes that the preconditions of an action are met
- WHENEVER a robot has just been at a location
  AND there is a connection to an other location
  THEN the robot may move to the other location
- WHENEVER holds(at(Robot,From),t-1)
  AND connection(From,To)
  THEN possible(Robot,move(To),t)
- ```
  possible(Robot,move(To),t) :-
    holds(at(Robot,From),t-1),
    connection(From,To).
  ```
  ```

# Preconditions of Actions

- possible describes that the preconditions of an action are met
- WHENEVER a robot has just been at a location
  AND there is a connection to an other location
  THEN the robot may move to the other location
- WHENEVER holds(at(Robot,From),t-1)
  AND connection(From,To)
  THEN possible(Robot,move(To),t)
- ```
  possible(Robot,move(To),t) :-
    holds(at(Robot,From),t-1),
    connection(From,To).
  ```
  ```

# Effects of Actions

- each action has effects on the world
- fluents are used to express these effects
- a robot moving to a location
  CAUSES the robot to be there
- executes(Robot,move(Location),t)
  CAUSES holds(at(Robot,Location),t)
- ```
  holds(at(Robot,Location),t) :-
    executes(Robot,move(Location),t).
  ```
  ```

# Effects of Actions

- each action has effects on the world
- fluents are used to express these effects
- a robot moving to a location
  CAUSES the robot to be there
- executes(Robot,move(Location),t)
  CAUSES holds(at(Robot,Location),t)
- ```
  holds(at(Robot,Location),t) :-
    executes(Robot,move(Location),t).
  ```
  ```

# Effects of Actions

- each action has effects on the world
- fluents are used to express these effects
- a robot moving to a location
  CAUSES the robot to be there
- executes(Robot,move(Location),t)
  CAUSES holds(at(Robot,Location),t)
- ```
  holds(at(Robot,Location),t) :-
    executes(Robot,move(Location),t).
  ```
  ```

# The Frame Problem

- what fluents stay the same
- what fluents must change
- a robot moving
  CEASES the robot to be at the current location
- executes(Robot,move(_),t)
  CEASES holds(at(Robot,Location),t)

```
abnormal(at(Robot,Location),t) :-
  holds(at(Robot,Location),t-1),
  executes(Robot,move(_),t).
holds(Fluent,t) :-
  holds(Fluent,t-1),
  not abnormal(Fluent,t).
```

# The Frame Problem

- what fluents stay the same
- what fluents must change
- a robot moving
  CEASES the robot to be at the current location
- executes(Robot,move(_),t)
  CEASES holds(at(Robot,Location),t)

```
abnormal(at(Robot,Location),t) :-
  holds(at(Robot,Location),t-1),
  executes(Robot,move(_),t).
holds(Fluent,t) :-
  holds(Fluent,t-1),
  not abnormal(Fluent,t).
```

# The Frame Problem

- what fluents stay the same
- what fluents must change
- a robot moving
  CEASES the robot to be at the current location
- executes(Robot,move(_),t)
  CEASES holds(at(Robot,Location),t)

```
abnormal(at(Robot,Location),t) :-
  holds(at(Robot,Location),t-1),
  executes(Robot,move(_),t).
holds(Fluent,t) :-
  holds(Fluent,t-1),
  not abnormal(Fluent,t).
```

# The Frame Problem

- what fluents stay the same
- what fluents must change
- a robot moving
  CEASES the robot to be at the current location
- executes(Robot,move(_),t)
  CEASES holds(at(Robot,Location),t)

```
abnormal(at(Robot,Location),t) :-
  holds(at(Robot,Location),t-1),
  executes(Robot,move(_),t).
holds(Fluent,t) :-
  holds(Fluent,t-1),
  not abnormal(Fluent,t).
```

# External Events

- requests are issued and canceled online
- actions executed may have unforseen consequences
- information about the world may be gathered during execution
- atoms modified from the outside are declared external

```
#external _request(ID,moveto(Content),t) : id(ID),content(Content).
#external _cancel(ID,t)                  : id(ID),content(Content).
#external _value(Robot,success,t)        : robot(Robot).
#external _value(Robot,failure,t)        : robot(Robot).
#external _value(Robot,Content,t)        : robot(Robot), content(Content).
```

# External Events

- requests are issued and canceled online
- actions executed may have unforseen consequences
- information about the world may be gathered during execution
- atoms modified from the outside are declared external

```
#external _request(ID,moveto(Content),t) : id(ID),content(Content).
#external _cancel(ID,t)                  : id(ID),content(Content).
#external _value(Robot,success,t)        : robot(Robot).
#external _value(Robot,failure,t)        : robot(Robot).
#external _value(Robot,Content,t)        : robot(Robot), content(Content).
```

# Goal + Horizon

- goal: no open requests
- find task plan to fulfill the goal in the end
- need to identify the last time step
- external horizon(t) identifies the last time step at t

```
goal(t) :- not holds(request(_,_),t).
#external horizon(t).
:- not goal(t), horizon(t).
```

# Goal + Horizon

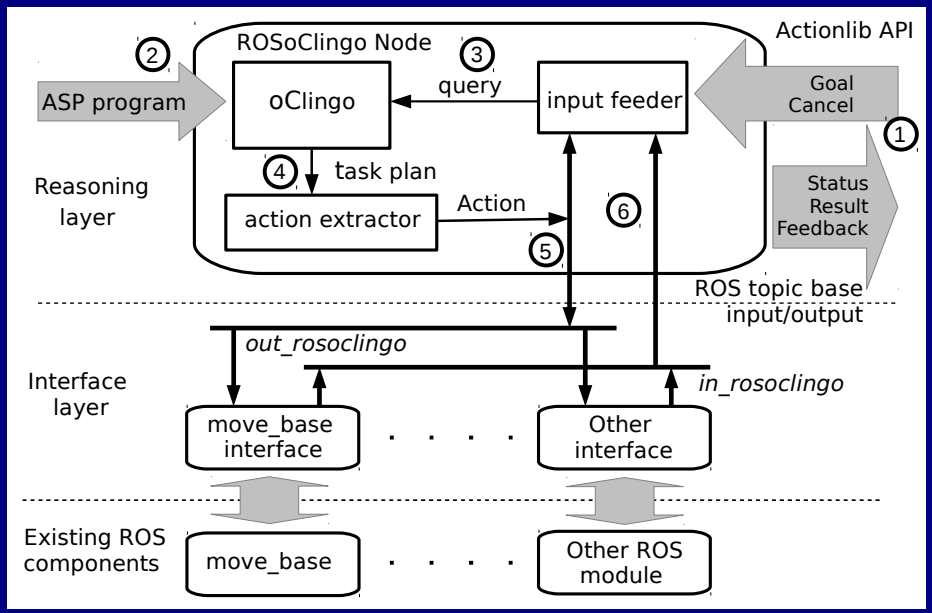
- goal: no open requests
- find task plan to fulfill the goal in the end
- need to identify the last time step
- external horizon(t) identifies the last time step at t

```
goal(t) :- not holds(request(_,_),t).
#external horizon(t).
:- not goal(t), horizon(t).
```

# ROSoClingo

- an interface between the ASP solver and ROS nodes
- functions as ROS action_lib and as controller for the ASP solver
- organizes the passage of time points in the ASP encoding
- transforms ROS messages into ASP facts
- and ASP task plans into ROS requests

# ROSoClingo




ROSoClingo Input:

_request(1,moveto(1),1).

---

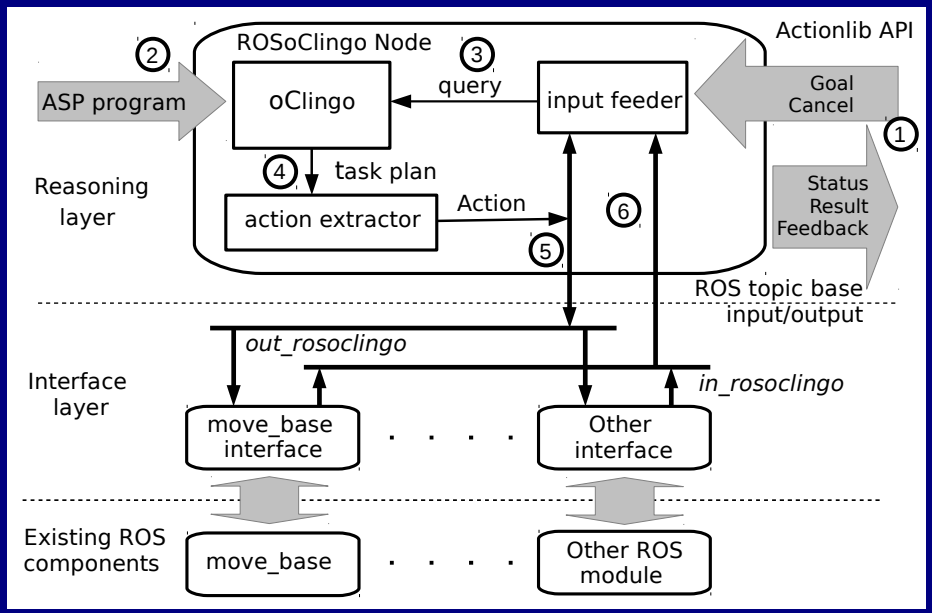
ROSoClingo Output:

**_action(turtlebot_1,move(2),1)**

_action(turtlebot_1,move(1),2)

_action(turtlebot_1,beep,3)

# ROSoClingo




ROSoClingo Input:

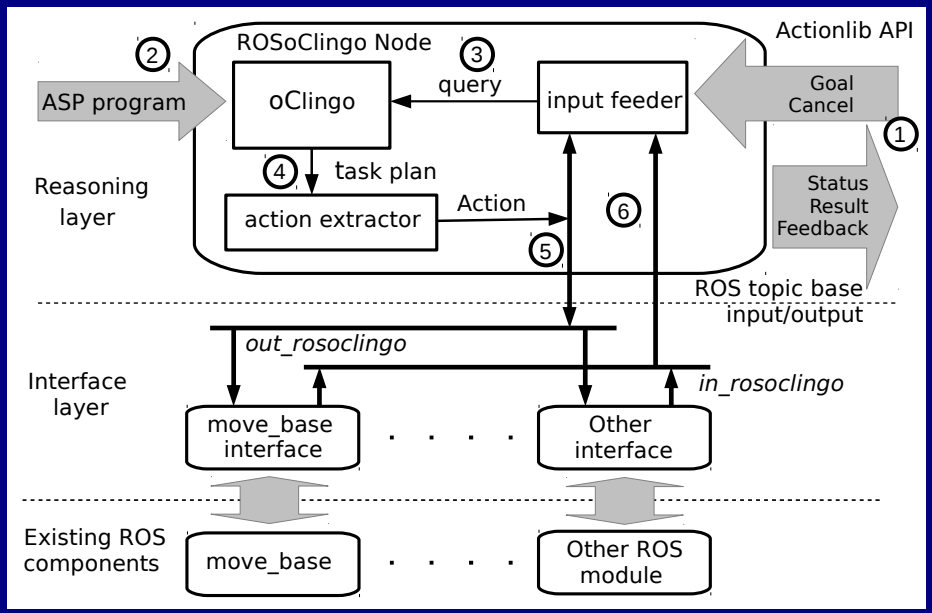
```
:- not _action(turtlebot_1,move(2),1).
_value(turtlebot_1,success,1).
```

---

ROSoClingo Output:

```
_action(turtlebot_1,move(2),1)
_action(turtlebot_1,move(1),2)
_action(turtlebot_1,beep,3)
```

# ROSoClingo



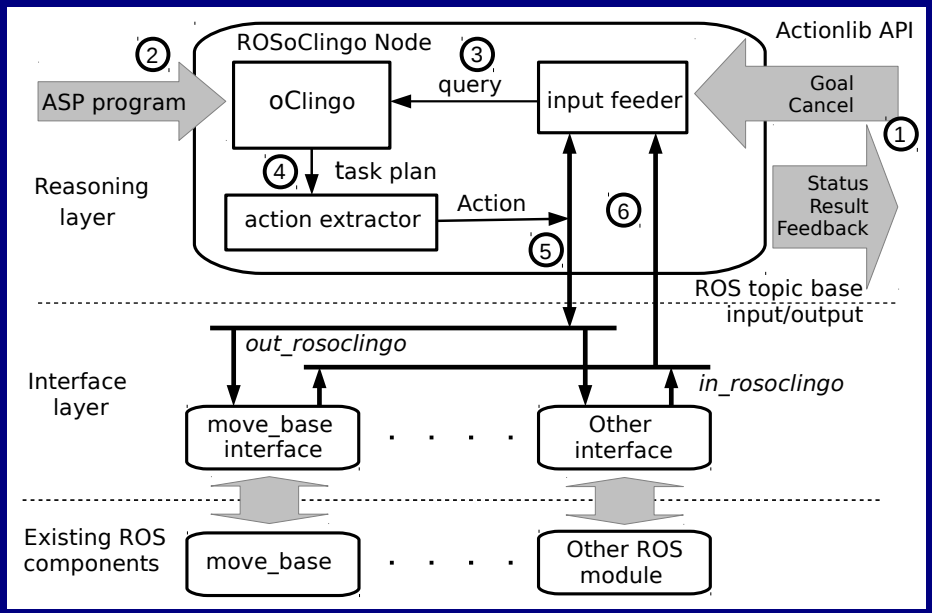

ROSoClingo Input:

```
:- not _action(turtlebot_1,move(1),2).
_value(turtlebot_1,success,2).
_cancel(1,3).
```

ROSoClingo Output:

```
_action(turtlebot_1,move(2),1)
_action(turtlebot_1,move(1),2)
```

# ROSoClingo




ROSoClingo Input:

_request(2,moveto(2),3).

---

ROSoClingo Output:

_action(turtlebot_1,move(2),1)

_action(turtlebot_1,move(1),2)

**_action(turtlebot_1,move(2),3)**

_action(turtlebot_1,move(3),4)

_action(turtlebot_1,beep,5)

# Improvements

- Cancellation of Requests
  - receiving an external event canceling a request CEASES the need to fulfill the request
- Indirect Knowledge

  WHENEVER the content of a QR-Code is known
  THEN the content of it's opposite is also known
- Scanning Actions
  - define when scanning is possible
  - define the result of scanning CAUSES + CEASES
  - define the goal anew

# Files and Running the System

Running in different terminals:

roscore

rosrun rosoclingo irun −−files: environment.lp winterschool.lp

- running ROSoClingo with no gazebo simulation:

  rosrun asp_session_files interface_winter_school_simulator.py

- running ROSoClingo with the better simulator

  roslaunch asp_session_files environment_asp.launch

  rosrun asp_session_files interface_winter_school.py

rosrun asp_session_files winterschool_manual.py

r 1

r 3