# Integrating ASP into ROS for Reasoning in Robots

Benjamin Andres[1], David Rajaratnam[2], Orkunt Sabuncu[1], and Torsten Schaub[1]

[1]University of Potsdam, Germany

[2]University of New South Wales, Australia

Potassco

# Outline

# Motivation

- High-level knowledge representation and reasoning capacities are vital to cognitive robotics
  - Answer Set Programming (ASP) is well suited for this
- Reactive behaviour is also expected from autonomous robots
  - *clingo* has reactive reasoning capacities

- Providing a highly capable reasoning framework for ROS by integrating the reactive answer set solver *clingo*

ROSoClingo
`potassco.sourceforge.net`

Potassco

# Motivation

- High-level knowledge representation and reasoning capacities are vital to cognitive robotics
  - Answer Set Programming (ASP) is well suited for this
- Reactive behaviour is also expected from autonomous robots
  - *clingo* has reactive reasoning capacities
- Providing a highly capable reasoning framework for ROS by integrating the reactive answer set solver *clingo*

ROSoClingo

`potassco.sourceforge.net`

## Motivation

- High-level knowledge representation and reasoning capacities are vital to cognitive robotics
  - Answer Set Programming (ASP) is well suited for this
- Reactive behaviour is also expected from autonomous robots
  - *clingo* has reactive reasoning capacities

- Providing a highly capable reasoning framework for ROS by integrating the reactive answer set solver *clingo*

ROSoClingo

`potassco.sourceforge.net`

# Motivation

- High-level knowledge representation and reasoning capacities are vital to cognitive robotics
  - Answer Set Programming (ASP) is well suited for this
- Reactive behaviour is also expected from autonomous robots
  - *clingo* has reactive reasoning capacities

- Providing a highly capable reasoning framework for ROS by integrating the reactive answer set solver *clingo*

ROSoClingo
potassco.sourceforge.net

Potassco

# Motivation

- High-level knowledge representation and reasoning capacities are vital to cognitive robotics
  - Answer Set Programming (ASP) is well suited for this
- Reactive behaviour is also expected from autonomous robots
  - *clingo* has reactive reasoning capacities

- Providing a highly capable reasoning framework for ROS by integrating the reactive answer set solver *clingo*

# Motivation

- High-level knowledge representation and reasoning capacities are vital to cognitive robotics
  - Answer Set Programming (ASP) is well suited for this
- Reactive behaviour is also expected from autonomous robots
  - *clingo* has reactive reasoning capacities
- Providing a highly capable reasoning framework for ROS by integrating the reactive answer set solver *clingo*

ROSoClingo

`potassco.sourceforge.net`

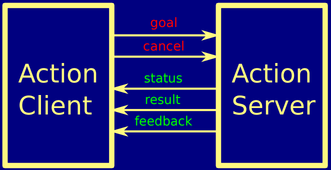
# Overview

# Robot Operating System (ROS)

- A middleware for robotic applications
  - Provides a communication framework for sending messages
  - Expandable through packages
  - Supports the Gazebo simulator
- ROS has a framework for complex behaviours: actionlib
  - client sets and cancels goals
  - server executes the goals and provides feedback

# Robot Operating System (ROS)

- A middleware for robotic applications
  - Provides a communication framework for sending messages
  - Expandable through packages
  - Supports the Gazebo simulator
- ROS has a framework for complex behaviours: actionlib
  - client sets and cancels goals
  - server executes the goals and provides feedback



# Overview



Potassco

# *clingo*

- *clingo* is an ASP solver providing an operative solving process for handling continuously changing logic programs
  - multi-shot solving rather than one-shot
  - explicit control knowledge via Lua/Python
  - avoids redundancies in grounding and solving in multi-shot setting
  - `import gringo` in Python
- *clingo* can react to incoming requests, environment changes, and new sensory information

# *clingo*

- *clingo* is an ASP solver providing an operative solving process for handling continuously changing logic programs
  - multi-shot solving rather than one-shot
  - explicit control knowledge via Lua/Python
  - avoids redundancies in grounding and solving in multi-shot setting
  - `import gringo` in Python
- *clingo* can react to incoming requests, environment changes, and new sensory information

# *clingo*

- *clingo* is an ASP solver providing an operative solving process for handling continuously changing logic programs
  - multi-shot solving rather than one-shot
  - explicit control knowledge via Lua/Python
  - avoids redundancies in grounding and solving in multi-shot setting
  - `import gringo` in Python

- *clingo* can react to incoming requests, environment changes, and new sensory information

# *clingo*

- *clingo* is an ASP solver providing an operative solving process for handling continuously changing logic programs
  - multi-shot solving rather than one-shot
  - explicit control knowledge via Lua/Python
  - avoids redundancies in grounding and solving in multi-shot setting
  - `import gringo` in Python
- *clingo* can react to incoming requests, environment changes, and new sensory information

Potassco

## *clingo*

- *clingo* is an ASP solver providing an operative solving process for handling continuously changing logic programs
  - multi-shot solving rather than one-shot
  - explicit control knowledge via Lua/Python
  - avoids redundancies in grounding and solving in multi-shot setting
  - `import gringo` in Python
- *clingo* can react to incoming requests, environment changes, and new sensory information

# *clingo*

- *clingo* is an ASP solver providing an operative solving process for handling continuously changing logic programs
  - multi-shot solving rather than one-shot
  - explicit control knowledge via Lua/Python
  - avoids redundancies in grounding and solving in multi-shot setting
  - `import gringo` in Python
- *clingo* can react to incoming requests, environment changes, and new sensory information

# Overview



Potassco

# Methodology

- Based on the general guidelines of representing dynamic domains and solving planning problems in ASP
  - Formalize the dynamic domain
  - Formalize the task as a planning problem
- Parametrizable subprograms
  - `base`: static knowledge, initial situation
  - `transition(t)`: causal laws, action preconditions, inertia
  - `query(t)`: goal condition
- Exogenous events are modelled by *clingo*'s external directives
- Execution failures are directly incorporated in the encoding

```
holds(at(R,W),t) :- do(R,go(W),t), not event(R,failure,t).
holds(blocked(W',W),t) :- do(R,go(W),t), event(R,failure,t),
holds(at(R,W'),t-1).
```

# Methodology

- Based on the general guidelines of representing dynamic domains and solving planning problems in ASP
  - Formalize the dynamic domain
  - Formalize the task as a planning problem
- Parametrizable subprograms
  - `base`: static knowledge, initial situation
  - `transition(t)`: causal laws, action preconditions, inertia
  - `query(t)`: goal condition
- Exogenous events are modelled by *clingo*'s external directives
- Execution failures are directly incorporated in the encoding

```
holds(at(R,W),t) :- do(R,go(W),t), not event(R,failure,t).
holds(blocked(W',W),t) :- do(R,go(W),t), event(R,failure,t),
holds(at(R,W'),t-1).
```

# Methodology

- Based on the general guidelines of representing dynamic domains and solving planning problems in ASP
  - Formalize the dynamic domain
  - Formalize the task as a planning problem
- Parametrizable subprograms
  - `base`: static knowledge, initial situation
  - `transition(t)`: causal laws, action preconditions, inertia
  - `query(t)`: goal condition
- Exogenous events are modelled by *clingo*'s external directives

- Execution failures are directly incorporated in the encoding

```
holds(at(R,W),t) :- do(R,go(W),t), not event(R,failure,t).
holds(blocked(W',W),t) :- do(R,go(W),t), event(R,failure,t),
holds(at(R,W'),t-1).
```

# Methodology

- Based on the general guidelines of representing dynamic domains and solving planning problems in ASP
  - Formalize the dynamic domain
  - Formalize the task as a planning problem
- Parametrizable subprograms
  - `base`: static knowledge, initial situation
  - `transition(t)`: causal laws, action preconditions, inertia
  - `query(t)`: goal condition
- Exogenous events are modelled by *clingo*'s external directives
- Execution failures are directly incorporated in the encoding

```
holds(at(R,W),t) :- do(R,go(W),t), not event(R,failure,t).
holds(blocked(W',W),t) :- do(R,go(W),t), event(R,failure,t),
holds(at(R,W'),t-1).
```

# Methodology

- Based on the general guidelines of representing dynamic domains and solving planning problems in ASP
  - Formalize the dynamic domain
  - Formalize the task as a planning problem
- Parametrizable subprograms
  - `base`: static knowledge, initial situation
  - `transition(t)`: causal laws, action preconditions, inertia
  - `query(t)`: goal condition
- Exogenous events are modelled by *clingo*'s external directives
- Execution failures are directly incorporated in the encoding

```
holds(at(R,W),t) :- do(R,go(W),t), not event(R,failure,t).
holds(blocked(W',W),t) :- do(R,go(W),t), event(R,failure,t),
holds(at(R,W'),t-1).
```

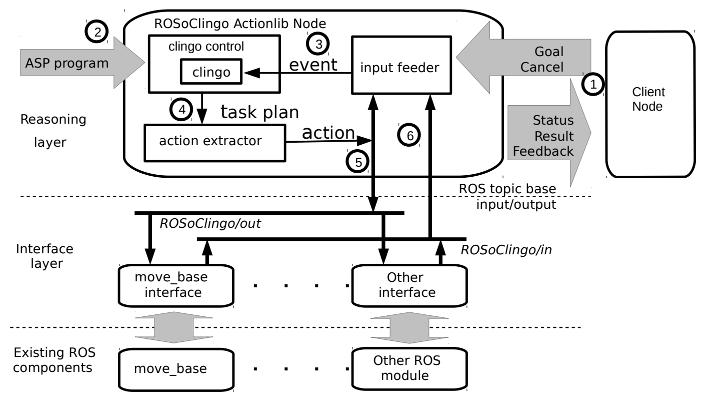
# Methodology

- Based on the general guidelines of representing dynamic domains and solving planning problems in ASP
  - Formalize the dynamic domain
  - Formalize the task as a planning problem
- Parametrizable subprograms
  - `base`: static knowledge, initial situation
  - `transition(t)`: causal laws, action preconditions, inertia
  - `query(t)`: goal condition
- Exogenous events are modelled by *clingo*'s external directives
- Execution failures are directly incorporated in the encoding

```
holds(at(R,W),t) :- do(R,go(W),t), not event(R,failure,t).
holds(blocked(W',W),t) :- do(R,go(W),t), event(R,failure,t),
holds(at(R,W'),t-1).
```

# Overview

# ROSoClingo

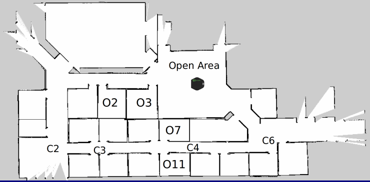

ROSoClingo Actionlib Node
2
ASP program
clingo control
clingo
3
event
input feeder
Goal
Cancel
1
Client
Node
4
task plan
Reasoning
layer
action extractor
action
6
5
Status
Result
Feedback
ROS topic base
input/output
ROSoClingo/out
ROSoClingo/in
Interface
layer
move_base
interface
Other
interface
Existing ROS
components
move_base
Other ROS
module

# Overview



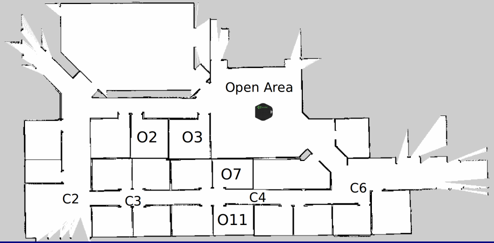
Potassco

# Mail Delivery Robot



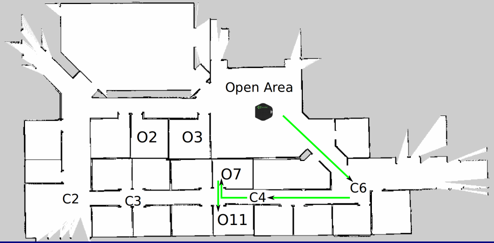

- Pick up and deliver packages
- Receive requests and cancellations
- Paths may be blocked
- Handling multiple requests at a time
- Gazebo simulation of an office environment

# Mail Delivery Robot




Input:
event(request(1),bring(o7,o11),1)
Plan:

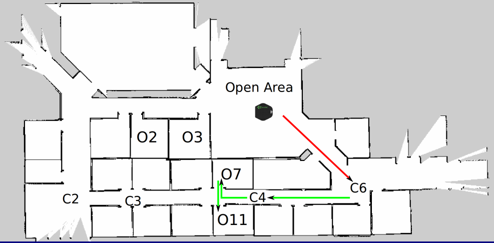
# Mail Delivery Robot




Input:

Plan:
do(robot,move(c6),1)
do(robot,move(c4),2)
do(robot,move(o7),3)
do(robot,pickup(request(1)),4)
do(robot,move(o11),5)
do(robot,deliver(deliver(1)),6)

# Mail Delivery Robot

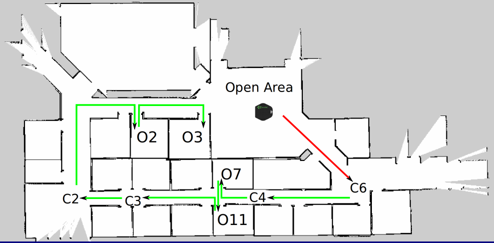



Input:
event(request(2),bring(o2,o3),1).
Plan:
do(robot,move(c6),1)

---

do(robot,move(c4),2)
do(robot,move(o7),3)
do(robot,pickup(request(1)),4)
do(robot,move(o11),5)
do(robot,deliver(deliver(1)),6)

# Mail Delivery Robot



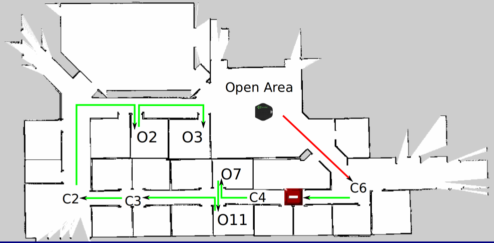

Input:

Plan:
do(robot,move(c6),1)

---

do(robot,move(c4),2)
do(robot,move(o7),3)
do(robot,pickup(request(1)),4)
do(robot,move(o11),5)
do(robot,deliver(request(1)),6)
do(robot,move(c3),7)
do(robot,move(c2),8)
do(robot,move(o2),9)
do(robot,pickup(request(2)),10)
do(robot,move(o3),11)
do(robot,deliver(request(2)),12)

# Mail Delivery Robot

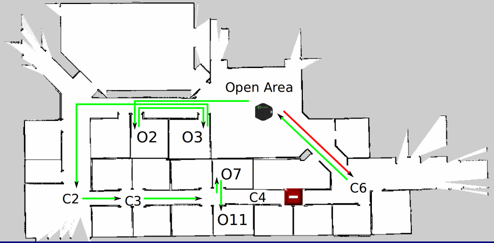



Input:
event(info,blocked(c6,c4),1)
Plan:
do(robot,move(c6),1)

---

do(robot,move(c4),2)
do(robot,move(o7),3)
do(robot,pickup(request(1)),4)
do(robot,move(o11),5)
do(robot,deliver(request(1)),6)
do(robot,move(c3),7)
do(robot,move(c2),8)
do(robot,move(o2),9)
do(robot,pickup(request(2)),10)
do(robot,move(o3),11)
do(robot,deliver(request(2)),12)

# Mail Delivery Robot




Input:

Plan:
do(robot,move(c6),1)

---

do(robot,move(open),2)
do(robot,move(o3),3)
do(robot,pickup(request(2)),4)
do(robot,move(o2),5)
do(robot,deliver(request(2)),6)
do(robot,move(c2),7)
do(robot,move(c3),8)
do(robot,move(c4),9)
do(robot,move(o7),10)
do(robot,pickup(request(1)),11)
do(robot,move(o11),12)
do(robot,deliver(request(1)),13)

# Demo Video

▶

# Overview



Potassco

# Conclusion

1. *ROSoClingo* fulfils the need for high-level KR&R capacities with ASP
2. Single framework declaratively controlling robots to do complex action planning while adapting to new information and environment changes
3. Eases the work of developer by making integration details transparent
4. Use cases & Collaborations
   - Mailbot with TurtleBot (Sydney)
   - Baxter solving soma and blocks world puzzles (Sydney)
   - Warehouse robot (Örebro)
   - Robot with an external geometric reasoner (Örebro)
5. Improvements and extensions
   - Formalizing the action domain with an action language
   - Handling possibly infinite requests with a fixed size request slots
   - Integrating external solvers like geometric reasoner
   - Fine-grained control of request statuses within the encoding

# Conclusion

1. *ROSoClingo* fulfils the need for high-level KR&R capacities with ASP
2. Single framework declaratively controlling robots to do complex action planning while adapting to new information and environment changes
3. Eases the work of developer by making integration details transparent
4. Use cases & Collaborations
   - Mailbot with TurtleBot (Sydney)
   - Baxter solving soma and blocks world puzzles (Sydney)
   - Warehouse robot (Örebro)
   - Robot with an external geometric reasoner (Örebro)
5. Improvements and extensions
   - Formalizing the action domain with an action language
   - Handling possibly infinite requests with a fixed size request slots
   - Integrating external solvers like geometric reasoner
   - Fine-grained control of request statuses within the encoding

# Conclusion

1. *ROSoClingo* fulfils the need for high-level KR&R capacities with ASP
2. Single framework declaratively controlling robots to do complex action planning while adapting to new information and environment changes
3. Eases the work of developer by making integration details transparent
4. Use cases & Collaborations
   - Mailbot with TurtleBot (Sydney)
   - Baxter solving soma and blocks world puzzles (Sydney)
   - Warehouse robot (Örebro)
   - Robot with an external geometric reasoner (Örebro)
5. Improvements and extensions
   - Formalizing the action domain with an action language
   - Handling possibly infinite requests with a fixed size request slots
   - Integrating external solvers like geometric reasoner
   - Fine-grained control of request statuses within the encoding

Potassco