COPENHAGEN
INTERNATIONAL
SOFTWARE DEVELOPMENT
CONFERENCE 2015

goto; conference

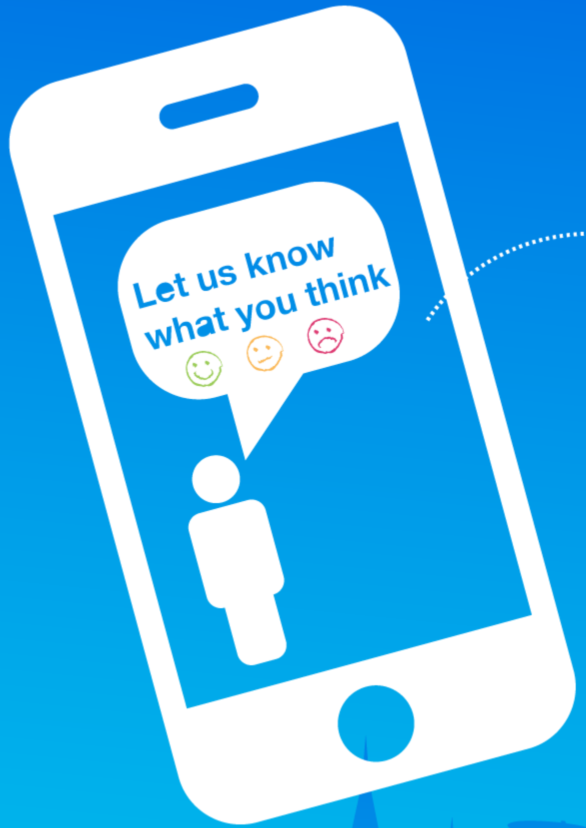
Building Languages for Self-Rebuilding Robots

Ulrik Pagh Schultz
University of Southern Denmark

(joint work with numerous colleagues)

Join the conversation #gotocph

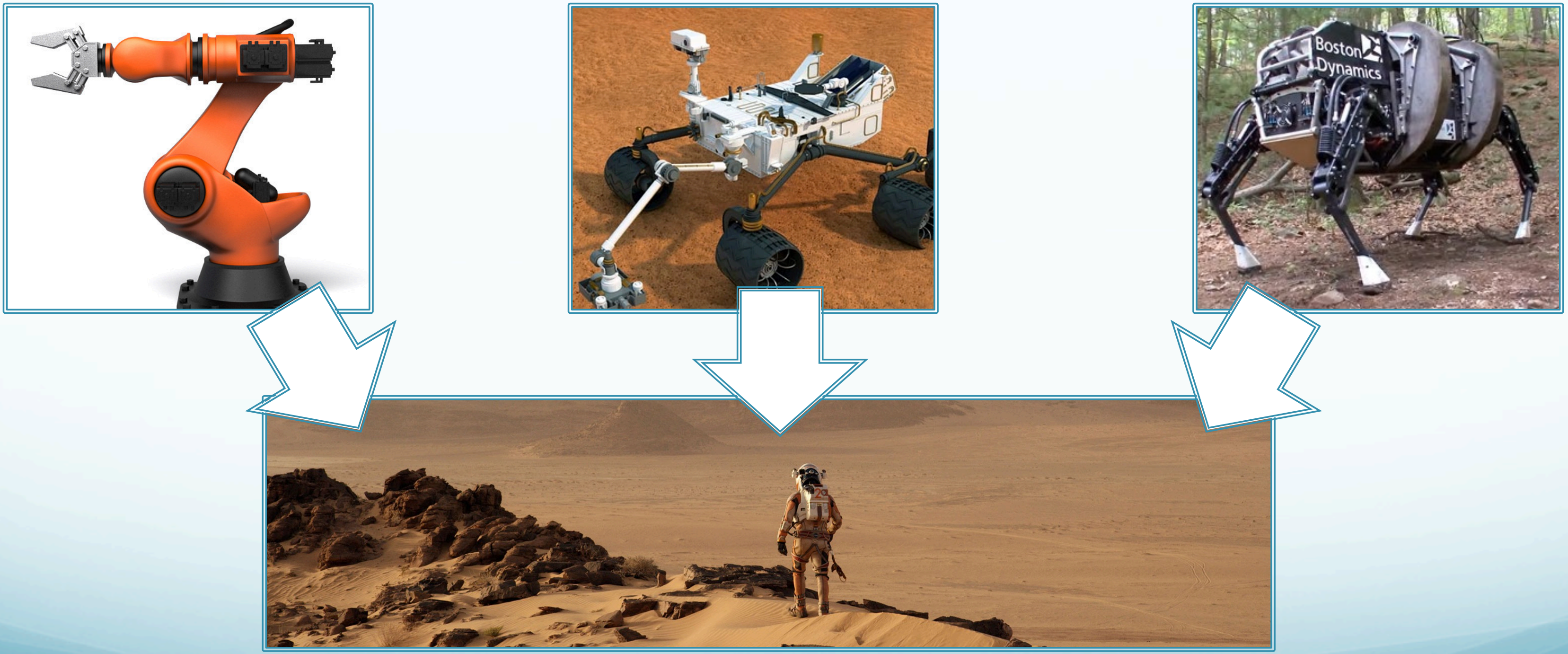
Conference: October 5-6 // Workshops: October 7-8, 2015 goto; conference



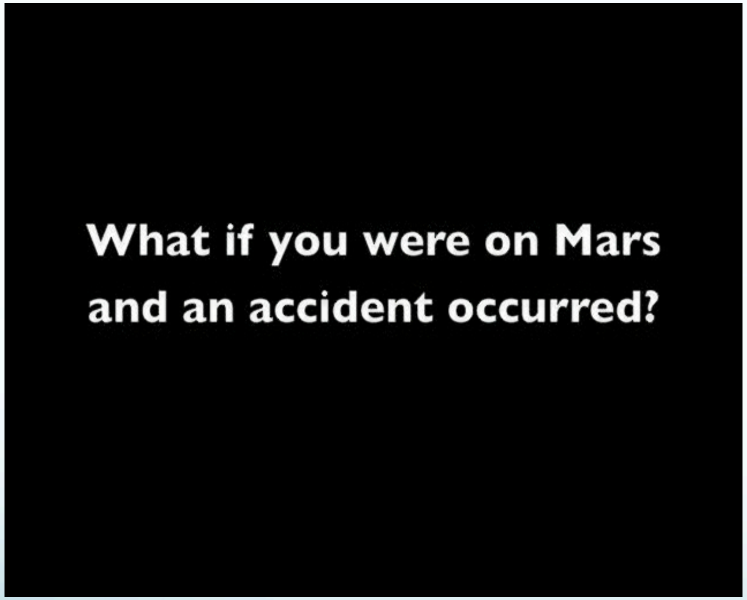
Click ‘engage’
to rate sessions
and ask questions

Join the conversation #gotocph

Q: Which robot would you bring to Mars?



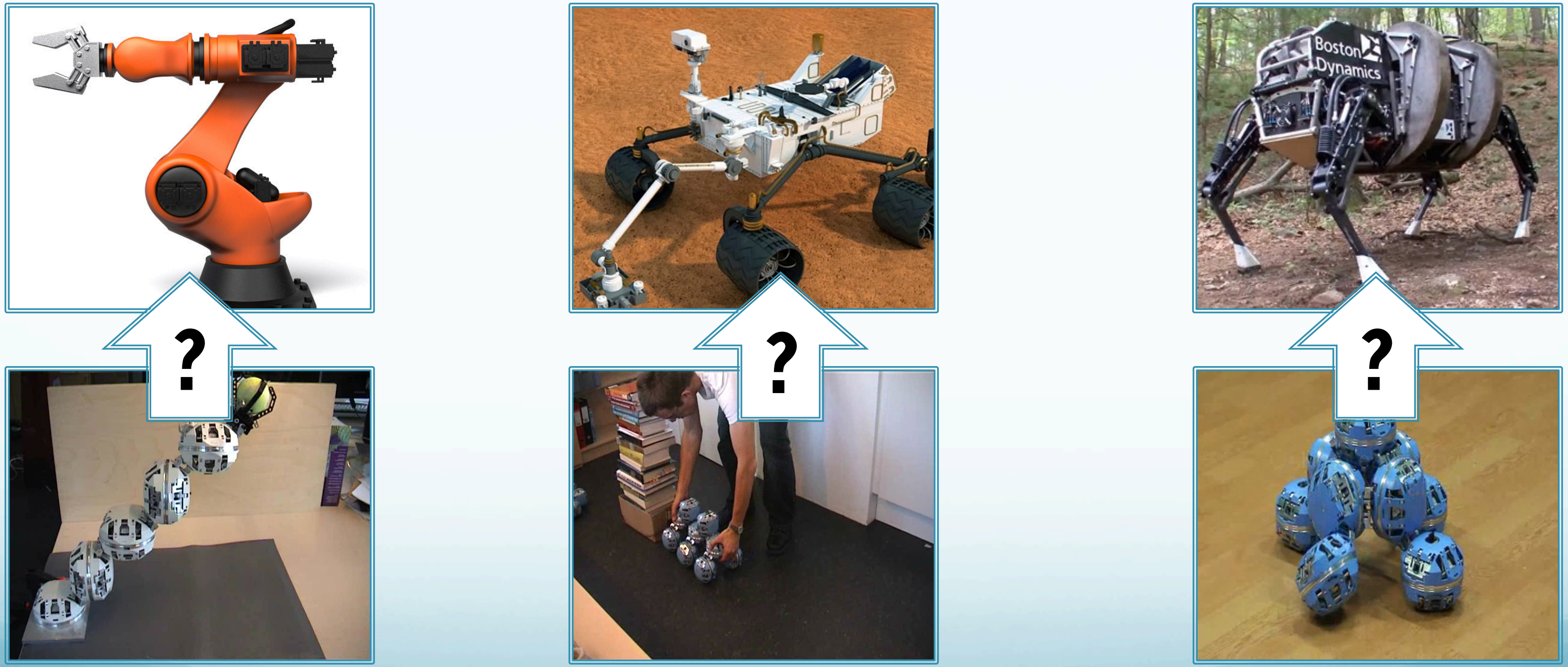
A: The robot building kit (*)



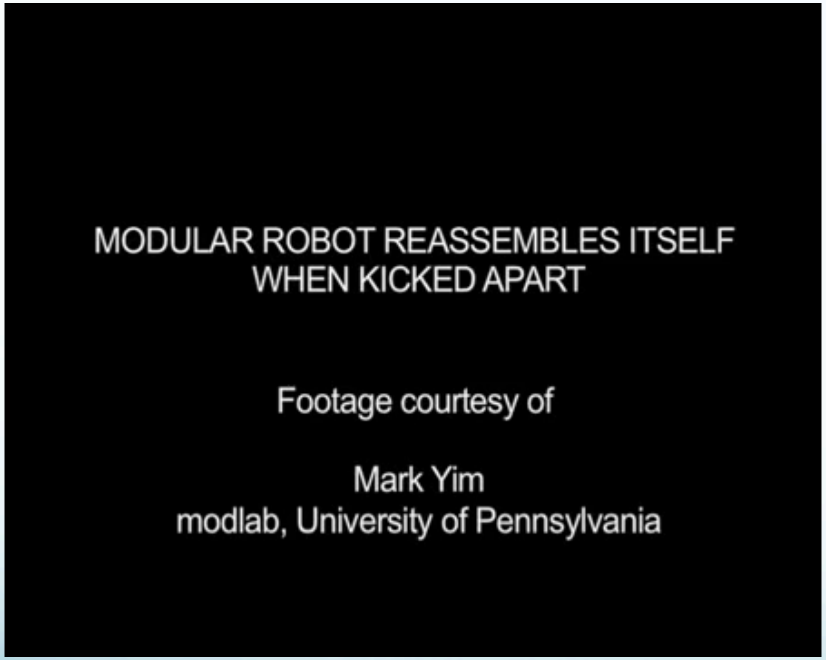
What if you were on Mars and an accident occurred?

(*): Disclaimer: Unlike this morning's keynote, we're not rocket scientists. Don't bring our (prototype) robots to Mars, you'll die!

Robot building kit?



Resilient & Adaptable

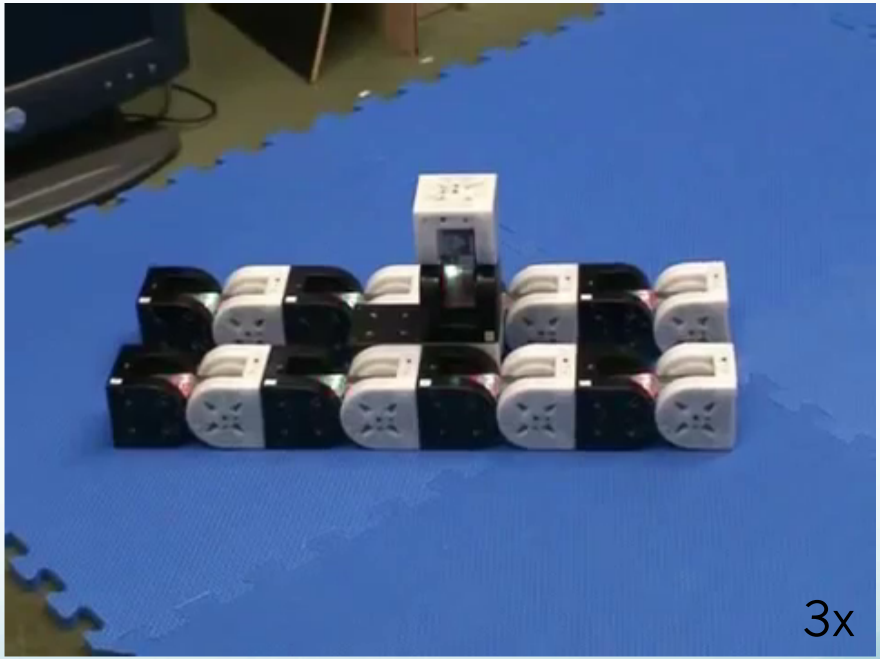


MODULAR ROBOT REASSEMBLES ITSELF
WHEN KICKED APART

Footage courtesy of

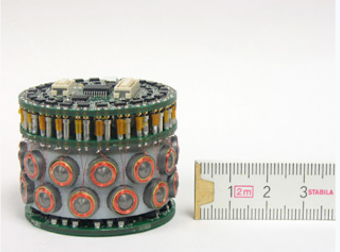
Mark Yim
modlab, University of Pennsylvania

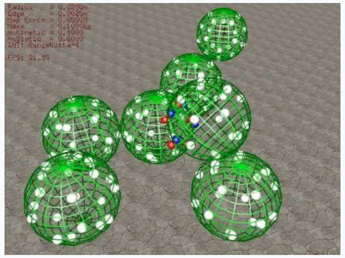
[CKBot, Yim]



[MTRAN, Kurokawa]

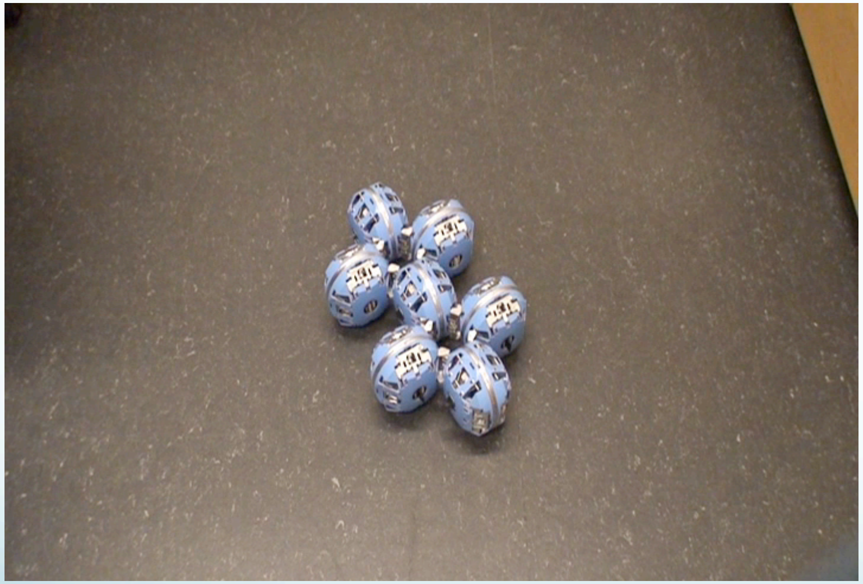
Programmable matter

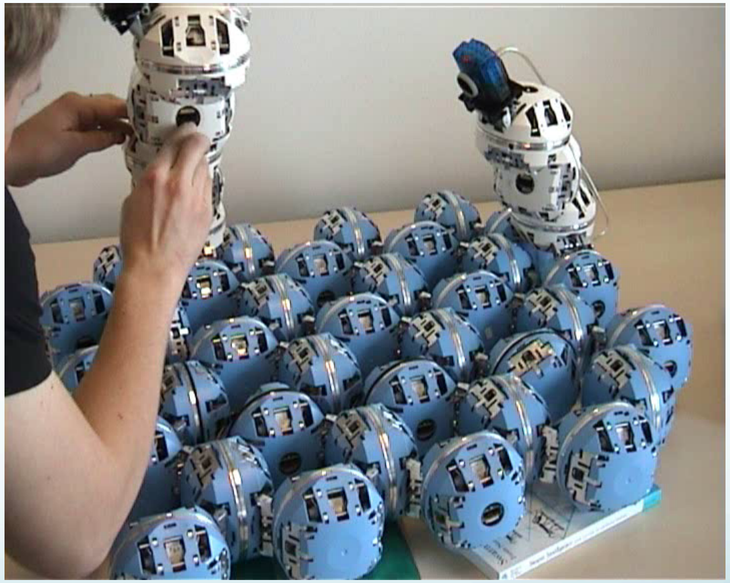




[Claytronics, Goldstein]

This talk: ATRON robot programming (*before* taking it to Mars)





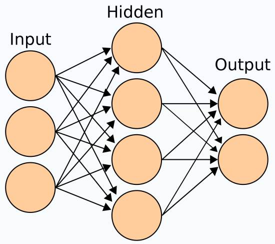
Robotics programming

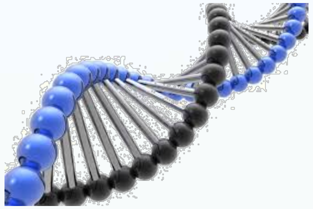
We need you!

Current state

Robotics programming

Intelligence

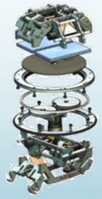


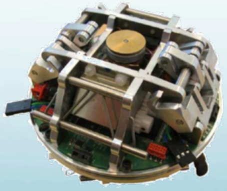


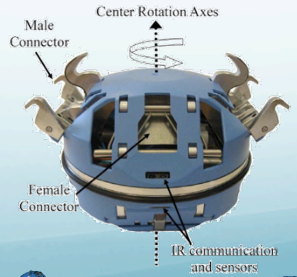
Software

C code: `val = ( *((type *&)(pIP))++ );`

Hardware

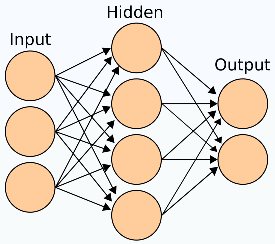


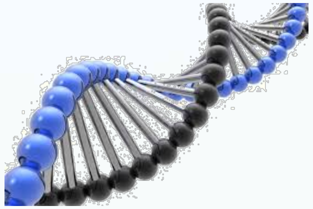




Robotics programming

Intelligence



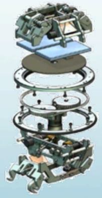


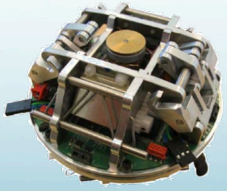
Software

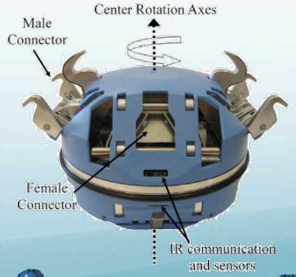
This talk: DSLs

C code: **val = (*((type *&)(pIP))++);**

Hardware







Domain-specific languages

Fowler's advantages:

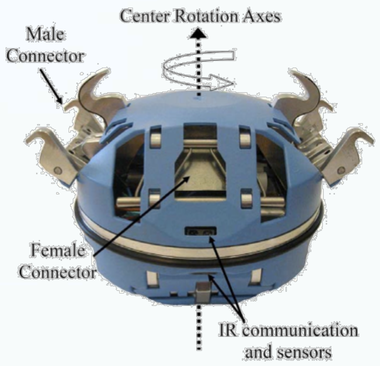
- **Improving development productivity**
- Communication with domain experts
- **Change in execution context**
- **Alternative computation model**
- **Opportunities for verification**

DSLs 101:

- Language for solving problems in a given domain
- Examples: SQL, XML, Excel, ...
- Key design issue: expressiveness vs abstraction
- Key value: abstraction mechanism
- Tools: xtext, MPS, spoofax, ...

ATRON programming?

- Modular, self-reconfigurable robot
  - 3D self-reconfiguration, hybrid/lattice-type
  - Atmel 8-bit processor with 4K RAM / 128K flash ROM
  - main joint and male connector actuation, 8 connectors total
  - neighbor communication (and proximity detection) via 8 IR ports
- (Real-time) embedded system with dynamically evolving topology
- Unreliable (bug/feature)

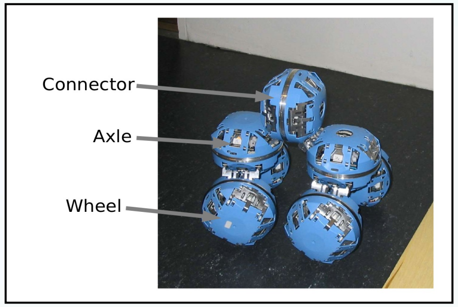


First language prototype: Everything’s an object?

```
module Connector implements Car {
  Axle front = Axle(channel#2);
  Axle rear = Axle(channel#6);
  move(v) { front.move(v); rear.move(v); }
  turn(d) { front.rotate(d/2); rear.rotate(-d/2); }
}
```

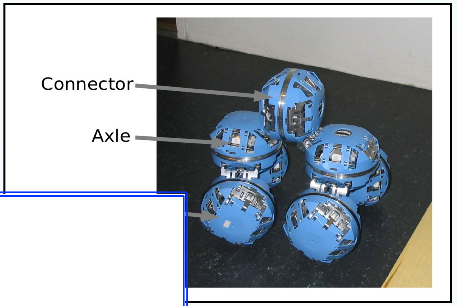
```
module Axle implements Car {
  Wheel left = Wheel(channel#0);
  Wheel right = Wheel(channel#2);
  Connector c = Connector(channel#5);
  move(v) { left.move(v); right.move(-v); }
}
```

```
module Wheel implements Car {
 Axle axle = Axle(channel#5);
}
```



```
whole Car {
 drive(v) {
  Connector.move(v);
 }
 turn(d) {
  Connector.turn(d);
 }
}
```

First language prototype: Everything's an object?



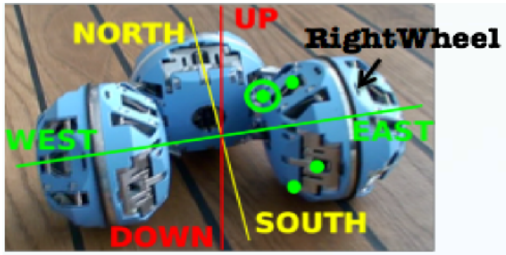
```
modul
  Axle
  Axle
  move
  turn(
}
```

```
mo
  W
  W
  C
  n
}
```

```
modul
 Axle a
}
```

```
ove(v);
rn(d);
```

- **Good**:
  - modularity
  - concise RPC syntax
- **Bad**:
  - hardcoded spatial structure
  - programming model not homogeneous
  - doesn't really work!



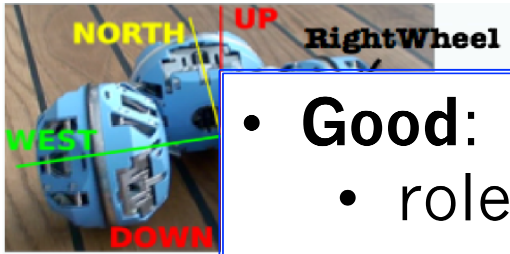
#2: Roles for shapes, functions for functionality?

- Functional reactive programming with physical pattern matching based on roles
  - roles defined using spatial constraints
  - behavior defined using distributed functions

- VM does distributed shape/role-based code application

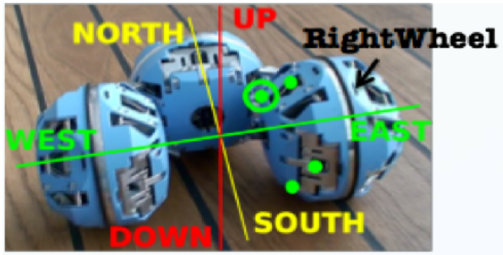
```
role Wheel (Module x) = (center_position EAST_WEST x) and ...
   | LeftWheel (Wheel x) = sizeof (connected WEST x)=1
   | ...
fun moveWheel speed (LeftWheel w) = @turnContinuous speed w
  | moveWheel speed (RightWheel w) = @turnContinuous -speed w
apply* (moveWheel 1)

nWheels = fold* (fn n (Wheel m) => (n+1)) 0
maxX = fold* (fn x (Module m) => if x>@getX m then x else @getX m) -127
```



#2: Roles for shapes

- **Good**:
  - roles for mapping structure to behavior
  - wonderful functional abstractions
- **Bad**:
  - wonderful functional abstractions
  - very difficult to implement properly (2K)
  - more well-suited to behavior-based control (continuous) than self-reconfiguration (state transitions)
  - doesn’t really work



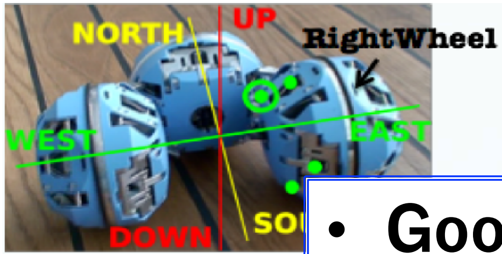
#3: Roles for shapes, Roles for functionality!

```
abstract role Wheel extends Module {
 …
 require self.center == EAST_WEST;
 require sizeof(self.connected(side)) == 1;
 behavior move() {
  self.@TURN_CONTINUOUSLY(turn_dir);
 }
 command evade() { … }
}

role RightWheel extends Wheel { … }

role Head extends Module {
 require self.center == NORTH_SOUTH;
 startup initialize() {
 handle PROXIM_1 PROXIM_3 {
   Wheel.evade(0);
…
```

- Role = hierarchy of behaviors in context
- Spatial constraints for activation and deployment
- Efficient and dynamic role-based distributed code deployment



#3: Roles for shapes, Roles for functionality!

- **Good**:
  - roles for mapping structure to behavior
  - OO-style reuse easy to implement
- **Bad**:
  - robot control algorithms are hard to read: distributed across roles
  - doesn't really work

```
abstract r
 ...
 require s
 require s
 behavior
  self.@TU
 }
 command
}

role Right

role Head
 require s
 startup i
 handle P
   Wheel.evade(0);
...
```

ent

- Efficient and dynamic role-based distributed code deployment

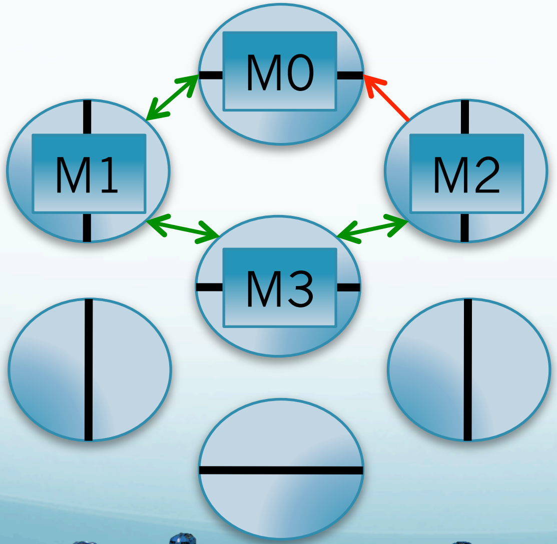
#4: The insight: One program distributed across the robot.

- Self-reconfiguration = group sequential/parallel behavior
  - Execution a “spatial wave of state changes”
  - Robust local/global execution in the presence of partial hardware failure
  - Manage physical parallelism easily
- Automatic derivation of reverse sequence
- Automatic scheduling of communication

```
sequence eight2car {
  M0.Connector[0].retract() &
  M3.Connector[4].retract();
  M3.Joint.rotateFromToBy(0,324,false,150);
  ... } ...
car2eight = reverse eight2car;
car2snake = car2eight + eight2snake;
snake2car = reverse car2snake;
```

#4 (details): State management

```
M0.connector[0].retract() |
M3.connector[4].retract();
M3.rotateFromTo(0,324); ...
```



Globally shared state

- Store current and pending states in all modules
- Continuously and independently of actions communicate local state to all neighbors
- *Merge* incoming global state to ensure progression

#4 (details): Properties [1/2], Robustness and *efficiency*

Robustness: partial failures

- *Order of magnitude improvement!*
- *Communication:*
  - continuous transmission of idempotent packets
  - broadcast communication
- *Module reset:*
  - idempotent operations
  - replication of global state

Efficiency:

- *Time:* continuous transmission ensures fastest safe progression
- *Steps:* massive opportunities for parallelization often unexploited
- *Experiments:* reversible experiments reduces need for reassembly

#4 (details): Properties [2/2], Program reversibility

- Reversible programs:
  - facilitated by API design
  - practical tool, not theoretical result

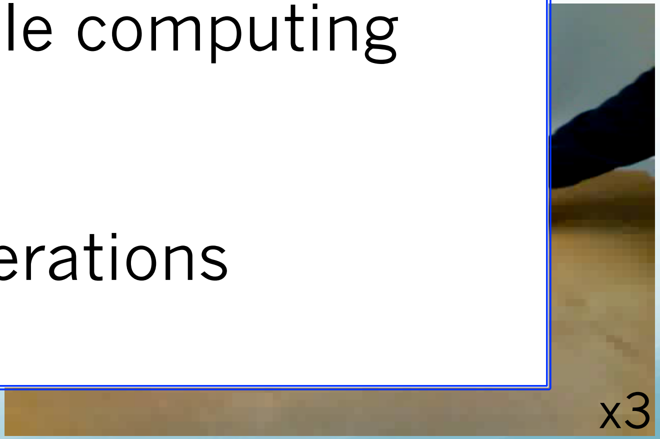
  (reverse, then generate)
- Not reversal in a purely semantic sense
- Perfect for self-reconfiguration

```
seq eight2car = {
 M0.connector[0].retract() |
 M3.connector[4].retract() ;
 M3.rotateFromTo(0,324); ... }
seq car2eight = rev eight2car;
```

#4 (details): Properties [2/2], Program reversibility

- **Good**:
  - whole-robot control easy to read
  - really works: order of magnitude robustness improvement
  - practical use of reversible computing

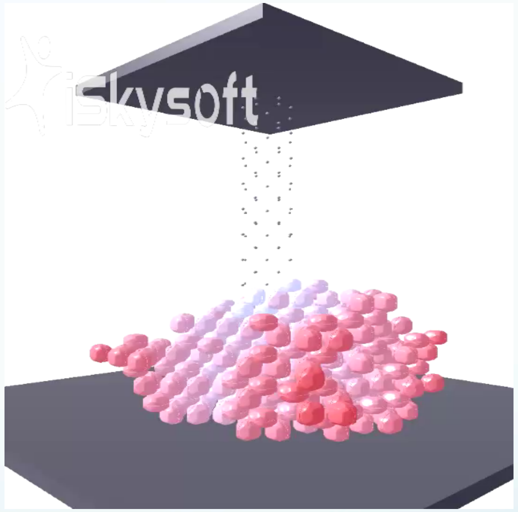
- **Bad**:
  - only does sequential operations

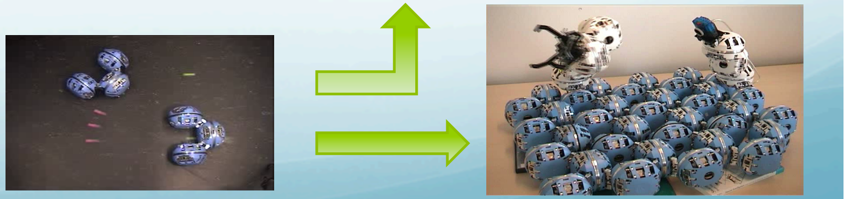


Perspectives

- Programming approach must match the hardware:
  - unreliable
  - distributed control and state
- Incremental language evolution:
  - the search for more abstractions
  - patterns & forces
- Impact: modularity & abstraction for more robots

Goal: morphogenesis



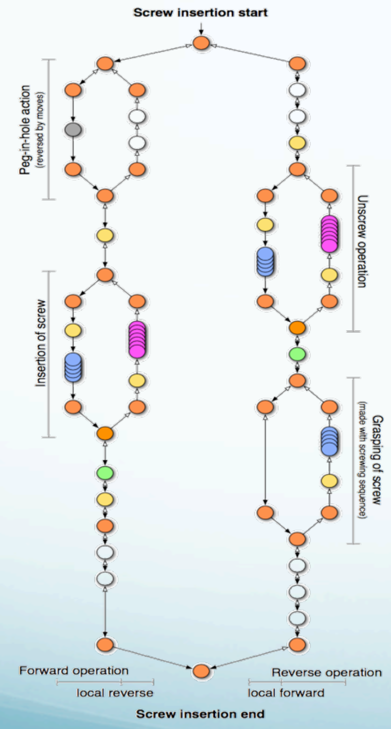


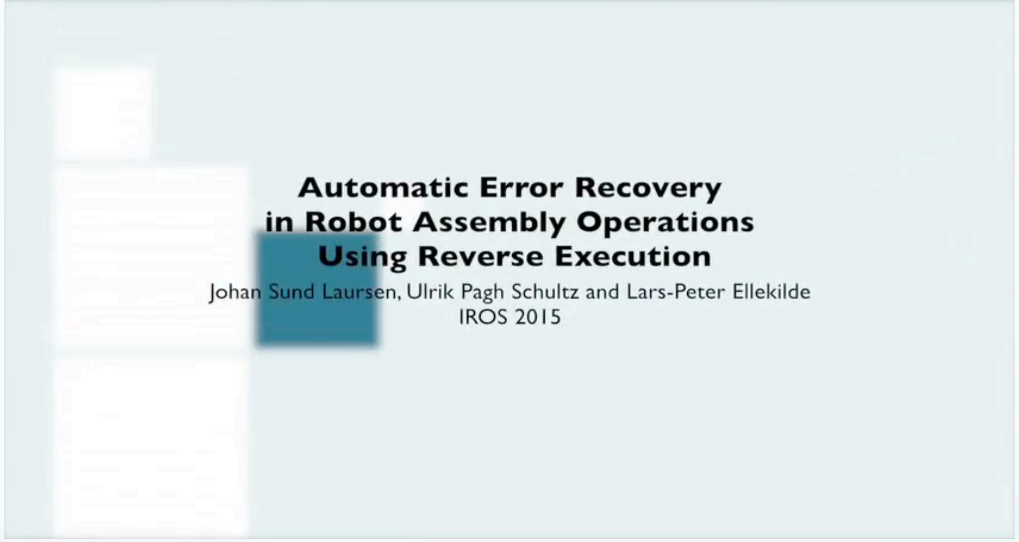
Industrial robots

UNIVERSAL ROBOTS

Source: Universal Robots
PS: They're hiring

DSL for Reversible Assembly Sequences





Automatic Error Recovery in Robot Assembly Operations Using Reverse Execution

Johan Sund Laursen, Ulrik Pagh Schultz and Lars-Peter Ellekilde

IROS 2015

```
pickup(nut,gripper2,nut_pos); moveto(above_table);
try(3: force<1N) {
  moveto(on_bolt); call apply_and_turn_nut
}
release(nut,gripper2,nut_attached_pos);
```

Agricultural robots

Example: Kongskilde Robotti...

...and earlier SDU prototypes

- Precision agriculture
- DSLs for safety
- Software: ROS

  Excellent pathway into (experimental) robotics

  (note: ROS≈javascript of robotics)

Unmanned Aerial Systems (i.e., flying robots aka drones)

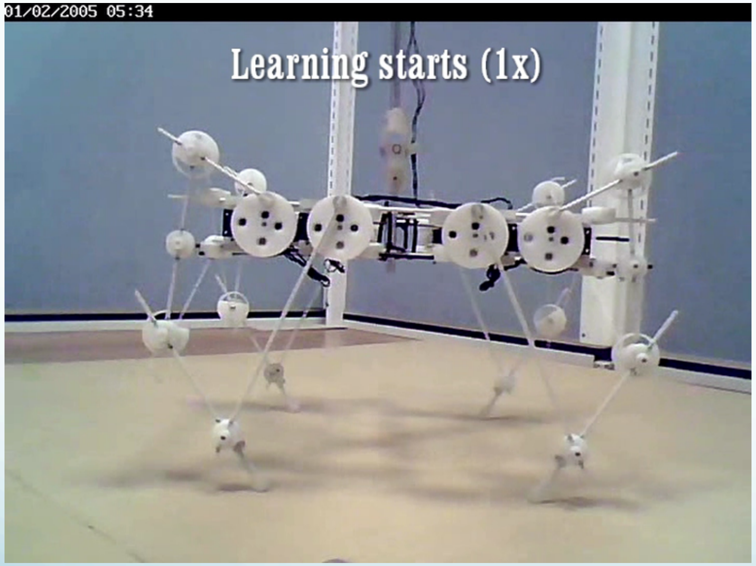
Software (& Hardware)

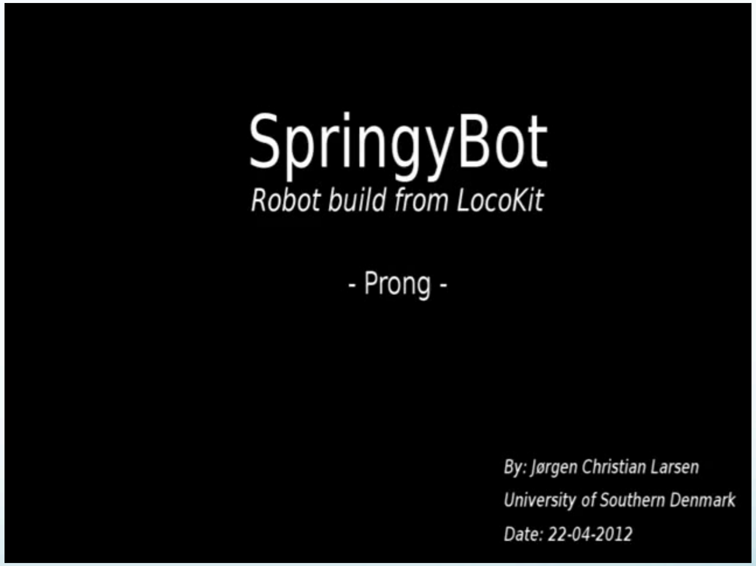
- Civilian applications, e.g., agriculture, environmental monitoring, ...
- Principles for a DSL for swarm coordination?
- Code generation for safety!

Infrastructure hotspots

- License plates
- UAS Test Center
- Pilot certification
- BVLOS legislation

Lightweight energy-efficient robots




SpringyBot

Robot build from LocoKit

- Prong -

By: Jørgen Christian Larsen

University of Southern Denmark

Date: 22-04-2012

Take-away

Robots

- Physical modularity
- Cognitive gap
- They’re coming (but they need your help)

DSLs

- Ultimate abstraction mechanism
- Abstractions require insights
- Systematic development?

goto; conference

Please

Remember to rate this session

Thank you!

Join the conversation #gotocph