Ambient Intelligence

Semantic Ambient Assistance Processes

Serge Autexier   Christoph Stahl

German Research Center for Artificial Intelligence (DFKI), Bremen, Germany

23.06.2014

AmI 2014

Cyber-Physical Systems

Universität Bremen

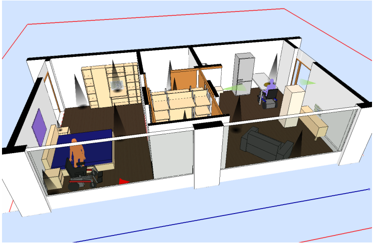
- Modelling the environment
- Acting on the environment
- Interfaces
- Observing/Monitoring the environment
- Intelligent Assistance: Planning vs. Programming
    - Reacting on failures

Modelling the environment

Modelling the environment

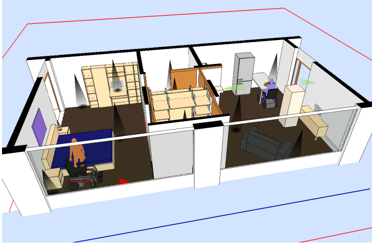
Cyber-Physical Systems

- The different objects, sensors, actors, locations, etc.
- The relationships between these
- Derivable properties



Modelling the environment

- The different objects, sensors, actors, locations, etc.
- The relationships between these
- Derivable properties



- Modelling using Ontologies

Using Description Logics

- Ontology-language interoperable with OWL 2
- Closer to DL languages
- Formalisation consists of **terminology** (TBOX) und **assertions** (ABOX):
  - TBOX:
    - Inclusions $\mathtt{C} \sqsubseteq \mathtt{D}$
    - Definitions $\mathtt{C} ::= \mathtt{D}$, C Name
    - Maximally one Definition per name
  - ABOX:

$$Steve : Parent, (Steve, John) : hasChild$$

Language

	Description Logic	Concrete SHIP Syntax
Inclusion	A ⊑ B	A <B
Definition	A ::= B	A ::= B
Union, Intersection	A⊔ B, A ⊓ B	A + B, A & B
Ex, All	∃R.A,∀R.A	ex R.A, all R.A
Disjointness	DisjointClasses(A,B)	**Disjoint** (A, B)
Functional Roles	FunctionalObjectProperty(R)	**Func**(A, B)
	FunctionalObjectProperty(R), ObjectPropertyDomain(R, A), ObjectPropertyRange(R, B)	R:A ->B
	Light ::= LightOn +LightOff, DisjointClasses(LightOn, LightOff)	Light ::= LightOn|LightOff

Language

Description Logic	Concrete SHIP Syntax
WheelChair $\sqsubseteq \exists$route.Route $\sqcap \exists$carries.OptPerson, FunctionalObjectProperty(route), FunctionalObjectProperty(carries), ObjectPropertyRange(route, Route), ObjectPropertyRange(carries,OptPerson), ObjectPropertyDomain(route, WheelChair), ObjectPropertyDomain(carries,WheelChair)	WheelChair ::= WheelChair(route:Route, carries:OptPerson)

Family of Description Logics

- ALC: only atomic Roles
- ALCN: unqualified number restrictions for roles

$$\leq nR, \geq nR$$

- ALCQ: qualified number restrictions for roles

$$\leq nR.C, \geq nR.C$$

- ALCI: Inverse roles

$$\forall \mathtt{R}^{-}.\mathtt{C}, \exists \mathtt{R}^{-}.\mathtt{C}, \ldots$$

- ALCO: nominal classes and roles

$$\{a\}, \{(c, d)\}, \ldots$$

Usage

- Describe environment and their attributes but also any other kind of basic information (≈ classes and attributes)

```
Light ::= LightOn | LightOff
WheelChair ::= WheelChair(route:Route, carries:OptPerson)
OptPerson = Person | Nobody
```

- **ABox**-facts represent current state and process information

```
livingroomlight1:LightOn      rolland:WheelChair
bathroomlight2:LightOff       (rolland, r1):route
                              (rolland, paul):carries
```

Using Description Logics

- Represent derived knowledge/properties (no counterpart in programming languages)

```
WCCarriesPerson = WheelChair ⊓∃carries . Person
WCNonEmptyroute = WheelChair ⊓∃route . NonEmptyRoute
```

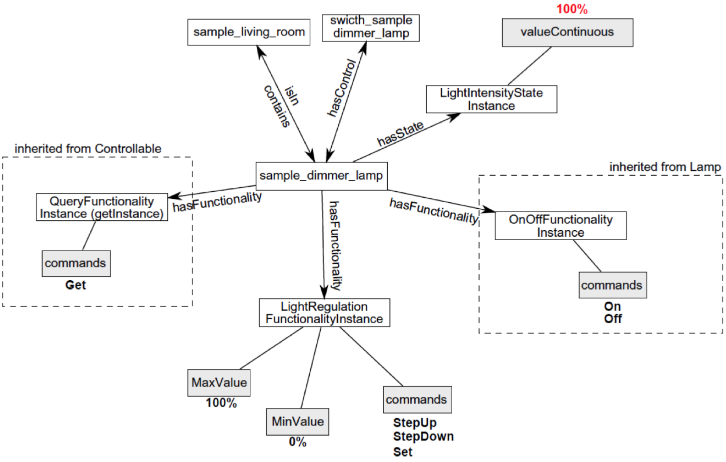
- Use it in queries to test/find sensors/actors/devices based on their properties

```
wc:WCNonEmptyroute and
(wc:WCCarriesPerson or
 ex person:(Person & (ex elementIsInArea . { livingroom })))
// carries a person or there is a person in the area... //
```

- Similar to queries over databases (e.g. SQL), but **logical database**.

Modellierung einer Lampe in DogOnt in vereinfachter Syntax

Cyber-Physical Systems



Changes in the Environment

Updating Knowledge

livingroomlight1:LightOn (rolland, nobody):carries (rolland,sofa):at	livingroomlight1:**LightOff** (rolland, nobody):carries (rolland,sofa):at	livingroomlight1:LightOff (rolland, nobody):carries (rolland, **table**):at

livingroomlight1:LightOff →　(rolland, table):at →

- Updates are provided on ABox-facts only
- Minimal and complete to keep ABoxes **constructive**
  - Derived from logical properties of declarations
  - Like only allowing sensor and actor updates, but also for process internal knowledge

Updating Knowledge

livingroomlight1:LightOn (rolland, paul):carries (rolland, sofa):at (paul, sofa):at	livingroomlight1:Off (rolland, paul):carries (rolland, **table**):at (paul, **sofa**):at

(rolland, table):at →

- Updates are provided on ABox-facts only
- Minimal and complete to keep ABoxes **constructive**
  - Derived from logical properties of declarations
  - Like only allowing sensor and actor updates, but also for process internal knowledge

Handling Frame Problem by Causal Relationships

Cyber-Physical Systems

```
indirect effect CarriedPersonMovesAsWell = {
 init = (wc,p):at
 causes = (x,p):at
 cond = (wc,x):carries, x:Person, wc:WheelChair
}
```

- Causal relationships: Baader F., et.al., 2010. *Using causal relationships to deal with the ramification problem in action formalisms based on description logics*, LPAR 2010.
- Include indirect effect rules declarations to ontology

Handling the Frame Problem

- Apply upon update, add additional new facts, which in turn removes more old facts (hence unlike onology rules)

livingroomlight1:LightOn (rolland, paul):carries (rolland, sofa):at (paul, sofa):at	livingroomlight1:LightOff (rolland, paul):carries (rolland, **table**):at (paul, **table**):at

(rolland, table):at

$\longrightarrow$

Acting on the environment

Acting on the environment

Action descriptions: $a(\varphi, (\alpha, \delta), (\varphi_1 \rightarrow (\alpha_1, \delta_1), \ldots)$

- Action name: a , possibly with parameters
- Preconditions φ: list of literals that need to hold
- Effects
  - Unconditional:
    - add α: list of atomic assertions that are added
    - del δ: list of atomic assertions that are deleted
  - Conditional:
    - If φ_i holds in previous world then add α_i and delete δ_i

(remember Jess, RETE network)

Examples in SHIP-DL


```
action switchOn (l) =
{
pre = l:LightOn
effect = l:lightOff
}
```

```
action switchLight (l) =
{
pre = l:Light
if (l:LightOn) l:LightOff
if (l:LightOff) l:LightOn
}
```

- Action application computes an update (change) and is applied like a change obtained from the environment

livingroomlight1:LightOn	livingroomlight1:LightOff

switchLight(livingroomlight1)

⟶

Action Application

- Assume there is a second light livingroomlight2, physically synchronized with livingroomlight1
- How would you model that connection?
- What happens on

livingroomlight1:LightOn
livingroomlight2:LightOn

- On

livingroomlight1:LightOn
livingroomlight2:LightOff

Observing/Monitoring the environment

Observing Properties of the Environment

- We want to formulate that those two lamps are always in the same state, and detect when they are not (malfunction).
- Observing events over time:

  When the livingroomlight1 has been turned on, then the lowerleftdoor is opened and the corridorlight switched on, then livingroomlight1 is turned off.

- Observing derived properties over time:

  A room remains illuminated at night as long as a person is in it until the person leaves the room or turns off all lights in that room

Monitoring over time

- Language to formulate behaviour over time: temporal logic over ABox properties
  - formulas over ABox atoms:

$$\alpha = n : C \mid \text{not}\,\alpha \mid \alpha \text{ and } \alpha \mid \alpha \text{ or } \alpha \mid \alpha \Rightarrow \alpha$$

  - temporal connectives
    - Globally $G\varphi$ Now and always in the future
    - Eventually $F\varphi$ Now or eventually in the future
    - Until $\varphi U \psi$ φ holds until ψ holds
  - bounded quantification
    - Forall $\forall n : C.\varphi$ for all n satisfying C (will) hold φ
    - Exists $\exists n : C.\varphi$ for some n satisfying C (will) hold φ

Observing Properties of the Environment I

- We want to formulate that those two lamps are always in the same state, and detect when they are not (malfunction).

```
G((livingroomlight1:LightOn and livingroomlight2:LightOn) or
  (livingroomlight1:LightOff and livingroomlight2:LightOff))
```

- Observing events over time:

  When the livingroomlight1 has been turned on, then the lowerleftdoor is opened and the corridorlight switched on, then livingroomlight1 is turned off.

```
livingroomlight1:LightOn ⇒F((lowerleftdoor:Open and
     corridorlight:LightOn) ⇒F(livingroomlight1:LightOff))
```

Observing Properties of the Environment II

Cyber-Physical Systems

- Observing derived properties over time:

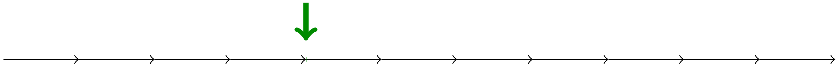
  A room remains illuminated at night as long as a person is in it until the person leaves the room or turns off all lights in that room

```
isinarea:(Person⊔Light) →Room
RoomIlluminated ::= Room ⊓∃inv(isinarea) . LightOn
RoomNotIlluminated ::= Room ⊓∀inv(isinarea) . LightOff

daytime:Night and r:RoomIlluminated
   and p:(Person ⊓(∃isinarea . {r})) ⇒
  (daytime:Night and r:RoomIlluminated U (r:RoomNotIlluminated or
      not((p,r):isinarea))
```

More Examples


```
rolland:WCNonEmptyRoute
(rolland,p1):at
(rolland,p2):WCNextPosition
```

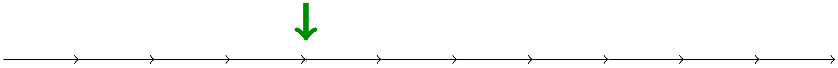


Progress: `init G(all wc:WCNonEmptyRoute .`
`WellBehaved(wc)U wc:WCEmptyRoute)`

```
G(all wc:WCNonEmptyRoute . WellBehaved(wc))
WellBehaved(wc) ::= ∀p1:Position .∀p2:Position .
          (wc,p1):at and (wc,p2):WCNextPosition . (wc,p1):at U
              (wc,p2)
```

More Examples


```
rolland:WCNonEmptyRoute
(rolland,p1):at
(rolland,p2):WCNextPosition
```



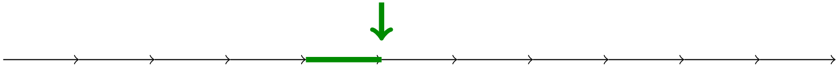
Progress:

```
(rolland,p1):at U (rolland,p2):at
and G(all wc:WCNonEmptyRoute . WellBehaved(wc)U wc:WCEmptyRoute)
```

```
G(all wc:WCNonEmptyRoute . WellBehaved(wc))
WellBehaved(wc) ::= ∀p1:Position .∀p2:Position .
           (wc,p1):at and (wc,p2):WCNextPosition . (wc,p1):at U
               (wc,p2)
```

More Examples

rolland:WCNonEmptyRoute
(rolland,p2):at
(rolland,p3):WCNextPosition

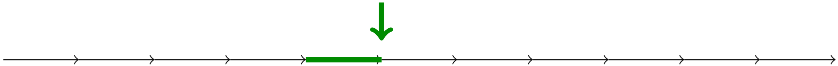


Progress: (rolland,p1):at U (rolland,p2):at
and G(all wc:WCNonEmptyRoute . WellBehaved(wc)U wc:WCEmptyRoute)

G(all wc:WCNonEmptyRoute . WellBehaved(wc))
WellBehaved(wc) ::= ∀p1:Position .∀p2:Position .
(wc,p1):at and (wc,p2):WCNextPosition . (wc,p1):at U
(wc,p2)

More Examples


```
rolland:WCNonEmptyRoute
(rolland,p2):at
(rolland,p3):WCNextPosition
```



Progress:
```
(rolland,p2):at U (rolland,p3):at
and G(all wc:WCNonEmptyRoute . WellBehaved(wc)U wc:WCEmptyRoute)
```

```
G(all wc:WCNonEmptyRoute . WellBehaved(wc))
WellBehaved(wc) ::= ∀p1:Position .∀p2:Position .
           (wc,p1):at and (wc,p2):WCNextPosition . (wc,p1):at U
               (wc,p2)
```

More Examples

`rolland:WCEmptyRoute`
`(rolland,p3):at`

Progress: `(rolland,p2):at U (rolland,p3):at`
`and G(all wc:WCNonEmptyRoute . WellBehaved(wc)U wc:WCEmptyRoute)`

```
G(all wc:WCNonEmptyRoute . WellBehaved(wc))
WellBehaved(wc) ::= ∀p1:Position .∀p2:Position .
           (wc,p1):at and (wc,p2):WCNextPosition . (wc,p1):at U
               (wc,p2)
```

More Examples

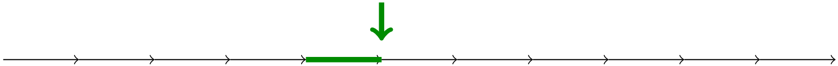
rolland:WCEmptyRoute
(rolland,p3):at

Progress: G(all wc:WCNonEmptyRoute .
WellBehaved(wc)U wc:WCEmptyRoute)

G(all wc:WCNonEmptyRoute . WellBehaved(wc))
WellBehaved(wc) $::=$ $\forall$p1:Position .$\forall$p2:Position .
(wc,p1):at and (wc,p2):WCNextPosition . (wc,p1):at U
(wc,p2)

More Examples


```
rolland:WCNonEmptyRoute
(rolland,p2):at
(rolland,p3):WCNextPosition
```



Progress:
```
(rolland,p2):at U (rolland,p3):at
and G(all wc:WCNonEmptyRoute . WellBehaved(wc)U wc:WCEmptyRoute)
```

```
G(all wc:WCNonEmptyRoute . WellBehaved(wc))
WellBehaved(wc) ::= ∀p1:Position .∀p2:Position .
        (wc,p1):at and (wc,p2):WCNextPosition . (wc,p1):at U
            (wc,p2)
```

More Examples

```
rolland:WCNonEmptyRoute
(rolland,p1):at
(rolland,p2):WCNextPosition
```

Progress: `(rolland,p2):at U (rolland,p3):at`
`and G(all wc:WCNonEmptyRoute . WellBehaved(wc)U wc:WCEmptyRoute)`

```
G(all wc:WCNonEmptyRoute . WellBehaved(wc))
WellBehaved(wc) ::= ∀p1:Position .∀p2:Position .
           (wc,p1):at and (wc,p2):WCNextPosition . (wc,p1):at U
               (wc,p2)
```

More Examples

rolland:WCNonEmptyRoute
(rolland,p1):at
(rolland,p2):WCNextPosition

Progress: False

```
G(all wc:WCNonEmptyRoute . WellBehaved(wc))
WellBehaved(wc) ::= ∀p1:Position .∀p2:Position .
            (wc,p1):at and (wc,p2):WCNextPosition . (wc,p1):at U
                (wc,p2)
```