Ambient Intelligence
Semantic Ambient Assistance Processes

Serge Autexier   Christoph Stahl

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

23.06.2014

Aml 2014
- 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

- 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 \( C \sqsubseteq D \)
    - Definitions \( C ::= D, \ C \text{ Name} \)
    - Maximally one Definition per name
  - **ABOX**:
    
    Steve : Parent, (Steve, John) : hasChild
# Description Logic

<table>
<thead>
<tr>
<th>Description Logic</th>
<th>Concrete SHIP Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inclusion</td>
<td>$A \sqsubseteq B$</td>
</tr>
<tr>
<td>Definition</td>
<td>$A ::= B$</td>
</tr>
<tr>
<td>Union, Intersection</td>
<td>$A \sqcup B, A \sqcap B$</td>
</tr>
<tr>
<td>Ex, All</td>
<td>$\exists R.A, \forall R.A$</td>
</tr>
<tr>
<td>Disjointness</td>
<td><code>DisjointClasses(A,B)</code></td>
</tr>
<tr>
<td>Functional Roles</td>
<td><code>FunctionalObjectProperty(R)</code></td>
</tr>
<tr>
<td></td>
<td>$\text{FunctionalObjectProperty}(R), \text{ObjectPropertyDomain}(R, A), \text{ObjectPropertyRange}(R, B)$</td>
</tr>
<tr>
<td></td>
<td>$R:A \rightarrow B$</td>
</tr>
<tr>
<td>Light</td>
<td>$\text{Light} ::= \text{LightOn} + \text{LightOff}, \text{DisjointClasses}(\text{LightOn}, \text{LightOff})$</td>
</tr>
<tr>
<td></td>
<td>$\text{Light} ::= \text{LightOn</td>
</tr>
</tbody>
</table>

---

Language

Cyber-Physical Systems

Universität Bremen

Aml 2014
<table>
<thead>
<tr>
<th>Description Logic</th>
<th>Concrete SHIP Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>WheelChair ⊑ ∃route.Route ⊓ ∃carries.OptPerson, FunctionalObjectProperty(route), FunctionalObjectProperty(carries), ObjectPropertyRange(route, Route), ObjectPropertyRange(carries, OptPerson), ObjectPropertyDomain(route, WheelChair), ObjectPropertyDomain(carries, WheelChair)</td>
<td>WheelChair ::= WheelChair(route:Route, carries:OptPerson)</td>
</tr>
</tbody>
</table>
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
  \[ \forall R^{-}.C, \exists R^{-}.C, \ldots \]
- 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  
  bathroomlight2:LightOff  
  rolland:WheelChair  
  (rolland, r1):route  
  (rolland, paul):carries
Using Description Logics

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

\[ \text{WCCarriesPerson} = \text{WheelChair} \sqcap \exists \text{carries} . \text{Person} \]
\[ \text{WCNonEmptyroute} = \text{WheelChair} \sqcap \exists \text{route} . \text{NonEmptyRoute} \]

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

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

- Similar to queries over databases (e.g. SQL), but logical database.
Modellierung einer Lampe in DogOnt in vereinfachter Syntax
Changes in the Environment
Updating Knowledge

- 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

- 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

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


- 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
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 \( \varphi \): list of literals that need to hold
- Effects
  - Unconditional:
    - add \( \alpha \): list of atomic assertions that are added
    - del \( \delta \): list of atomic assertions that are deleted
  - Conditional:
    - If \( \varphi_i \) holds in previous world then add \( \alpha_i \) and delete \( \delta_i \)

(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)
```
Assume there is a second light `livingroomlight2`, physically synchronized with `livingroomlight1`

How would you model that connection?

What happens on

On
Observing/Monitoring 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 \( G\varphi \) \( \text{Now and always in the future} \)
  ▶ Eventually \( F\varphi \) \( \text{Now or eventually in the future} \)
  ▶ Until \( \varphi U\psi \) \( \varphi \text{ holds until } \psi \text{ holds} \)

▶ bounded quantification
  ▶ Forall \( \forall n : C.\varphi \) \( \text{for all } n \text{ satisfying } C \text{ (will) hold } \varphi \)
  ▶ Exists \( \exists n : C.\varphi \) \( \text{for some } n \text{ satisfying } C \text{ (will) hold } \varphi \)
We want to formulate that those two lamps are always in the same state, and detect when they are not (malfunction).

\[ G((\text{livingroomlight1}:\text{LightOn} \ \text{and} \ \text{livingroomlight2}:\text{LightOn}) \ \text{or} \ (\text{livingroomlight1}:\text{LightOff} \ \text{and} \ \text{livingroomlight2}:\text{LightOff})) \]

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.*

\[ \text{livingroomlight1}:\text{LightOn} \ \Rightarrow F((\text{lowerleftdoor}:\text{Open} \ \text{and} \ \text{corridorlight}:\text{LightOn}) \ \Rightarrow F(\text{livingroomlight1}:\text{LightOff})) \]
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

\[
\text{isinarea} : (\text{Person} \sqcup \text{Light}) \rightarrow \text{Room}
\]

\[
\text{RoomIlluminated} ::= \text{Room} \sqcap \exists \text{inv(isinarea)} . \text{LightOn}
\]

\[
\text{RoomNotIlluminated} ::= \text{Room} \sqcap \forall \text{inv(isinarea)} . \text{LightOff}
\]

daytime: \text{Night and r:RoomIlluminated}

and p: (\text{Person} \sqcap (\exists \text{isinarea} . \{r\})) \Rightarrow

(daytime: \text{Night and r:RoomIlluminated} U (r: \text{RoomNotIlluminated} \text{ 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))
(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))
               (wc,p2)
More Examples

\begin{align*}
\text{rolland}\text{:} \text{WCNonEmptyRoute} \\
(\text{rolland}, p_2)\text{:} \text{at} \\
(\text{rolland}, p_3)\text{:} \text{WCNextPosition}
\end{align*}

\textbf{Progress:} \quad (\text{rolland}, p_1)\text{:} \text{at} \text{ U } (\text{rolland}, p_2)\text{:} \text{at} \\
\text{and } G(\text{all wc}\text{:} \text{WCNonEmptyRoute} \text{. WellBehaved(wc)} \text{U wc}\text{:} \text{WCEmptyRoute})

\begin{align*}
G(\text{all wc}\text{:} \text{WCNonEmptyRoute} \text{. WellBehaved(wc)}) \\
\text{WellBehaved(wc)} \overset{:=}{=} \forall p_1\text{:} \text{Position} \text{. } \forall p_2\text{:} \text{Position} \text{.} \\
(\text{wc}, p_1)\text{:} \text{at} \text{ and } (\text{wc}, p_2)\text{:} \text{WCNextPosition} \text{. } (\text{wc}, p_1)\text{:} \text{at} \text{ U } \\
(\text{wc}, p_2)
\end{align*}
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) ::= \forall p1:Position . \forall p2:Position .
(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))
(wc,p2)
rolland:WCEmptyRoute
(rolland,p3):at

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

G(all wc:WNCNonEmptyRoute . WellBehaved(wc))
WellBehaved(wc) ::= \forall p1:Position \forall p2:Position .
(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) ::= \forall p1:Position . \forall p2:Position .
   (wc,p2)
More Examples

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

Progress: (rolland,p2):at \(\bigcup\) (rolland,p3):at
and \(G(\text{all } wc:\text{WCNonEmptyRoute} . \text{WellBehaved}(wc) \bigcup wc:\text{WCEmptyRoute})\)
\[G(\text{all } wc:\text{WCNonEmptyRoute} . \text{WellBehaved}(wc))\]
WellBehaved(wc) \(:=\ \forall p1:\text{Position} . \forall p2:\text{Position} .\)
\((wc,p1):\text{at \ and } (wc,p2):\text{WCNextPosition} . (wc,p1):\text{at} \bigcup (wc,p2)\)
More Examples

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

Progress: False

G(all wc: WCNonEmptyRoute . WellBehaved(wc))
WellBehaved(wc) := \forall p1: Position . \forall p2: Position .
  (wc, p2)