

Agent Environments in the Ambient Event Calculus

Kostas Stathis

Department of Computer Science - DICE Lab
Royal Holloway, University of London, UK
`kostas.stathis@cs.rhul.ac.uk`

Keynote Speech at WOA2015
Naples, 17-18 June 2015

- 1 Introduction
- 2 Motivation
- 3 Modelling an Agent Environment
- 4 The Ambient Event Calculus
- 5 Applications
- 6 Conclusions

Welcome to dice**lab**



Lab Overview

The *distributed and intelligent computing environments* laboratory (dice*lab*) at Royal Holloway carries out research that spans theoretical and practical studies of complex applications by combining distributed systems techniques, intelligent agent models, and automated reasoning using computational logic.

Recent Posts

PhD student Bedour Alrayes
wins prize with CONAN
Kostas Stathis

Thursday, April 2nd, 2015

CONAN, a negotiation agent system developed in the DICE lab as part of Bedour M. Alrayes' PhD thesis, won second place in the Innovation and Entrepreneurship Prize for Saudi Students in the UK. The picture below show Mrs Alrayes after she has received the prize. Well done Bedour!

Workshop on Personal Health Systems for Chronic Diseases

Kostas Stathis

Saturday, February 21st, 2015

Workshop on Personal Health
Systems for Chronic Diseases
(PHSCD 2015) at Pervasive Health

Computational logic approach

Reasoning: Resolution, Unification & Backtracking

KB

```
should_take(Person, Drug) ←  
  complains_of(Person, Symptom),  
  surpresses(Drug, Symptom),  
  not unsuitable(Drug, Person).
```

```
unsuitable(Drug, Person) ←  
  aggravates(Drug, Condition),  
  suffers_from(Person, Condition).
```

```
surpresses(aspirin, pain).  
complains_of(peter, pain).  
aggravates(aspirin, peptic_ulcer).
```

$KB \vdash \text{should_take}(\text{peter}, \text{aspirin})$

is expressed as:

?- `should_take(peter, aspirin).`

Interpreter

```
solve(KB, not Goal) ←  
  not solve(KB, Goal).  
solve(KB, Goal) ←  
  rule(KB, Goal, SubGoals),  
  solveAll(KB, SubGoals).  
solve(KB, Goal) ←  
  askUser(Goal).
```

```
solveAll(_, []).  
solveAll(KB, [Goal | Goals]) ←  
  solve(KB, Goal),  
  solveAll(KB, Goals).
```

?- `solve('KB', should_take(peter, aspirin)).`

No: `if suffers(peter, peptic_ulcer)`

Yes: otherwise.

Computational logic approach

Reasoning: Resolution, Unification & Backtracking

KB

```
should_take(Person, Drug) ←  
  complains_of(Person, Symptom),  
  surpresses(Drug, Symptom),  
  not unsuitable(Drug, Person).
```

```
unsuitable(Drug, Person) ←  
  aggravates(Drug, Condition),  
  suffers_from(Person, Condition).
```

```
surpresses(aspirin, pain).  
complains_of(peter, pain).  
aggravates(aspirin, peptic_ulcer).
```

$KB \vdash \text{should_take}(\text{peter}, \text{aspirin})$

is expressed as:

?- `should_take(peter, aspirin).`

Interpreter

```
solve(KB, not Goal) ←  
  not solve(KB, Goal).  
solve(KB, Goal) ←  
  rule(KB, Goal, SubGoals),  
  solveAll(KB, SubGoals).  
solve(KB, Goal) ←  
  askUser(Goal).
```

```
solveAll(_, []).  
solveAll(KB, [Goal | Goals]) ←  
  solve(KB, Goal),  
  solveAll(KB, Goals).
```

?- `solve('KB', should_take(peter, aspirin)).`

No: `if suffers(peter, peptic_ulcer)`

Yes: otherwise.

kb

initially(rank(john) = lecturer).
initially(rank(mary) = ra).

next_rank(ra, lecturer).
next_rank(lecturer, senior_lecturer).
next_rank(senior_lecturer, reader).
next_rank(reader, professor).

initiates(promote(X), rank(X) = N, T) ←
holds_at(rank(X) = O, T),
next_rank(O, N).

happens_at(promote(mary), 1).
happens_at(promote(john), 3).
happens_at(promote(john), 4).

Event Calculus [8]

holds_at(F = V, T) ←
initially(F = V),
not broken(F = V, 0, T).
holds_at(F = V, Tn) ←
happens(Ei, Ti),
Ti < Tn,
initiates(Ei, F = V, Ti),
not broken(F = V, Ti, Tn).

broken(F=V, Ti, Tn) ←
happens(Ej, Tj),
Ti < Tj, Tj < Tn,
terminates(Ej, F=V, Tj).

terminates(E, F = V, T) ←
initiates(E, F = Vnew, T),
holds_at(F = V, T),
V ≠ Vnew.

kb

initially(rank(john) = lecturer).
initially(rank(mary) = ra).

next_rank(ra, lecturer).
next_rank(lecturer, senior_lecturer).
next_rank(senior_lecturer, reader).
next_rank(reader, professor).

initiates(promote(X), rank(X) = N, T) ←
holds_at(rank(X) = O, T),
next_rank(O, N).

happens_at(promote(mary), 1).
happens_at(promote(john), 3).
happens_at(promote(john), 4).

Event Calculus [8]

holds_at($F = V$, T) ←
 initially($F = V$),
 not broken($F = V$, 0, T).
holds_at($F = V$, T_n) ←
 happens(E_i , T_i),
 $T_i < T_n$,
 initiates(E_i , $F = V$, T_i),
 not broken($F = V$, T_i , T_n).

broken($F=V$, T_i , T_n) ←
 happens(E_j , T_j),
 $T_i < T_j$, $T_j < T_n$,
 terminates(E_j , $F=V$, T_j).

terminates(E , $F = V$, T) ←
 initiates(E , $F = V_{new}$, T),
 holds_at($F = V$, T),
 $V \neq V_{new}$.

kb

initially(rank(john) = lecturer).
initially(rank(mary) = ra).

next_rank(ra, lecturer).
next_rank(lecturer, senior_lecturer).
next_rank(senior_lecturer, reader).
next_rank(reader, professor).

initiates(promote(X), rank(X) = N, T) ←
holds_at(rank(X) = O, T),
next_rank(O, N).

happens_at(promote(mary), 1).
happens_at(promote(john), 3).
happens_at(promote(john), 4).

Event Calculus [8]

holds_at(F = V, T) ←
initially(F = V),
not broken(F = V, 0, T).
holds_at(F = V, T_n) ←
happens(E_i, T_i),
T_i < T_n,
initiates(E_i, F = V, T_i),
not broken(F = V, T_i, T_n).

broken(F=V, T_i, T_n) ←
happens(E_j, T_j),
T_i < T_j, T_j < T_n,
terminates(E_j, F=V, T_j).

terminates(E, F = V, T) ←
initiates(E, F = V_{new}, T),
holds_at(F = V, T),
V ≠ V_{new}.

kb

initially(rank(john) = lecturer).
initially(rank(mary) = ra).

next_rank(ra, lecturer).
next_rank(lecturer, senior_lecturer).
next_rank(senior_lecturer, reader).
next_rank(reader, professor).

initiates(promote(X), rank(X) = N, T) $\leftarrow$
holds_at(rank(X) = O, T),
next_rank(O, N).

happens_at(promote(mary), 1).
happens_at(promote(john), 3).
happens_at(promote(john), 4).

?- holds_at(rank(john) = X, 5).

X= reader.

Event Calculus [8]

holds_at(F = V, T) $\leftarrow$
initially(F = V),
not broken(F = V, 0, T).
holds_at(F = V, T_n) $\leftarrow$
happens(E_i, T_i),
T_i < T_n,
initiates(E_i, F = V, T_i),
not broken(F = V, T_i, T_n).

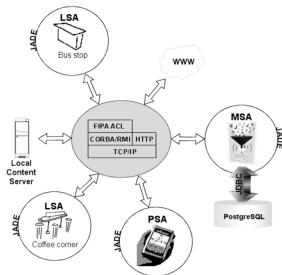
broken(F=V, T_i, T_n) $\leftarrow$
happens(E_j, T_j),
T_i < T_j, T_j < T_n,
terminates(E_j, F=V, T_j).

terminates(E, F = V, T) $\leftarrow$
initiates(E, F = V_{new}, T),
holds_at(F = V, T),
V $\neq$ V_{new}.

- 1 Introduction
- 2 Motivation**
- 3 Modelling an Agent Environment
- 4 The Ambient Event Calculus
- 5 Applications
- 6 Conclusions

The Living Memory Project

Agent technology for neighbourhoods (Stathis et al [11])



Social cohesion in a community:

- personal agents manage user's profiles and acquaintances
- system agents as disseminators of community events
- agents implemented in *Jade* containers [10]
- deployed in a community in Edinburgh

Jade Issues

Everything a reactive agent?
Directory facilitator unintuitive!

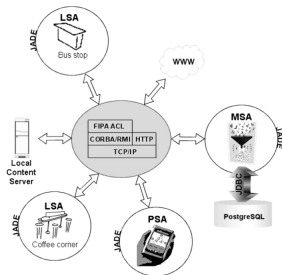
The Living Memory Project

Agent technology for neighbourhoods (Stathis et al [11])



Social cohesion in a community:

- personal agents manage user's profiles and acquaintances
- system agents as disseminators of community events
- agents implemented in *Jade* containers [10]
- deployed in a community in Edinburgh



Jade Issues

Everything a reactive agent?
Directory facilitator unintuitive!

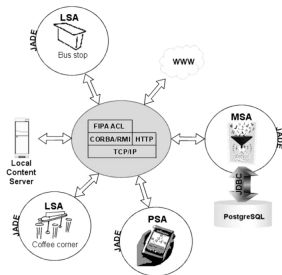
The Living Memory Project

Agent technology for neighbourhoods (Stathis et al [11])



Social cohesion in a community:

- personal agents manage user's profiles and acquaintances
- system agents as disseminators of community events
- agents implemented in *Jade* containers [10]
- deployed in a community in Edinburgh

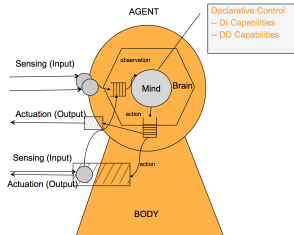


Jade Issues

Everything a reactive agent?
Directory facilitator unintuitive!

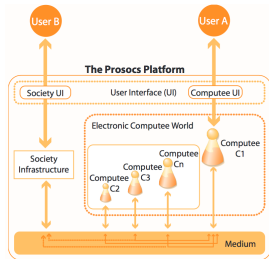
The SOCS Project

Societies of Logic-based Agents



Mind-body distinction (Bell [3]):

- “mind” with logical and planning capabilities [7]
- “body” can be discovered
- observations & actions as events [9]
- society infrastructure [1]
- P2P implementation [4]

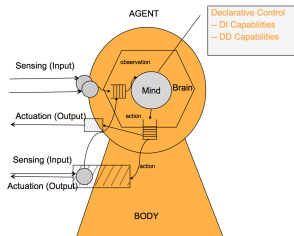


PROSOCS Issues

No explicit environment mediation.
Interaction with objects an issue.
Grounding.

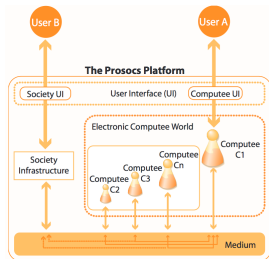
The SOCS Project

Societies of Logic-based Agents



Mind-body distinction (Bell [3]):

- “mind” with logical and planning capabilities [7]
- “body” can be discovered
- observations & actions as events [9]
- society infrastructure [1]
- P2P implementation [4]

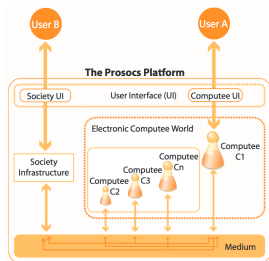
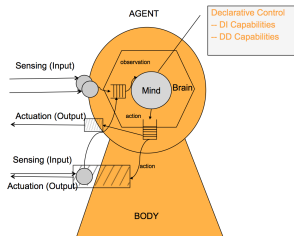


PROSOCS Issues

No explicit environment mediation.
Interaction with objects an issue.
Grounding.

The SOCS Project

Societies of Logic-based Agents



Mind-body distinction (Bell [3]):

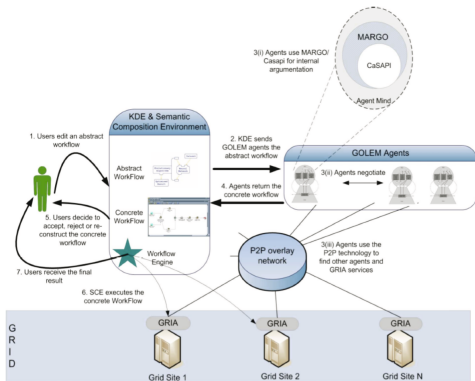
- “mind” with logical and planning capabilities [7]
- “body” can be discovered
- observations & actions as events [9]
- society infrastructure [1]
- P2P implementation [4]

PROSOCS Issues

No explicit environment mediation.
Interaction with objects an issue.
Grounding.

The ARGUGRID Project

Argumentation for GRID services



Agents operate services and manage contracts:

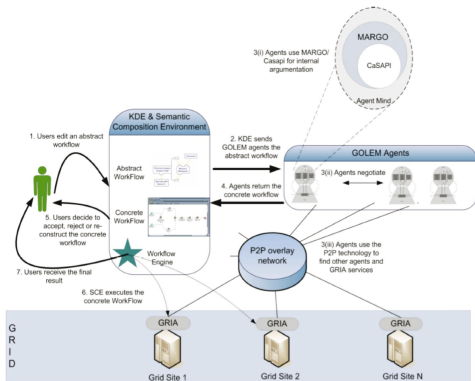
- “mind” with argumentation [7]
- interaction with objects
- explicit environment mediation
- PROSOCS $\Rightarrow$ GOLEM.

GOLEM Requirements

How do we model the environment?
How is interaction mediated?
Environment distribution?

The ARGUGRID Project

Argumentation for GRID services



Agents operate services and manage contracts:

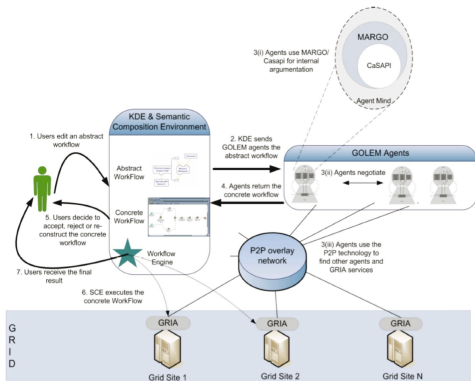
- “mind” with argumentation [7]
- interaction with objects
- explicit environment mediation
- PROSOCS $\Rightarrow$ GOLEM.

GOLEM Requirements

How do we model the environment?
How is interaction mediated?
Environment distribution?

The ARGUGRID Project

Argumentation for GRID services



Agents operate services and manage contracts:

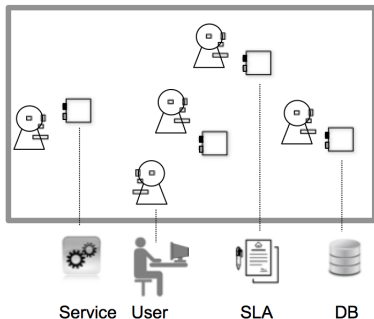
- “mind” with argumentation [7]
- interaction with objects
- explicit environment mediation
- PROSOCS $\Rightarrow$ GOLEM.

GOLEM Requirements

How do we model the environment?
How is interaction mediated?
Environment distribution?

- 1 Introduction
- 2 Motivation
- 3 Modelling an Agent Environment**
- 4 The Ambient Event Calculus
- 5 Applications
- 6 Conclusions

Container



A container is (a portion of) the environment.

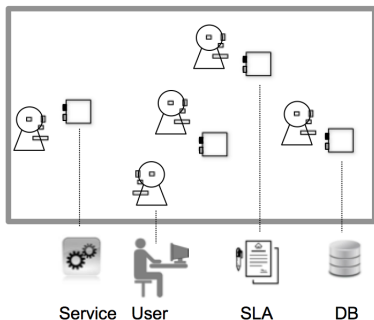
- Its state describes bodies of
 - agents (software/avatars)
 - objects (passive/active, internal/external)
- Mediates interaction between bodies.
- Can be open or closed.

Key issues

Approach, particularly choice of

- evolving container state
- mediated interaction

Container



External resources

A container is (a portion of) the environment.

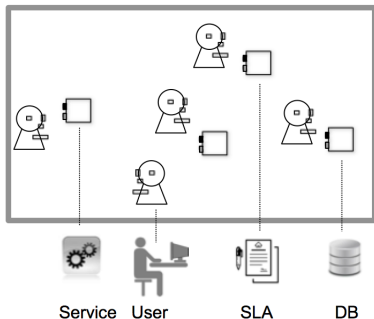
- Its state describes bodies of
 - agents (software/avatars)
 - objects (passive/active, internal/external)
- Mediates interaction between bodies.
- Can be open or closed.

Key issues

Approach, particularly choice of

- evolving container state
- mediated interaction

Container



External resources

A container is (a portion of) the environment.

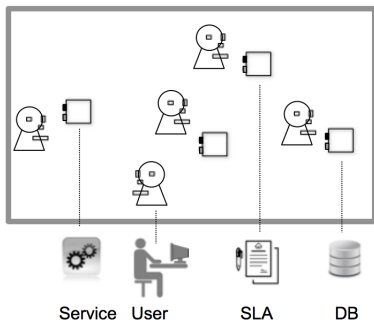
- Its state describes bodies of
 - agents (software/avatars)
 - objects (passive/active, internal/external)
- Mediates interaction between bodies.
- Can be open or closed.

Key issues

Approach, particularly choice of

- evolving container state
- mediated interaction

Container



External resources

Interactors are producers and consumers of **events**:

- sensors/actuators for agents
- triggers/emitters for active objects
- gateways for containers

State representation

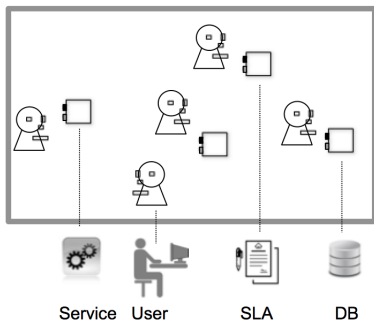
Temporal logic program

Physics = what mediates events

Use the Event Calculus

Environment as a Distributed Event Processing System (Bromuri & Stathis [5]).

Container



External resources

Interactors are producers and consumers of **events**:

- sensors/actuators for agents
- triggers/emitters for active objects
- gateways for containers

State representation

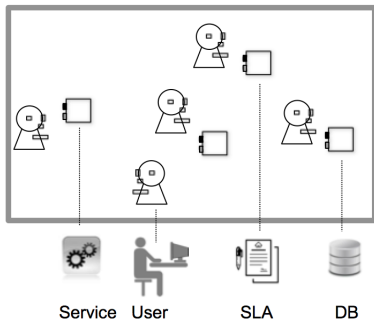
Temporal logic program

Physics = what mediates events

Use the Event Calculus

Environment as a Distributed Event Processing System (Bromuri & Stathis [5]).

Container



External resources

Interactors are producers and consumers of **events**:

- sensors/actuators for agents
- triggers/emitters for active objects
- gateways for containers

State representation

Temporal logic program

Physics = what mediates events

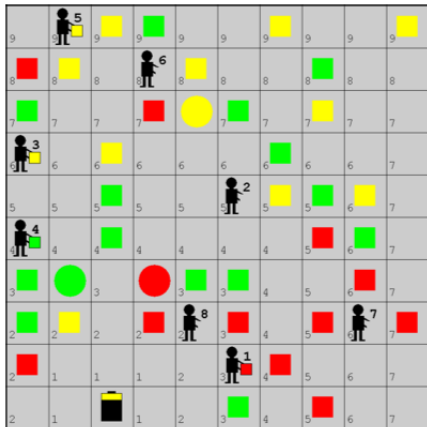
Use the Event Calculus

Environment as a Distributed Event Processing System (Bromuri & Stathis [5]).

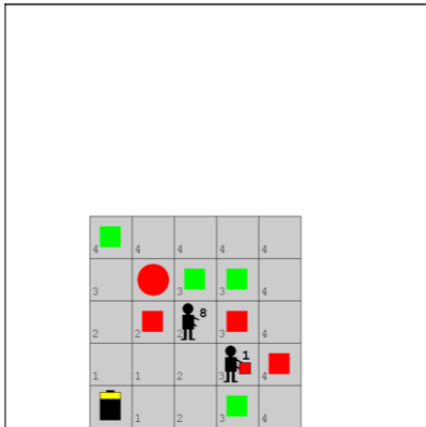
- 1 Introduction
- 2 Motivation
- 3 Modelling an Agent Environment
- 4 The Ambient Event Calculus**
- 5 Applications
- 6 Conclusions

Test-bed

Packet-World (Weyns [12])



(a) A Packet-World of 10x10.

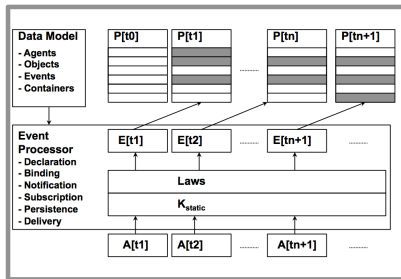


(b) Local view of agent 8.

Container State

Representation

Container



Data model:

```
class(agent).  
attr(agent, sensor_of, multi).  
attr(agent, location, single).  
...  
isa(X,Z) ← isa(X,Y), isa(Y,Z).  
...
```

Static knowledge:

```
isa(do, physical_act).  
isa(look, sensing_act).  
...  
location((0, 1)).  
location((0, 2)).  
...
```

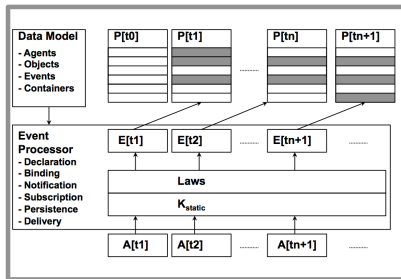
Initial state:

```
initially(agent(a1) = true).  
initially(location(a1) = (1, 1)).  
initially(sensor_of(a1) = s1).  
initially(listener(s1) = true).  
initially(status(s1) = open).  
...
```

Container State

Representation

Container



Data model:

```
class(agent).  
attr(agent, sensor_of, multi).  
attr(agent, location, single).  
...  
isa(X,Z) ← isa(X,Y), isa(Y,Z).  
...
```

Static knowledge:

```
isa(do, physical_act).  
isa(look, sensing_act).  
...  
location((0, 1)).  
location((0, 2)).  
...
```

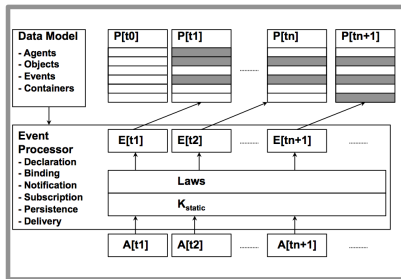
Initial state:

```
initially(agent(a1) = true).  
initially(location(a1) = (1, 1)).  
initially(sensor_of(a1) = s1).  
initially(listener(s1) = true).  
initially(status(s1) = open).  
...
```

Container State

Representation

Container



Data model:

```
class(agent).  
attr(agent, sensor_of, multi).  
attr(agent, location, single).  
...  
isa(X,Z) ← isa(X,Y), isa(Y,Z).  
...
```

Static knowledge:

```
isa(do, physical_act).  
isa(look, sensing_act).  
...  
location((0, 1)).  
location((0, 2)).  
...
```

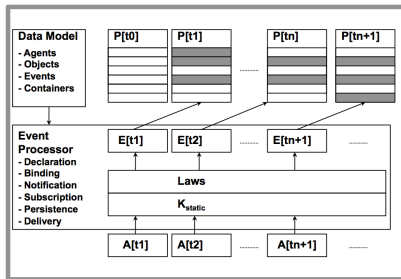
Initial state:

```
initially(agent(a1) = true).  
initially(location(a1) = (1, 1)).  
initially(sensor_of(a1) = s1).  
initially(listener(s1) = true).  
initially(status(s1) = open).  
...
```

Container State

Representation

Container



Data model:

```
class(agent).  
attr(agent, sensor_of, multi).  
attr(agent, location, single).  
...  
isa(X,Z) ← isa(X,Y), isa(Y,Z).  
...
```

Static knowledge:

```
isa(do, physical_act).  
isa(look, sensing_act).  
...  
location((0, 1)).  
location((0, 2)).  
...
```

Initial state:

```
initially(agent(a1) = true).  
initially(location(a1) = (1, 1)).  
initially(sensor_of(ag1) = s1).  
initially(listener(s1) = true).  
initially(status(s1) = open).  
...
```

Environment events

Event Declaration

Domain events are logical terms according to the data model e.g.:

do(Agent, move(ToB)).

Events are instantiated by agents actions as:

do(ag1, move((0, 2))).

Instances of events must be available in the container:

*available(do(Ag, move(ToB), T) ←
holds_at(agent(Ag) = true, T),
location(ToB)).*

Environment events

Event Declaration

Domain events are logical terms according to the data model e.g.:

do(Agent, move(ToB)).

Events are instantiated by agents actions as:

do(ag1, move((0, 2))).

Instances of events must be available in the container:

*available(do(Ag, move(ToB), T) ←
holds_at(agent(Ag) = true, T),
location(ToB).*

Environment events

Event Declaration

Domain events are logical terms according to the data model e.g.:

do(Agent, move(ToB)).

Events are instantiated by agents actions as:

do(ag1, move((0, 2))).

Instances of events must be available in the container:

*available(do(Ag, move(ToB), T) ←
holds_at(agent(Ag) = true, T),
location(ToB).*

Mediated occurrences

Action execution

Notion of attempts. They happen if they are valid:

$\text{happens}(\text{Event}, T) \leftarrow \text{attempt}(\text{Event}, T), \text{valid}(\text{Event}, T).$

Validity means that the event is physically feasible/executable:

$\text{valid}(\text{Event}, T) \leftarrow \text{available}(\text{Event}, T), \text{possible}(\text{Event}, T).$

Domain specific definitions of what is possible are written as:

*$\text{possible}(\text{do}(\text{Ag}, \text{move}(\text{ToB})), T) \leftarrow$
 $\text{holds_at}(\text{position}(\text{Ag}) = \text{FromA}, T),$
 $\text{within_boarder}(\text{ToB}),$
 $\text{adjacent}(\text{FromA}, \text{ToB}),$
 $\text{not_occupied}(\text{ToB}, T).$*

Mediated occurrences

Action execution

Notion of attempts. They happen if they are valid:

$\text{happens}(\text{Event}, T) \leftarrow \text{attempt}(\text{Event}, T), \text{valid}(\text{Event}, T).$

Validity means that the event is physically feasible/executable:

$\text{valid}(\text{Event}, T) \leftarrow \text{available}(\text{Event}, T), \text{possible}(\text{Event}, T).$

Domain specific definitions of what is possible are written as:

$\text{possible}(\text{do}(\text{Ag}, \text{move}(\text{ToB})), T) \leftarrow$
 $\text{holds_at}(\text{position}(\text{Ag}) = \text{FromA}, T),$
 $\text{within_boarder}(\text{ToB}),$
 $\text{adjacent}(\text{FromA}, \text{ToB}),$
 $\text{not_occupied}(\text{ToB}, T).$

Mediated occurrences

Action execution

Notion of attempts. They happen if they are valid:

$\text{happens}(\text{Event}, T) \leftarrow \text{attempt}(\text{Event}, T), \text{valid}(\text{Event}, T).$

Validity means that the event is physically feasible/executable:

$\text{valid}(\text{Event}, T) \leftarrow \text{available}(\text{Event}, T), \text{possible}(\text{Event}, T).$

Domain specific definitions of what is possible are written as:

*$\text{possible}(\text{do}(\text{Ag}, \text{move}(\text{ToB})), T) \leftarrow$
 $\text{holds_at}(\text{position}(\text{Ag}) = \text{FromA}, T),$
 $\text{within_boarder}(\text{ToB}),$
 $\text{adjacent}(\text{FromA}, \text{ToB}),$
 $\text{not_occupied}(\text{ToB}, T).$*

Mediated occurrences

Action execution

Notion of attempts. They happen if they are valid:

$$\text{happens}(\text{Event}, T) \leftarrow \text{attempt}(\text{Event}, T), \text{valid}(\text{Event}, T).$$

Validity means that the event is physically feasible/executable:

$$\text{valid}(\text{Event}, T) \leftarrow \text{available}(\text{Event}, T), \text{possible}(\text{Event}, T).$$

Domain specific definitions of what is possible are written as:

$$\begin{aligned} \text{possible}(\text{do}(\text{Ag}, \text{move}(\text{ToB})), T) \leftarrow \\ \text{holds_at}(\text{position}(\text{Ag}) = \text{FromA}, T), \\ \text{within_boarder}(\text{ToB}), \\ \text{adjacent}(\text{FromA}, \text{ToB}), \\ \text{not_occupied}(\text{ToB}, T). \end{aligned}$$

Mediated occurrences

Active sensing

Some events happens necessarily as a result of the environment:

$\text{happens}(\text{Event}, T) \leftarrow \text{necessary}(\text{Event}, T).$

We may define necessary events as:

$\text{necessary}(\text{Event}, T) \leftarrow$
 $\text{happens}(\text{Event}^*, T),$
 $\text{causes}(\text{Event}^*, \text{Event}, T),$
 $\text{available}(\text{Event}, T).$

E.g. Active sensing of an agent perceiving a specific environment property:

$\text{attempt}(\text{look}(\text{ag1}, s2, \text{pos}(\text{ag2})), 120).$

Environment's physics "response" is formulated as:

$\text{causes}(\text{look}(A, S, P), \text{see}(A, S, P = V), T) \leftarrow \text{holds_at}(P = V, T).$

Mediated occurrences

Active sensing

Some events happens necessarily as a result of the environment:

$\text{happens}(\text{Event}, T) \leftarrow \text{necessary}(\text{Event}, T).$

We may define necessary events as:

**$\text{necessary}(\text{Event}, T) \leftarrow$
 **$\text{happens}(\text{Event}^*, T),$
 **$\text{causes}(\text{Event}^*, \text{Event}, T),$
 $\text{available}(\text{Event}, T).$******

E.g. Active sensing of an agent perceiving a specific environment property:

$\text{attempt}(\text{look}(\text{ag1}, s2, \text{pos}(\text{ag2})), 120).$

Environment's physics "response" is formulated as:

$\text{causes}(\text{look}(A, S, P), \text{see}(A, S, P = V), T) \leftarrow \text{holds_at}(P = V, T).$

Mediated occurrences

Active sensing

Some events happens necessarily as a result of the environment:

$$\text{happens}(\text{Event}, T) \leftarrow \text{necessary}(\text{Event}, T).$$

We may define necessary events as:

$$\begin{aligned} \text{necessary}(\text{Event}, T) \leftarrow \\ & \text{happens}(\text{Event}^*, T), \\ & \text{causes}(\text{Event}^*, \text{Event}, T), \\ & \text{available}(\text{Event}, T). \end{aligned}$$

E.g. Active sensing of an agent perceiving a specific environment property:

$$\text{attempt}(\text{look}(\text{ag1}, s2, \text{pos}(\text{ag2})), 120).$$

Environment's physics “response” is formulated as:

$$\text{causes}(\text{look}(A, S, P), \text{see}(A, S, P = V), T) \leftarrow \text{holds_at}(P = V, T).$$

Mediated occurrences

Active sensing

Some events happens necessarily as a result of the environment:

$$\text{happens}(\text{Event}, T) \leftarrow \text{necessary}(\text{Event}, T).$$

We may define necessary events as:

$$\begin{aligned} \text{necessary}(\text{Event}, T) \leftarrow \\ & \text{happens}(\text{Event}^*, T), \\ & \text{causes}(\text{Event}^*, \text{Event}, T), \\ & \text{available}(\text{Event}, T). \end{aligned}$$

E.g. Active sensing of an agent perceiving a specific environment property:

$$\text{attempt}(\text{look}(\text{ag1}, s2, \text{pos}(\text{ag2})), 120).$$

Environment's physics “response” is formulated as:

$$\text{causes}(\text{look}(A, S, P), \text{see}(A, S, P = V), T) \leftarrow \text{holds_at}(P = V, T).$$

Mediated occurrences

Event subscription

How to link happenings of events to sensors:

- speech acts - simulate “audioception”;
- sensing and physical acts - simulate “ophthalmoception”.

Default subscription to events:

```
subscribes(S, E, T) ←  
  holds_at(senses(S) = EventClass, T),  
  instance_of(E, EventSubClass),  
  isa(EventSubClass, EventClass),  
  holds_at(status(S) = open, T).
```

E.g. Requests:

```
instance_of(request(−, −, −), request).  
isa(request, speech_act).
```

Mediated occurrences

Event subscription

How to link happenings of events to sensors:

- speech acts - simulate “audioception”;
- sensing and physical acts - simulate “ophthalmoception”.

Default subscription to events:

```
subscribes(S, E, T) ←  
  holds_at(senses(S) = EventClass), T),  
  instance_of(E, EventSubClass),  
  isa(EventSubClass, EventClass),  
  holds_at(status(S) = open, T).
```

E.g. Requests:

```
instance_of(request(−, −, −), request).  
isa(request, speech_act).
```

Mediated occurrences

Event subscription

How to link happenings of events to sensors:

- speech acts - simulate “audioception”;
- sensing and physical acts - simulate “ophthalmoception”.

Default subscription to events:

```
subscribes(S, E, T) ←  
  holds_at(senses(S) = EventClass, T),  
  instance_of(E, EventSubClass),  
  isa(EventSubClass, EventClass),  
  holds_at(status(S) = open, T).
```

E.g. Requests:

```
instance_of(request(-, -, -), request).  
isa(request, speech_act).
```

Event Notification

Passive perception

Events are notified to sensors once they happen:

notify(E, S, T) $\leftarrow$ happens(E, T), detectable(E, S, T).

**detectable(E, S, T) $\leftarrow$
subscribes(S, E, T), perceivable(E, S, T), not interfered(E, S, T).**

An interfering entity between two agents in packet-world:

*interfered(E, S, T) $\leftarrow$
instance_of(E, physical_act),
actor_of(E, AgentB),
holds_at(position(AgentB) = PosB, T),
holds_at(sensor(AgentA) = S, T),
holds_at(position(Entity) = PosE, T),
in_between(PosE, AgentA, AgentB).*

Event Notification

Passive perception

Events are notified to sensors once they happen:

notify(E, S, T) $\leftarrow$ happens(E, T), detectable(E, S, T).

**detectable(E, S, T) $\leftarrow$
subscribes(S, E, T), perceivable(E, S, T), not interfered(E, S, T).**

An interfering entity between two agents in packet-world:

*interfered(E, S, T) $\leftarrow$
instance_of(E, physical_act),
actor_of(E, AgentB),
holds_at(position(AgentB) = PosB, T),
holds_at(sensor(AgentA) = S, T),
holds_at(position(Entity) = PosE, T),
in_between(PosE, AgentA, AgentB).*

Event Notification

Passive perception

Events are notified to sensors once they happen:

$\text{notify}(E, S, T) \leftarrow \text{happens}(E, T), \text{detectable}(E, S, T).$

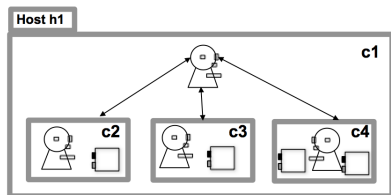
$\text{detectable}(E, S, T) \leftarrow$
 $\text{subscribes}(S, E, T), \text{perceivable}(E, S, T), \text{not interfered}(E, S, T).$

An interfering entity between two agents in packet-world:

$\text{interfered}(E, S, T) \leftarrow$
 $\text{instance_of}(E, \text{physical_act}),$
 $\text{actor_of}(E, \text{Agent}B),$
 $\text{holds_at}(\text{position}(\text{Agent}B) = \text{Pos}B, T),$
 $\text{holds_at}(\text{sensor}(\text{Agent}A) = S, T),$
 $\text{holds_at}(\text{position}(\text{Entity}) = \text{Pos}E, T),$
 $\text{in_between}(\text{Pos}E, \text{Agent}A, \text{Agent}B).$

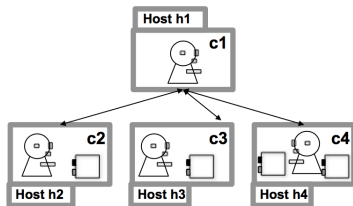
Distributing containers (1)

Super/sub containers



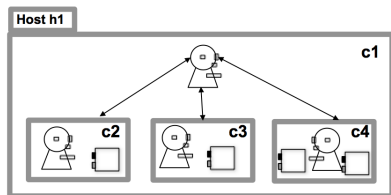
locally_at/4:

```
locally_at(C, Path, [C | Path], Query) ←  
    not member(C, Path),  
    solve(C, Query).  
  
locally_at(C, Path, Path*, Query) ←  
    time_of(Query, T),  
    solve(C, holds_at(sub_container(C)=Ci, T)),  
    locally_at(Ci, [C | Path], Path*, Query).
```



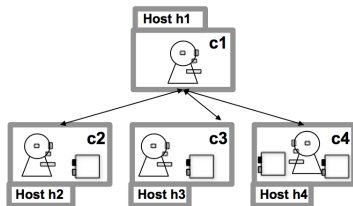
Distributing containers (1)

Super/sub containers



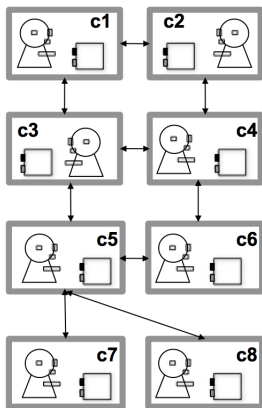
locally_at/4:

```
locally_at(C, Path, [C | Path], Query) ←  
    not member(C, Path),  
    solve(C, Query).  
  
locally_at(C, Path, Path*, Query) ←  
    time_of(Query, T),  
    solve(C, holds_at(sub_container(C)=Ci, T)),  
    locally_at(Ci, [C | Path], Path*, Query).
```



Distributing containers (2)

Neighbourhoods

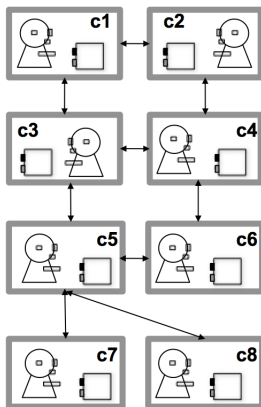


neighbouring_at/5:

```
neighbouring_at(C, Path, Path*, Max, Query) ←  
  Max ≥ 0,  
  locally_at(C, Path, Path*, Query).  
neighbouring_at(C, Path, Path*, Max, Query) ←  
  Max > 0,  
  solve(C, holds_at(neighbour(C)=Ni, T)),  
  not member(Ni, Path),  
  Max* is Max - 1,  
  neighbouring_at(Ni, [C | Path], Path*, Max*, Query).
```

Distributing containers (2)

Neighbourhoods

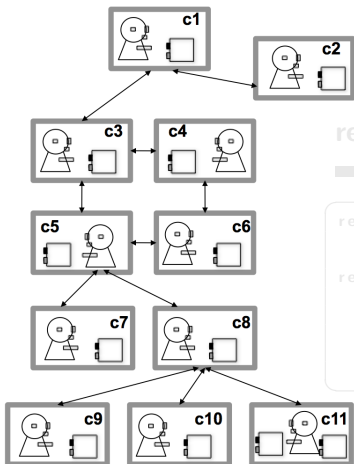


neighbouring_at/5:

```
neighbouring_at(C, Path, Path*, Max, Query) ←  
    Max >= 0,  
    locally_at(C, Path, Path*, Query).  
neighbouring_at(C, Path, Path*, Max, Query) ←  
    Max > 0,  
    solve(C, holds_at(neighbour(C)=Ni, T)),  
    not member(Ni, Path),  
    Max* is Max - 1,  
    neighbouring_at(Ni, [C | Path], Path*, Max*, Query).
```

Distributing containers (3)

Regions

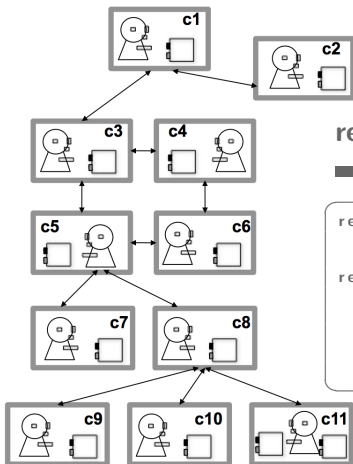


regionally_at/6:

```
regionally_at(C, Path, Path*, MaxH, MaxW, Query) ←  
    MaxH >= 0,  
    neighbouring_at(C, Path, Path*, MaxW, Query).  
regionally_at(C, Path, Path*, MaxH, MaxW, Query) ←  
    MaxH > 0,  
    holds_at(super_container(C)=Si, T),  
    not_member(Si, Path),  
    MaxH* = MaxH - 1,  
    regionally_at(Si, [C|Path], Path*, MaxH*, MaxW, Query)
```

Distributing containers (3)

Regions



regionally_at/6:

```
regionally_at(C, Path, Path*, MaxH, MaxW, Query) ←  
    MaxH >= 0,  
    neighbouring_at(C, Path, Path*, MaxW, Query).  
regionally_at(C, Path, Path*, MaxH, MaxW, Query) ←  
    MaxH > 0,  
    holds_at(super_container(C)=Si, T),  
    not_member(Si, Path),  
    MaxH* = MaxH - 1,  
    regionally_at(Si, [C|Path], Path*, MaxH*, MaxW, Query)
```

Distributing containers (4)

Example of distributed physics

Picking up a packet:



$possible(do(A, pick(Packet)), T) \leftarrow$
 $holds_at(hand(A) = free, T),$
 $holds_at(position(A) = PosA, T),$
 $neighbouring_at(this, [], -, 1, holds_at(packet(Packet) = PosB, T)),$
 $adjacent(PosA, PosB).$

Distributing containers (4)

Example of distributed physics

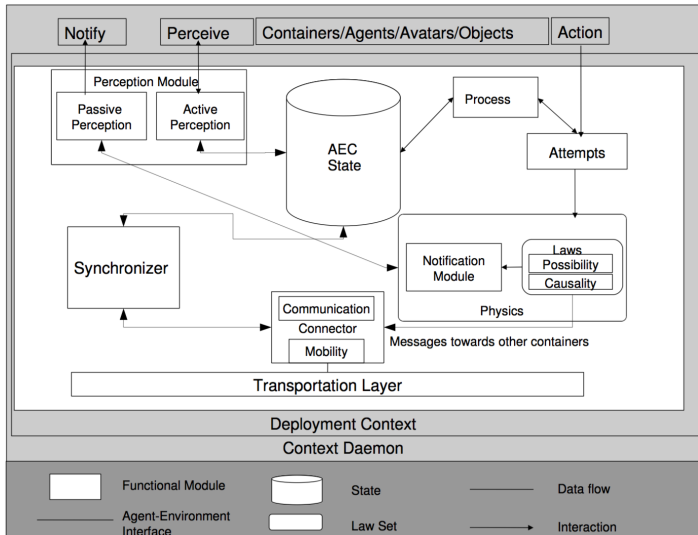
Picking up a packet:



$possible(do(A, pick(Packet)), T) \leftarrow$
 $holds_at(hand(A) = free, T),$
 $holds_at(position(A) = PosA, T),$
 $neighbouring_at(this, [], -, 1, holds_at(packet(Packet) = PosB, T)),$
 $adjacent(PosA, PosB).$

Implementation Issues

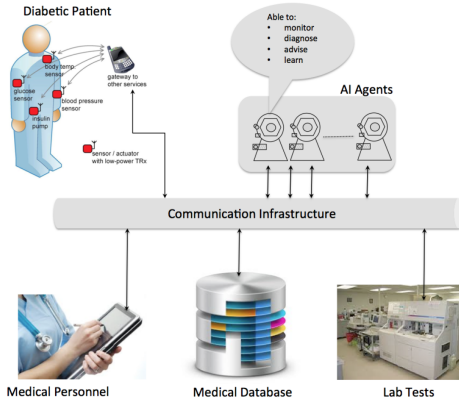
GOLEM Reference Model [5]



- 1 Introduction
- 2 Motivation
- 3 Modelling an Agent Environment
- 4 The Ambient Event Calculus
- 5 Applications**
- 6 Conclusions

The COMMODITY₁₂ Project

Personal Health System for Diabetes



Agents support a variety of tasks:

- activity recognition, monitoring and online advice to patient
- recommendations to doctors
- model based diagnosis vs data driven predictions

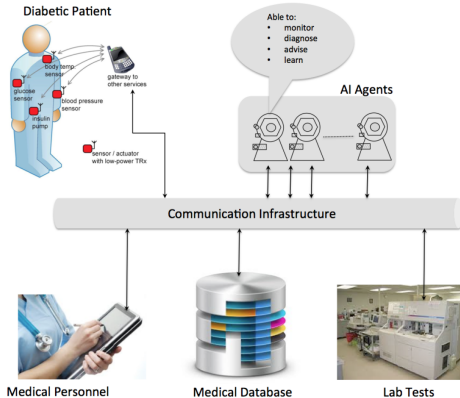
Agent Environment Issues

Linking agent and physical environment

- grounding agents
- what approach is best to take?

The COMMODITY₁₂ Project

Personal Health System for Diabetes



Agents support a variety of tasks:

- activity recognition, monitoring and online advice to patient
- recommendations to doctors
- model based diagnosis vs data driven predictions

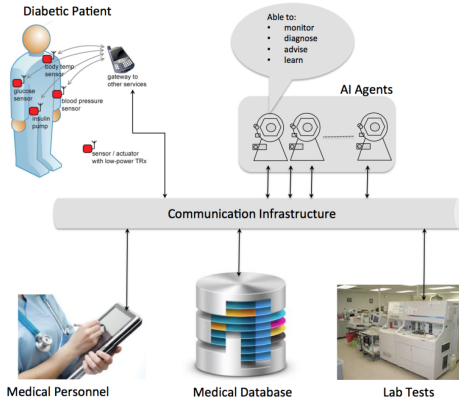
Agent Environment Issues

Linking agent and physical environment

- grounding agents
- what approach is best to take?

The COMMODITY₁₂ Project

Personal Health System for Diabetes



Agents support a variety of tasks:

- activity recognition, monitoring and online advice to patient
- recommendations to doctors
- model based diagnosis vs data driven predictions

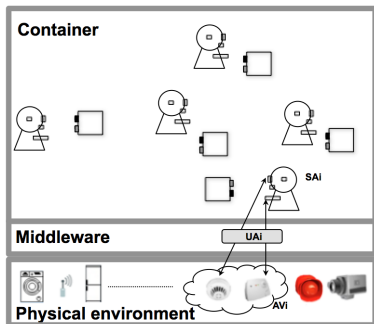
Agent Environment Issues

Linking agent and physical environment

- grounding agents
- what approach is best to take?

Cyber-Physical Environments

Ubiquitous agents



Ambient Intelligence & IoT:

- $UI_i = SA_i + AV_i$
- BIL - *body interaction language*
- Service-oriented middleware
EVATAR [6]

Key Issues

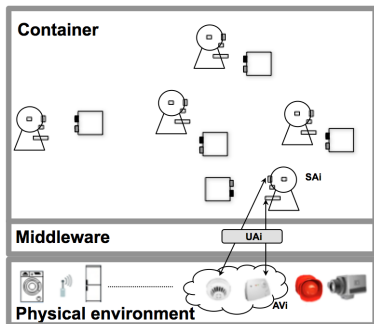
Mapping between environments e.g.:

- Robots as Agent Avatars?
- Agents as Robot Avatars?



Cyber-Physical Environments

Ubiquitous agents



Ambient Intelligence & IoT:

- $UI_i = SA_i + AV_i$
- BIL - *body interaction language*
- Service-oriented middleware
EVATAR [6]

Key Issues

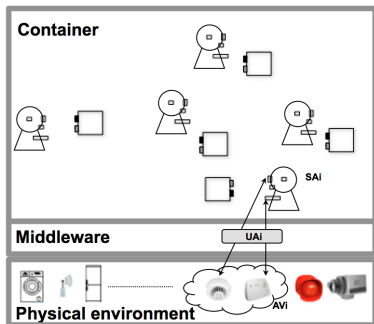
Mapping between environments e.g.:

- Robots as Agent Avatars?
- Agents as Robot Avatars?



Cyber-Physical Environments

Ubiquitous agents



Ambient Intelligence & IoT:

- $UI_i = SA_i + AV_i$
- BIL - *body interaction language*
- Service-oriented middleware
EVATAR [6]

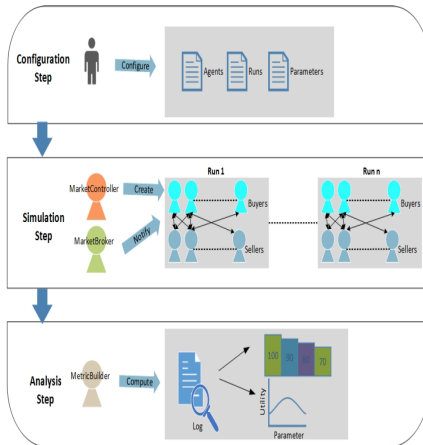
Key Issues

Mapping between environments e.g.:

- Robots as Agent Avatars?
- Agents as Robot Avatars?

Large-scale simulations

Negotiation in Electronic Markets



Experimental Evaluation of Concurrent Negotiation Strategies:

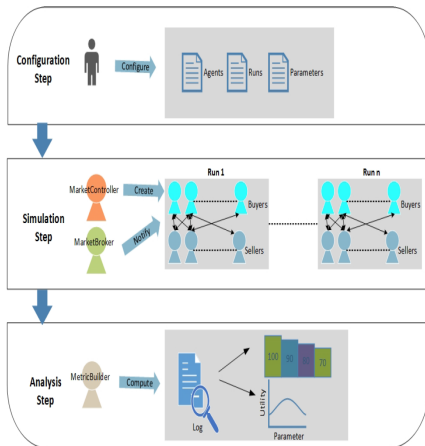
- Simulation of an Electronic Market (e.g. E-Bay);
- Large no. of agents & models;
- RECON: Specialisation of GOLEM [2];
- Simulation vs deployment.

Key Issues

Stochastic Environments e.g.:
– Analytics & UIs

Large-scale simulations

Negotiation in Electronic Markets



Experimental Evaluation of Concurrent Negotiation Strategies:

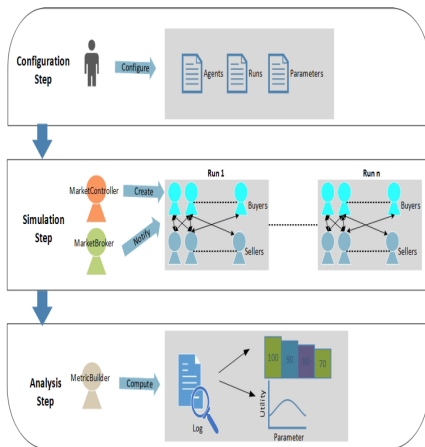
- Simulation of an Electronic Market (e.g. E-Bay);
- Large no. of agents & models;
- RECON: Specialisation of GOLEM [2];
- Simulation vs deployment.

Key Issues

Stochastic Environments e.g.:
– Analytics & UIs

Large-scale simulations

Negotiation in Electronic Markets



Experimental Evaluation of Concurrent Negotiation Strategies:

- Simulation of an Electronic Market (e.g. E-Bay);
- Large no. of agents & models;
- RECON: Specialisation of GOLEM [2];
- Simulation vs deployment.

Key Issues

Stochastic Environments e.g.:
– Analytics & UIs

Outline

- 1 Introduction
- 2 Motivation
- 3 Modelling an Agent Environment
- 4 The Ambient Event Calculus
- 5 Applications
- 6 Conclusions**

Ambient Event Calculus

- Framework for specifying **agent environments**
- Abstractions to model the environment in terms of
 - **containers, agents, objects, events** and **physics**
- Underlying **distributed event processing** for interactions
- Implementation in **GOLEM**

Challenges

- **Hybrid languages** for Agent Environments
- **Openness, Reuse** and **Plug-and-play**
- **Managing large scale**
- **Applications** (e.g. serious games in mixed realities)



Marco Alberti et al. “Compliance verification of agent interaction: a logic-based software tool”. In: *Applied Artificial Intelligence* 20.2-4 (2006), pp. 133–157.



Bedour Alrayes, Ozgur Kafali, and Kostas Stathis. “RECON: a Robust multi-agent Environment for simulating CONcurrent Negotiations”. In: *In Seventh International Workshop on Agent-based Complex Automated Negotiations (ACAN), AAMAS*. 2014.



John Bell. “A Planning Theory of Practical Rationality”. In: *Proceedings of AAAI’95 Fall Symposium on Rational Agency*. 1995, pp. 1–4.



Andrea Bracciali et al. “Crafting the mind of PROSOCS agents”. In: *Applied Artificial Intelligence* 20.2-4 (2006), pp. 105–131.



Stefano Bromuri and Kostas Stathis. “Distributed Agent Environments in the Ambient Event Calculus”. In: *3rd ACM International Conference on Distributed Event-Based Systems (DEBS’09)*. 2009, pp. 1–12.



Nikolaos Dipsis and Kostas Stathis. “Ubiquitous Agents for Ambient Ecologies”. In: *Pervasive and Mobile Computing* 8.4 (2012), pp. 562–574.



Antonis C. Kakas et al. “Computational Logic Foundations of KGP Agents”. In: *J. Artif. Intell. Res. (JAIR)* 33 (2008), pp. 285–348.



R Kowalski and M Sergot. “A Logic-based Calculus of Events”. In: *New Gen. Comput.* 4.1 (Jan. 1986).



Anand S. Rao. “AgentSpeak(L): BDI Agents Speak out in a Logical Computable Language”. In: *Proceedings of the 7th European Workshop on Modelling Autonomous Agents in a Multi-agent World :Agents Breaking Away*. MAAMAW '96. Eindhoven, The Netherlands: Springer-Verlag New York, Inc., 1996, pp. 42–55.



Giovanni Rimassa. “Runtime support for distributed multi-agent systems”. PhD thesis. Università degli Studi di Parma, 2003.



Kostas Stathis, Oscar de Bruijn, and Silvio Macedo. “Living memory: agent-based information management for connected local communities”. In: *Interacting with Computers* 14.6 (2002), pp. 663–688.



Danny Weyns, Alexander Helleboogh, and Tom Holvoet. “The Packet-World: A Test Bed for Investigating Situated Multi-Agent Systems”. In: *Software Agent-Based Applications, Platforms and Development Kits*. Ed. by Rainer Unland, Monique Calisti, and Matthias Klusch. Whitestein Series in Software Agent Technologies. Birkhuser Basel, 2005, pp. 383–408.