Autonomic Multi-Agent Systems
Awareness and Self-Awareness in Autonomic Systems
Academic Course; September; 2012

Jeremy Pitt
Imperial College London
Lecture Objectives and Outcomes

- Objectives
  - objectives ...

- Outcomes – lecture attendees will:
  - Understand the fundamental properties of autonomic multi-agent systems, different approaches, and how categorise/evaluate them
  - Understand how agents, embedded in an environment, can use awareness of that environment for self-organisation (a form of self-aware autonomics)
Lecture Structure

- Multi-Agent Systems
- Autonomic Systems
- Representative Approaches
- In detail: Dynamic norm-governed systems
- Voting (awareness in autonomic systems)
Multi-Agent Systems

Two key considerations

- Level of abstraction
- Type of communication
Abstraction (1)

Multi-Agent abstraction

- Distributed systems
  - physical distribution of data and methods
  - tightly coupled
  - location transparency

- Multi-agent systems
  - logical distribution of responsibility and control
  - loosely coupled
  - location significant

- Organization of collective intelligence for:
  - functionality and knowledge encapsulation (ontology)
  - cooperation, delegation and negotiation
  - planning and decision making wrt. own goals

Only self-modify the logical layer/virtual machine
Abstraction (2)

Intelligent Agent abstraction

- Highest level of abstraction
  - ownership: delegated responsibility for task
  - internal states: large number exhibit some form of ‘intelligent’ behaviour
  - asynchrony: communication and coordination with other agents/users

- Lowest level of abstraction
  - embedded software process
  - encapsulates some notion of state
  - communicates by message passing

The nature of the message passing is absolutely critical
How To Do Things . . .

(... if you absolutely have to . . .)

- . . . with physical objects
  - Change the state of the physical world
  - Given the ‘ideal’ physics (physical capability)
- . . . with software objects
  - Change the state of the object
  - Given ‘appropriate’ programming language semantics and the semantics of the call
- . . . with words
  - Change the state of the the conventional world
  - Given ‘validity’ of the of the action (*institutionalised power*)
Institutionalised Power

(Not to be confused with real or reactive power)

- Searle: counts as in Speech act Theory
- Jones and Sergot (1996): Formal Characterisation of...
- ... with software objects
  - Change the state of the object
  - Given ‘appropriate’ programming language semantics and the semantics of the call
- A standard feature of any norm-governed system whereby designated agents, when acting in specified roles, are empowered by the system to create or modify facts of special significance conventionally agreed within the context an institution
- This matters, especially in socio-technical systems, electronic institutions, and self-organising systems
Agent-Oriented Software Engineering

Many methodologies, languages and platforms

- GAIA, etc.
- PRS, Jack, Jam, Jason, etc.
- Jade, etc.

FIPA

- FIPA ACL: could not be more wrong
Autonomic Systems: An Overview

Systems are expected to show some kind of ‘agency’ / autonomic behaviour

- Sensor networks – monitor environment, respond to events, situations, ...
- Data centres – monitor and respond to changing/expected workloads
- Power systems – maintain frequency

Systems need to exhibit some sort of ‘autonomic’ properties

- Improve some facet of behaviour
- ... which must be the expected facet
- ... and which must change in an expected way
Adaptation

Complexity of built systems

- Change dynamics (due to Bertrand Meyer)
  - Linear – cost proportional to size of change being made
  - Non-linear – cost proportional to size of system being changed
- Complexity and inter-dependence make all changes non-linear
- Become too expensive to change in any significant respect

Incompatible with human management

- Change too fast, too frequent, too far away, too tangled, ... 
- Responses need to be guaranteed ... 
- ... But may need to represent human artefacts – contracts, laws, conventional rules, etc.
Autonomic Systems

Originally an IBM notion focusing on TCO

- Inspiration from the biological autonomic nervous system: things keep breathing, (almost) no matter what
- Since broadened to include systems with sensorised, feedback-driven management
  - Use technology to manage technology

Significant Approaches

- IBM Autonomic Toolkit
- Morphogen-gradient methods use biological inspiration to manage dynamic (especially mobile) services
- Dynamical systems models treat adaptation as movement within an adaptive space
- Multi-agent systems using qualitative knowledge representation and reasoning for multi-criteria optimisation
Autonomic Control Loop

Stages in Autonomic Control

- Collect data from sensors or instrumented managed elements
- Analyse using some model of the system being managed
- Decide on actions taken to accomplish control goal
- Act on decision, possibly with learning, feedback, etc
Example: Data Centre Manager

Power-saving

- Assign agents to monitor and manage components
- Global utility function for different configurations
- Combine aspects of system into a single utility which is then maximised
- Single (Global) Variable/Diverse (Local) Objectives
(Some More) Self-* properties

“Things the machine can (should) do (for/to) itself”

- Self-configuring  select configuration options, policies and interaction with other components automatically (or at least make suggestions)
- Self-managing  perform ‘housekeeping’ operations automatically
- Self-optimising  reflect on own behaviour and adjust configuration accordingly
- Self-healing  respond to component failures or attacks to minimise damage
- Self-organisation  manage the orchestration between components adaptively
Representative Approaches

- Max-flow Networks
- Unity
- OMACS (Organisational Model for Adaptive Computational Systems)
- Adaptive Decision-Making Frameworks
- Dynamic Argument Systems
- Organic Computing
- Law-Governed Interaction
- Dynamic Norm-Governed Systems
Analytic Framework

- *What is adapted* – identifying the specific changeable components.
- *Why is adaptation performed* – the motivating cause(s), in response to exogenous or endogenous stimuli, which prompt adaptation.
- *How is adaptation performed* – detailing the specific mechanisms or processes used to bring about the adaptation.
- *Evaluation of adaptation* – how the adapted system is evaluated as an improvement (or not) on the previous configuration.
Dynamic Norm-governed Systems

Dynamic Norm-Governed Multi-Agent Systems

- Social Constraints
  - Physical power, institutionalised power, and permission
  - Obligations, and other complex normative relations
  - Sanctions and penalties
  - Roles and actions (communication language)

- Communication Protocols
  - Protocol stack: object-/meta-/meta-meta-/etc. level protocols
  - Transition protocols to instigate and implement change

- Specification Space
  - Identify changeable components of a specification (Degrees of Freedom: DoF)
  - Define a ‘space’ of specification instances, and a notion of distance
  - Define rules about moving between instances
Social Constraints

- Three types of ‘can’
  - Physical capability
  - Institutional power
    - The performance by a designated agent, occupying a specific role, of a certain action, which has conventional significance, in the context of an institution
    - A special kind of ‘certain action’ is the *speech act*
  - Permission (& obligation)
  - Can have (physical or institutional) power with/without permission
    - Sometimes power implies permission
- Sanctions and enforcement policies
- Right, duty, entitlement, and other more complex relations
- Social constraints can be adapted for intentional, run-time modification of the institution
Any protocol for norm-governed systems can be in level 0.

Any protocol for decision-making over rule modification can be in level \( n, n > 0 \).

Attention is also payed to the transition protocols: the procedures with which a meta-protocol is initiated.
We define the **Degrees of Freedom (DoF)** of a protocol.

A protocol specification with $n$ DoF creates an $n$-dimensional specification space, where each dimension corresponds to a DoF.

A specification point represents a complete protocol specification — a specification instance — and is denoted by a $n$-tuple, where each element of the tuple expresses the value of a DoF.
Awareness and Self-healing

Congruence of appropriation/provision rules and state

- Interleave rules of social-\{order | exchange | choice\}
  - Brute facts (resource level in past and current state)
  - Individual beliefs (resource level in future states)
  - Common beliefs formed by *gossiping* (opinion formation)
  - Expressed preferences mapped to collective choice
  - Collective choice represented as institutional fact
Voting Protocol: Informal Description

- Informal specification of a decision-making procedure according to *Robert’s Rules of Order (Newly Revised)*
  - a committee meets and the chair opens a session
  - a committee member requests and is granted the floor
  - that member proposes a motion
  - another member seconds the motion
  - the members debate the motion
  - the chair calls for those in favour to cast their vote
  - the chair calls for those against to cast their vote
  - the motion is carried or not, according to the standing rules of the committee
Voting Protocol: Graphical Description

- Various options for graphical representation
  - UML Sequence diagrams
  - State diagrams

- Note certain simplifications to RONR specification
  - No floor request, debate or agenda
  - Voting, changing of votes etc., concurrently
An Event Calculus Specification

- Basic Items: Events and Fluents
- Institutional Powers
- Voting and Counting Votes
- Permission and Obligation
- Sanctions
- Objection
## Actions

<table>
<thead>
<tr>
<th>Action</th>
<th>Indicating...</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>open_session(Ag, S)</code></td>
<td>open and close a session</td>
</tr>
<tr>
<td><code>close_session(Ag, S)</code></td>
<td></td>
</tr>
<tr>
<td><code>propose(Ag, M)</code></td>
<td>propose and second a motion</td>
</tr>
<tr>
<td><code>second(Ag, M)</code></td>
<td></td>
</tr>
<tr>
<td><code>open_ballot(Ag, M)</code></td>
<td>open and close a ballot</td>
</tr>
<tr>
<td><code>close_ballot(Ag, M)</code></td>
<td></td>
</tr>
<tr>
<td><code>vote(Ag, M, aye)</code></td>
<td>vote for or against a motion, abstain or change vote</td>
</tr>
<tr>
<td><code>vote(Ag, M, nay)</code></td>
<td></td>
</tr>
<tr>
<td><code>abstain(Ag, M)</code></td>
<td></td>
</tr>
<tr>
<td><code>revoke(Ag, M)</code></td>
<td></td>
</tr>
<tr>
<td><code>declare(Ag, M, carried)</code></td>
<td>declare the result of a vote</td>
</tr>
<tr>
<td><code>declare(Ag, M, not_carried)</code></td>
<td></td>
</tr>
</tbody>
</table>
### Fluents

<table>
<thead>
<tr>
<th>Fluent</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>sitting($S$)</td>
<td>boolean</td>
</tr>
<tr>
<td>status($M$)</td>
<td>{ pending, proposed, seconded voting($T$), voted, resolved }</td>
</tr>
<tr>
<td>votes($M$)</td>
<td>$N \times N$</td>
</tr>
<tr>
<td>voted($Ag$, $M$)</td>
<td>{ nil, aye, nay, abs }</td>
</tr>
<tr>
<td>resolutions($S$)</td>
<td>list of motions</td>
</tr>
<tr>
<td>qualifies($Ag$, $R$)</td>
<td>boolean</td>
</tr>
<tr>
<td>role_of($Ag$, $R$)</td>
<td>boolean</td>
</tr>
<tr>
<td>pow($Ag$, $Act$)</td>
<td>boolean</td>
</tr>
<tr>
<td>per($Ag$, $Act$)</td>
<td>boolean</td>
</tr>
<tr>
<td>obl($Ag$, $Act$)</td>
<td>boolean</td>
</tr>
<tr>
<td>sanction($Ag$)</td>
<td>list of integers</td>
</tr>
</tbody>
</table>
Recall: an empowered agent performs a designated action in context which creates or changes an institutional fact.

We want to express the effects of the designated protocol (speech) actions, in particular:

- vote
- open_session and open_ballot
- declare

For the specification of the effects of these actions, it is important to distinguish between:

- the act of (‘successfully’) casting a vote, and
- the act by means of which the casting of the vote is signalled (e.g. sending a message of a particular form via a TCP/IP socket connection).
Institutional Power

- Institutional power to open the ballot on a motion:

  \[
  \text{pow}(C, \text{open\_ballot}(C, M)) = true \quad \text{holdsat} \quad T \leftarrow \\
  \text{status}(M) = seconded \quad \text{holdsat} \quad T \quad \land \\
  \text{role\_of}(C, \text{chair}) = true \quad \text{holdsat} \quad T
  \]

- Institutional power to cast a vote:

  \[
  \text{pow}(V, \text{vote}(V, M, \_)) = true \quad \text{holdsat} \quad T \leftarrow \\
  \text{status}(M) = \text{voting}(\_) \quad \text{holdsat} \quad T \quad \land \\
  \text{role\_of}(V, \text{voter}) = true \quad \text{holdsat} \quad T \quad \land \\
  \text{not} \quad \text{role\_of}(V, \text{chair}) = true \quad \text{holdsat} \quad T \quad \land \\
  \text{voted}(V, M) = \text{nil} \quad \text{holdsat} \quad T
  \]
Effects of Institutional Power (1)

- Chair performs \( \text{open\_ballot}(C, M) \)
  \[ \text{open\_ballot}(C, M) \text{ initiates } \text{votes}(M) = (0, 0) \text{ at } T \leftarrow \]
  \[ \text{pow}(C, \text{open\_ballot}(C, M)) = true \text{ holdsat } T \]

- \( \text{open\_ballot}(C, M) \text{ initiates } \text{voted}(V, M) = nil \text{ at } T \leftarrow \]
  \[ \text{pow}(C, \text{open\_ballot}(C, M)) = true \text{ holdsat } T \land \]
  \[ \text{role\_of}(V, \text{voter}) = true \text{ holdsat } T \]

- \( \text{open\_ballot}(C, M) \text{ initiates } \text{status}(M) = \text{voting}(T) \text{ at } T \leftarrow \]
  \[ \text{pow}(C, \text{open\_ballot}(C, M)) = true \text{ holdsat } T \]

- Now voters have power to cast votes
Effects of Institutional Power (2)

- Casting and counting votes
  
  \[
  \text{vote}(V,M,\text{aye}) \text{ initiates } \text{votes}(M) = (F1,A) \text{ at } T \leftarrow \\
  \text{pow}(V,\text{vote}(V,M)) = \text{true} \text{ holdsat } T \land \\
  \text{votes}(M) = (F,A) \text{ holdsat } T \land \\
  F1 = F + 1
  \]

  \[
  \text{vote}(V,M,\text{aye}) \text{ initiates } \text{voted}(V,M) = \text{aye} \text{ at } T \leftarrow \\
  \text{pow}(V,\text{vote}(V,M,\_)) = \text{true} \text{ holdsat } T
  \]

- Power to revoke vote now granted (revocation without vote was ‘meaningless’)

- Power also used to advance status of motion, perform role assignment, etc.
Permission

- ‘Right’ aspect of enfranchisement
  - Agents have the power to vote
  - Agents have the permission to vote
    - In this case (although not always) power implies permission
  - Nobody should stop them from exercising their power
    - Therefore the chair’s power to close the ballot is not always permitted

\[
pow(C, close\_ballot(C, M)) = true \quad \text{holdsat} \quad T \leftarrow \\
\quad status(M) = voting \quad \text{holdsat} \quad T \quad \land \\
\quad role\_of(C, chair) = true \quad \text{holdsat} \quad T
\]

\[
per(C, close\_ballot(C, M)) = true \quad \text{holdsat} \quad T \leftarrow \\
\quad role\_of(C, chair) = true \quad \text{holdsat} \quad T \quad \land \\
\quad status(M) = voting(T') \quad \text{holdsat} \quad T \quad \land \quad T > T' + 10
\]
Obligation

- ‘Entitlement’ aspect of enfranchisement
  - ‘Access’ to ‘voting machine’ is a ‘physical’ issue
  - Correct vote count: as above
  - A ‘fair’ outcome: obligation to declare the result correctly: e.g.
    a simple majority vote

\[
\text{obl}(C, \text{declare}(C, M, \text{carried})) = \text{true} \quad \text{holdsat} \quad T \leftarrow \\
\text{role_of}(C, \text{chair}) = \text{true} \quad \text{holdsat} \quad T \quad \land \\
\text{status}(M) = \text{voted} \quad \text{holdsat} \quad T \quad \land \\
\text{votes}(M) = (F, A) \quad \text{holdsat} \quad T \quad \land \\
F > A
\]
The chair always has the power to close a ballot
- It has permission to exercise the power only after some time has elapsed
- If it closes the ballot early, it may be sanctioned

\[
\text{close\_ballot}(C, M) \text{ initiates } \text{sanction}(C) = [(102, M)|S] \text{ at } T \leftarrow \text{role\_of}(C, \text{chair}) = \text{true} \text{ holdsat } T \land \\
\text{per}(C, \text{close\_ballot}(C, M)) = \text{false} \text{ holdsat } T \land \\
\text{sanction}(C) = S \text{ holdsat } T
\]

- The sanction results in penalty only if someone objects
- Feature of RONR: ‘anything goes unless someone objects’
EC has been mainly used for narrative assimilation:
  - Given a narrative, check that it is consistent
  - Given a consistent narrative, check what holds when

There are many EC variants and implementations, for different purposes or requirements
  - Discrete Event Calculus Reasoner, for proving properties & planning
  - Cached Event Calculus, for efficient narrative assimilation
  - etc. . .

We will show the use of the Simplified EC for narrative assimilation
  - Pre-process directly into Prolog
The Event Calculus: Narrative Assimilation

Initial Social State
initially( role_of(cAgent,chair) = true ).
initially( role_of(cAgent,voter) = true ).
initially( role_of(pAgent,voter) = true ).

…

Narrative
happens( open_session(cAgent, sesh), 1).
happens( propose(pAgent, m1), 2).
happens( second(sAgent, m1), 3).
happens( open_ballot(cAgent, m1), 4).
happens( vote(pAgent, m1, aye), 5).
happens( vote(sAgent, m1, nay), 6).
happens( vote(vAgent, m1, nay), 7).
happens( revoke(sAgent,m1), 8).
happens( vote(sAgent, m1, aye), 9).
happens( close_ballot(cAgent, m1), 10).
happens( declare(cAgent, m1, not_carried), 11).
happens( close_session(cAgent, sesh), 12).

Resulting Social State
roles
powers
permissions
obligations
sanctions

Social Constraints
…
holdsAt( obl(C, declare(C,M,carried))=true, T ) :-
holdsAt( role_of(C,chair)=true, T ),
holdsAt( status(M)=voted, T ),
holdsAt( votes(M)=(F,A), T ),
F > A.
…
Example: The Voting Protocol

- EC Specification pre-processed into Prolog program
- Process narratives for consistency and ‘what holds when’

<table>
<thead>
<tr>
<th>agent</th>
<th>roles</th>
<th>powers</th>
<th>permissions</th>
<th>obligations</th>
<th>sanctions</th>
</tr>
</thead>
<tbody>
<tr>
<td>cAgent</td>
<td>chair voter</td>
<td>close_ballot close_session</td>
<td>close_ballot</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pAgent</td>
<td>voter proposer</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sAgent</td>
<td>voter proposer</td>
<td>vote</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>vAgent</td>
<td>voter</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

happens(vote(sAgent, m1, aye))

<table>
<thead>
<tr>
<th>agent</th>
<th>roles</th>
<th>powers</th>
<th>permissions</th>
<th>obligations</th>
<th>sanctions</th>
</tr>
</thead>
<tbody>
<tr>
<td>cAgent</td>
<td>chair voter</td>
<td>close_ballot close_session</td>
<td>close_ballot</td>
<td>close_ballot</td>
<td></td>
</tr>
<tr>
<td>pAgent</td>
<td>voter proposer</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sAgent</td>
<td>voter proposer</td>
<td>vote</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>vAgent</td>
<td>voter</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

happens(close_ballot(cAgent, m1))

<table>
<thead>
<tr>
<th>agent</th>
<th>roles</th>
<th>powers</th>
<th>permissions</th>
<th>obligations</th>
<th>sanctions</th>
</tr>
</thead>
<tbody>
<tr>
<td>cAgent</td>
<td>chair voter</td>
<td>declare close_session</td>
<td>declare(carried)</td>
<td>declare(carried)</td>
<td></td>
</tr>
<tr>
<td>pAgent</td>
<td>voter proposer</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sAgent</td>
<td>voter proposer</td>
<td>vote</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>vAgent</td>
<td>voter</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

happens(declare(cAgent, m1, not_carried))

<table>
<thead>
<tr>
<th>agent</th>
<th>roles</th>
<th>powers</th>
<th>permissions</th>
<th>obligations</th>
<th>sanctions</th>
</tr>
</thead>
<tbody>
<tr>
<td>cAgent</td>
<td>chair voter</td>
<td>close_session</td>
<td>close_session</td>
<td></td>
<td>102</td>
</tr>
<tr>
<td>pAgent</td>
<td>voter proposer</td>
<td>propose</td>
<td>propose</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sAgent</td>
<td>voter proposer</td>
<td>propose</td>
<td>propose</td>
<td></td>
<td></td>
</tr>
<tr>
<td>vAgent</td>
<td>voter</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Summary

- Given an overview of multi-agent systems and intelligent agents
- Introduced the requirement for some multi-agent systems to show awareness and autonomic properties, e.g. for multi-criteria control in decentralised systems with heterogenous components
- Given an overview of approaches and an analytic framework
- Presented one approach in detail: dynamic norm-governed systems
- Specified a voting protocol for specification change
- Voting allows (self-)aware agents to adapt system specifications at run-time to meet dynamic environmental conditions (autonomics with conventional rules)
Bibliography I

- **Open Systems:**

- **Norm-Governed Systems:**