Lecture 10: Methodologies

An Introduction to Multiagent Systems

http://www.csc.liv.ac.uk/~mjw/pubs/imas
Pitfalls of Agent Development

Lots of (single and multi-) agent projects... but agent-oriented development receives little attention.

We now consider pragmatics of AO software projects.

Seven categories:

- Implementation.
- Macro (society) level.
- Micro (agent) level.
- Analysis and design.
- Conceptual.
- Management.
- Political.

Identifies key pitfalls.

1 Pitfalls of Agent Development
1.1 You oversell agents and AI.

- Don’t equate agents and AI.
- Agents are not AI by a back door.
- ... but they do not make the impossible possible.
- Agents may make it easier to solve certain classes of problems using techniques.
- No evidence that any system developed using agent technology could not have been built just as easily using non-agent techniques.
- If you can’t do it with ordinary software, you probably can’t do it with agents.
- Agents are not magical.

http://www.csc.liv.ac.uk/~mjw/pubs/imas/
Agents have been used in a wide range of applications, but they are not a universal solution. For many applications, conventional software paradigms (e.g., OO) are more appropriate. Given a problem for which an agent and a non-agent approach appear equally good, prefer non-agent solution.

In summary: danger of believing that agents are the right solution to every problem. Other form of dogma: believing in your agent definition.

1.2 You Get Religious
1.3 Don't know why you want agents

- New technology = lots of hype!
- “Agents will generate US$2.6 billion in revenue by the year 2000”
- Pure research?
- Technology vendor?
- Solutions vendor?
- "Having agents" will buy them.

Managers often propose agent projects without having clear idea about what "having agents" will buy them.

Managerial reaction:

- "We can get 10% of that.
- No business plan for the project:

http://www.csc.liv.ac.uk/~mjw/pubspapers/imas
Often, projects appear to be going well. ("We have agents!") But no vision about where to go with them.

The lesson: understand your reasons for attempting an agent development project, and what you expect to gain from it.

- Often, projects appear to be going well. ("We have agents!"") But no vision about where to go with them.

An Introduction to Multiagent Systems
Having developed some agent technology, you search for an application to use them.

Putting the cart before the horse.

Leads to mismatches/disatisfaction.

The lesson: be sure you understand how and where your new technology may be most usefully applied.

Do not attempt to apply it to arbitrary problems & resist temptation to apply it to every problem.

Do not attempt to apply it to arbitrary problems & resist temptation to apply it to every problem.

Don’t know what agents are good for.
1.5 Generic Solutions to 1-Off Problems

General solutions are more difficult and more costly to develop, often need tailoring to different applications.

- General solutions are more difficult and more costly to develop.
- A close knit range of problems with similar characteristics.
- Re-use is difficult to attain unless development is undertaken for a close knit range of problems with similar characteristics.
- Reuse is difficult to attain unless development is undertaken for a close knit range of problems with similar characteristics.
- Re-use is difficult to attain unless development is undertaken for a close knit range of problems with similar characteristics.
- Re-use is difficult to attain unless development is undertaken for a close knit range of problems with similar characteristics.

Devising an architecture or testbed that supposedly enables a range agent systems to be built, when you really need a one-off system.

The "yet another agent testbed" syndrome.
Confuse Prototypes with Systems

Prototypes are easy (particularly with nice GUI builders!)
Field tested production systems are hard.
Process of scaling up from single-machine multi-threaded Java app to multi-user system much harder than it appears.
Silver Bullet

Holy grail of software engineering is a "silver bullet":

- Technologies promoted as the silver bullet:
  - COBOL :-)
  - Formal methods (i)
  - Graphical programming;
  - Expert systems;
  - Automatic programming;
  - Agent technology

Good reasons to believe that agents are useful way of tackling

But these arguments largely untested in practice.

Agent technology is not a silver bullet.

Some problems.
Agents are another abstraction.

Useful developments in software engineering: abstractions.
The idea of an agent is extremely intuitive.

1.8 Confuse Buzzwords & Concepts

- Label "BDI" now been applied to WWW pages/perl scripts.
  - Logic of practical reasoning (Rao & Georgeff).
  - Serious applications (NASA, ...);
  - Agent architectures (PRS, D MAR S, ...);
  - Theory of human practical reasoning (Bartman et al.);

Good example: the belief-desire-intention (BDI) model.

The 8 party syndrome: everyone has an opinion. However when they do not.

Encourages developers to believe that they understand concepts extremely intuitively.
Our system is a BDI system... Implication that this is like being a computer with 64MB memory: a quantifiable property, with measurable associated benefits.
Developing any agent system is essentially experimental.

Project plans focus on the agent bits.

This encourages developers to forget they are developing

No tried and trusted techniques

Frequent justification: software engineering for agent systems is

Neglected.

Result a foregone conclusion: project founders, not because
gent problems, but because basic software engineering

Mundane software engineering (requirements analysis,
specification, design, verification, testing) is forgotton.

Project plans focus on the agent bits.

Frequent justification: software engineering for agent systems is

none-existent.

Lecture 10
An Introduction to Multiagent Systems
But almost any principled software development technique is better than none.
Forget its distributed

Distributed systems = one of the most complex classes of computer system to design and implement.

Multi-agent systems tend to be distributed!

Problems of distribution do not go away, just because a system is agent-based.

Typical multi-agent system will be more complex than a typical distributed system.

Recognise distributed systems problems.

Make use of DS expertise.

http://www.csc.liv.ac.uk/~mjw/pubs/imas/
1.10 Don’t Exploit Related Technology

In any agent system, percentage of the system that is agent-specific is comparatively small. Therefore, important that conventional technologies and techniques are exploited wherever possible.

Example: CORBA.

- Focuses effort on agent component.
- Avoids re-inventing wheel.
- Speeds up development.

Exploitation of related technology:

- Don’t reinvent the wheel. (Yet another communication framework.)
- The rising bread model of Winston.

In any agent system, percentage of the system that is agent-specific is comparatively small.

http://www.csc.liv.ac.uk/~mjw/pubs/imas/
1.1 Don’t exploit concurrency, why have an agent solution?

- Don’t exploited.
- Most important potential advantages of multi-agent solutions not exploited.
- Only ever a single thread of control: concurrency, one of the most obvious features of a poor multi-agent design is non-existent.
- Serial processing in distributed systems.
- Small or even in extreme cases non-existent.
- Examples: decompose along functional, organisational, physical.
- Many ways of cutting up any problem.

Many ways of cutting up any problem.
Agent architectures: designs for building agents. Many agent architectures have been proposed over the years. Great temptation to imagine you need your own.

Driving forces behind this belief:

- Not designed here mindset;
- Intellectual property;
- "Not designed here" mindset;
- No clear payback.

Problems:

- Architecture development takes years;
- Architecture development takes years;
- Architecture development takes years;
- Architecture development takes years.

Recommendation: buy one, take one off the shelf, or do without.

1.12 Want your own architecture

http://www.csc.liv.ac.uk/~mjw/pubs/tmas/
Think your architecture is generic

If you develop an architecture, resist temptation to believe it is generic.

- Only apply the architecture to problems with similar characteristics.

- If you have developed an architecture that has successfully been applied to some particular problem, understand why it succeeded with that particular problem.

- Any architecture that is truly generic is by definition not an architecture...

- Different architectures good for different problems.

- Leads one to apply an architecture to problem for which it is patentily unsuited.

- Different architectures good for different problems.

1.13 Think Your Architecture is Generic
"Too Much AI"

1.1.4 Use Too Much AI

What Etzioni calls “useful first” strategy.

• Richer systems.
  Obtained with such systems, progressively evolve them into
  • The lesson: build agents with a minimum of AI; as success is
  • Your agent system.
  Resist the temptation to believe such features are essential in
  • Fueled by “feature envy”, where one reads about agents that
  • All techniques to be usable.
  Result: an agent framework too overburdened with experimental
  • Temptation to focus on the agent specific aspects of the

An Introduction to Multiagent Systems

http://www.csc.liv.ac.uk/~mjw/pubs/tamas/
1.15 Not Enough AI

Don’t call your on-off switch an agent!

Be realistic: it is becoming common to find everyday distributed systems referred to as multi-agent systems.

Another common example: referring to WWW pages that have any behind the scenes processing as “agents”.

Lead to the term “agent” losing any meaning;

Raises expectations of software recipients

– leads to cynicism on the part of software developers

Problems:

Don’t call your on-off switch an agent!

1.15 Not Enough AI

An Introduction to Multiagent Systems
1.16 See agents everywhere

- Pure A-O system = everything is an agent
- Naively viewing everything as an agent is inappropriate.
- Agents for addition, subtraction, ...
- More than 10 agents = big system.
- Choose the right grain size.

http://www.csc.liv.ac.uk/~mjw/pubs/imas/
1.1.7 Too Many Agents

Agents don’t have to be complex to generate complex behavior.

Lessons:
- Keep protocols simple.
- Keep interactions to a minimum.
- Chaotic behavior.
- Emergent functionality.

Large number of agents:

Largenumberofagents:
1.18 Toofew agents

Some designers imagine a separate agent for every possible task.

Others don’t recognise value of a multi-agent approach at all.

One “all powerful” agent.

Result is like OO program with 1 class.

Fails software engineering test of coherence.

http://www.csc.liv.ac.uk/~mjw/pubs/imas/
There are now widely-used software platforms for developing agents systems. Such platforms would provide all the basic infrastructure required to create a multi-agent system. The result: everyone builds their own. By the time this is developed, project resources gone! No effort devoted to agent-specifics.

\[ T.19 \text{ Implementing Infrastructure} \]
1.20 System is anarchic

- Cannot simply bundle a group of agents together.
- Most agent systems require system-level engineering.
- For large systems, or for systems in which the society is supposed to act with some commonality of purpose, this is particularly true.
- Organisation structure (even in the form of formal communication channels) is essential.
1.21 Confuses simulated with real parallelism

Every multi-agent system starts life on a single computer. Agents are often implemented as UNIX processes, lightweight threads in C or JAVA threads. Not possible.

- With simulated distribution, there is the possibility of centralized control. In truly distributed systems, such centralized control is impossible.
- A tendency to assume that results obtained with simulated distribution will immediately scale up to real distribution.
- A dangerous fallacy: distributed systems are an order of magnitude more difficult to design, implement, test, debug, and manage.
- Many practical problems in building distributed systems, from mundane to research level.

1.21 Confuses simulated with real parallelism
When building systems using new technology, often it is necessary to start from a “blank slate.” However, most important components of a software system will be those which cannot readily be rebuilt.

Such components, which can be incorporated into an agent system by wrapping them with an agent layer, can often be mission-critical.

They can be essential to the software solution, functionally, but technologically obsolete software components which cannot readily be rebuilt.

Such systems often incorporate legacy software, which is technologically obsolete but functionally essential.

When proposing a new software solution, it is necessary to start from a “blank slate.” Such systems often incorporate legacy software, which is technologically obsolete but functionally essential.

An introduction to Multiagent Systems

http://www.csc.liv.ac.uk/~mjw/pubs/tmas/28
There are no established agent standards. Developers often believe they have no choice but to design and build all agent-specific components from scratch. But here are some defacto standards.

Examples:
- FIPA
- KQML
- HTML
- CORBA

There are no established defacto standards.
Remote procedure calls (a) versus mobile agents (b):
An Introduction to Multiagent Systems

Lecture 10

Why mobile agents?

– low-bandwidth networks (hand-held PDAs, such as NEWTON);
– efficient use of network resources;
– heterogeneity of hosts;
– security for hosts and agents;
– dynamic linking.

There are many issues that need to be addressed when building software tools that can support mobile agents:

• Why mobile agents?
We do not want to execute foreign programs on our machine, as this would present enormous security risks.

If the agent programming language supports pointers, then there is the danger of agents corrupting the address space of the host. UNIX-like access rights don’t have pointers!

many agent languages don’t have pointers! Is the danger of agents exceeding the address space of the host?

Safe libraries for access to filestore, process space, etc;

some actions (e.g., sending mail) are harmless in some circumstances, but dangerous in others — how to tell?

http://www.csc.liv.ac.uk/~mjw/pubs/imas/
Trusted agents?

an owner when their agent crashes?

Hosts must handle crashed programs cleanly — what do you tell

verified on receipt.

Some agent languages allow security properties of an agent to be

verify co-processors are a solution — have a physically

secure co-processors

access;

access;

amount of e.g., memory & processor time that an agent can

some agent languages (e.g., TELESCRIPT) provide limits on the
Agents have a right to privacy.

In order to ensure that an agent is not tampered with, it is possible to use digital watermarks — rather like check digits.

- An agent can be protected in transit by using conventional encryption techniques (e.g., PGP).
- The agent might be modified (sabotaged) in some way, without its owner's knowledge or approval. It might enable the recipient to determine its purpose, and hence we often do not want to send out our programs, as to do so:
  - Might enable the recipient to determine its purpose, and hence
  - Might enable the recipient to determine its purpose, and hence
  - Might enable the recipient to determine its purpose, and hence

http://www.csc.liv.ac.uk/~mjw/pubs/imas/34
Unless we are happy for our agents to be executed on just one type of machine (Mac, PC, SPARC, ...), then we must provide facilities for executing the same agent on many different types of machine. This implies:

- **interpreted language**: compiled languages imply reduction to machine code, which is clearly system dependent — reduced efficiency; (perhaps use virtual machine technology);
- **dynamic linking**: libraries that access local resources must provide a common interface to different environments.
We can divide mobile agents into at least three types:

- autonomous
- on-demand
- reactive mail-type

A Typology for Mobile Agents
By autonomous mobile, we mean agents that are able to decide for themselves where to go, when to go, and what to do when they get there (subject to certain resource constraints, e.g., how much money they can spend.

Such agents are generally programmed in a special language that provides a go instruction… best known example is TELESCRIPT.

An introduction to Multiagent Systems
The idea here is that a host is only required to execute an agent when it explicitly demands the agent. The best known example of such functionality is that provided by On-Demand Mobility.

Java itself is a general purpose, C++ like programming language, (that does not have pointers!)...

The user's machine.

The best known example of such functionality is that provided by On-Demand Mobility. A user with a JAVA-compatible browser (e.g., Netscape 2.0) can request HTML pages that contain applets – small programs implemented in the JAVA language. These applets are downloaded along with all other images, text, forms, etc., on the page, and, once downloaded, are executed on the user's machine.

A user with a JAVA-compatible browser (e.g., Netscape 2.0) can request HTML pages that contain applets – small programs implemented in the JAVA language. These applets are downloaded along with all other images, text, forms, etc., on the page, and, once downloaded, are executed on the user's machine.

Java itself is a general purpose, C++ like programming language, (that does not have pointers!)...
The idea here is to piggy-back agent programs onto mail.

When email is received, the agent is unpacked, and the script executed. Hence the email is no longer passive, but active.

The best-known example of this work is the mime extension to email, allowing Safe-Tcl scripts to be sent.

The idea here is to 'piggy-back' agent programs onto mail.

Active-Mail Agents
TELESCRIPT was a language-based environment for constructing mobile agent systems. Places are virtual locations occupied by agents. There are two key concepts in TELESCRIPT technology:

- **Places**: virtual locations occupied by agents. A place may correspond to a single machine, or a family of machines.
- **Agents**: TELESCRIPT technology is the name given by General Magic to a family of concepts and techniques they have developed to underpin their products.

An Introduction to Multiagent Systems
Agents are the providers and consumers of goods in the electronic marketplace that TELESCRIPT was developed to support.

- completion time
- destination

which specifies the parameters of its journey:

In order to travel across the network, an agent uses a ticket, 

- recognized

transmitted across a network to another place, where execution recommences, in which case their program and state are encoded and another, in which case they are able to move from one place to another.

Agents are mobile — they are able to move from one place to another, in which case their programs, rather like TCL, are interpreted programs, rather like TCL.

Agents are mobile — they are able to move from one place to another, in which case their programs, rather like TCL, are interpreted programs, rather like TCL.
Agents can communicate with another:

- If they occupy different places, then they can connect across a network;
- If they occupy the same location, then they can meet one another.
An engine is a kind of agent operating system — agents correspond to operating system processes.

- Agents and places are executed by an engine.
- Agents have an associated permit, which specifies:
  - what the agent can do (e.g., limitations on travel);
  - what resources the agent can use;
  - lifetime (measured in seconds);
  - money (measured in teleclicks, which correspond to real money);
  - size (measured in bytes).

The most important resources are:

- 'money', measured in 'teleclicks', which correspond to real
- lifetime (measured in seconds);
- size (measured in bytes).

An Introduction to Multiagent Systems

http://www.csc.liv.ac.uk/~mjw/pubs/imas/
Just as operating systems can limit the access provided to a process (e.g., in UNIX, via access rights), so an engine limits the way an agent can access its environment.
Engines continually monitor agent's resource consumption, and kill agents that exceed their limit.

Engines provide (C/C++) links to other applications via application program interfaces (APIs).

Agents and places are programmed using the TELESCRIPT language:

- persistent;
- a process class, of which agent and place are sub-classes;
- semi-compiled language for efficient execution;
- two levels — high (the visible language), and low (a semi-compiled language);
- interpreted;
- apparently based on SMALLTALK;
- pure object-oriented language — everything is an object —
- a 'process' class, of which agent and place are sub-classes;

Agents and places are programmed using the TELESCRIPT language.

Engines provide (C/C++) links to other applications via application program interfaces (APIs).

Kill agents that exceed their limit.
General Magic claim that the sophisticated built-in communications services make TELESCRIPT ideal for agent applications!
Summary:

- a rich set of primitives for building distributed applications,
- agents are ultimately interpreted programs;
  - no notion of strong agency!
- likely to have a significant impact (support from Apple, AT&T, Motorola, Philips, Sony).
- not heard of anyone who has yet actually used it!
2.2 TCL/TK and Scripting Languages

TCL/TK are now often mentioned in connection with agent based systems.

The Tool Control Language (TCL — pronounced "tickle") is its companion TK, and is primarily intended as a standard command language.

TCL/TK is an X window based widget toolkit — it provides facilities for making GUI features such as buttons, labels, text and graphic windows (much like other X widget sets).

TCL was primarily intended as a standard command language.

TCL provides the facilities to easily implement your own command languages.

 lots of applications provide such languages, (databases, spreadsheets, ...) but every time a new application is developed, a new command language must be as well.

TK is an X window based widget toolkit — it provides facilities for communication, via the exchange of TCL scripts.

TCL/TK can be embedded as C++ code, which can be embedded in an application, and as required:

- TCL/TK can be embedded — the interpreter itself is available implemented in C/C++, and allows the user to build on these
- TCL is extendable — it provides a core set of primitives,
- TCL is an interpreted language;

much more interesting:

TCL/TK combined, make an attractive and simple to use GUI development tool; however, they have features that make them

An Introduction to Multiagent Systems

Lecture 10

http://www.csc.liv.ac.uk/~mjw/pubs/tmas/
TCL programs are called **scripts**. TCL scripts have many of the properties that UNIX shell scripts have.

As TCL programs are **interpreted**, they are very much easier to prototype and debug than compiled languages like C/C++ — they also provide more powerful control constructs.

- they can call up various other programs and obtain results from these programs (cf. procedure calls).
- they can be executed by a shell program (**tclsh** or **wish**).
- they are plain text programs, that contain control structures just like a normal programming language; variables, lists, and arrays (e.g., iteration, sequence, selection) and data structures (e.g., iteration, sequence, selection) and data structures.

TCL programs are called **scripts**.
Lecture 10: An Introduction to Multiagent Systems

Also, the structuring constructs provided by TCL leave something to be desired.

http://www.csc.liv.ac.uk/~mjw/pubs/imas/
Lecture 10: Introduction to Multiagent Systems

Where does the idea of an agent come in?

It is easy to build applications where TCL scripts are exchanged across a network and executed on remote machines. But the safety issue has not yet been fully resolved in TCL. This limits its attractiveness as an agent programming environment.

A key issue is safety. You don't want to provide someone else's script with the full access to your computer that an ordinary scripting language (e.g., csh) provides. Thus TCL scripts become sort of agents.

Example: Safe TCL controls the access that a script has to the UI, access provided to a script.

This led to Safe TCL, which provides mechanisms for limiting the limitations on the number of times a window can be modified by a script.

By placing limits on the number of times a window can be modified by a script.

But the safety issue has not yet been fully resolved in TCL. This limits its attractiveness as an agent programming environment.

http://www.csc.liv.ac.uk/~mjw/pubs/tmas/
Summary:

- TCL/TK provide a rich environment for building agent environments.
- The core primitives may be used for building agent environments — the source code is free, stable, well-designed, and easily modified.
- But they are not/were not intended as agent programming language-based applications, particularly GUI-based ones.
- TCL/TK provide a rich environment for building agent programming environments — the source code is free, stable, well-designed, and easily modified.