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

An Introduction to Multiagent Systems

CHAPTER 9: METHODOLOGIES
Pitfalls of Agent Development

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

We now consider pragmatics of AO software projects.

Seven categories:

- micro (agent) level;
- analysis and design;
- conceptual;
- management;
- political;

Identifies key pitfalls.

1 Pitfalls of Agent Development

Chapter 9
An Introduction to Multiagent Systems

– Implementation.
– Macro (society) level.
1.1 You oversell agents.

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

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 solutions.

In summary: danger of believing that agents are the right solution to every problem.

Other form of dogma: believing in your agent definition.

• 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"

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.

- Pure research?
- Technology vendor?
– solutions vendor?
– ...

• 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.
1.4 Don't Know What Agents Are Good For

• Having developed some agent technology, you search for an application to use them.

• Putting the cart before the horse! Leads to mismatches/dissatisfaction

• 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.

• Don't know what agents are good for

Don't attempt to apply it to arbitrary problems & resist temptation to apply it to every problem.

http://www.csc.liv.ac.uk/~mjw/pubs/imas/
1.5 Generic Solutions to 1-Off Problems

- The "yet another agent testbed" syndrome.
- Devising an architecture or testbed that supposedly enables a range of agent systems to be built, when you really need a one-off system.
- Re-use is difficult to attain unless development is undertaken for a close knit range of problems with similar characteristics.
- General solutions are more difficult and more costly to develop, often need tailoring to different applications.

http://www.csc.liv.ac.uk/~mjw/pubs/imas/
1.6 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.

http://www.csc.liv.ac.uk/~mjw/pubs/imas/
Agent technology is not a silver bullet.

- technologies promoted as the silver bullet:
  - formal methods (i)
  - expert systems;
  - graphical programming;
  - automated programming;
  - COBOL :-)

Technologies promoted as the silver bullet development.

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

1.7 Believe Agents = Silver Bullet

An Introduction to Multiagent Systems 2e
Good reasons to believe that agents are useful way of tackling some problems.

Agents are another abstraction.

Abstracts:

• Useful developments in software engineering:

  • But these arguments largely untested in practice.

Chapter 9
An Introduction to Multiagent Systems 26
1.8 Confuse Buzzwords & Concepts

- The idea of an agent is extremely intuitive.
- The theory of human practical reasoning (Bratman et al).
- Good example: the belief-desire-intention (BDI) model.
- Multiagent architectures (PRS, DMARS, ...).

The AI & party syndrome: everyone has an opinion.
(However uninformed.)
Encourages developers to believe that they understand concepts when they do not.

1.8 Confuse Buzzwords & Concepts
serious applications (NASA, . . . );

- logic of practical reasoning (Rao & Georgeff).
- serious applications (NASA, . . . );

- quantifiable property, with measurable associated benefits.

- Our system is a BDI system . . . implication that this is like being a computer with 64M memory: a quantifiable property, with measurable associated benefits.

- Label "BDI" now been applied to WWW pages/Perl scripts.
1.9 Forget it’s Software

Developing any agent system is essentially experimentation.

- No tried and trusted techniques are developing software!
- This encourages developers to forget they are experimenting.
- Project plans focus on the agenty bits.
- Mundane software engineering (requirements analysis, specification, design, verification, testing) is forgotten.

[http://www.csc.liv.ac.uk/~mjw/pubs/imas/]
technique is better than none. But almost any principled software development system is non-existent.

Frequent justification: software engineering for agent engineering ignored.

because agent problems, but because basic software engineering

Result a foregone conclusion: project founders, not
Chapter 9
An Introduction to Multiagent Systems

Forget its distributed.

- Make use of DS expertise.
- Recognise distributed systems problems.
  A typical distributed system.
- Typical multi-agent system will be more complex than
  a system is agent-based.
- Problems of distribution do not go away, just because
  Multi-agent systems tend to be distributed!
- Classes of computer system to design and implement:
  Distributed systems = one of the most complex

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

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

Therefore important that conventional technologies and techniques are exploited wherever possible.

Exploitation of related technology:

- speeds up development;
- avoids re-inventing wheel;
- framework.

Therefore raising bread model of Winston.

Agent-specific is comparatively small.

In any agent system, percentage of the system that is

1.10 Don’t Exploit Related Technology
Example: CORBA.

http://www.csc.liv.ac.uk/~mjw/pubs/imas/
multi-agent solutions not exploited.

• Only ever a single thread of control: concurrency, one
• Serial processing in distributed systems.
  • Extreme cases: non-existent.
  • Problem solving is comparitively small or even in
  • Multi-agent design is that the amount of concurrent
  • One of the most obvious features of a poor
  • Organisational, physical, or resource related lines.
  • Examples: decompose along functional,
  • Many ways of cutting up any problem.

1.1.1 Don’t exploit concurrency
If you don’t exploit concurrency, why have an agent?
1.12 Want Your Own Architecture

- Agent architectures: designs for building agents.
  - Many agent architectures have been proposed over the years.

Problems:
  - Intellectual property.

Driving forces behind this belief:
  - "not designed here" mindset.
  - Great temptation to imagine you need your own.

1.2 Want Your Own Architecture
Recommendation: buy one, take one off the shelf, or do without.

- no clear payback.

http://www.csc.liv.ac.uk/~mjw/pubs/imas/23
1.3 Think Your Architecture is Generic

- Any architecture that is truly generic is by definition not an architecture.
- Different architectures are good for different problems.
- Which it is patently unsuited.
- Leads one to apply an architecture to problems for which it is generic.
- If you do develop an architecture, resist temptation to believe it is generic.

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

• Only apply the architecture to problems with similar characteristics.
1.4 Use Too Much AI

- Resist the temptation to believe such features are essential in your agent system.
- Resist the temptation to believe such features are essential in your agent system.
- Resist the temptation to believe such features are essential in your agent system.
- Resist the temptation to believe such features are essential in your agent system.

Fuelled by "feature envy", where one reads about agents that have the ability to learn, plan, talk, sing, dance... Experimental AI techniques to be usable.

Result: an agent framework too overloaded with the application.

Templating to focus on the agent specific aspects of the application.
• What Etzioni calls “useful first” strategy.
  
  - Evolve them into richer systems.
  - Success is obtained with such systems progressively.

• The lesson: build agents with a minimum of AI, as

An Introduction to Multiagent Systems 2e
1.15 Not Enough AI

• Don't call your on-off switch an agent!
• Be realistic: it is becoming common to find everyday systems referred to as multi-agent systems that have any behind the scenes processing as agents.
• Another common example: referring to WWW pages that have any behind the scenes processing as "agents".
• Problems: raise expectations of software recipients - lead to the term "agent" losing any meaning;

1.15 Not Enough AI
leads to cynicism on the part of software developers.
1.16 See agents everywhere

• "Pure" A-O system = everything is an agent
• Naively viewing everything as an agent is inappropriate.
• More than 10 agents = big system.
• Choose the right grain size.

Agents for addition, subtraction, ...
1.17 Too Many Agents

- Agents don't have to be complex
to generate complex
behaviour.
- Large number of agents:
Emergent functionality;
Chaotic behaviour;
Keep protocols simple;
Keep interactions to a minimum;
Lessons:
- Keep interactions to a minimum;
- Keep protocols simple;
- Large number of agents:

Too Many Agents

1.17 Too Many Agents

http://www.csc.liv.ac.uk/~mjw/pubs/imas/
Somedesignersimagineaseparateagentforevery
possibletask.

Othersdon’trecognisethevalueofamulti-agent
approachatall.

Some designers imagine a separate agent for every
task.

Fails software engineering test of coherence.

Result is like a single class.

One “all powerful” agent.

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

1.19 Implementing Infrastructure
System is anarchic.

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

1.20 System is anarchic
1.21 Confuse simulated with real parallelism.

- Every multi-agent system starts life on a single computer.

- 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.

- High-weight processes in C, or JAVA threads, agents are often implemented as UNIX processes.

- Every multi-agent system starts life on a single computer.
Many practical problems in building distributed systems.

Centralised control is not possible.

With simulated distribution, there is the possibility of

centralised control, in truly distributed systems, such
from mundane to research level.

An Introduction to Multiagent Systems 2e

Chapter 9
When proposing a new software solution, essential to work with such components.

- When building systems using new technology, it is necessary to start from a “blank slate”.

- Often, most important components of a software system will be legacy.

- Such systems often mission critical.

- Functionally essential, but technologically obsolete software components, which cannot readily be rebuilt.

- Often, most important components of a software system will be legacy.
They can be incorporated into an agent system by wrapping them with an agent layer.
There are no established agent standards.

Examples:
- FIPA.
- KQML.
- HTML.
- CORBA.

But here are some de facto standards.

Developers often believe they have no choice but to design and build all agent-specific components from scratch.

1. Ignore de facto standards.

http://www.csc.liv.ac.uk/~mjw/pubs/imas/
Remote procedure calls (a) versus mobile agents (b):

2 Mobile Agents
Why mobile agents?

- Low-bandwidth networks (hand-held PDAs, such as NEWTON);
- Dynamic linking;
- Heterogeneity of hosts;
- Security for hosts and agents;
- Efficient use of network resources;
- Security for hosts and agents;
- Many issues that need to be addressed when building software tools that can support mobile agents.
We do not want to execute foreign programs on our machine, as this would present enormous security risks.

- Security for Hosts

We do not want to execute foreign programs on our machine, as this would present enormous security risks. It is important to ensure that the security of the host system is maintained. This can be achieved through several means:

- Safe libraries for access to file stores, process space, etc.
- UNIX-like access rights on host;
- Many agent languages don't have pointers!
- If the agent programming language supports pointers, then there is the danger of agents corrupting the address space of the host.

In addition, it is important to consider the security of the agent programming language itself. Many agent languages do not support pointers, which can help to prevent security vulnerabilities. It is also important to ensure that the host system has safe libraries for accessing files, processes, etc., in order to maintain security.

http://www.csc.liv.ac.uk/~mjw/pubs/imas/42
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?

Do you tell an owner when their agent crashes?

Hosts must handle crashed programs cleanly — what agent to be verified on receipt.

Some agent languages allow security properties of an agent to be verified on receipt.

Some agent languages allow security properties of an agent to be verified on receipt.

Trusted agents?

Do you tell an owner when their agent crashes?

Hosts must handle crashed programs cleanly — what agent to be verified on receipt.

Some agent languages allow security properties of an agent to be verified on receipt.

Trusted agents?

Do you tell an owner when their agent crashes?

Hosts must handle crashed programs cleanly — what agent to be verified on receipt.

Some agent languages allow security properties of an agent to be verified on receipt.

Trusted agents?

Do you tell an owner when their agent crashes?

Hosts must handle crashed programs cleanly — what agent to be verified on receipt.

Some agent languages allow security properties of an agent to be verified on receipt.

Trusted agents?

Do you tell an owner when their agent crashes?

Hosts must handle crashed programs cleanly — what agent to be verified on receipt.

Some agent languages allow security properties of an agent to be verified on receipt.

Trusted agents?

Do you tell an owner when their agent crashes?

Hosts must handle crashed programs cleanly — what agent to be verified on receipt.

Some agent languages allow security properties of an agent to be verified on receipt.

Trusted agents?

Do you tell an owner when their agent crashes?

Hosts must handle crashed programs cleanly — what agent to be verified on receipt.

Some agent languages allow security properties of an agent to be verified on receipt.

Trusted agents?

Do you tell an owner when their agent crashes?

Hosts must handle crashed programs cleanly — what agent to be verified on receipt.

Some agent languages allow security properties of an agent to be verified on receipt.

Trusted agents?

Do you tell an owner when their agent crashes?

Hosts must handle crashed programs cleanly — what agent to be verified on receipt.

Some agent languages allow security properties of an agent to be verified on receipt.

Trusted agents?

Do you tell an owner when their agent crashes?

Hosts must handle crashed programs cleanly — what agent to be verified on receipt.

Some agent languages allow security properties of an agent to be verified on receipt.

Trusted agents?

Do you tell an owner when their agent crashes?

Hosts must handle crashed programs cleanly — what agent to be verified on receipt.

Some agent languages allow security properties of an agent to be verified on receipt.

Trusted agents?

Do you tell an owner when their agent crashes?

Hosts must handle crashed programs cleanly — what agent to be verified on receipt.

Some agent languages allow security properties of an agent to be verified on receipt.

Trusted agents?

Do you tell an owner when their agent crashes?

Hosts must handle crashed programs cleanly — what agent to be verified on receipt.

Some agent languages allow security properties of an agent to be verified on receipt.

Trusted agents?

Do you tell an owner when their agent crashes?

Hosts must handle crashed programs cleanly — what agent to be verified on receipt.

Some agent languages allow security properties of an agent to be verified on receipt.

Trusted agents?

Do you tell an owner when their agent crashes?

Hosts must handle crashed programs cleanly — what agent to be verified on receipt.

Some agent languages allow security properties of an agent to be verified on receipt.

Trusted agents?

Do you tell an owner when their agent crashes?

Hosts must handle crashed programs cleanly — what agent to be verified on receipt.

Some agent languages allow security properties of an agent to be verified on receipt.

Trusted agents?

Do you tell an owner when their agent crashes?

Hosts must handle crashed programs cleanly — what agent to be verified on receipt.

Some agent languages allow security properties of an agent to be verified on receipt.

Trusted agents?

Do you tell an owner when their agent crashes?

Hosts must handle crashed programs cleanly — what agent to be verified on receipt.

Some agent languages allow security properties of an agent to be verified on receipt.

Trusted agents?

Do you tell an owner when their agent crashes?

Hosts must handle crashed programs cleanly — what agent to be verified on receipt.

Some agent languages allow security properties of an agent to be verified on receipt.

Trusted agents?

Do you tell an owner when their agent crashes?

Hosts must handle crashed programs cleanly — what agent to be verified on receipt.

Some agent languages allow security properties of an agent to be verified on receipt.

Trusted agents?

Do you tell an owner when their agent crashes?

Hosts must handle crashed programs cleanly — what agent to be verified on receipt.

Some agent languages allow security properties of an agent to be verified on receipt.

Trusted agents?

Do you tell an owner when their agent crashes?

Hosts must handle crashed programs cleanly — what agent to be verified on receipt.

Some agent languages allow security properties of an agent to be verified on receipt.

Trusted agents?

Do you tell an owner when their agent crashes?

Hosts must handle crashed programs cleanly — what agent to be verified on receipt.

Some agent languages allow security properties of an agent to be verified on receipt.

Trusted agents?

Do you tell an owner when their agent crashes?

Hosts must handle crashed programs cleanly — what agent to be verified on receipt.

Some agent languages allow security properties of an agent to be verified on receipt.

Trusted agents?

Do you tell an owner when their agent crashes?

Hosts must handle crashed programs cleanly — what agent to be verified on receipt.

Some agent languages allow security properties of an agent to be verified on receipt.

Trusted agents?

Do you tell an owner when their agent crashes?

Hosts must handle crashed programs cleanly — what agent to be verified on receipt.

Some agent languages allow security properties of an agent to be verified on receipt.

Trusted agents?

Do you tell an owner when their agent crashes?

Hosts must handle crashed programs cleanly — what agent to be verified on receipt.

Some agent languages allow security properties of an agent to be verified on receipt.

Trusted agents?

Do you tell an owner when their agent crashes?

Hosts must handle crashed programs cleanly — what agent to be verified on receipt.

Some agent languages allow security properties of an agent to be verified on receipt.

Trusted agents?

Do you tell an owner when their agent crashes?

Hosts must handle crashed programs cleanly — what agent to be verified on receipt.

Some agent languages allow security properties of an agent to be verified on receipt.

Trusted agents?

Do you tell an owner when their agent crashes?

Hosts must handle crashed programs cleanly — what agent to be verified on receipt.

Some agent languages allow security properties of an agent to be verified on receipt.
Agents have a right to privacy.

In order to ensure that an agent is not tampered with,
- An agent can be protected in transit by using conventional encryption techniques (e.g., PGP).
- Digital watermarks—rather like check digits—enable to use digital watermarks
- Agents have a right to privacy.
- An agent might be modified (sabotaged) in some way, without its owner’s knowledge or approval.
- The purpose, and hence our intent, do so: might enable the recipient to determine its own do not want to send out our programs, as to
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.
Wecandividemobileagentsintoatleastthreetypes:

− autonomous
− on-demand
− "active mail"-type

A Typology for Mobile Agents
By autonomous mobile, we mean agents that are able to decide for themselves where to go, when, and what to do when they get there (subject to certain resource constraints, e.g., how much ‘emoney’ they can spend. Such agents are generally programmed in a special language that provides a go instruction. Its best known example is TELESCRIPT.
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 the JAVA language. 

A user with a JAVA-compatible browser (e.g., NETSCAPE 2.0) can request HTML pages that contain small programs implemented in the JAVA language — small programs implemented in the JAVA language — as embedded within HTML.

• On-Demand Mobility

http://www.csc.liv.ac.uk/~mjw/pubs/imas/
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/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 then the script executive. 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 attached to email.

Active-Mail Agents

http://www.csc.liv.ac.uk/~mjw/pubs/imas/
2.1 Telescript

- agents
- places

There are two key concepts in Telescript:

- Telescript technology is the name given by General Magic to a family of concepts and techniques they have developed to underpin their products.
- Telescript was a language-based environment for constructing mobile agent systems.

http://www.csc.liv.ac.uk/~mjw/pubs/imas/
Places are virtual locations occupied by agents. A place may correspond to a single machine, or a family of machines.
Agents are the providers and consumers of goods in the electronic marketplace that TELESCRIPT was developed to support.

- Agents are mobile — they are able to move from one place to another, in which case their program and state are encoded and transmitted across a network.
- Agents are interpreted programs, rather like TCL.
- Agents are the providers and consumers of goods in the electronic marketplace.
An Introduction to Multiagent Systems 2e

Chapter 9

http://www.csc.liv.ac.uk/~mjw/pubs/imas/
Agents can communicate with one another:

- If they occupy the same location, then they can meet one another.
- If they occupy different places, then they can connect across a network.
- If they occupy different locations, then they can connect across a network.
Agents and places are executed by an engine.

- size (measured in bytes);
- lifetime (measured in seconds);
- \textit{money}, measured in \textit{teleclicks} (which correspond to real money);

The most important resources are:

- what resources the agent can use;
- what the agent can do (e.g., limitations on travel);
- \textit{TELESCRIPT} agents have an associated \textit{permit}, which
An engine is a kind of agent operating system—agents correspond to operating system processes. 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++) interfaces to other applications.

Agents and places are programmed using the TELESCRIPT language:

- a semi-compiled language for efficient execution;
- two levels — high (the 'visible' language), and low — interpreted;
- object — apparently based on SMALLTALK;
- pure object oriented language — everything is an object;

via application program interfaces (APIs).

An Introduction to Multiagent Systems 2e

http://www.csc.liv.ac.uk/~mjw/pubs/imas/
GeneralMagic claim that the sophisticated
communications services make TELESCRIPT ideal for
agent applications!

• persistent;
• sub-classes;
• a ‘process’ class, of which ‘agent’ and ‘place’ are;
Summary:

- A rich set of primitives for building distributed applications, with a fairly powerful notion of agency.
- No notion of strong agency.
- Agents are ultimately interpreted programs.
- 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 was primarily intended as a standard command language—lots of applications provide such languages, (databases, spreadsheets, ...), but every time a new application is developed, a new command language must be as well. The (free) Tool Control Language (TCL) and its companion TK, are now often mentioned in connection with agent based systems. The (free) Tool Control Language (TCL) (pronounced 'tickle', and its companion TK, are now often mentioned in connection with agent based systems.}
TK is an X window based widget toolkit. It provides powerful facilities for interprocess communication, via the exchange of TCL scripts. TK also provides powerful facilities for making GUI features such as buttons, labels, text and graphic windows (much like other X widget sets).
TCL/TK combined, make an attractive and simple to use GUI development tool; however, they have features that make them much more interesting:

- TCL/TK can be embedded — the interpreter itself is an application, and can itself be extended.
- TCL/TK can be embedded — the interpreter itself is available as C++ code, which can be embedded in user to build on these as required;
- TCL is extendable — it provides a core set of primitives, implemented in C/C++, and allows the user to develop and extend the interpreter's functions;
- TCL is an interpreted language;
- TCL/TK combined, make an attractive and simple to use GUI development tool; however, they have
TCL programs are called scripts.

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 (e.g., variables, lists, and arrays) just like a normal programming language;
- They are executable by a shell program (`tclsh` or `wish`);
- They 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. . .

– . . . but this power comes at the expense of speed.
– Also, the structuring constructs provided by TCL leave something to be desired.
So where does the idea of an agent come in?

Thus TCL scripts become sort of agents.

A key issue is safety. You don’t want to provide

computer that an ordinary scripting language (e.g.,
exchanged across a network, and executed on remote

http://www.csc.liv.ac.uk/~mjw/pubs/imas/
This led to SafeTCL, which provides mechanisms for
programming environment.

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

Example: SafeTCL controls the access that a script
has to the UI by placing limits on the number of times
a window can be modified by a script.

This led to SafeTCL, which provides mechanisms for
limiting the access provided to a script.
Summary:

- TCL/TK provide a rich environment for building agent programming environments. The core primitives may be used for building agent programming environments.
- But they are not/were not intended as agent GUI-based ones.
- Particularly language-based applications, particularly
- An Introduction to Multiagent Systems 2e

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