

## Dealing with conflicts and failure in multi-agent systems

Michael Winikoff, Lin Padgham, James Harland  
RMIT University  
{winikoff,linpa,jah}@cs.rmit.edu.au  
<http://www.cs.rmit.edu.au/~{winikoff,linpa,jah}>

**Abstract:** Real world multi-agent systems, such as dMARS, are concurrent, distributed, and situated. In common with any concurrent system there are issues involving resource conflicts and failure. In this talk I will discuss why agent based systems differ from classical concurrent systems and what issues arise in dealing with conflicts and failure in the context of multi-agent systems.

- \* Collaboration with AAIL
- \* Funded by ARC under SPIRT program
- \* Early report

1

## Agents

### What is an agent? (many definitions)

Computer program situated in *environment* capable of *autonomous* action.

### What is an *intelligent agent*?

... capable of *flexible* autonomous action.  
flexible = responsive, proactive, social  
responsive = reactive to changes in environment  
proactive = goal directed, opportunistic

Applications of agents are in complex, changing environments.

Other traits of agents include mobility and learning.

2

## The BDI Model

- Beliefs, Desires (or goals), Intentions – and plans
- Area of application is *environments* which are *complex* and *changing*
- *Complex*: hence incomplete and imperfect information
- *Changing*: limited processing time hence local view of world
- *Beliefs* store the local state of the world.
- *Desires* are needed to handle failure (“why was I running this plan?”) and to exploit opportunities.
- *Intentions* relate to the persistence of plans in a changing environment.

3

## Project Aims

- To develop reasoning methods appropriate for agent systems in which actions are not instantaneous, may fail, and may interact with other actions;
- To use these methods to control, predict and analyse the behaviour of the execution of such actions;
- To develop the appropriate software technology to enable the integration of the resulting features into dMARS, or similar systems.

4

## Classic concurrent system

- Resources known at programming time
- Resources computer controlled (e.g. databases) so rollback possible
- Caters for infrequent system failure
- Actions are assumed to be well behaved (i.e. crashes are random)
- Often at OS level
- Low-level primitives (semaphores, RPC)

5

## Important characteristics of real-world multi-agent systems

- Resources dynamic (world is complex – not the “neat” OS level)
- No rollback!
- Failure is “frequent” and non-random – but localised.
- Distributed: no central locking.
- Situated: hence lack of control, limited time to make decisions. In particular robust recovery from failure isn't optional.

6

## A range of issues

Concurrency conflicts:

- Resources
- Goals
- Assumptions

Deadlock and livelock

Conflicts both between and *within* agents. (since, for example dMARS, has human written plans and limited run-time intelligence)

7

## Approach

- Avoid conflicts where possible
- Handle failure where unavoidable

8



## Assumption Violation (cont.)

The point isn't that failure has to be handled, the point is that S turned the light on even though it "knew" that L2 was still changing the bulb.

Compared with classical concurrency, the responsibility is with S, not L/L2.

Why? L can't be expected to know which globes share a power source.

13

## Resource Conflict Between Agents

L and L2 asked to change same globe.

14

## Part II

### Failure Recovery

In the context of concurrent/distributed systems failure is usually handled by rolling back. Unfortunately, the real world can't be rolled back.

Failure handling in the general programming context is handled by either:

- Checking the return value (too cumbersome and not done), or
- Exceptions (too low level)

15

### Failure Recovery (cont.)

In order to recover from failure we need to be able to both

1. Detect when failure has occurred, and
2. Respond appropriately

Failure detection can be done either by assuming that primitive actions always detect failure, or by checking for failure after every action.

16

## “What happened? Where am I?”

In order to respond we need to know what actually happened.

If a complex action consisting of primitive actions failed because a single primitive action failed and we know which action failed and what its effects were we can reason and determine the state of the world and what needs to be done to undo the damage.

In some situations a primitive action might have unpredictable consequences upon failure, or we might not know which parts of a complex action failed. In this case some of our beliefs about the world become undetermined and a recovery plan must begin by determining the state of the world.

This is the “Kaboom! where am I?” scenario.

17

### Example (2)

In the case of a power cut, don't keep on changing globes!

This illustrates the need for autonomy and why hard-wired recovery isn't sufficient.

### Example (3)

Action: “move from (x,y) to (a,b)”

Where are you if the action failed?

Illustrates loss of beliefs.

19

## Example

book flight:

ask airline to reserve seat

ask bank to transfer money to airline

- If money transfer fails, un-book seat.
- If seat booking fails, don't un-book seat.

Explicit solution:

book flight:

ask airline to reserve seat

if airline fails, fail

ask bank to transfer money to airline

if bank fails, ask airline to free seat

if airline fails, ???

18

### Current Work

- Logic: durational, failure, concurrent
- Language constructs

The role of logic: important as foundation for proving correctness, and analysing plans. Also potentially useful at runtime for reasoning.

Need to know about resources “in advance” (for classical system at programming time, for agent system runtime?) to avoid deadlock.

**Note:** Proposed solutions must operate at the level of plans and agents, rather than at the level of semaphores and exception handlers.

20

## Current Work

- Collect examples of conflict and exceptions
- Categorise examples
- Look at existing language constructs (e.g. maintenance conditions)
- Propose alternatives/extensions to language constructs
- Give proposals formal semantics (using Sandewall's logic, BDI logic, etc.)

21

## Failure Reaction

### What:

- An action attached to a plan
- Executed when plan fails (but doesn't cause failure)

### Problems:

- No information on where plan failed

22

## Maintenance Conditions

### What:

- Logic formula  $F$  attached to plan
- Plan aborts if  $F$  becomes false

### Problems:

- Can't be suspended for authorised changes (hence race conditions)
- Only apply while plan is running

23

## Related Work

- Current theories of BDI agents do not model actions.
- Theories of actions exist. Some model durational actions. Some model failing actions. These theories do not model goals and do not support reasoning for recovery from failing actions.
- Work at MIT (\*) uses medical analogy of "coordination doctor".
  - Looks at coordination/communication failures rather than primitive action failures.
  - Uses heuristic strategies (300+)

(\*) Mark Klein and Chrysanthos Dellarocas. Exception handling in agent systems. In *AGENTS'99, Proceedings of the third annual conference on Autonomous Agents*. <http://ccs.mit.edu/ases/agents.htm>

24

## Conclusions

- Multi-agent systems are concurrent
- Multi-agent systems differ from classical concurrent systems
- Failure is an issue
- Different types of conflict
- Conflict can also occur *within* an agent
- Current failure recovery strategies aren't intelligent