SIM-AGENTS
A TOOLKIT FOR PHILOSOPHERS AND ENGINEERS

A Sloman

Rapporteur: Professor J E Dobson

SIM_AGENT
A Toolkit
for Philosophers and Engineers

Aaron Sloman
School of Computer Science
The University of Birmingham

Including ideas from:

Riccardo Poli, Brian Logan,
Luc Beaudoin, Darryl Davis,
Catriona Kennedy, Ian Wright,
Peter Waudby Jeremy Baxter (DERA),
Richard Hepplewhite (DERA)

THE TOOLKIT IS AVAILABLE ONLINE

IN THE BIRMINGHAM FREE POPLOG DIRECTORY

http://www.cs.bham.ac.uk/research/poplog/

For an up to date version of this document see
http://www.cs.bham.ac.uk/ axs/misc/draft/toolkit.ps

ABSTRACT

In order to support exploration of a variety of architectures without commitment to any particular architecture (e.g. SOAR, PRS, ACT-R, etc.) we have developed a very flexible toolkit based on and supporting multiple programming paradigms, including standard procedural programming, functional programming, list processing, rule-based programming, event-based programming, object oriented programming (with multiple inheritance), and logic programming, in an environment that supports rapid prototyping and the use of incremental compilation to support exploration of ideas.

We wanted it to be easy to do fairly simple things, so that the toolkit could be used for teaching and for student projects (for which it has already proved very successful here in Birmingham) while also supporting more complex and difficult programming tasks in research projects, including for instance combining neural and symbolic mechanisms in the same architecture, and allowing agents with different architectures to interact with one another and with objects in a simulated environment.

The talk will present some of the motivating ideas, and give some simple demonstrations. Further information about the toolkit is available at

http://www.cs.bham.ac.uk/ axs/cog_affect/sim_agent.html

It uses the Poplog multi-language software development environment which is now available free of charge with full system sources here:

http://www.cs.bham.ac.uk/research/poplog/freepoplog.html

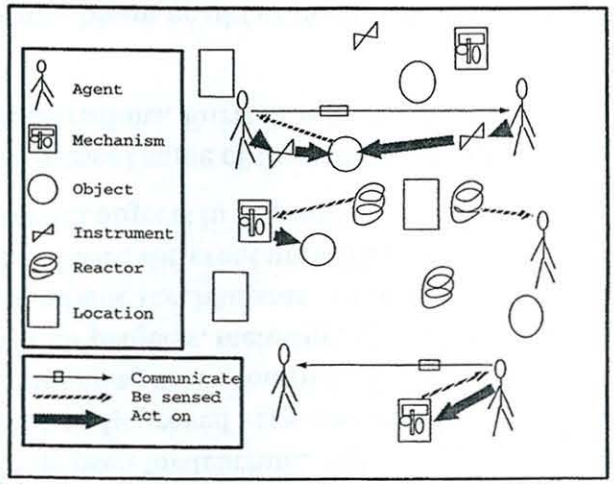
See also:

Aaron Sloman and Brian Logan,
Building cognitively rich agents using the Sim_agent toolkit,
Communications of the Association of Computing Machinery,
42, 3, pp. 71–77, March, 1999,

A recent presentation on the toolkit is available in postscript and PDF here:

http://www.cs.bham.ac.uk/ axs/misc/draft/toolkit.ps
http://www.cs.bham.ac.uk/ axs/misc/draft/toolkit.pdf

We need to support SCENARIOS WITH RICH ONTOLOGIES

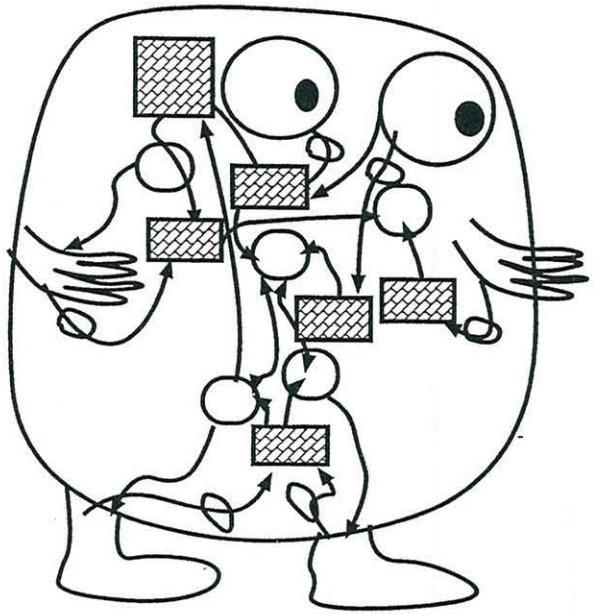


Various kinds of concurrently active entities, e.g.:

- AGENTS: **which can communicate with one another,**
- MECHANISMS: **which sense and react to other things,**
- INSTRUMENTS: **which can act if controlled by an agent,**
- "REACTORS" **which do nothing unless acted on (e.g. a mouse-trap)**
- LOCATIONS: **of arbitrary extents with various properties, including continuously varying heights**

etc., etc.

INSIDE ONE AGENT



Agents can have complex internal architectures

- Rectangles represent short or long term databases
- Ovals represent processing units.
- Arrows represent flow of information, including control information

Some components are linked to sensors and motors (physical or simulated)

Some are connected only to other internal components

The toolkit should support different sorts of agents

- performing different sorts of tasks
- with various kinds of sensors and motors (either simulated or physical)
- connected to different kinds of internal processing modules
- with different kinds of internal short term and long term databases,
- with some components monitoring or controlling others

Concurrency is required at all levels

Agents, their components and other objects can all perform various sub-tasks concurrently and asynchronously.

With information flowing in all directions simultaneously.

(Compare M.Minsky The Society of Mind. Perhaps we need to think of an "ecosystem of mind".)

MORE ON INTERNAL COMPLEXITY AND VARIETY

It should be possible to have different sorts of agents with different architectures geared to different tasks and requirements.

Since architectures may develop over time, the tools must support changing hierarchical process structures.

We know from several decades of work in AI that different TYPES of mechanisms are likely to be required, including rule-based reactive systems, neural nets, parsers, meaning generators, sentence generators, pattern-directed associative knowledge stores, low level image analysers mainly crunching numbers, high level perceptual mechanisms mainly manipulating structures, simulations of other agents, event-driven and interrupt-driven modules etc.

This in turn imposes a requirement for using different kinds of language for different subtasks.

Architectural bootstrapping

Individual agents, through learning or development may need to be able to modify THEIR OWN architectures,

either

- to simulate biological processes of growth and development,

or because

- applications of artificial agents require changes of competence at run time (e.g. agents extending themselves with new "plug-in" components at any level).

The toolkit should support varying resource allocations

E.g. some agents or components of agents should be allowed to run faster or slower than others.

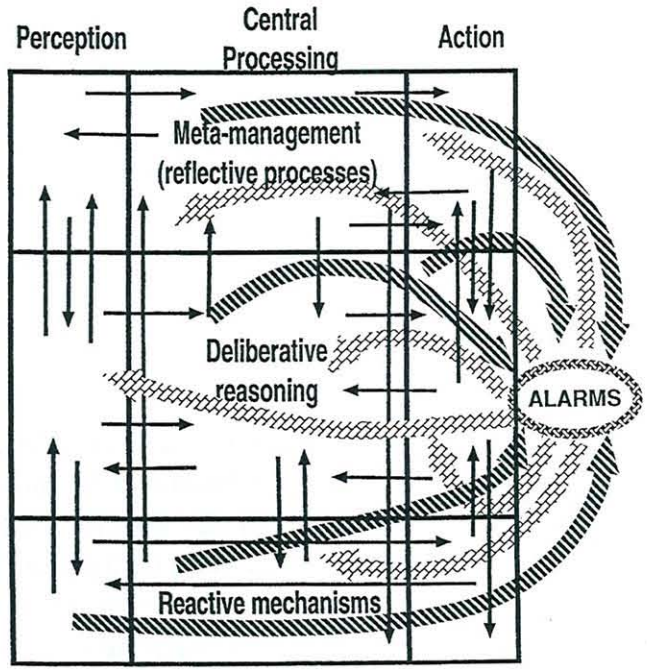
It should also be possible to vary the speeds of components,

e.g. to investigate the ability of an architecture to cope with resource limits.

FOR INSTANCE WE ARE INTERESTED IN THE ABILITY OF ATTENTION FILTERS TO COMPENSATE FOR RESOURCE LIMITS IN DELIBERATIVE MECHANISMS.

The Birmingham 'CogAff' Architecture (A partial view)

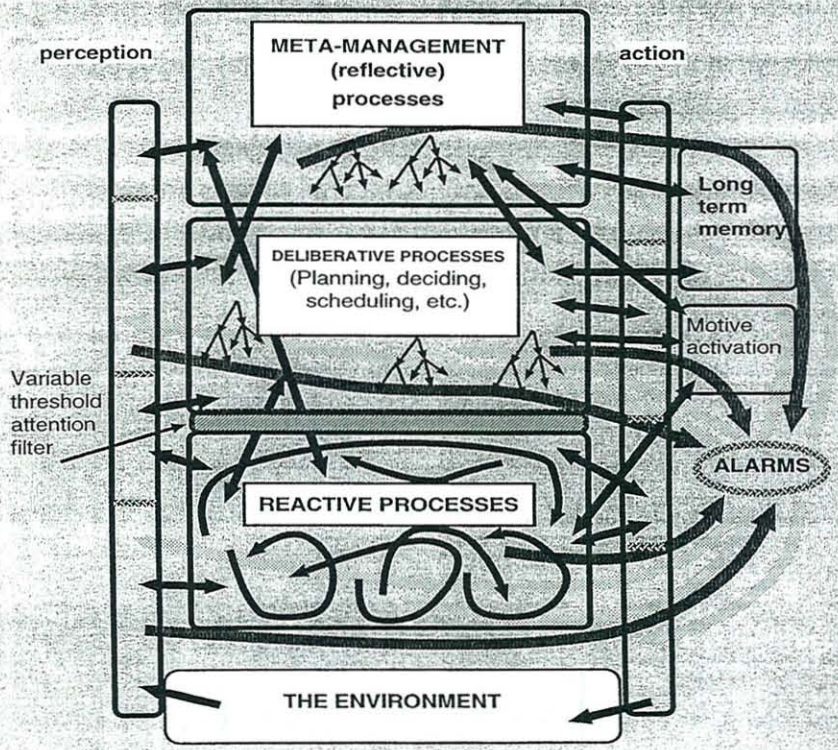
We wish to develop models with multi-level concurrently active components within perceptual, central and motor sub-systems.



This model is explained in more detail in separate presentations and publications available at

http://www.cs.bham.ac.uk/research/cogaff/

ANOTHER VIEW AUTONOMOUS REFLECTIVE/HYBRID AGENTS WITH ALARMS



Not yet implemented as a whole.
Compare Steve Allen's PhD thesis
http://www.cs.bham.ac.uk/~sra

APPROACHES TO DIVERSITY

Tools to support this diversity cannot be expected to anticipate all types of entities, causal and non-causal relationships, states, processes, etc. which can occur.

So users should be able to extend the ontology as needed. E.g.

- **User provides axioms defining new classes and subclasses and new behaviours**
- **User assembles architectures diagrammatically.**
- **User defines new classes and sub-classes using an object oriented programming language (e.g. with multiple-inheritance).**
  (THIS IS, AT PRESENT, THE APPROACH SUPPORTED BY SIM_AGENT.)

Which approach works will depend on various factors

E.G.

- **Whether there's only one type of architecture supported by the toolkit (e.g. SOAR, ACT-R, PRS, COGENT, etc.)**
- **How well defined the scenario is at the start: e.g. whether the only variation allowed is in a limited set of parameters.**

THERE IS A TRADE-OFF BETWEEN FLEXIBILITY/GENERALITY OF THE TOOLKIT AND EASE OF USE.

Sim_agent aims for flexibility and generality.

So it takes longer to learn to use than some more restricted toolkits.

But this can be alleviated by developing higher level tools and libraries aimed at specific classes of architectures.

SIM_AGENT: A POPLOG-BASED TOOLKIT

POPLOG: a multi-language AI development environment with incremental compilers for

- **Pop-11 (a Lisp-like language, with a Pascal-like syntax)**
- **Prolog**
- **Common Lisp**
- **Standard ML**
- **facilities for adding new incremental compilers**
- **a rich interface to the X window system**
- **a very fast general garbage collector**
- **large and easily extended collection of code and documentation libraries and AI teaching materials**

"POPLOG" IS A TRADE MARK OF THE UNIVERSITY OF SUSSEX WHERE IT WAS ORIGINALLY DEVELOPED.

POPLOG AVAILABILITY

From the early 80s until the end of 1998, Poplog was a commercial product sold world-wide by ISL, who were co-developers after about 1989.

ISL was taken over by SPSS December 1988 and it was then decided by ISL and Sussex university that Poplog should be made available free of charge.

It is still used as the core of Clementine, a data-mining toolkit sold world-wide, and other commercial products.

It is robust and stable.

The free Poplog system is now available with full system sources for various platforms:

http://www.cs.bham.ac.uk/research/poplog/freepoplog.html

It runs on several Unix/Linux systems and VAX/VMS

Windows/NT Poplog does not yet include graphics (unless used with an X window emulation package.)

THE ARCHITECTURE OF THE TOOLKIT:

Built on Pop-11 extended with:

- OBJECTCLASS (designed by Steve Leach). Like CLOS, it supports object oriented programming with multiple inheritance and generic functions (multi-methods)
- RCLIB – An object-oriented "relative coordinates" 2-D graphical package, making it easy to produce graphical interfaces linked to a simulation, including declaratively specified control panels.
- POPRULEBASE – an unusually flexible forward chaining pattern-driven production system interpreter, able to invoke arbitrary procedures in its conditions and actions, with meta-rules, and support for hybrid architectures (e.g. a rule's conditions can run a neural net)
- SIM_AGENT – a scheduler and some default classes and methods for sensing, communicating, and a growing library of utilities. The object-oriented design provides a set of basic classes and methods which can be extended for particular applications.

RCLIB and Poprulebase can be used independently of each other and the rest of the toolkit.

The toolkit depends on a large number of re-usable Pop-11 libraries, forming part of Poplog,

including many low-level libraries concerned with the X window capabilities and the Poplog X widget set.

HOW BIG IS IT?

The Sim_agent toolkit has been under development since 1994.

There are now about 245 code files with about 60,000 lines of code and comment occupying about 1.8Mbytes, and about 90 documentation files with about 67,000 lines of text and example code, occupying about 2.3Mbytes.

It is implemented in Pop-11 in the Poplog[tm] development environment.

Run time memory requirements

The size of a running system will of course depend on the application. As an example: a simple simulation of a sheepdog herding five sheep into a pen uses about 12 Mbytes on Sparc/Solaris Poplog (as shown by "top") including the whole poplog system and editor.

It is a little smaller in PC Linux Poplog.

The same demonstration running in Poplog on Digital Alpha Unix (with 64 bit addresses) takes about 21Mbytes.

Availability of Poplog and Sim_agent

It is available at the Free Poplog web site (with full sources).

- Fetch poplog for your system (machine+OS)
- Fetch bhamteach.tar.gz
- Fetch rclib.tar.gz
- Fetch newkit.tar.gz

(There are other packages, for vision, neural nets, etc. and more will be added.)

USERS

Sim_agent has been used at various times by a collection of undergraduates, MSc students, PhD students, colleagues and collaborators at DERA Malvern.

Suggestions from users have led to many improvements and extensions, e.g. including support for self-monitoring.

It is expected that the process of designing extensions guided by user requirements will continue.

Some extensions may be built deep into the system, while others will be optional libraries.

LIBRARIES

It is intended that, with collaborators, we'll develop a set of libraries for different sorts of classes of agents and environments.

A library can define

- Environmental object classes and mixins
- Agent classes and mixins
- Re-usable sensor methods and action methods
- Re-usable rulesets, and behaviours
- Graphical appearances
- Low level utilities

And can use Poplog's documentation mechanisms

including hypertext links.

(There is work in progress to provide support for HTML and XML in Poplog, independently of the toolkit.)

HOW WE DO IT
A MULTI-PARADIGM APPROACH

Combine many styles of programming:

- **Conventional procedural and functional programming** POP-11
- **List processing and pattern matching** POP-11
- **Rule-based programming** POPRULEBASE
- **Object oriented programming** OBJECTCLASS
  Including generic functions and multiple inheritance
- **Event-driven programming** X WINDOW SYSTEM AND RCLIB
- **Other computational paradigms needed for particular applications, e.g. neural nets or evolutionary mechanisms.**
- **Extendable syntax and semantics (macros and beyond)**
- **Invocation of other languages as needed**
  PROLOG, ML, LISP, C, ...
- **Automatic store management and** FAST **garbage collection.**

Distributed agents

The basic system supports multiple agents in one Unix process on a single CPU.

However, some users have used Poplog facilities such as the socket library to implement distributed systems.

RAPID PROTOTYPING AND SELF-MODIFYING SOFTWARE

Use of Pop-11's INCREMENTAL COMPILER **makes it easy to experiment with changes and extensions to a running system without having to re-start every time.**

- **Dynamic replacement of modules (at run time)**
- **Essential for debugging complex systems**
- **Also for** RAPID PROTOTYPING
  i.e. rapid evaluation, exploration, etc.
  (Required when you don't start off with a precisely defined well understood problem.)
- **And for self-modifying systems**

As far as I know, ONLY AI programming languages combine all of these features, with Java probably the closest contender, some way behind.

There will be continued development of increasingly high level languages to express the design ideas, along with compilers (or interpreters) to translate them into the sort of code which now has to be designed by hand.

That has been the dominant form of progress in computer science and software engineering in the last half century, apart from hardware developments.

OBJECT ORIENTATION IN SIM_AGENT

Use multiple inheritance with powerful default methods.
Define new subclasses combining capabilities of old classes.

See TEACH OOP

Default sim_agent classes, with associated methods

- **object – the top level class in Sim_agent**
- **agent – the next level down, with additional capabilities,**
  e.g. message sending.

Graphical classes and mixins in RCLIB

e.g. rc_window_object, rc_linepic, rc_movable, rc_rotatable, rc_selectable, rc_constrained_mover, etc.

With support for buttons, sliders, scrolling text, etc.,

All with user-extendable methods.

Additional classes are provided in libraries.

SIM_PICAGENT library combines RCLIB and SIM_AGENT

- **New class of graphical window**
  class sim_picagent_window;
  is rc_window_object;
- **New types of objects using sim_agent + rclib**
  mixin sim_multiwin;
  (For movable objects in multiple windows)
  mixin sim_multiwin_static;
  IS SIM_MULTIWIN;
  mixin sim_multiwin_mobile;
  IS SIM_MULTIWIN RC_LINEPIC_MOVABLE;
  mixin sim_immobile;
  IS SIM_MULTIWIN_STATIC SIM_OBJECT;
  mixin sim_immobile_agent;
  IS SIM_MULTIWIN_STATIC SIM_AGENT;
  mixin sim_movable;
  IS SIM_MULTIWIN_MOBILE SIM_OBJECT;
  mixin sim_movable_agent;
  IS SIM_MULTIWIN_MOBILE SIM_AGENT;

Users can define new subclasses, and extend or replace the methods.

THERE IS NO FIXED ARCHITECTURE: BUT A VERY FLEXIBLE FRAMEWORK FOR EXPLORING A VARIETY OF ARCHITECTURES.

Each agent's architecture can include:

- **condition-action rules**
  In a flexible, user-extendable formalism.
- **rulesets**
  composed of a collection of rules which work together to perform some task
- **ruleclusters**
  Consisting of a group of rulesets, only one of which is active at any time.
  (Previously called 'rulefamilies')
- **a rulesystem**
  Made up of a collection of ruleclusters which run in parallel. Each agent has one rulesystem.
- **Various methods for sensing, acting, communicating, tracing**

ADDITIONAL FEATURES

- **The rules within an agent communicate via private databases and message channels in the agent.**

- **Conditions and actions in rules can access arbitrary Pop-11 code: including code for neural nets and other "sub-symbolic" mechanisms (e.g.** [WHERE ...] **conditions).**

- **Pop-11 code can access all the poplog libraries, including pipes and sockets, and can invoke 'external' procedures, e.g. C procedures, such as the X window facilities.**

- **Rulesets can be turned on and off while an agent is running.**

- **The rulesystems of different agents run in simulated parallelism.**

- **The rulesystems within an agent run in simulated parallelism.**

- **An agent can inspect and alter its own architecture.**

Each agent's rulesystem is represented as a collection of items in its database. I.e. the architecture consists of mechanisms for operating on a database which contains the architecture.

CODE EXAMPLES

```
define :class sim_object;
    ;;; Top level class
    slot sim_name = gensym("object");

    ;;; A table for mapping words to rulesets, etc
    slot sim_valof =
        newproperty([], 17, false, "tmparg");

    slot sim_speed == 1;
    slot sim_cycle_limit == 1;
    slot sim_interval == 1;

    slot sim_status == undef; ;;; e.g. 'alive', e

    slot sim_data = prb_newdatabase(sim_dbsize,[])

    slot sim_rulesystem == [];

    slot sim_sensors =
        [{sim_sense_agent 1000}];

    slot sim_sensor_data == [];

    slot sim_actions == [];

    slot sim_setup_done = false;
enddefine;
```

```
define :class sim_agent; is sim_object;

    slot sim_name = gensym("agent");

    slot sim_in_messages == [];
    slot sim_out_messages == [];
enddefine;


define :class trial_agent;
    is rc_rotatable rc_linepic_movable
        rc_selectable sim_agent;

    slot trial_heading == 0;
    slot trial_size == 10;
    slot rc_picx == 0;
    slot rc_picy == 0;
    slot sim_sensors = [];

enddefine;
```

```
define :class trial_sheep; is trial_agent;
    ;;; The class defining the sheep's attributes

    slot trial_hunger == 1;
    slot trial_fatigue == 20;
    slot trial_speed == 0;
    slot trial_pspace == 30;
    slot trial_pack_range == 60;
    slot trial_flock_range == 100;
    slot trial_obstacle_range == 40;

    slot rc_mouse_limit == {-20 -20 20 20};

    slot rc_pic_lines ==
          [
            WIDTH 3
            [CIRCLE {0 0 10}]        ;;; Round body
            [CIRCLE {13 0 5}]        ;;; Round head
          ];

    slot sim_rulesystem = trial_sheep_rulesystem;
    slot sim_sensors =
        [{sim_sense_agent ^trial_visual_range}];
enddefine;
```

```
define :class trial_dog; is trial_agent;
  slot trial_speed == 0;
  slot rc_pic_lines ==
    [
      WIDTH 3
      [CLOSED {-10 10} {10 10} {10 -10} {-10 -10}]
      [CLOSED {8 8}  {8 -8} {17 0}]
    ];
  slot sim_rulesystem = trial_dog_rulesystem;
  slot sim_sensors =
        [{sim_sense_agent ^trial_visual_range}];
  slot trial_list == [];
  slot trial_current == [];
  slot trial_goal == [];
  slot trial_leftpost == [];
  slot trial_rightpost == [];
  slot trial_postlist == [];
  slot trial_sector = [];
  slot trial_side = [];
  slot trial_sheepside = [];
  slot trial_in_pen = false;
  slot trial_deshead = false;
  slot trial_problempost = false;
  slot trial_problemtree = false;
  slot trial_personalspace = 30;
  slot trial_behav = [];
  slot trial_memory = [];
  slot trial_trees = [];
  slot counter = 0;
enddefine;
```

```
define :rulesystem trial_dog_rulesystem;

    debug = false;
    cycle_limit = 1;

    include: dog_pen_rules
    include: find_new_sheep
    include: dog_perception_rules
    include: dog_target_rules
    include: dog_side_rules
    include: dog_sheepside_rules
    include: dog_tracing
    include: behaviour_rules
    include: dog_activity
    include: memory_testing

enddefine;


define :rulefamily dog_activity;

    ruleset: join
    ruleset: steer
    ruleset: take
    ruleset: treedetection

enddefine;
```

```
define :ruleset take;

RULE flipttotd
  [WHERE tree_detect(sim_myself)]
      ==>
  [RESTORERULESET treedetection]

RULE flipttoj
  [behaviour join]
  [WHERE
     sim_distance(
        sim_myself, sim_myself.trial_current)
              > 100]
      ==>
  [RESTORERULESET join]

RULE flipttoj2
  [WHERE
     sim_distance_from(
        trial_coords(sim_myself),
         trial_coords(sim_myself.trial_current))
            > 100]
      ==>
  [RESTORERULESET join]

RULE flipttoj3
  [side pen]
      ==>
  [RESTORERULESET join]

RULE flipttoj4
  [targ ron]
      ==>
  [RESTORERULESET join]

RULE flipttos
  [behaviour steer]
      ==>
  [RESTORERULESET steer]

RULE inpen
  [side pen]
      ==>
  [POP11
      [in pen]==>;
      lvars speed, heading, a, dist;
      sim_distance_from(
         trial_coords(sim_myself),
         trial_coords(sim_myself.trial_current))
                     -> dist;

      20 -> speed;

      pen.orientation - 90 -> heading;
      move_dog( sim_myself, speed, heading );
  ]
```

```
;;;At a distance...Approach the sheep directly

RULE noproblem
  [NOT targ on]
  [NOT targ ron]
  [WHERE
     sim_distance(
          sim_myself, sim_myself.trial_current)
              > 100]
     ==>
  [POP11
      lvars dist, speed, heading;
      sim_distance_from(
        trial_coords(sim_myself),
          trial_coords(sim_myself.trial_current))
                  -> dist;

      if dist > 50 then 10
      else round(dist/25)
      endif    -> speed;

      sim_heading_from(
        trial_coords(sim_myself),
           trial_coords(sim_myself.trial_current))
                  -> heading;
      move_dog(sim_myself, speed, heading);
```

AN EXAMPLE METARULE USING AN [ALL ...] CONDITION

```
define :ruleset check_rules;

  RULE check_constraints
    [constraint ?name ?checks ?message]
    [ALL ?checks]
    ==>
    [SAY Constraint ?name violated]
    [SAY ??message]
    [RESTORERULESET backtrack_rules]

  RULE checks_ok
    ==>
    [RESTORERULESET solve_rules]
enddefine;
```

In the above, the condition

```
[constraint ?name ?checks ?message]
```

Causes the variable `checks` **to pick up from the database a list of conditions (which may include variables.**

Example constraint, from TEACH PRB_RIVER.P

```
;;; Now the constraints - checked by rule check
;;; first constraint -
;;; fail if something can eat something
[constraint Eat
    [[?thing1 isat ?side]
        [NOT man isat ?side]
        [?thing1 can eat ?thing2]
        [?thing2 isat ?side]]
    [?thing1 can eat ?thing2 GO BACK]]

;;; second constraint, is the current state
;;; one that's in the history?
[constraint Loop
    [[state ?state] [history == [= ?state] == ]]
    ['LOOP found - Was previously in state: ' ?st.
```

Then this condition

```
    [ALL ?checks]
```

tests whether all those conditions are currently satisfied in the database: as if the conditions had been made explicit in this rule.

THE VIRTUAL TIME SCHEDULER

SIM_AGENT provides a scheduler which 'runs' objects in a virtual time frame composed of a succession of time slices.

It uses Objectclass methods that can be redefined for different sub-classes of agents without altering the scheduler.

The default 'run' method gives every agent a chance to do three things in each time-slice:

- **sense its environment**
- **run internal processes that interpret sensory data and incoming messages, and manipulate internal states**
- **produce actions or messages for other agents**

After that's done for each agent, default methods are used:

- **to transfer messages between agents**
- **to perform the actions for each agent**

So each agent's sensory processes and internal processes run with the 'external' world in the same state in the same time-slice.

Changeable resource limits associated with rulesets supports exploration of effects of speeding up or slowing down different modules relative to each other and environmental speeds.

This will help us evaluate the need for meta-management mechanisms, and various ways of meeting that need by letting meta-management compensate for lack of speed in some contexts

DEVELOPMENT ENVIRONMENT

It has proved quite difficult to design and implement such agents. One reason is the difficulty of knowing what sort of design is required. This suggests a need for tools to explore possible designs and design requirements (e.g. by examining how instances of first draft designs succeed or fail in various domains and tasks). I.e. support for very rapid prototyping is essential.

Trade-off: compile time checking etc. vs flexibility.

Many agent toolkits exist which are geared to support a particular type of agent (e.g. agents built from a collection of neural nets, or agents which all have a particular sort of cognitive architecture such as SOAR).

For researchers who don't yet know which architecture to use, these impose a premature commitment (the field is still in its infancy).

So we need a toolkit which not only supports the kind of complexity described above, but which does not prescribe any particular architecture for agents, so that we can learn by exploring new forms.

However it should support the re-use of components of previous designs so that not all designers have to start from scratch. Libraries are needed for this.

It should support both explicit design by human researchers and also automated design of agents e.g. using evolutionary mechanisms.

CHALLENGES FOR THEORISTS

- It seems likely that the sort of complexity outlined above will be required even in some safety critical systems. Can we possibly hope to understand such complex systems well enough to trust them?
- Will we ever be able to automate the checking of important features of such designs?
- The design of systems of such complexity poses a formidable challenge. Can it be automated to any useful extent?
- Do we yet have good languages for expressing the REQUIREMENTS for such systems (e.g. what does "coherent integration" mean? What does "adaptive learning" mean in connection with a multi-functional system?)
- Do we have languages adequate for describing DESIGNS for such systems at a high enough level of abstraction for us to be able understand them (as opposed to millions of lines of low level detail)?
- Will we ever understand the workings of systems of such complexity?
- How should we teach our students to think about such things?

FUTURE WORK

- Adding more libraries, including libraries supporting particular kinds of architectures
- Extending the "harness", e.g. with tools to make it easier to assemble and run scenarios (including architecture-specific graphical tools).
- Making it easier for agents to inspect and modify their own architectures (e.g. to model various kinds of cognitive development or self-awareness).
- Adding a more "neural like" database mechanism, with "sloppy" matching and spreading activation (as in ACT-R)

For more on sim_agent and its subsystems see

http://www.cs.bham.ac.uk/~axs/cog_affect/sim_agent.html

Overview

http://www.cs.bham.ac.uk/research/poplog/sim/help/sim_agent

Main integrating library

http://www.cs.bham.ac.uk/research/poplog/prb/help/rulesystems

How to express agent internals

http://www.cs.bham.ac.uk/research/poplog/prb/help/poprulebase

More details

http://www.cs.bham.ac.uk/research/poplog/rclib/help/rclib

The graphical tools.

There are gzipped tar files containing all the above.

See the file:

http://www.cs.bham.ac.uk/research/poplog/freepoplog.html

DISCUSSION

Rapporteur: Professor J E Dobson

(Report for lecture one not available)

Lecture Two

Dr Elliott asked how quickly the software was changing and what things, if any, were stable. Professor Sloman answered that the graphics side was changing fairly fast due to incremental availability of extra components. The insertion of the architecture into the database was taken fairly slowly, mainly because of the control imposed by the requirement to be stable during the lifetime of student projects.

Dr Cunningham asked about how the results could be validated in accordance with best practice of an experimental science. Professor Sloman replied that there were ways of justifying design decisions on the basis of theory (e.g. theories of neuroscience or evolution). Validation of an implementation could be expressed in terms of understanding the problem. Ultimately, though, his claims about humans and brains were probably false and unrealistic and to some extent he didn't care; what mattered was the evaluation of an application in terms of whether it did what we want it to do (with respect to dependability as well as functionality).

Professor Henderson observed that if a certain subset was in Java we'd probably use it; but what constitutes that subset? Professor Sloman said that it would be a subset of the language that allowed for totally universal typing, as was required for pattern matching, constraint propagation etc.

Dr Watson asked whether this meant the speaker was asking for lists where every member was of a subtype of the universal type. Professor Sloman replied that he was, but observed that this was incompatible with Java security. Dr Watson thought that this might restrict the list operations to the basic ones (*car*, *cdr* etc.) Professor Sloman replied that his incremental compiler technique allows runtime adaption to the types of the list elements as they are found to be.

Dr von Sydow observed that the MLJ compiler allows combination of ML and Java. Professor Sloman replied that ML has some restrictions in that lists of elements of arbitrary type are not permitted.