Oxford Computing Lab
30 Jan 2001

SimAgent:

A TOOLKIT FOR PHILOSOPHERS AND ENGINEERS

Aaron Sloman
http://www.cs.bham.ac.uk/˜axs/

School of Computer Science
The University of Birmingham

INCLUDING IDEAS FROM:

Riccardo Poli, Brian Logan,

Catriona Kennedy Matthias Scheutz,

Jeremy Baxter (DERA), Richard Hepplewhite (DERA)

THE TOOLKIT IS AVAILABLE ONLINE WITH SOURCES

IN THE BIRMINGHAM FREE POPLOG DIRECTORY

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

Agents and agent toolkits are very fashionable.
So why another toolkit?

Partly because our research objectives are very ambitious: we wished to explore ideas relevant to exploring models of human-like information processing architectures, with many complex, interacting, components.

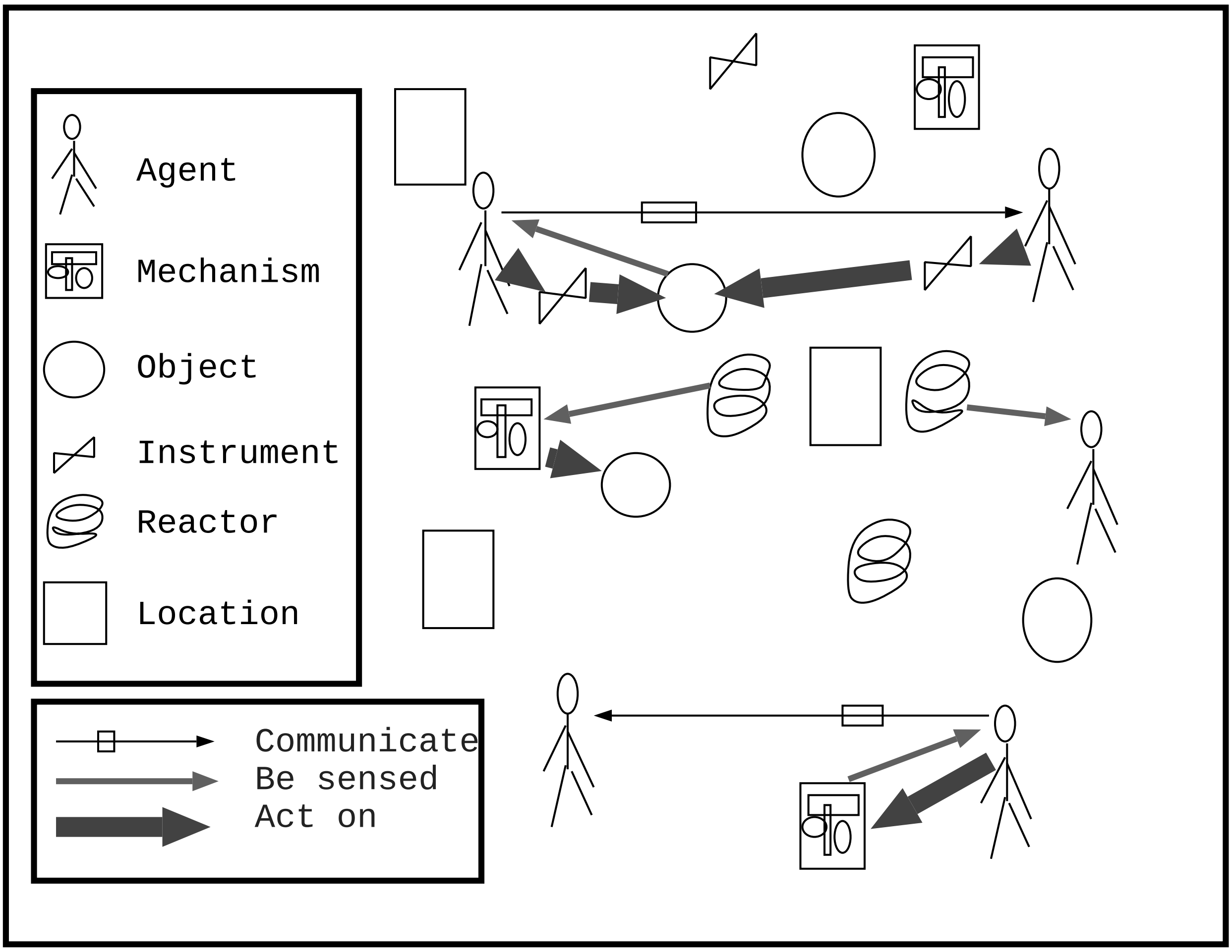
E.g. models explaining human emotions in terms of an information processing model of mind – with several concurrently active processing layers explaining different sorts of emotions.

We did not know how these models would develop. So we did not want a toolkit that was commited to a particular type of architecture, such as SOAR, ACT-R, BDI, ...

Other toolkit developers were more concerned with issues like security, provability, distribution, portability, efficiency, etc. We were primarily concerned with flexibility, exploratory design, support for multiple programming paradigms, rapid prototyping, interactive testing, and easy extendability.

We also wanted the tookit to have a core that could easily be used by students for interesting projects, at the same time as supporting advanced research and development.

The toolkit must support SCENARIOS WITH RICH ONTOLOGIES

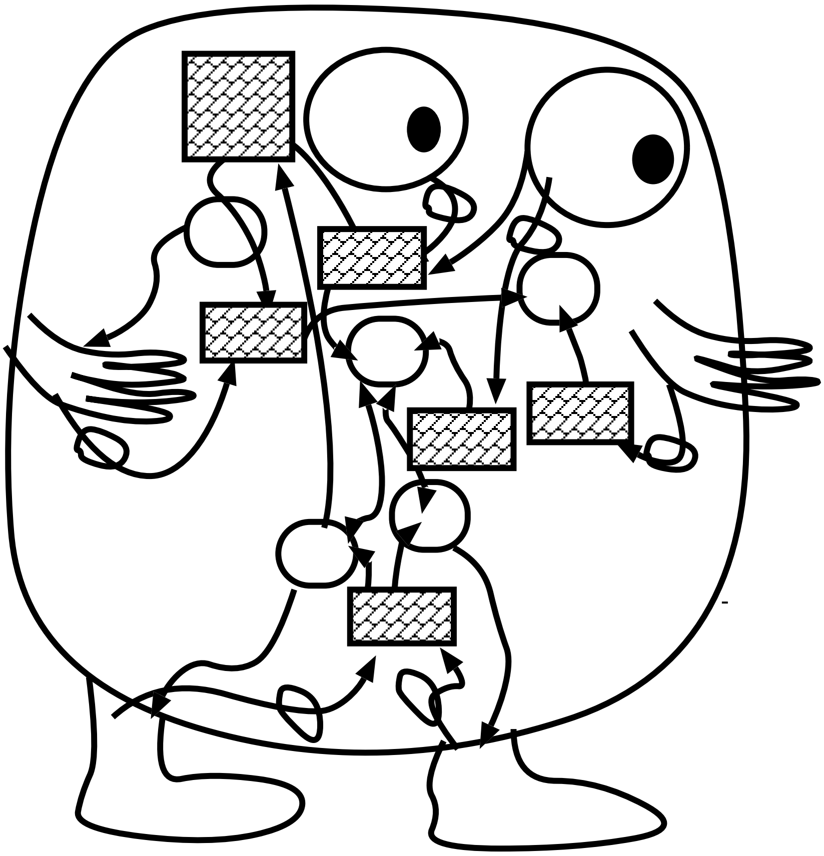


Various kinds of concurrently active entities, e.g.:

- 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, with different architectures

E.g. 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
- with different levels of abstraction, and different levels of control

See the slides for my Cognitive Neuroscience talk in Oxford last week

http://www.cs.bham.ac.uk/˜axs/misc/oxford/

Concurrency is required at all levels

Different entities in the "world" run concurrently.

Different components of individual agents run concurrently, performing various tasks in parallel, e.g.

- **Sensing,**
- **Different levels of perceptual analysis,**
- **Reacting through tight feedback loops,**
- **Linguistic processing (understanding, generating)**
- **Learning of various kinds,**
- **Triggering "alarms" (various kinds of emotion)**
- **Generating new motives, evaluating motives, comparing motives,**
- **Planning, executing plans,**
- **Monitoring internal processes, etc.**
- **Monitoring and reasoning about (simulating) other agents, etc. etc.**

Compare M.Minsky *The Society of Mind*.

Perhaps we need to think of an "ecosystem of mind".

There's more on this in papers in the Cogaff Web directory:

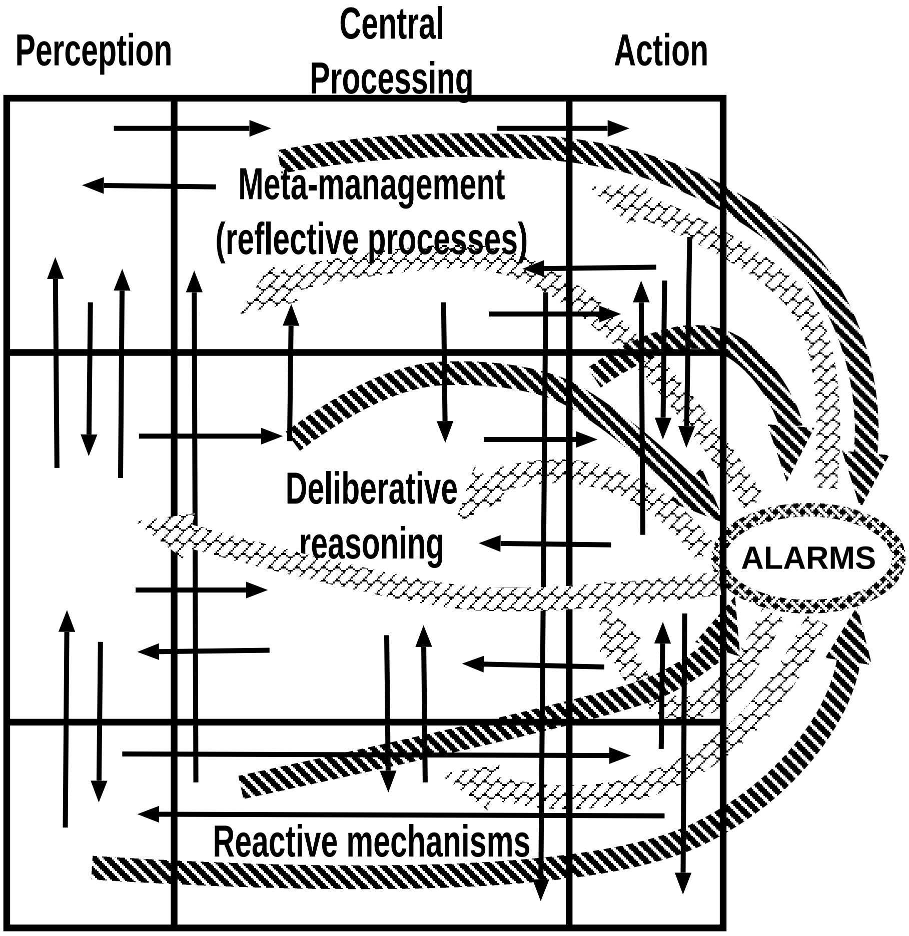
http://www.cs.bham.ac.uk/research/cogaff/

The Birmingham 'CogAff' Architecture Schema

A framework for models with multi-level concurrently active components within perceptual, central and motor sub-systems.

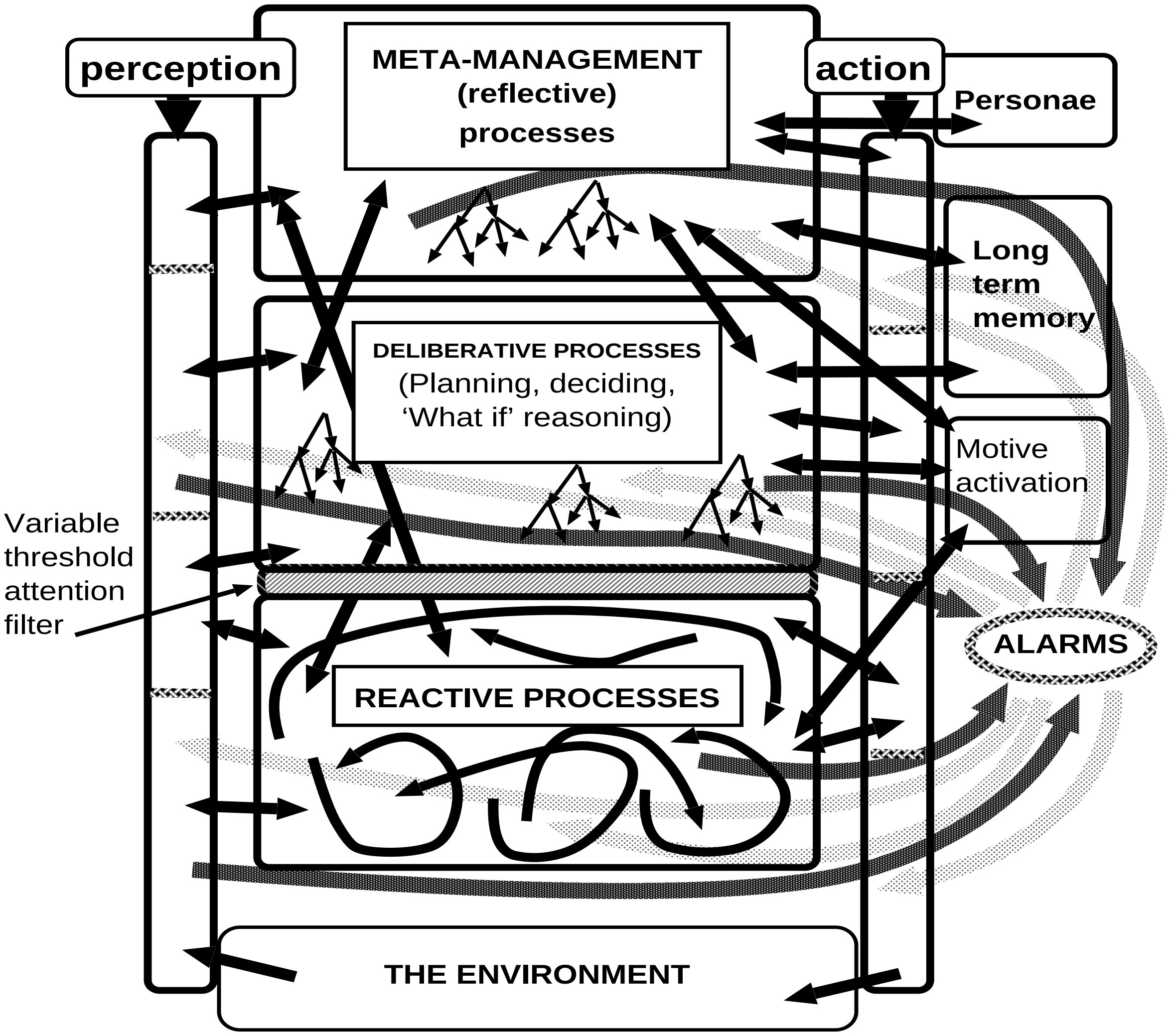
The different levels correspond to different stages in evolution (not all levels found in all animals, or new-born infants!).

They also involve different types of abstraction, different forms of representation, etc.



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

A SPECIAL CASE
H-COGAFF: A HUMAN-LIKE ARCHITECTURE


perception
META-MANAGEMENT (reflective) processes
action
Personae
Long term memory
DELIBERATIVE PROCESSES (Planning, deciding, 'What if' reasoning)
Motive activation
Variable threshold attention filter
ALARMS
REACTIVE PROCESSES
THE ENVIRONMENT

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.

There are various approaches possible, including:

- **User provides *logical axioms* defining new classes and subclasses and new behaviours**
- **User assembles architectures diagrammatically.**
- **User writes lots of low level code!**
- **User defines new classes and sub-classes using an object oriented programming language (e.g. with multiple-inheritance).**

THE LAST IS, AT PRESENT, THE APPROACH SUPPORTED BY SIM_AGENT.

Some tutorial examples

1. The marching platoons demo.

2. Sim_feelings

3. using RCLIB to build a control panel.

4. The sheepdog scenario (Peter Waudby, Tom Carter)

5. the gblocks demo (not strictly sim_agent, but the linguistic and planning modules could be invoked by agent actions.)

Earliest demo

Riccardo Poli’s RIB (Robot in a Box - IJCAI/Atal 1995)

Some examples of student work (Undergraduate, MSc, PhD)

1. The space wars scenario (Morgan Beeby)
2. The Braitenberg vehicles scenario (Duncan Fewkes)
3. Predator-prey (Nick Hawes)
4. Bee-scenarios (various)
5. Ian Wright (Minder 1)
6. Steve Allen (Abbot)
7. Catriona Kennedy (mutual and self monitoring of monitoring)

Jurassic park, various games, etc.

Others ...

E.g. Brian Logan, Darryl Davis, Matthias Scheutz, DERA collaborators

TTHERE ARE MANY TRADE-OFFS IN THE DESIGN OF TOOLS

E.g.

There is a trade-off between flexibility/generality of the toolkit and ease of use, and also between flexibility and efficiency.

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.

(Future work: supported by RCLIB).

SIM_AGENT: IS BASED ON POPLOG ESPECIALLY POP-11

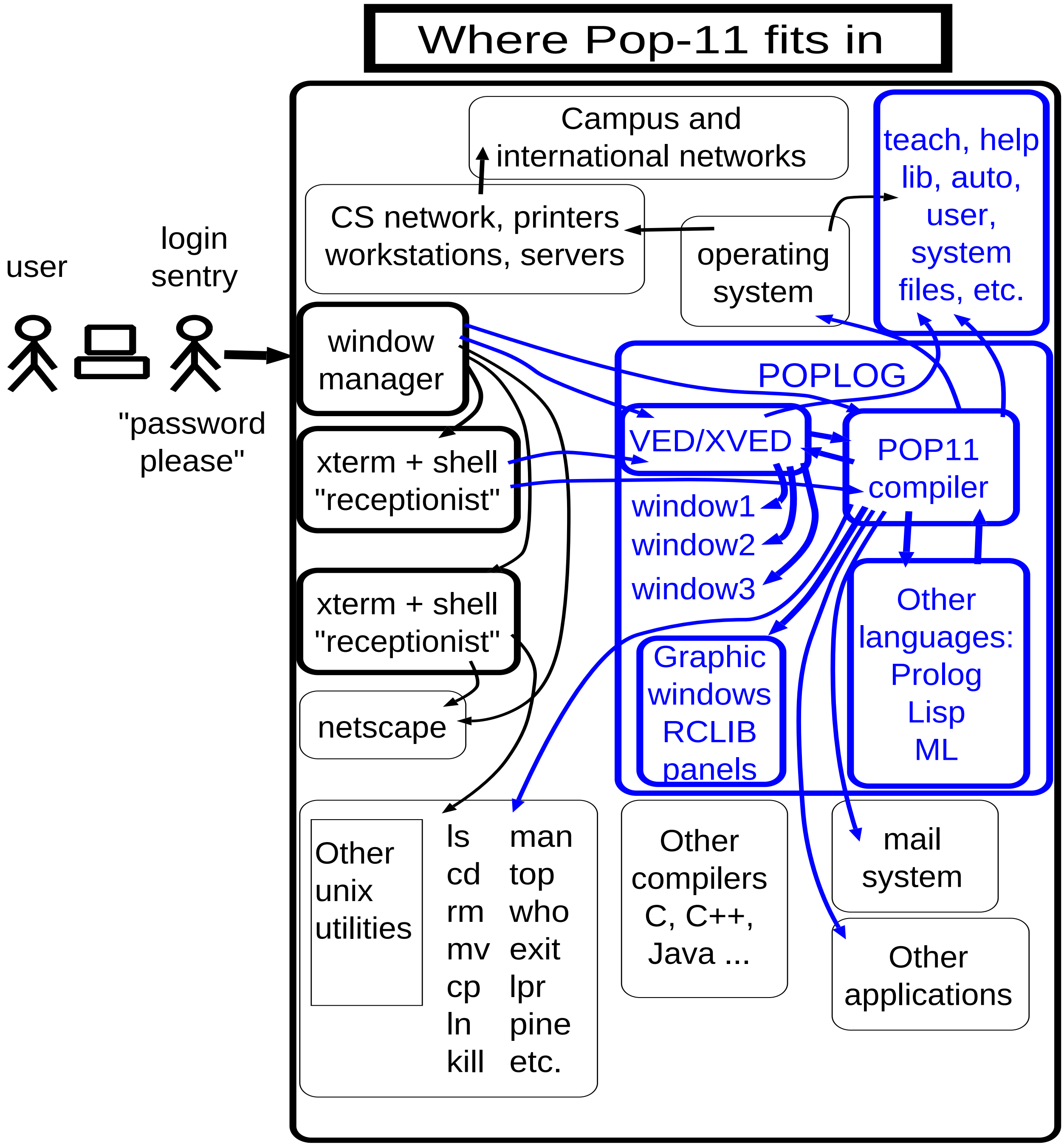
POPLOG: a multi-language AI development environment with incremental compilers for

- **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/Cognitive Science teaching materials**

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

A (PARTIAL) PICTURE OF POPLOG AND POP-11

A diagram produced for student users:



THE ARCHITECTURE OF THE TOOLKIT
Overview:

The toolkit uses Pop-11 extended with

1. Objectclass
(Object Oriented package partly like CLOS)

2. RCLIB (Graphical toolkit)

3. Poprulebase (Rule-based system)

4. SimAgent core libraries
(classes, scheduler, methods etc.)

5. Sim_picagent (combining sim_agent with RCLIB

6. Sim_harness (Tools for setting up scenarios)

... later application libraries, sitting on the Poplog library mechanisms

THE ARCHITECTURE OF THE TOOLKIT: more details

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).**
  IMPORTANT FOR RE-USABILITY AND EVENT DRIVEN PROGRAMMING.
- **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.**
  MULTIPLE-INHERITANCE HAS BEEN USED THROUGHOUT TO ACHIEVE MODULARITY.
  MANY METHODS ATTACHED TO INDIVIDUAL INSTANCES RATHER THAN CLASSES.

..........AND...........

continued...

- **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). Poprulebase supports:**
  - **Rulesets: collections of condition-action rules with various processing strategies.**
  - **Ruleclusters: families of Rulesets, with control switching between them**
  - **Threads: different concurrently active rulesets/clusters**
  - **Conditions and actions can invoke 'lower level' mechanisms (Pop-11, C, Prolog, neural nets...)**
  - **Ruleset-specific and/or dynamic setting of operation modes, e.g. conflict resolution, tracing modes**
  - **Interactive debugging and development: edit and recompile a ruleset after a run has begun**

..........AND...........

continued...

- SIM_AGENT
  provides a scheduler and some default classes and methods for sensing, communicating, acting.

  There is a growing library of utilities.

  The object-oriented design provides a set of basic classes and methods which can be extended for particular applications.

  A collection of rulesets and ruleclusters, executed “in parallel”, forms a *rulesystem* defining an agent’s internal processing architecture.

  There are also user-definable types of sensors and external actions.

  Default communication protocol supported

- SIM_PICAGENT
  Allows individual agents to be in multiple windows

- SIM_HARNESS
  Provides default control panel and startup mechanisms

SUPPORT FOR SELF-MONITORING (E.g. meta-management)

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.

Poprulebase allows meta-rules which get their conditions and actions from the database.

(Some of the self-monitoring mechanisms were partly specified by Catriona Kennedy.)

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.

(Basic X facilities developed at Sussex and ISL – now SPSS)

Distributed agents

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

However, some users have used Pop-11 facilities such as the socket library to implement distributed systems.

That needs to be better packaged for less expert users. It is work to be done. Collaborators welcome!

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.**

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**

What are the language requirements to support this?

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;

SIM_HARNESS automatically sets up control panels

See example from sim_feelings.

But changing defaults, even at run time, is not hard because everything is in Pop-11, which is incrementally compiled, and re-compilable at run time (like Lisp, Prolog, Scheme, (??ML??)...).

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

THERE IS NO FIXED ARCHITECTURE: 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**

Also a database (or several databases) which function as:

- **long term knowledge stores**
- **temporary workspaces**
- **communication channels between subsystems**

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).**
- **Conditions and actions can, if needed, access packages written in other languages supported by Poplog, etc. Prolog, Lisp, ML.**
- **Pop-11 code can access all the Pop-11 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.**

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’, etc.

    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_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 );
  ]

 .... ETC ....
```

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: ‘
           ?state]]
```

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.

There can be unexpected interactions, though, when the external actions are performed.

Developers have to take care.

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.

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)

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.

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?**

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.

Further information:
http://www.poplog.org

To obtain Poplog, and the toolkit libraries, see:

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