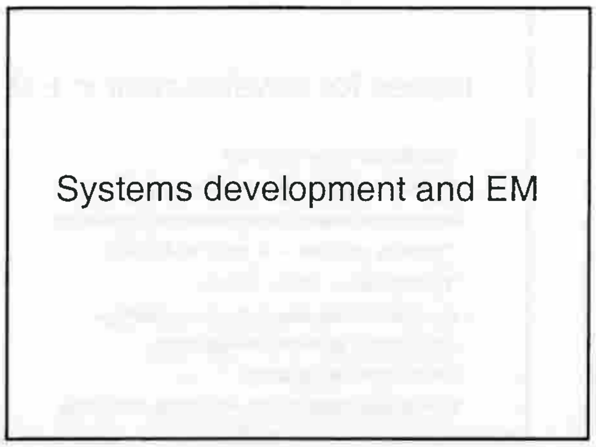
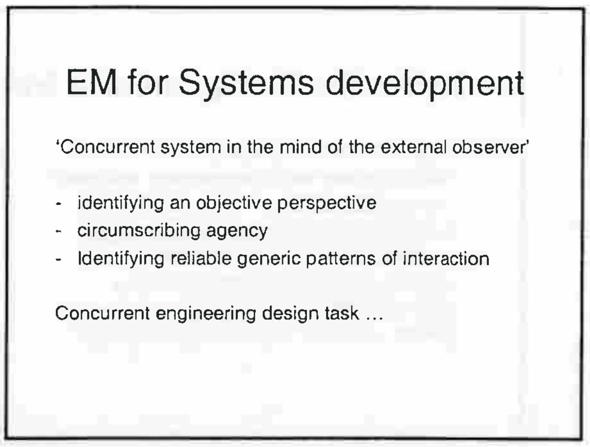


Systems development and EM

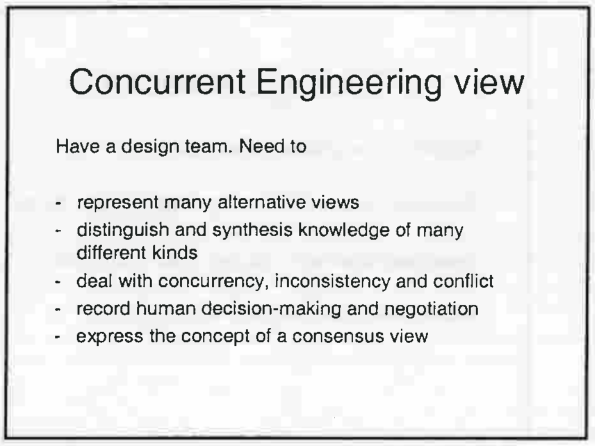


EM for Systems development

'Concurrent system in the mind of the external observer'

- identifying an objective perspective
- circumscribing agency
- Identifying reliable generic patterns of interaction

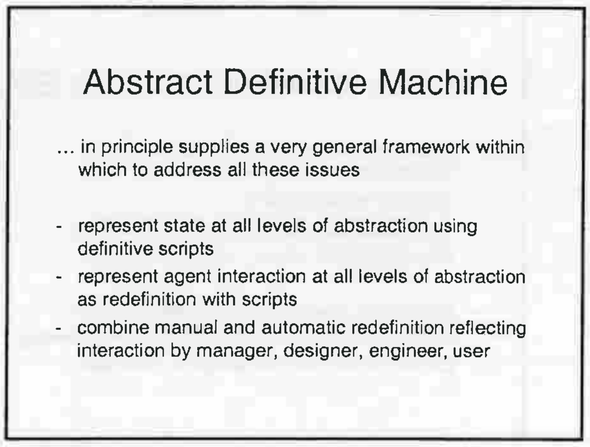
Concurrent engineering design task ...



Concurrent Engineering view

Have a design team. Need to

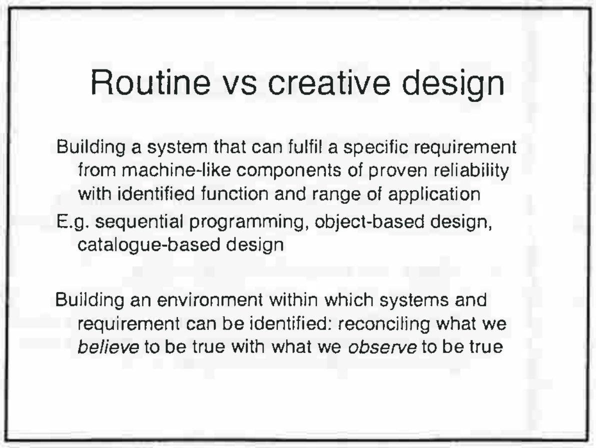
- represent many alternative views
- distinguish and synthesis knowledge of many different kinds
- deal with concurrency, inconsistency and conflict
- record human decision-making and negotiation
- express the concept of a consensus view



Abstract Definitive Machine

... in principle supplies a very general framework within which to address all these issues

- represent state at all levels of abstraction using definitive scripts
- represent agent interaction at all levels of abstraction as redefinition with scripts
- combine manual and automatic redefinition reflecting interaction by manager, designer, engineer, user

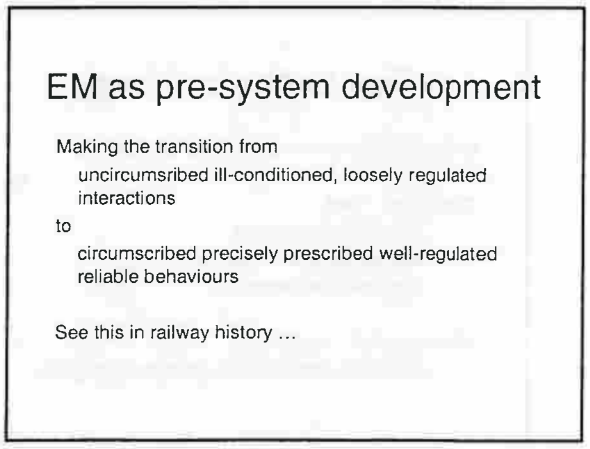


Routine vs creative design

Building a system that can fulfil a specific requirement from machine-like components of proven reliability with identified function and range of application

E.g. sequential programming, object-based design, catalogue-based design

Building an environment within which systems and requirement can be identified: reconciling what we *believe* to be true with what we *observe* to be true



EM as pre-system development

Making the transition from

uncircumsribed ill-conditioned, loosely regulated interactions

to

circumscribed precisely prescribed well-regulated reliable behaviours

See this in railway history ...

Issues for development in EM

- negotiation and elaboration
- learning as involved in requirements and design
  "growing software"
- development as situated problem-solving – amethodical software development
- traditional systems/programs derived by circumscription and optimisation
- how far can object / agent abstractions help?

Issues for development in EM

- *negotiation and elaboration*

  cf. experiential framework for learning
- *learning as involved in requirements and design*

  "growing software" – Brooks, WMB+SBR

  Racing Cars – Simon Gardner
- *development as situated problem-solving – amethodical software development*

  car maintenance analogy

  Pi-Hwa Sun PhD Thesis, Paul Ness, Y-C Chen

Issues for development in EM

- *traditional systems/programs derived by circumscription and optimisation*

cf. OXO in Pascal programs

(~wmb/public/projects/games/OXO/PASCAL)

Allan Wong PhD Thesis – 'beyond systems'

- *how far can object / agent abstractions help?*

  original aspirations of object-oriented programming

  Lind – notion of 'very weak agent' - McCarthy

The logicist debate

McDermott – a Critique of Pure Reason

Celebrated renunciation of faith in logic as basis for AI

Responses collected in Comput. Intell. Vol 3 1987

Brian Cantwell Smith – a non-logicist stance ...

Two Lessons of Logic

Smith identifies two factors of a symbol system:

"first factor" - form

shapes of the symbols

how put together / taken apart

operations and behaviour

"second factor" – content

what the symbols mean, what they are about

The two lessons

Lesson one: the irreducibility of content to form

content relations aren't computed: how symbols 'reach out and touch someone' - almost total mystery

Lesson two: a single theoretical stance

The two factors are relatively independent, but have to be ultimately related: a single, unified theory must provide an account of both factors

Formal logic tenets *not* for AI

1. Use can be ignored
2. Locally the two factors can be treated independently, even though must be globally related
3. Language and modelling should be treated completely differently (promiscuous modelling: can substitute model of X for X with theoretical abandon)

Formal logic tenets *not* for AI

1. Use can be ignored

cf. natural language – context dependence, resolving 'now', pronouns etc

2. Locally the the two factors can be treated independently, even though must be globally related

'if the first factor could be cleft from the second factor, would make sense to write things down first and build programs second' (McD) – not like thought

Formal logic tenets *not* for AI

3. Language and modelling should be treated completely differently

Whole new theories of representation and correspondence will be required:

- explaining computational practice
- promiscuous modelling pernicious where fine grained questions are concerned
- representations in current computational systems range continuously from linguistic to virtually iconic

What Then?

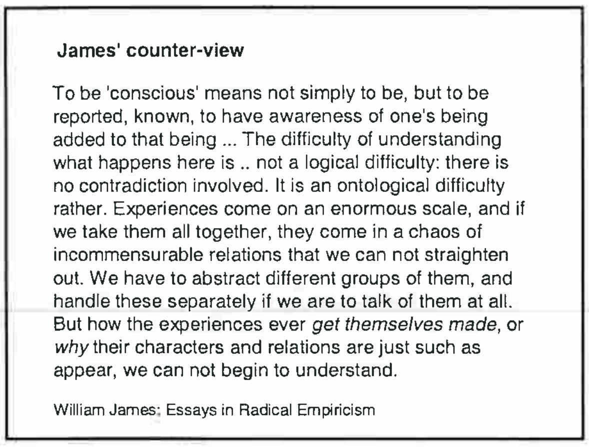
[In] the empirical view ... as reality is created temporally day by day, concepts ... can never fitly supersede perception

William James: A Pluralist Universe c.1900

A rationalist critique of William James's position

... mere experience ... furnishes no consistent view. [The direct products of experience] I find that my intellect rejects because they contradict themselves. They offer a complex of diversities conjoined in a way which it feels is not its way and which it can not repeat as its own ... For to be satisfied, my intellect must understand, and it can not understand a congeries [ie an aggregate] in the lump.

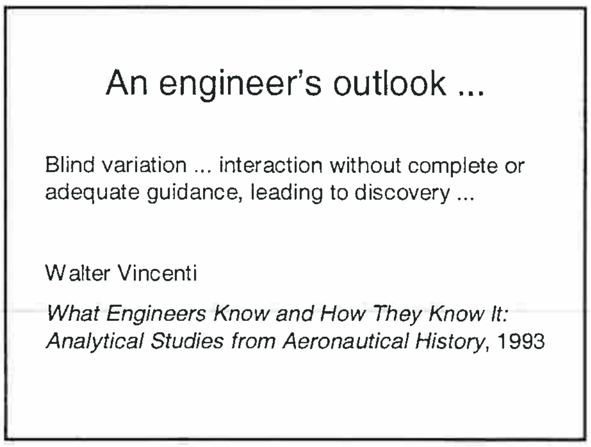
F H Bradley: Appearance and Reality



James' counter-view

To be 'conscious' means not simply to be, but to be reported, known, to have awareness of one's being added to that being ... The difficulty of understanding what happens here is .. not a logical difficulty: there is no contradiction involved. It is an ontological difficulty rather. Experiences come on an enormous scale, and if we take them all together, they come in a chaos of incommensurable relations that we can not straighten out. We have to abstract different groups of them, and handle these separately if we are to talk of them at all. But how the experiences ever *get themselves made*, or *why* their characters and relations are just such as appear, we can not begin to understand.

William James: Essays in Radical Empiricism



An engineer's outlook ...

Blind variation ... interaction without complete or adequate guidance, leading to discovery ...

Walter Vincenti

What Engineers Know and How They Know It: Analytical Studies from Aeronautical History, 1993

Closed world vs. open development cultures in engineering .... [Brödner]

One position, ... the *closed world* paradigm, suggests that all real-world phenomena, the properties and relations of its objects, can ultimately, and at least in principle, be transformed by human cognition into objectified, explicitly stated, propositional knowledge.

The counterposition, ... the *open development* paradigm ... contests the completeness of this knowledge. In contrast, it assumes the primary existence of practical experience, a body of tacit knowledge grown with a person's acting in the world. This can be transformed into explicit theoretical knowledge under specific circumstances and to a principally limited extent only .... Human interaction with the environment, thus, unfolds a dialectic of form and process through which practical experience is partly formalized and objectified as language, tools or machines (i.e. form) the use of which, in turn, produces new experience (i.e. process) as basis for further objectification.

The "Good Experiment" Paradox

A Good Experiment is one in which we really don't know what to expect ...

BUT

... must be some preconception of what to expect

A Good Experiment is one in which we know exactly what will happen ...

BUT

... must be some possibility of being confounded

Essence of Empirical Modelling process

Conviction based on empirical evidence converts experimental activity from 1st to 2nd category

Only a change of **perspective** is involved

NOT

a change in the experimental activity

David Gooding on Science and Philosophy

Most received philosophies of science focus so exclusively on the literary world of representations that they cannot begin to address the philosophical problems arising from the interaction of these worlds: empirical access as a source of knowledge, meaning and reference, and, of course, realism.

John Keats on Negative Capability

... several things dovetailed in my mind, and at once it struck me what quality went to form a Man of Achievement ... - I mean Negative Capability, that is when a man is capable of being in Uncertainties, Mysteries, Doubts without any irritable reaching after fact and reason ...

Letter to George and Thomas Keats
December 1817

... a Challenge and a Caution!

The scientist has a lot of experience with ignorance and doubt and uncertainty, and this experience is of very great importance, I think. When a scientist doesn't know the answer to a problem, he is uncertain. And when he is pretty darn sure of what the result is going to be, he is still in some doubt. We have found it of paramount importance that in order to progress we must recognise our ignorance and leave room for doubt. Scientific knowledge is a body of statements of varying degrees of certainty - some most unsure, some nearly sure, but none absolutely certain.

Richard Feynman (cited in a letter to the Guardia

Communicating from myself to another agent like myself

Empirical Modelling identifies the primitive with 1st person activities: I assert that
"this experience is like this experience"
No-one else can experience this likeness as I experience it in the sense that my experience is divorced from your consciousness and vice versa

[cf the idea that logic provides the most primitive foundation for human activity]

There may be what empirically seems to be a corresponding likeness between two experiences of your own

I may construct an artefact specifically to represent some experience in this manner

It may be that this artefact serves a mediating role in leading you to apprehend what empirically seems to be a similar likeness between your own experiences

Personal validity of my use of artefacts to represent experiences doesn't rely on this

e.g. snooker player uses an idiosyncratic trick to align the cue for a particular shot

Mediation needn't be via a commonly perceived artefact, e.g. invoked linguistically

cf "though I sang in my chains like the sea"

... where words evoke the likeness *but* two private experiences are necessary for the likeness to be apprehended

"singing in chains" as sense of protesting against being captive
"the sea singing in chains" as evoking breaking waves over pebbles

Communication between observers introduces intersubjectivity issues

cf the commonly accepted different nature of certain observables, such as traffic speed readings

cf the way in which mathematical models e.g. finite state machines, geometric references are used

Motivations for Empirical Modelling

Investigation of blends

Explanatory modelling, especially where conflict is involved (e.g. accidents)

Identification of stable contexts

Designing behaviours for concurrent systems of agents

Supporting activities in which concurrent decision-making is involved

Enabling construction through experimentation and human intervention

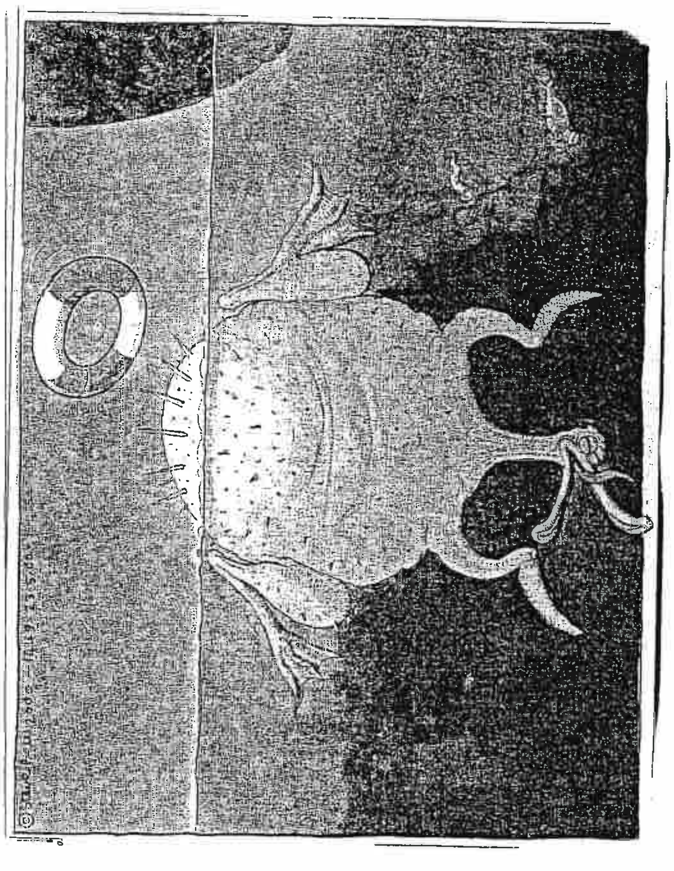
Migration through different paradigms of knowledge represented in the Empiricist Perspective on Learning

Ways in which to organise and interpret the use of definitive scripts

A. 1-agent modelling

B. Modelling the interaction of many agents in a concurrent system

C. Modelling phenomena from several different perspectives concurrently



The SQLEDDI environment

The development of the SQLEDDI environment illustrates a particularly ambitious use of EM in conjunction with education. The overall educational aims of building the environment, and the relevance of EM to achieving these objectives is set out in the following extract from a research proposal. The proposal contains some claims that will be elaborated upon below.

Project Overview

Title of Project: A lightweight portable environment to support teaching of the theory and practice of relational databases

Brief Description:

Anticipated Outcomes: Our proposed environment will
- support a richer learning experience in which practice and theory in relational databases are better integrated, enabling students from various backgrounds to appreciate the significance of relational theory and the flaws in SQL
- supply lightweight open source software that subsumes those aspects of a commercial database system that are most relevant to teaching relational database principles and can be run on any platform
- allow query processing to be integrated with interfaces and visualisation models that are problem specific (in our particular context, relating to timetabling)
- provide students and staff with more flexible and less cumbersome interactive learning and marking strategies.

Aims: In this project, we aim
- to address the problems of motivation encountered by students who find it difficult to relate practical use of complex commercial relational database systems with abstract relational theory.
- to reduce the overheads associated with deploying a commercial RDB system and delineating its core functionality from an academic viewpoint.
- to disseminate new principles for constructing educational software for which we have established proof-of-concept through prior project work at Warwick, and that have wide application.

Methodology: The project will ... illustrate unusual development features that would be exceptionally hard to replicate in a conventional programming paradigm. These include: on-the-fly evolution of notations, visualisation and evaluation techniques specific to the problem under consideration; scope to extend and modify a software environment interactively to take account of needs of different users and issues (e.g. to customise interfaces for particular groups of students, to take account of an issue such as query optimisation, to introduce software agents to monitor or evaluate student responses); potential for convenient reconfiguration for different architectures (e.g. client-server distribution) and modes of use (e.g. amenable to customisation by students, adapted for peer assessment). The evaluation of the project will be integrated with the preparation, delivery and assessment of well-established second and third year database modules (CS233 and CS319).

Deliverables:

The main deliverable will be an extensible environment in which principles of relational database (RDB) query languages can be taught with reference to mathematical foundations and the flaws of SQL. This relational query language (RQL) environment will combine the basic DML and DDL functionality of commercial RDB systems (such as Oracle) with a simple environment for defining and evaluating pure relational algebra expressions (such as was designed by Todd in ISBL). It will have an interface supporting layers of interaction of different degrees of sophistication that is oriented towards teaching basic SQL in a sound theoretical framework and is useable in a consistent way across UNIX, Linux and Windows platforms.

For purposes of assessment, our specific focus of interest will be on relational database use in a timetabling application. For this purpose, we shall integrate our RQL environment with a pre-existing timetabling model and develop an interface to this environment that will (in particular) enable users to visualise the results of queries, support the creation of assignments that are different for each student, and allow such coursework to be marked efficiently and consistently by tutors.

Outline rationale for embedding innovation / method within teaching and learning (include any evaluation outcomes)

A central concept behind our project is the open-ended development of educational resources that are specifically targetted at supporting a particular academic requirement (viz. relational query languages in theory and practice) and a specific application (viz. timetabling). This kind of development is quite different in character from the development of a monolithic environment whose functionality is carefully preconceived to admit many different applications, but is effectively not open to modification by its users. The model-building approach we have developed is especially well-suited to situations in which there is a specific focus of interest, but there are a whole range of different agents and perspectives that must be recruited on-the-fly as a situation evolves. In the long term, subject to appropriate development both of our tools and of understanding of the principles behind their use, we envisage that our RQL environment will be amenable to customisation by academics with different teaching agendas and styles and by students with different interests and learning strategies.

Comments on the proposal

The history of the development of SQLEDDI to this point provides some context.

The original version of eddi from which SQLEDDI originated is to be found in eddipTruong1996. Another version, very similar in functionality, but integrated more closely with tkeden is found in agentparserBrown2001. In its original form (July 2001), Brown's eddi parser was quite primitive. For instance: it had no catalogue, had no garbage collection, didn't handle sytax errors well, didn't allow renaming of attributes. The transition from this state to the current eddi interpreter (stabilised in November 2001) was an incremental activity that is arguably (to some extent - and with apologies to Ashley) self-documented in the library file
~empublic/emp/lib/lib-tkeden-1.46/eddi.eden.
The significant point is that the entire development was associated with incremental on-the-fly modification of the interpreter which took place in the very same context as the testing and exploration of potential design and implementation strategies. It was also accompanied by and motivated by a parallel enhancement of the interpreter to handle the translation of SQL to eddi, for which the essential framework had been supplied by Michael Evans's implementation of an agent-oriented SQL parser / translator.

A good analogy for traditional development is that, in (say) building a car, you road test it until you find a context in which it doesn't function properly, then return to the garage, explain the problem, then go out again in your modified car trying to 'revisit' the context in which you formerly had difficulties. In EM, the aspiration is quite different. The current state always subsumes the current status of your artefact and its present state and context, so that modifying the artefact is like repairing / redesigning your car in the context where it isn't meeting your requirements. In distinguishing EM development from other forms of use of interpreters in development, the contention is that fixing the interpreter is actually redesigning the context in a way that is inconsistent with the 'continuous relation' that is characteristic of an ongoing experience.

From an educational perspective, the development of SQLEDDI was carried out whilst teaching CS233 in exactly the above spirit. The best documentation of this activity from an educational perspective is to be found in the worksheets (worksheets 5 and 6 on the CS233 website) that accompanied the software construction. The implementation of a subset of standard SQL that emerged from this activity was designed on-the-fly after experiments that showed the infeasibility of merely enhancing the eddi evaluation strategies. The development of SQL0 can be seen as a form of scaffolding that is essential to understanding the problematic nature of giving good mathematical semantics to standard SQL.

~empublic/lib/lib-tkeden-1.46/eddi.eden

Some extracts from the eddi parser development

670 where the eddi parser specification first begins

```
eddi_statement = ["suffix", ";", "eddi_statement_1", ["fail",
"eddi_statement_1"]];
```

673 [comment]
```
eddi_statement_1 = ["prefix", "?", "table_val", ["script", ["declare",
"relname"], ["setparas", ["relname"]], ["later", "showrel (%%);", "relname"]],
["fail", "eddi_statement_2"]];
```

676
```
eddi_statement_1 = ["prefix", "?", "rel_exp", ["script", ["declare",
"relname"], ["setparas", ["relname"]], ["later", "showrel (%%);", "relname"]],
["fail", "eddi_statement_2"]];
eddi_statement_2 = ["prefix", "~", "table_name", ["script", ["declare",
"relname"], ["setparas", ["relname"]], ["later", "droptable (%%);",
"relname"]], ["fail", "eddi_statement_3"]];

eddi_statement_3 = ["pivot", "<<", ["table_name", "tuples"], ["script",
["declare", "relname"], ["setparas", ["relname"], []], ["later", "$$ = addvals
($$, %%);", "relname", "relname", "v_substrs[2]"]], ["fail",
"eddi_statement_4"]];
eddi_statement_4 = ["pivot", "!!", ["table_name", "tuples"], ["script",
["declare", "relname"], ["setparas", ["relname"], []], ["later", "$$ = delvals
($$, %%);", "relname", "relname", "v_substrs[2]"]], ["fail",
"eddi_statement_5"]];

eddi_statement_5 = ["pivot", "is", ["table_name", "rel_exp"], ["script",
["declare", "relname", "expr"], ["setparas", ["relname"], ["expr"]], ["later",
"$$ is %%;", "relname", "expr"]], ["fail", "eddi_statement_6"]];
eddi_statement_6 = ["pivot", "(", ["table_name", "create_state"], ["script",
["declare", "relname", "cdata"], ["setparas", ["relname"], ["cdata"]],
["later", "$$ = create ($$); addtotablelist (\"$$\");", "relname", "cdata",
"relname"]], ["fail", "eddi_statement_7"]];
```

Initial functionality - query expressions (1), drop tables (2), insert tuples (3), delete tuples (4), define views (5), and create intensional relation (6).

685 precursor to the catalogue table - lists the currently known tables

```
eddi_statement_7 = ["literal", "#", ["script", ["execute", "listtables ();"]]];
```

The function of eddi_statement_7 is later overwritten by assignment of
relations. The

1899 the listing of tables in response to '#;' is replaced by display of the CATALOGUE:

```
eddi_statement_8 = ["literal", "#", ["script", ["execute", "curr_agent=8;
showrel(CATALOGUE);"]]];
```

1677 notion of a 'current agent' added to disambiguate observation

```
eddi_statement_1 = ["prefix", "?", "rel_exp", ["script", ["declare",
"relname"], ["setparas", ["relname"]], ["later", "showrel (%%);", "relname"],
["execute", "curr_agent = 1;"] ], ["fail", "eddi_statement_2"]];
```

1716 gives a new definition of the pivot() function, which describes a generic form of observation in the parser. Now the parser consults the current agent to determine whether to pivot on a key symbol or not. For instance, it is appropriate to pivot on the '.' in X.Y, where X and Y are relational tables, but not such a good idea to pivot on the decimal point in 3.14. Observation qualified by context is an important idea here.

1870 CATALOGUE added, and so checking of CATALOGUE became necessary in the evaluation of a query

extract of code and comment below suggests that (though idea of checking consistency with the CATALOGUE when observing the entire expression to be evaluated was explored) checking of the CATALOGUE was subsequently effected step-by-step during the evaluation of relational expressions

1901

```
eddi_statement_1 = ["prefix", "?", "rel_exp", ["script", ["declare",
"relname"], ["setparas", ["relname"]], ["later", "showrel (%%);", "relname"],
["execute", "curr_agent=1;"]],["fail", "eddi_statement_2"]];

/* handle this checking elsewhere in parsing rel expressions

eddi_statement_1 = ["prefix", "?", "rel_exp", ["script", ["declare",
"relname"], ["setparas", ["relname"]], ["later", "if (search_CAT(%%)!=0)
showrel (%%); else writeln(\"Table not in catalogue\");", "relname",
"relname"], ["execute", "curr_agent=1;"]],["fail", "eddi_statement_2"]];

*/
```

1869 The introduction of the CATALOGUE involves some bootstrapping. There was initially no type checking on the insertion of tuples, but it became necessary to add this later. The addition of this type checking was assisted by a happy coincidence: that Truong had implemented some (misconceived) structure for handling key attributes in his table representations - this was redundant to this point in the eddi implementation using the agent-oriented parser, but was appropriated here to serve a role in typechecking.

```
%eddi
_CATALOGUE ( relname CHAR, reltype CHAR, defn CHAR, ptr POINTER);

%eden

/* this line was added after type checking: it makes use of redundant fields in
the relation table record that had been inappropriately used for key
information */


_CATALOGUE[1][1] = ["string", "string", "string", "pointer"];

%eddi

CATALOGUE is _CATALOGUE % relname, reltype;
```

Evolution of view definitions in eddi

682 most primitive form of view definition

```
eddi_statement_5 = ["pivot", "is", ["table_name", "rel_exp"], ["script",
["declare", "relname", "expr"], ["setparas", ["relname"], ["expr"]], ["later",
"$$ is %%;", "relname", "expr"]], ["fail", "eddi_statement_6"]];
```

1683 with a current agent setting added

```
eddi_statement_5 = ["pivot", "is", ["table_name", "rel_exp"], ["script",
["declare", "relname", "expr"], ["setparas", ["relname"], ["expr"]], ["later",
"$$ is %%;", "relname", "expr"], ["execute", "curr_agent = 5;"] ], ["fail",
"eddi_statement_6"]];
```

1909 Placing views in the CATALOGUE

```
eddi_statement_5 = ["pivot", "is", ["table_name", "rel_exp"], ["script",
["declare", "relname", "expr"], ["setparas", ["relname"], ["expr"]],
["execute", "curr_agent=5; instr = v_string;"], ["later", "if
(search_CAT(%%)==0) { $$ is %%; _CATALOGUE = addvals (_CATALOGUE, [\"$$\",
\"view\", instr, &$$]); } else writeln(\"Table name already in use\");",
"relname", "relname", "expr", "relname", "relname"]], ["fail",
"eddi_statement_6"]];
```

1991 Ensuring that views that are not inapproriately defined don't get placed in the CATALOGUE

```
eddi_statement_5 = ["pivot", "is", ["table_name", "rel_exp"], ["script",
["declare", "relname", "expr"], ["setparas", ["relname"], ["expr"]],
["execute", "curr_agent=5; instr = v_string;"], ["later", "if
(search_CAT(%%)==0) { if (%%!=@) {$$ is %%; _CATALOGUE = addvals (_CATALOGUE,
[\"$$\", \"view\", instr, &$$]);} else writeln(\"Invalid definition\"); } else
writeln(\"Table name already in use\");", "relname", "expr", "relname", "expr",
"relname", "relname"]], ["fail", "eddi_statement_6"]];
```

Evaluation strategy for relational DB expressions

Even when simply querying or assigning a relational value, the expression is (temporarily) built up using dependency links. For instance: when executing the query

```
? allfruits : begin < 3 % name, end;
```

an expression tree similar to that needed to store the view definition

```
X is allfruits : begin < 3 % name, end;
```

is constructed (see AOP.dmt). After an eddi statement has been executed all temporary variables (of the form var_*) that have a dependee in the CATALOGUE are retained. Those that do not can be garbage collected: for this purpose, each var_* is "assigned to itself", after which it csan be forgotten. (Note that forgetting a variable in eden requires that no variable should depend upon it.)

How the eddi statement parsing evolves

1984 accommodating the describe table feature (?? X;)

```
eddi_statement = ["suffix", ";", "eddi_statement_0", ["fail",
"eddi_statement_0"]];

eddi_statement_0 = ["prefix", "??", ["describetbl"], ["fail",
"eddi_statement_1"]];
```

2151 Introducing comments into eddi (## ...)

```
eddi_statement = ["suffix", ";", "eddi_comment", ["fail", "eddi_comment"]];

eddi_comment = ["prefix", "##", ["anything"], ["fail", "eddi_statement_0"]];
```

2274 post garbage collection in eddi

```
eddi_statement = ["suffix", ";", "eddi_comment", ["script", ["later",
"garbage_collect();"]], ["fail", "eddi_comment"]];
```

How the evaluation of relational expressions evolves.

695 - 705 original parsing of rel_exp is as follows:

```
rel_exp   = op_exp ('+', "rel_exp_1");
rel_exp_1 = op_exp ('-', "rel_exp_2");

rel_exp_2 = op_exp ('*', "rel_exp_3");
rel_exp_3 = op_exp ('.', "rel_exp_4");

rel_exp_4 = ["rev_pivot", "%", ["rel_exp", "attr_list"], ["script", ["declare",
"expr_part", "attr_part"], ["setparas", ["expr_part"], ["attr_part"]],
["execute", "%% is " // lookupop ('%') // "(%%, %%);", "v_paras[1]",
"expr_part", "attr_part"]], ["ignore", ["bras"]], ["fail", "rel_exp_5"]];
rel_exp_5 = ["rev_pivot", ":", ["rel_exp", "predicate"], ["script", ["declare",
"expr_part"], ["setparas", ["expr_part"], []], ["execute", "%% is " // lookupop
(':') // "(%%, ##);", "v_paras[1]", "expr_part", "v_substrs[2]"]], ["ignore",
["bras"]], ["fail", "rel_exp_6"]];

rel_exp_6 = ["prefix", "(", "rel_exp_7", ["fail", "table_val"], ["script",
["setparas", ["v_paras[1]"]]]];
rel_exp_7 = ["suffix", ")", "rel_exp", ["script", ["setparas",
["v_paras[1]"]]]];
```

The file prec.e changes the precedence of the relational expression evaluation

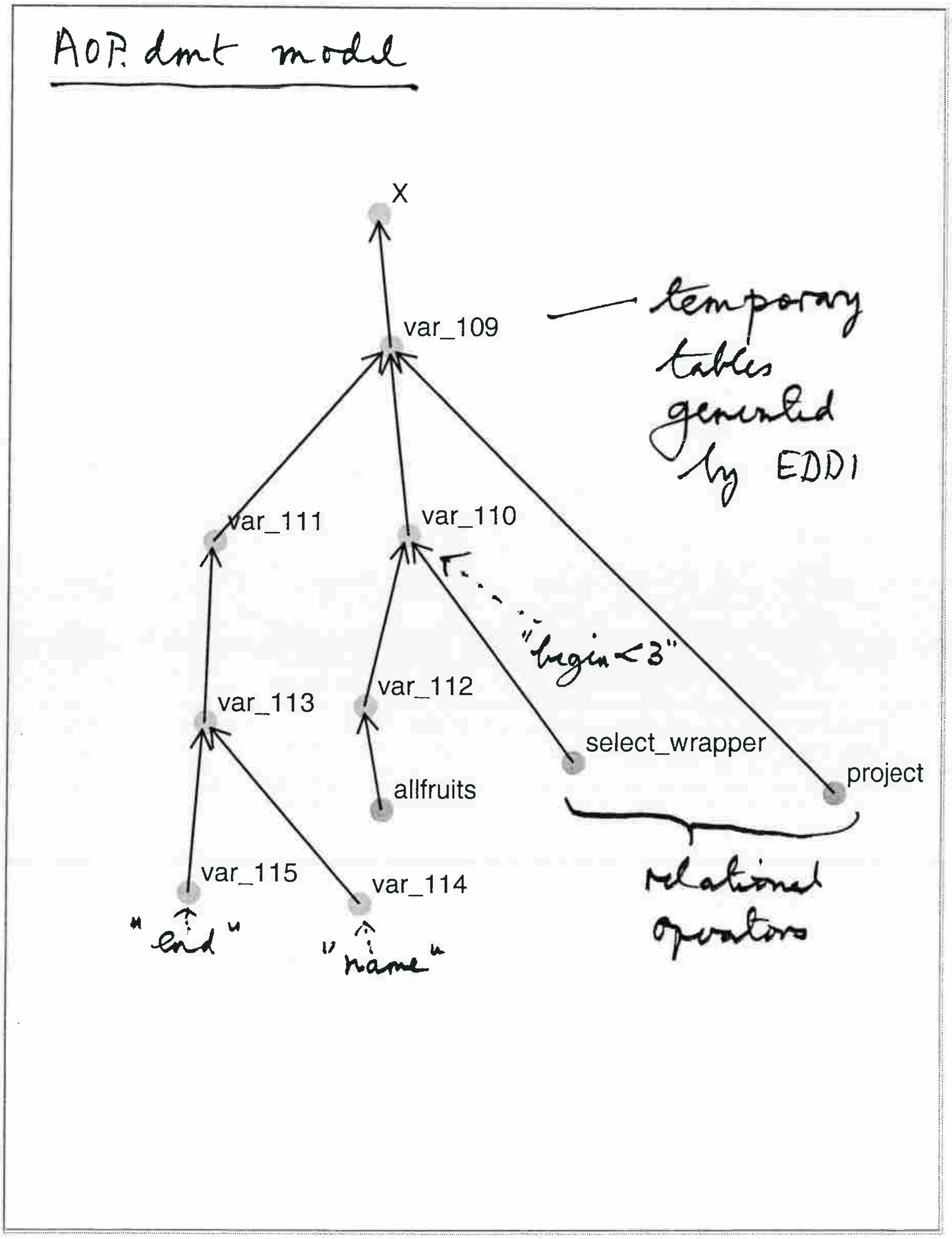
1836 - 1847

```
%eden
/* this file changes the precedence of the eddi ops - giving higher precedence
to * than to % and : and lowest precedence to +, - and .
*/
rel_exp   = op_exp ('+', "rel_exp_1");
rel_exp_1 = op_exp ('-', "rel_exp_2");

rel_exp_5 = op_exp ('*', "rel_exp_6");
rel_exp_2 = op_exp ('.', "rel_exp_3");

rel_exp_3 = ["rev_pivot", "%", ["rel_exp", "attr_list"], ["script", ["declare",
"expr_part", "attr_part"], ["setparas", ["expr_part"], ["attr_part"]],
["execute", "%% is " // lookupop ('%') // "(%%, %%);", "v_paras[1]",
"expr_part", "attr_part"]], ["ignore", ["bras"]], ["fail", "rel_exp_4"]];
rel_exp_4 = ["rev_pivot", ":", ["rel_exp", "predicate"], ["script", ["declare",
"expr_part"], ["setparas", ["expr_part"], []], ["execute", "%% is " // lookupop
(':') // "(%%, ##);", "v_paras[1]", "expr_part", "v_substrs[2]"]], ["ignore",
["bras"]], ["fail", "rel_exp_5"]];
```

First rules applied are those that correspond to operators of lowest precedence in the evaluation process.



Internal dependency structure generated by view definition:

X is allfruits : begin < 3 % name, end.

Analysis of the model used for the nodes of the heap suggests some refinement of the design of the heap artefact.

The interpretation of the clusters of observables are as follows:

```
_N3 is  cart(75 + 3 * _sep, 70);
_S3 is  cart(75 + 3 * _sep, 50);
_V3 is  line(_S3, _N3);
_elt3 is  scalar_div((vector_add(_S2, _N3)), 2);
_v3 is label(str(val3), _elt3);
A_v3 is  "color="// ((( val3==minelt) || (val3==maxelt))? RED: GREEN);
etc
```

refers to vertical lines in the boxed array of values displayed at the top of the screen

_elt3 is the location of the third value in the boxed array: it is labelled by the string _v3, which is RED if the element is maximal or minimal, and is otherwise GREEN

```
_l37 is  newline(_p3,_p7);
_l36 is  newline(_p3,_p6);
_p3 is cart(600,500);
_c3 is  circle(_p3, _radius);
etc
```

refers to the display of node 3 in the tree, the circle around it, and the lines incident with it. The value associated with node 3 in the tree is recorded in the label _lab3. The line that connects the donald points p3 and p7 associated with nodes 3 and 7 is an adaptation of the standard donald line() function that produces a short line.

The attributes of the circle surrounding node 3 are determined by A_c3, which is associated with the subscript:

```
A_c3 is  "color="//((inhp3)? c3col: WHITE);
c3col is  (hc3)?BLUE: RED;
ord37 is  cf(3,7,htree);
ord36 is  cf(3,6,htree);
ord13 is  cf(1,3,htree);
inhp3 is  inheap(3, first,last);
hc3 is  (!inhp3) || (inhp3 && (!inhp6 || (inhp6 && (ord36 >= 0)) && (!inhp7 ||
(ord37 >= 0))));
```

hc3 expresses the heap condition at node 3, according to whether node 6 is in the heap (as expressed by inhp6) etc.
cf() is a comparison function: htree is the list of values - identical to val, but computed from the definitions:

```
htree is proj2(harray);
harray is [node1,node2,node3,node4,node5,node6,node7];
node1 is [1, val1];
val1 is val[1];
etc
```

As A_c3 ensures, the colour of the circle around node 3 is WHITE (the "background" colour) is the node is not in the heap, and is otherwise BLUE or RED according to whether the heap condition is / isn't satisfied at node 3.

```
A_l37 is  "color="//((inhp3&&inhp7)?((ord37==0) ?BLACK:((ord37== 1)?BLUE:
RED)): WHITE);
```

is the definition that determines the colour of the line that connects node 3 to node 7 - if both are in the heap, it makes the line BLACK / RED / BLUE according to whether the values at node 3 is equal to / is less than / is greater than that at node 7.

The following definition computes the index of the child node of 3 that has the greater value, subject to being within the heap.

```
ixgtch3 is  (!inhp7) ? 6: ((val[6]>val[7]) ? 6: 7);
```

The coincidental choice of colours for labelling values in the boxed array and the circles and lines associated with tree nodes creates spurious edges in the dependency model.

It is apparent that if the colour of circles were not chosen to be BLUE or RED (coinciding with the colours of lines), there would be no edges between the c3col and the BLUE and RED nodes, nor between the RED node and the A_v3 node.

Morals of the exercise - controversial!

About EM

It is possible through modelling with artefacts to make highly sensitive and discriminating semantic distinctions
- cf promiscuous modelling [Cantwell-Smith] using abstract mathematical models

It is possible to construct computer models without predetermined goals, formal IO behaviours and use-cases: uncertain interaction in the world is conceptually prior to formalisation

We are just at the beginning of understanding how to organise and manage EM model development: need better tools, but also need unorthodox strategies for dealing with issues such as versioning and documenting.

About classical computing

Much of classical computer science is motivated by optimisation to specific goals (as was historically expedient); clearly specification of function is a prerequisite for optimisation, but it is a mistake to imagine that such specification is an essential prerequisite for constructing computer models.

Functional specification is not very prescriptive where program development is concerned

Abstract data types do not really capture the analogue characteristics of data types as pseudo-physical artefacts that embody understanding, and underpin mental models to aid the imagination

About 'computers for learning'

How we construct models is highy relevant to how useful they can be in supporting learning cf. Logo as a vision for constructionism

Interaction that is creative in character has to accommodate situation, ignorance and nonsense (the SIN principle) - essence of learning is the capacity not to know, to be suspicious of perfect knowledge, and to be able to make mistakes; these require support from modelling approaches that negotiate meaning through interactive development rather than prescribe the interpretation of interaction in advance

About the relationship between EM and classical programming / software development

The distinction between EM and rapid prototyping and iterative development as in XP is ontological rather than being simply a matter of degree: the basis for EM is in the notion of construal, and continuity in EM development is in the 'stream of thought' (cf car maintenance analogy in the discussion of the sqleddi environment)

The 'experimental paradox' is central to the distinction between EM and the classical theory of computation

It is more appropriate to seek a foundation for specification and logic in EM than to seek a foundation for EM in specification and logic

The single square of the draughts model can be isolated and used to disclose problems in definition. The model should be read in conjunction with the dmt model: draughtssq68sde.dmt in the Models directory associated with DMT072.

This shows one of the squares of the draughts board: square number 68

To load the display it is necessary to execute the Run.e file. Because of the way the model is built, it is not enough to define the position of a single piece, as in

```
b11 = [6,8];
```

since the visualisation operates with dependency driven by the lists wpieces, bpieces, wcrowned, bcrowned etc.

```
/* valid positions for pieces are:
        [1,2], [3,1], [5,1], [7,1], [2,2], [2,4] etc
 and a piece that is placed off the board is at [1000,1000] - initially:

 b1 = [2,6];
 b10 = [4,8];
 b11 = [6,8];
 b12 = [8,8];
 b2 = [4,6];
 b3 = [6,6];
 b4 = [8,6];
 b5 = [1,7];
 b6 = [3,7];
 b7 = [5,7];
 b8 = [7,7];
 b9 = [2,8];
 bcrowned = [0,0,0,0,0,0,0,0,0,0,0,0];
 w1 = [1,1];
 w10 = [3,3];
 w11 = [5,3];
 w12 = [7,3];
 w2 = [3,1];
 w3 = [5,1];
 w4 = [7,1];
 w5 = [2,2];
 w6 = [4,2];
 w7 = [6,2];
 w8 = [8,2];
 w9 = [1,3];
 wcrowned = [0,0,0,0,0,0,0,0,0,0,0,0];

*/
```

An unfortunate aspect of checkcol() is the hidden role for a global variable: blank, that is set to the value "blank" in the actual script:

```
func checkcol {
        para coord;
        auto ans,i;
        ans = blank;    /* 'blank' is global */

        for (i=1;i<=12;i++){
        if (`"b"//str(i)` == coord)
                ans = black;
                }
        for (i=1;i<=12;i++){
        if(`"w"//str(i)` == coord)
                ans = white;
                }
        return ans;
}
```

Can move a piece off the square [6,8] by redefining

```
b11 = [1000,1000];
```

say: can then move the piece w11 to this square:

```
w11 = [6,8];
```

The model works because bgcol in the definition of square68:

```
window square68 = {
    type: DONALD
    box: [p68, q68]
    pict: "piece68"
    bgcolor: bgcol
    sensitive: ON
};
```

happens to be the same as bgcolor, as used in the definitions:

```
colsqr68 is (Square68 == white)? white:((Square68 == black)? black: bgcolor);
A_Piece68 is "fill=solid,color=" // colsqr68;
```

which are used to determine the colour of the donald circle representing the draughts piece.

If we change the background of the scout window to red, via

```
%scout
bgcol = "red";
```

and then move the piece from this square, via

```
%eden
w11 = [1000,1000];
```

the hidden problem in the colour definition for the square is disclosed.

Can check the effect of crowning a piece also:

```
wcrowned = [0,0,0,0,0,0,0,0,0,0,0,1,0];
```

This (unexpectedly!) has no effect on the colour of the king dot: kingcol68

This is because there is a dependency of the form:

```
iscrowned ~> [kingcol68];
```

but nothing to ensure that iscrowned is redefined when wcrowned or bcrowned is altered. To fix this, we would need to change the definition of

```
kingcol68 is (iscrowned([6,8]) == 1)? red: colsqr68;
```

to the form

```
kingcol68 is (iscrowned([6,8], wcrowned, bcrowned)==1) ? red : colsqr68;
```

as in the definition of Square68 with reference to bpieces and wpieces which checkcol does not use explicitly in computing its result:

```
Square68 is checkcol([6,8], bpieces, wpieces);
```

A further (unresolved) problem is that there is no layering in DoNaLD, so that when we arrange for iscrowned([6,8]) to be true, the white circle representing the piece may not get displayed beneath the red dot.