# Empirical Modelling and Geometry: Design and Implementation

Meurig Beynon and Richard Cartwright

**Department of Computer Science**
**University of Warwick**
**Coventry CV4 7AL**

**Content of talk**

1. Review relationship of Empirical Modelling to Geometry
2. Examine the current EDEN implementation
3. Identify issues for further research
4. Discuss the geometric modelling notation CADNO
5. Describe the JaM and DAM abstract machine models
6. Draw conclusions

**Background to Empirical Modelling (EM)**

EM is a new approach to computer-based modelling

new fundamental principles for the construction of models

unusual association between form and content

systems analysis wrt observables, dependencies and agency

construct computer-based artefacts whose states metaphorically represent the observed states of the system being modelled.

**Empirical Modelling and Geometry**

Metaphor typically visual - so geometry important in EM

design of definitive notations for geometry

| | |
|---|---|
| DoNaLD | line drawing |
| ARCA | combinatorial graphs |
| Scout | window layout |
| CADNO | geometric modelling |

EM also potentially helpful in specifying graphics

construction of images guided by semantics

scope to explore computer as geometric instrument

**Implementation using EDEN**

EDEN = Evaluator for Definitive Notations

**A Simple Illustrative Example**

The Jugs model

*definitions*

```
contentA = ...
capA = ...
fullA = capA==contentA
avail_option_fillA = not fullA
colour_button_fillA = if avail_option_fillA
                      then white else black
```

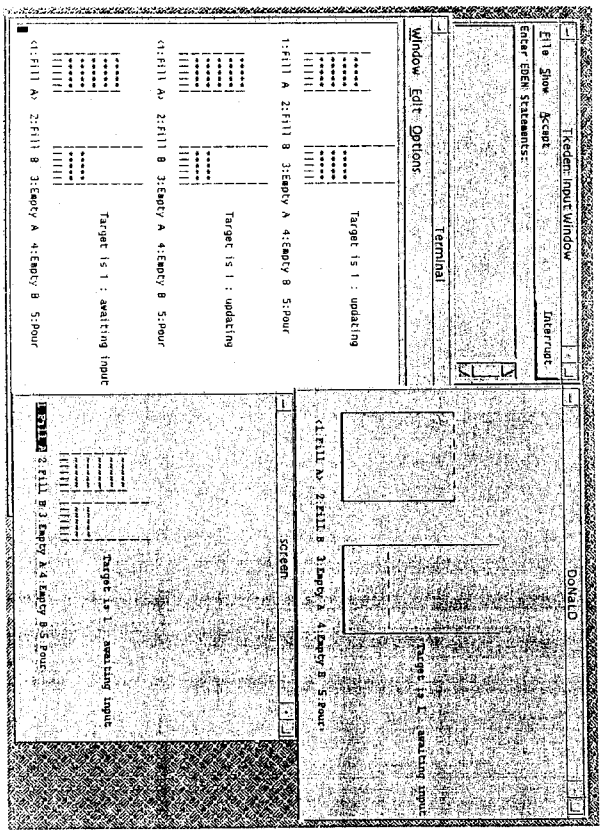
*super_agent action* `contentA = target`

*user_interaction*

```
if avail_option_pour then input=pour
```

*event-driven action*

```
if input==pour and not emptyA and not fullB then
{     input=pourAB;
      contentB=|contentA+contentB|-contentA
}

if input==pourAB and not emptyA and not fullB
then contentA=|contentA|-1
```



### Visualisation of the jugs

Graphical elements are constructed as line-drawings or windows defined in terms of relevant parameters.

```
jugA = F (contentA, capA)
```

where `F` is a user-defined function

Appropriate functions `F` can return
- a DoNaLD line-drawing
- a SCOUT window
- a textual string

In the JUGS model ...

**observables** are parameters such as

- content of jugs (e.g. `contentA`)
- capacities of jugs (e.g. `capA`)
- whether can be filled (e.g. `avail_option_fillA`)
- whether target has been attained (`status`)
- etc

**dependencies** connect visualisation and menu button status to contents and capacities

possible **agents** are
- the pupil, in roles as represented by buttons
- the teacher, with discretion to change parameters

## Modelling in EDEN

**observables** are represented by EDEN variables

**dependencies** by definitions, with operators by pure functions

automatic **agents** by actions triggered by active values

user of the model can act as super-agent to emulate any agent

## EDEN Implementation of Definitive Notations

EDEN has scalar, string and recursive non-homogeneous lists

Interpretation of complex types in geometric notations involves

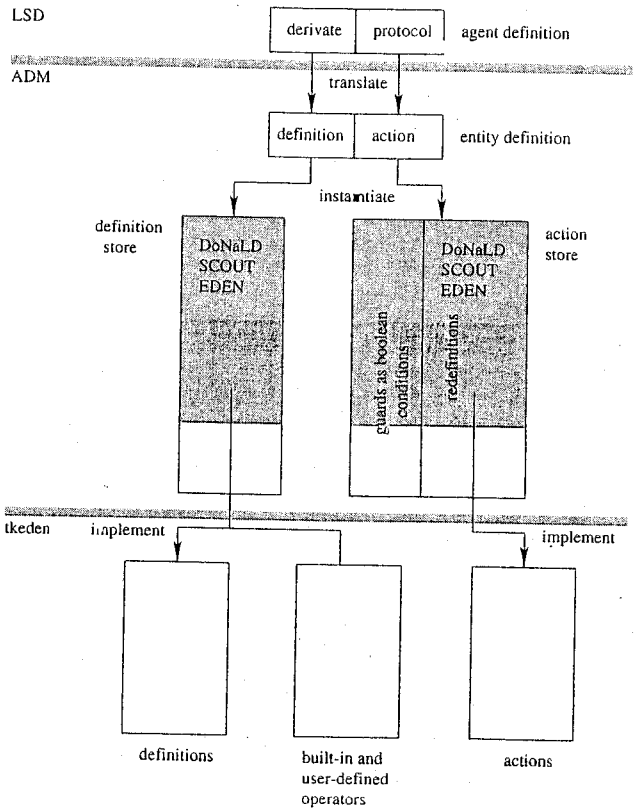
- conversion to internal representation
- associated actions to maintain visualisation

DoNaLD **point** variable p translated into e.g.

```
p = [CART, 2.3, 2.4]
p = cartadd ( q, r )
```

toge ner with an EDEN action to handle visualisation

```
proc display_p : p {
        <update imace of p on screen>
}
```



**Extension to Multi-Agent systems**

Have extended Empirical Modelling Principles to Concurrent System Visualisation cf. railway animation.

Quality of the interaction not the graphics!

*Research issues examined ...*

Concurrent Systems Modelling with the ADM
(ADM = Abstract Definitive Machine)

VR issues: experimentation, open vs. closed simulation

Concurrent Engineering

*Relevant issues for further research ...*

dealing with complex structures and operators

control over privileges of agents

context-dependent references

semantics through interaction: conceptual model

**Objectives addressed in this talk**

Incorporating more sophisticated geometry in EM ...

design of a definitive notation for geometric modelling

Developing more effective implementation strategies ...

Java Maintainer

Definitive Assembler Maintainer

**Progress of research**

Two aspects for which prototyping has been explored:

1. representation for complex structures
2. more flexible and efficient implementation strategies for interaction with objects

**1. Representation for Complex Structures ...**

Design of CADNO

3 perspectives on geometry

*Conceptual*

combinatorial structure associated with abstract geometric references

defined by complexes of abstract simplices ie points, lines, faces etc

*Geometric*

situated geometry, complexes with associated coordinate information

defined by associating coordinates with points to create frames

serve as skeletal information for construction

*Realisation*

Recipes to attach point sets to lists of frames

Can use a variety of methods

B-rep, F-rep, CSG, rectilinear

## The Definitive Notation CADNO

Combines definitive notations for abstract geometry & realisation

**Illustrative Example**

The CADNO table script

... defines complexes of various dimension

... defines associated frames by adding coordinate info

Two alternative realisations

... use an auxiliary definitive notation

3d-DoNaLD *or* front-end to VRML

Specify objects with reference to definitive script for extent

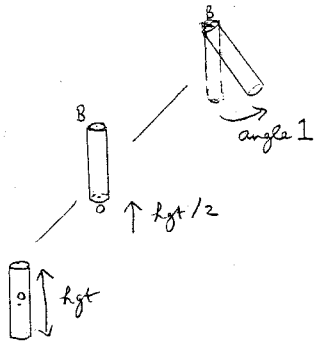
Parameters used in the realisation process may include references to the underlying frame

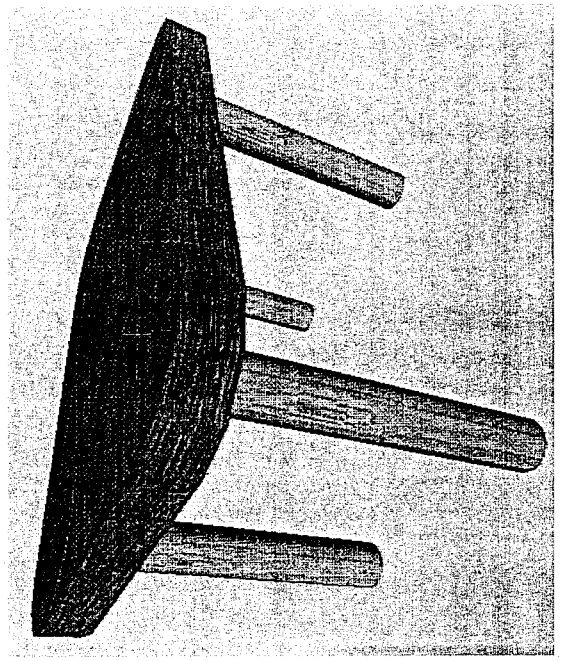
3d-Donald realisation via lines only

VRML via pointsets: extent specified via function representation

```
pict1 = object on [table1, angle1, dimensions1]
                with extent(donald3d) {

          line leg1, leg2, leg3, leg4, top1, top2, top3, top4

          leg1 = [A, E]
          leg2 = [B, F]
          leg3 = [C, G]
          leg4 = [D, H]

          top1 = [E, F]
          top2 = [F, G]
          top3 = [G, H]
          top4 = [H, E]
        }

 }
```

```
pict2 = object on [table1, angle1, dimensions1, radii1]
                with extent(VRML) {

        c1 = Transform { center B
                         rotation 0 0 1  angle1
                         children [
                           Transform {
                             translation 0 hgt/2 0
                             children [
                                Cylinder { height hgt }
                             ]
                           }
                         ]
                       }

                       .... }
```



```
table = 3-complex on [A, B, C, D, E, F, G, H]
                with {[A, E], [C, G], [D, H], [B, F], [E, G, H, F]}

base = 3-complex on [origin]                              # User's handles
angles = 1-complex on [angle1, angle2, angle3, angle4]

dimensions = 1-complex on [hgt, width, length]  # Designer's handles
radii = 1-complex on [legRad, cornerRad]

table1 = frame on [table, base]
                where   A = B - hgt * (sin(angle1), cos(angle1), 0},
                        B = origin + (0, hgt, 0},
                        C = D - hgt * (sin(angle2), cos(angle2), 0},
                        D = origin + (0, hgt, depth},
                        E = F - hgt * (-sin(angle3), cos(angle3), 0},
                        F = origin + (width, hgt, depth},
                        G = H - hgt * (-sin(angle4), cos(angle4), 0},
                        H = origin + (width, hgt, 0}
                        origin = (0, 0, 0}
radii1 = frame on radii
                where   legRad = 3.0,
                        cornerRad = 10.0
                subject to
                        0.5 <= legRad <= 20.0,
                        legRad <= cornerRad <= 25.0
angle1 = frame on angles
                where angle1 = 0, angle2 = 0, angle3 = 0, angle4 = 0
                subject to
                        0 <= angle1 <= 90,
                        0 <= angle2 <= 90,
                        0 <= angle3 <= 90,
                        0 <= angle4 <= 90
dimensions1 = frame on dimensions
                where hgt = 40, width = 50, depth = 30
                subject to hgt > 0, width > 0, depth > 0
```

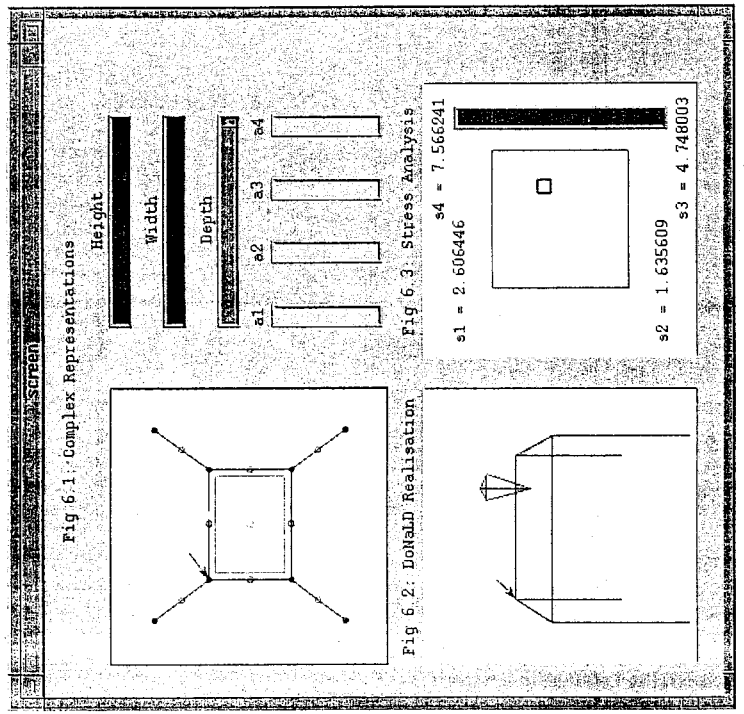



Fig 6.1: Complex Representations

Fig 6.3: Stress Analysis

Fig 6.2: DoNaLD Realisation

## JaM vs. EDEN implementation

+ User-defined data types
+ Facilities for object based data provided by Java
    - Inheritance
    - Interfaces and Abstract Classes
+ Compiled methods to implement algebras: speed-up
+ Large libraries of classes available from Java
+ More than one agent concurrently interacting with a script
+ Simultaneous update of several definitions

- Underlying Algebra and Actions cannot be modified on-the-fly
- Initial work to design and implement type & function classes
- No undefined - order of definitions more important

# The JaM API

An Application Programming Interface
for Empirical Modelling

**JaM** = **Ja**va **M**aintainer

A programmer can extend abstract classes
DefnType and DefnFunc
to implement new definitive notations (like DoNaLD & SCOUT).

DefnType supports

- Type Structures

DefnFunc supports

- Operators from Underlying Algebras for those types
- Actions associated with datatype manipulation
  e.g. visualisation of values of a particular type

JaM will maintain all dependencies between values.

Applications of JaM so far:

sophisticated arithmetic dependency maintainer

prototype CADNO implementation

## Using JaM

A simplistic step-by-step guide to using the JaM API.

1) Design a notation.

2) Implement subclasses of DefnType to provide the types of the notation. Devise an Internal Data Representation and include methods to ....

- Convert from String to Internal Representation
- Convert from Internal Representation to String
- Recognise whether a particular string is valid
- Take action every time value of type is updated

3) Implement subclasses of DefnFunc to provide the operators of the underlying algebra. Include methods to ....

- Type check passed arguments
- Calculate the new value and type
- Take action every time the function is called

NB actions are optional e.g. only DisplayWorld() in CADNO

4) Create a program to call JaM ....

```
Script s = new Script("password_file");
s.addType(new FloatType());
s.addFunc(new AddFunc());

int john = s.login("john", "password");
s.addToQ(john, "val1 = 23.7");
s.addToQ(john, "val2 = 24.9");
s.addToQ(john, "def1 = +(val1, val2)");
s.update(john);
```

## CADNO

**Computer Aided Design NOtation** -
a JaM based definitive notation

Abstract Geometry and Realisation
via two complementary definitive notations:

1) Notation for underlying combinatorial structures for geometry:

**d-complex**: list-of label × list-of list-of label
**frame**: list-of vector-of real × d-complex

2) Realisation of the geometry of an object:

**object**: list-of frame ×
(list-of frame → (vector-of real → boolean))

Realisation via a notation for geometry, a script within a script:

- Syntax based on VRML (version 2.0)
- All geometry represented in F-rep.
- Output is a VRML world description ...
  link to a browser dynamically

**Types**: Box, Cylinder, Sphere, Cone, Union, World, Intersection, Cut, Blend, Transformation, Vectors, ....
- all values given explicitly

**Functions**: box, cylinder, union, world, blend, rotate, mirror, transform, ....
- value returned implied by arguments

## Implementing CADNO using JaM

Realisation for objects achieved using function representation

Implementation strategy relies upon following abstract classes:

| *Abstract class* | *methods* |
|---|---|
| DefnType | conversion to / from internal rep<br>recogniser for strings<br>action (optional) |
| VRMLType<br>*extends* DefnType | createVRML() |
| FrepType<br>*extends* VRMLType | point-inclusion function f(x,y,z)<br>polygonalise() |

The polygonalise() method can be used to generate a createVRML() from the Frep function f.

When implementing an FrepType:

specify methods of DefnType
specify point-inclusion function f(x,y,z)
specify createVRML()
**either** as VRML primitive
**or** using polygonalise()

## Simple Examples

This slide demonstrates the difference between explicit and implicit specification of geometry in CADNO.

*Explicit*

`b1 = Box { }` - a standard box centered at the origin O

```
b2 = Box {size 3.2 4.3 2.1 }
```
- a box of dimension 3.2 x 4.3 x 2.1 centered on O

```
c1 = Cylinder {height 7.5 radius 3 }
```
- a cylinder of height 7.5, radius 3, centered on O

```
tr1 = Transform {1 2.5 7
                 children [ Box { } ] }
```
- a standard box centered at (1, 2.5, 7)

*Implicit*

```
v1 = {1 2.5 7}
tr2 = translate(b1, v1)
```

The value of `tr2` is the same as the value of `tr1`. However, if subsequent modification is made to `b1` or `v1`, `tr2` will be updated accordingly.

```
u1 = Union {[ Box {size 1 1 1 } Cylinder { } ] }
```
- a union of a box and a cylinder

`u2 = union(b2, c1)` - a definition of a union

`w = world(u2, tr2)` - a world containing `u2` and `tr2`

## CADNO Geometry

- Possible to associate attributes to objects:
  - Colour, shine, texture, ....
- F-rep representations rendered by a polygonalisation algorithm:
  - quality of render also a definition ...
    quality vs. render time trade off
  - VRML browsers often hardware accelerated
- Future plans: add a Hyper style, on-the-fly description of shape:
  - slow whilst being very open ended
  - shape can later be encapsulated in a FrepType class
  - exploit Java's ability to dynamically load classes

## Linking CADNO Abstract Geometry and Realisation

Object realisation provided by a CADNO geometry script

Abstract Geometry of CADNO incorporates the geometry scripts for realisation ...

The object description contains a recipe, using labels from the complex.

The correct geometry taken from a frame is substituted for these labels (like macro expansion).

Represents a special kind of higher-order definition.

### Merits of JaM implementation

portability / platform independence

interaction possible across the web

high-level support for concurrent engineering

enriched types and underlying algebras

2. ALTERNATIVE IMPLEMENTATION STRATEGIES...

## The Definitive Assembler Machine: DAM

Concept:
*carry definitive model of state to lowest levels of abstraction*

Construct an abstract machine in which the modeller can establish definitive relationships between machine words

Operators on the RHS of such definitions can be any piece of function code that accepts arguments from a particular set of registers and returns its result to a special register.

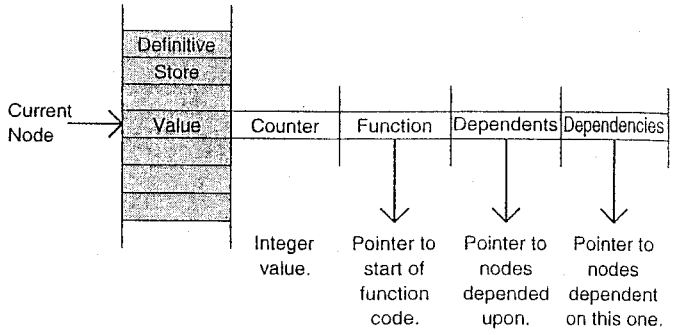
Implemented by R I Cartwright on ARM chip of Acorn RISC PC.

Uses Knuth's topological sorting algorithm for efficient update

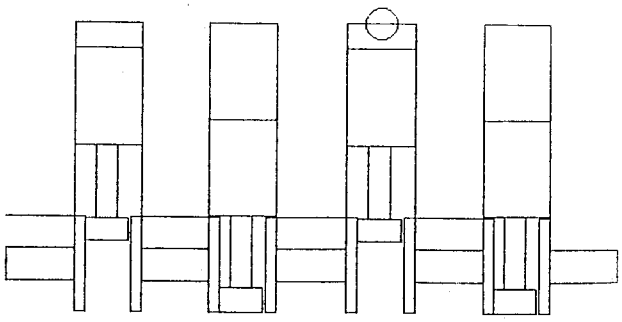
Structure associated with a word w of definitive store reflects this:

| | |
|---|---|
| *value* | current contents of word w |
| *counter* | used in Knuth algorithm |
| *function* | operator on RHS of definition of w |
| *dependents* | pointer to words used as arguments in defn of w |
| *dependencies* | pointer to words that depend on w |

To drive interface, can also use actions linked to pseudo-defns




STRUCTURE OF DEFINITIVE STORE



ENGINE PROFILE





## A DoNaLD to DAM Compiler

Implement a high-level definitive notation on DAM by converting high-level dependencies into low-level dependencies

Done by James Allderidge as final year project in 1996-7 for definitive notation for line drawing DoNaLD

| | |
|---|---|
| *parsing* | convert DoNaLD script into family of typed abstract defns |
| | transform into symbolic DAM script by factorising into words |
| *code generation* | realise symbolic DAM script in DAM store |

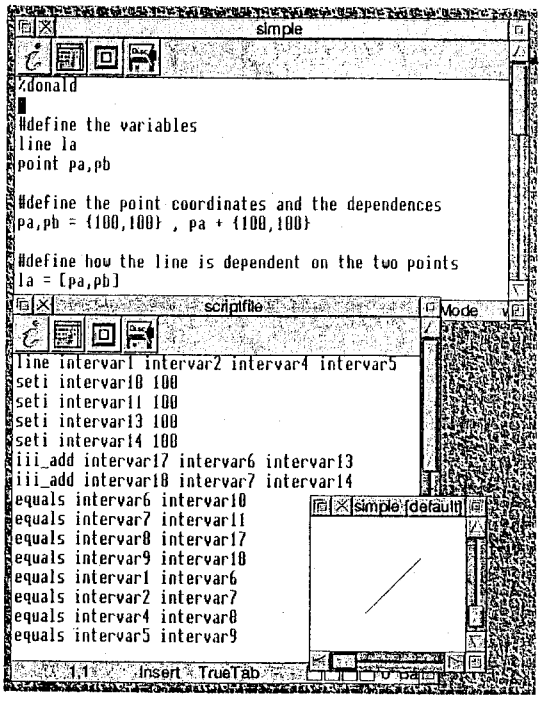
For example: line in DoNaLD is represented by 4 DAM words

Illustration of use

profile of 4 cylinder engine uses 70+ definitions

DAM store representation has 890 entries

smooth animation at 25 frames per second
(a tenfold speed-up over EDEN on SUN W/S)



**Prospects for the DAM abstract machine**

implement other definitive notations
(e.g. tkeden on Acorn RISC PC platform)

introduce greater dynamism in update of DoNaLD script

(optionally) link screen content directly to definitive store

exploit dependency-linked word & function referencing

investigate application for hidden functionality in code

implement DAM on other platforms e.g. IBM PC

## Conclusion

Need to combine qualities of several implementations:

current implementation via tkeden

implementation via JAM

implementations as in HyperJazz

DAM implementation

Major research project
much time and specialist expertise needed
basis for a Joint Project?

## References


EG'UK96

*Higher-Order Constructs for Interactive Graphics and Design in a Definitive Programming Context*

Warwick CS RR#319

*Abstract Geometry for Design in an Empirical Modelling Context*

Warwick CS RR#329

*Enabling Technologies for Empirical Modelling in Graphics*