6 - Practical modelling

6.1 The four strands of practical modelling that I undertook

In this section I will explain the practical modelling that I have undertaken for this project. This has been chosen carefully to try and emphasise the principles that I have talked about in preceding chapters and to provide a practical outlet to explain the way in which I believe educational software should be created with children in mind.

There were initially to be four strands to the modelling work that I was going to undertake. They were :-

i) to look at the models that the group had already created and pick one or more that had some educational value and write some investigations or activities that used them, These may or may not require the model to be updated in some way. This was to be very much in the same vein as the activities and investigations described in conjunction with Logiblocs.

ii) to write an original model in some educational field that would benefit from a computer based approach but would also benefit from being written using an Empirical Modelling approach. This was to be the major piece of work to be undertaken.

iii) to look at the possibilities for linking together different models across a network and see the potential for the use of these models together. This was to see whether the Empirical Modelling tools could address a problem diSessa believed that modern software had, “In terms of interaction, it is currently very difficult to connect different tools-as-applications. Suppose you want to connect a nice simulation, say of an ecological system, with a statistical analysis tool and a graphing tool. This is sometimes possible these days, but as often it is awkward, and too often it is just impossible” [22,p51]. The aim was to demonstrate whether this was possible using the Empirical Modelling approach, as a proof of concept, not to spend time refining the approach but to leave it open for future development.

iv) to look at massively parallel simulations. Mitchel Resnick’s book [44] on how colony level behaviours arise from creature level interactions was the stimulus for this strand,

along with the recently developed *dtkeden* development environment. Some of the models that have been developed by the Empirical Modelling group have been demonstrated over a number of workstations with multiple agents but this has been limited to about 7 agents in the Clayton Tunnel railway accident simulation [10]. I wanted to see if it was possible to develop simulations with thousands of simple interacting agents over a network, created using the virtual agency feature provided in *dtkeden*. Issues of speed, load on processors and communication between the server and the clients would be important and could lead to an unsuccessful implementation. However, due to time constraints, this strand of work was abandoned although I will talk briefly later about how massively parallel simulations may be designed in *dtkeden*.

6.2 Investigations for existing models

The Empirical Modelling group has an ever growing library of case studies developed either as academic exercises or for student projects. I decided that some of these must be suitable for use in a classroom in conjunction with activities or investigations which achieve some learning outcome desired by the National Curriculum. I wanted to find a simple model that had a fairly instructive interface and could be used mainly at the interface level without having to resort too much to using the input window for typing in redefinitions. The limited use of the input window would mean that children would be learning some basics about the underlying language and the power of entering new definitions whilst actively engaged in a task which was of pedagogical value. In this way we would be scaffolding the use of the development environment [39,p2].

After investigating and experimenting with a large number of the case studies I settled for the jugs model, as it had the characteristics that I wanted it to possess. A screenshot of this model is shown on the next page.

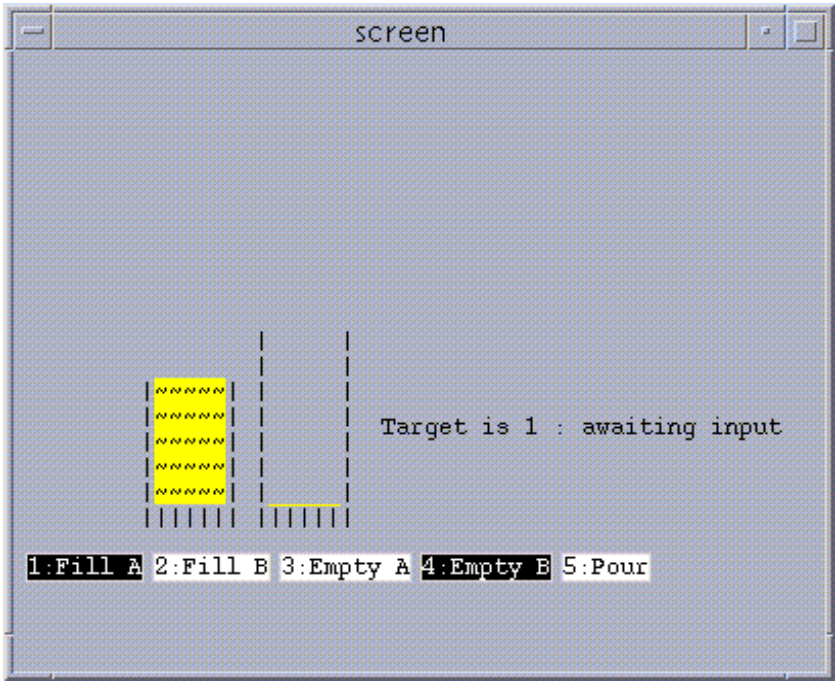

Fig 6-1 : A screenshot of the jugs model

The idea for the jugs model is to measure a specified quantity of liquid in one jug, given two jugs A and B of different capacities. They have no distinguishing marks apart from the fact that their capacities are known. The operations that can be performed are to fill either of the jugs, to empty either of the jugs, or to pour from one jug to the other. This last operation is unidirectional since at any one time only one jug can be neither full nor empty. The major general mathematical question that this model raises is :-

Given two jugs of different capacities, A and B, and a target amount of water T such that 0<=T<=max(A,B) can it be determined whether achieving T using only the five operations defined above is possible before pouring any liquid, or hypothesising about sequences of operations?

The answer to the question is yes, but this is not the place to explain how to do so. The two investigations that I have devised are aimed at two different age groups. They can be seen in Appendix C (pages 130-138). The first is for pupils around the concrete-formal boundary (i.e. upper primary-lower secondary) and the second is aimed at mathematics undergraduates in their introduction to group theory courses. Both investigations attempt

to guide pupils towards the answer to the above question, but obviously the investigation aimed at the older pupils gives far less help. The older group should be able to derive the formula fairly easily. The main aim for their activity is to illustrate jugs as a practical example of some topics from group theory, namely cyclic groups, order of a cyclic group and generators of a group. The investigation aimed at the younger children starts at a very simple level before gradually increasing the complexity trying to engage them with the task of helping the professor, guiding them towards the formula but ultimately leaving the conceptual understanding as the final leap they have to make themselves. This leap is to realising that the relationship between the capacities of the jugs is most important and they need to be familiar with the notion of a highest common factor to identify it.

The third investigation is designed as a follow on to the younger age investigation and attempts to generate a solution to the following different question that can be answered with the jugs model.

Given two jugs of differing capacities, A and B, can we give an upper limit on the amount of water we need to use in order to achieve every target T, 1<=T<=max(A,B).

The answer again is yes but this is not the place to explain how. The formula again depends on the relationship between the capacities of the jugs but this time it is the concept of the lowest common multiple that is important. One of the extension questions to this investigation is phrased to attempt to elicit from the children the relationship **hcf(a,b)*lcm(a,b) = a*b**. This particular fact I was never made aware of until studying at university. We are not aiming at a proof, just a conjecture of this relationship, which would generate the same kind of learning before taught at school that was achieved with the Logiblocs investigations, described in section 4.6.

6.3 The coordinate geometry model

6.3.1 Design criteria for the model

The major strand of practical work undertaken was an attempt to construct a model using the Empirical Modelling tools that satisfied the following criteria :-

i) It would be of use in the classroom. It had to satisfy aims and objectives from the National Curriculum or have investigations and activities designed that could be carried out using it. It also would ideally be flexible to use with many curriculum levels.

ii) It had to illustrate the use of the definitive principles underlying the Empirical Modelling tools.

iii) It had to be easy to use with an aesthetically pleasing interface and be suitable for children.

iv) It had to show in a practical situation some of the theoretical ideas introduced in this project. I wanted the model to be a constructionist work environment where children could play and discover things for themselves or discover knowledge embedded in a microworld.

A discussion with teachers and an examination of the National Curriculum to discover which topics would benefit most from the use of a dynamic tool like a computer and of an approach using Empirical Modelling methods. I eventually settled on coordinate geometry as a suitable domain to situate the model within due to its prominent position in mathematics across a wide range of age groups and abilities. This domain is ideally suited to Empirical Modelling tools since dependencies will naturally arise in geometry (e.g. this shape is that shape translated by 6 units horizontally). If any entity of the original shape is then altered the underlying definition manager should handle the updates without requiring the modeller to explicitly intervene. The third criterion is a more difficult one to judge. There are models that are well designed and pleasant to use and there are those that are not. I will leave the question as to whether I have been successful with this aim to the reader. To provide a coordinate geometry based model that would let children explore for

themselves would have to be organised so that they could create drawings and manipulate them as they desired with tools that they needed to achieve their aims. This is a very loose overview of the system criteria that I will explain in more detail. One major design aim was to provide a set of tools for the children to use which could be easily expanded on either by myself or another person familiar enough with the development environment and the language to do so. This was so that the model embodied the idea of diSessa's open toolsets [22].

6.3.2 Existing dynamic geometry software

After deciding on the model domain I was made aware of a couple of some existing dynamic geometry packages. These are commercial packages developed over a number of years by a team of programmers. I have included brief descriptions of these packages with some example screenshots to illustrate the power of these dynamic geometry environments.

The first is called Geometer's sketchpad made by Key Curriculum Press [25].

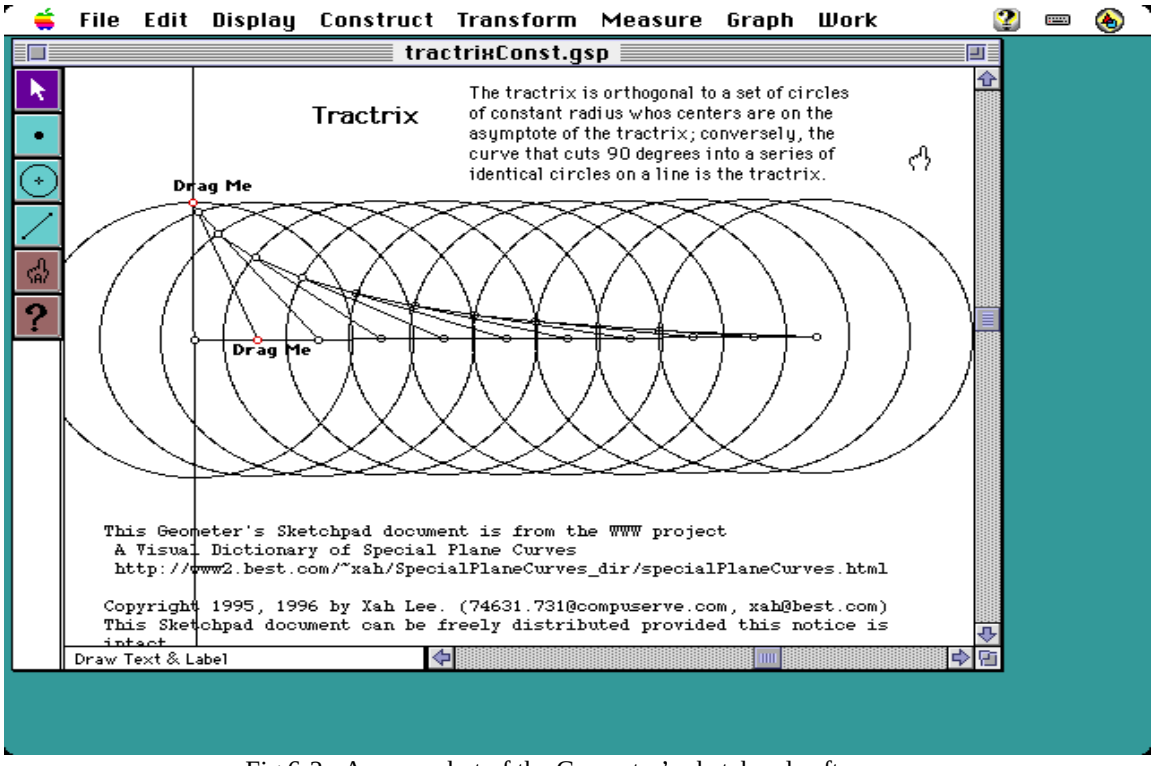

Fig 6-2 : A screenshot of the Geometer's sketchpad software

The following description is taken from their website [25]. It is an interactive plane geometry environment. For example you can construct a circle inscribed in a triangle and then drag a vertex of the triangle and the inscribed circle will follow it in real time. The authors claim that this property easily leads users to anticipate theorems in classic geometry.

The other program that has been in direct competition with Geometer's sketchpad is Cabri Geometry [18]. It is marketed by Texas instruments [46]. It is thought of as being more powerful than Geometer's sketchpad. It allows users to create the basic geometrical objects such as circles, lines and points as well as more complex shapes such as arc, conics and parabolas. There is a suite of construction tools built-in, which allow the user to build parallel lines, perpendicular and angle bisectors and intersection points. There are measuring tools that allow lengths of lines or arcs to be measured and the angle between two lines. There are many other facilities such as equations of lines and transformation tools such as rotation, translation and reflection. All of these are achieved through a point and click interface.

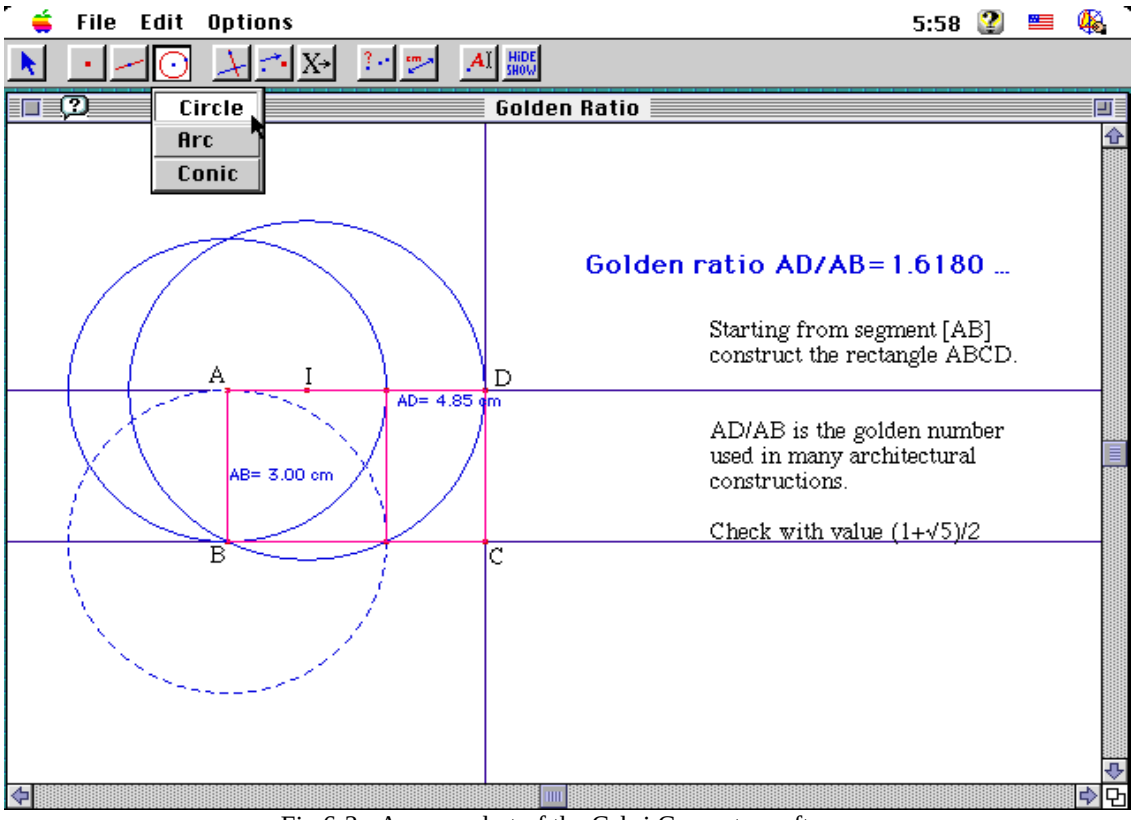

Fig 6-3 : A screenshot of the Cabri Geometry software

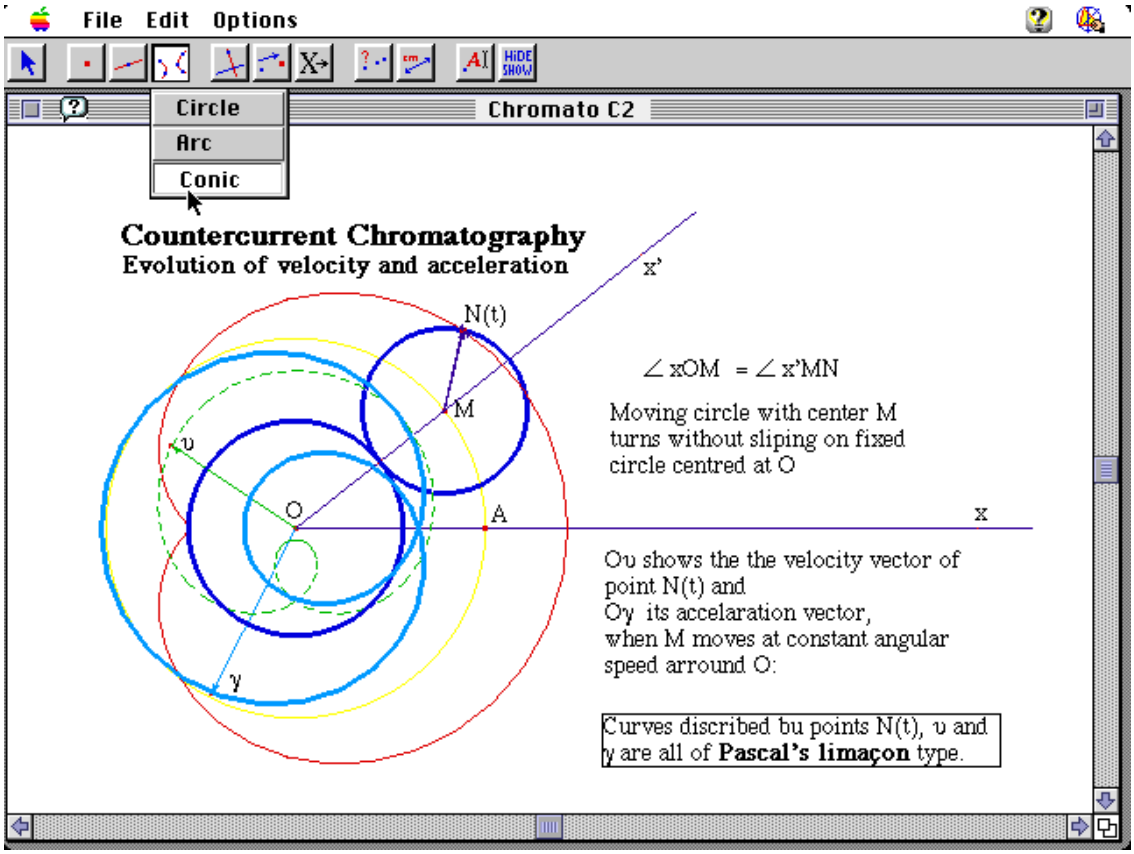

Fig 6-4 : A more advanced example in Cabri Geometry

These are the two established leaders in the field of dynamic geometry software. A new package that has become available in the second half of 1999 is claimed by its authors to overcome some of the shortcomings of the above two packages. This is not the place to debate the comparative merits of these packages. The reader is referred to the products themselves to make up their own minds.

The new package is called Cinderella [19]. It is written in pure Java so applets constructed in the environment can be placed on Internet web pages and explored. One advantage it claims over Cabri Geometry is that it uses the principle of continuity to establish which point to select when the mouse is over a collection of points, without the need to prompt the user to choose. This package is attempting to break into a market that already has two established packages with large and dedicated user bases. If it is to succeed it has to convince these users that their product offers more than existing packages. Some screenshots of Cinderella are shown on the next page.

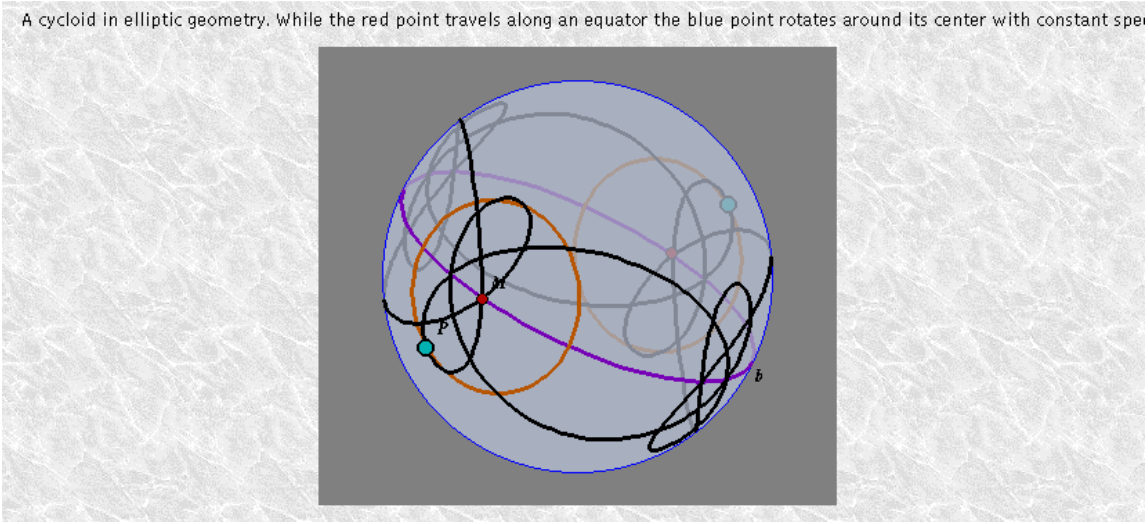

Fig 6-5 : A mathematical example produced in Cinderella

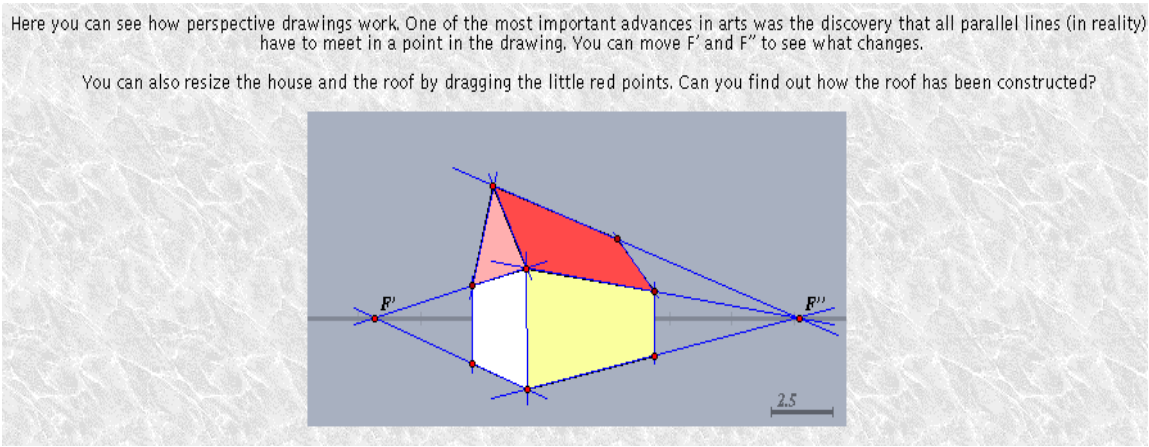

Fig 6-6 : An artistic example produced in Cinderella

6.3.3 Reasons for developing a dynamic geometry model using the Empirical Modelling approach

If these packages are so highly thought of why build a model using Empirical Modelling principles to perform similar kinds of operations?

i) The flexibility that an environment such as this can be provided with when built using Empirical Modelling tools means that new tools can be added and hence new domains of knowledge can be investigated and explored. Example domains could be graphing tools, calculus or curve sketching. Also a layered implementation (e.g. the script's functionality is refined by successive scripts being added which describe how the model works) would

mean that the grid, the interface, or even any part of the model could be used as the basis of a completely different model. This could lead to use as a cellular automata simulation, sieves of prime numbers or even as a game such as noughts and crosses.

ii) The model can be refined in light of experience in the classroom with children using it and from the comments of them and their teachers. Any annoying features in commercial software have to be endured, with Empirical Modelling the teacher can change the way a model works or an 'expert' modeller could work with the school on site to develop more satisfactory alternatives.

iii) The model can be used to test the feasibility of the Empirical Modelling approach to tool construction in line with diSessa's notion of open toolsets [22].

iv) If it is possible to imitate some reasonable percentage of the features in these commercial packages with the prototype tools that the Empirical Modelling group currently possesses it shows the power of definitive notations in developing applications with these types of dependencies.

6.3.4 Script organisation and design

Having made a final decision on the model domain some thought was given to the ideal structure of the overall script and the following seemed appropriate.

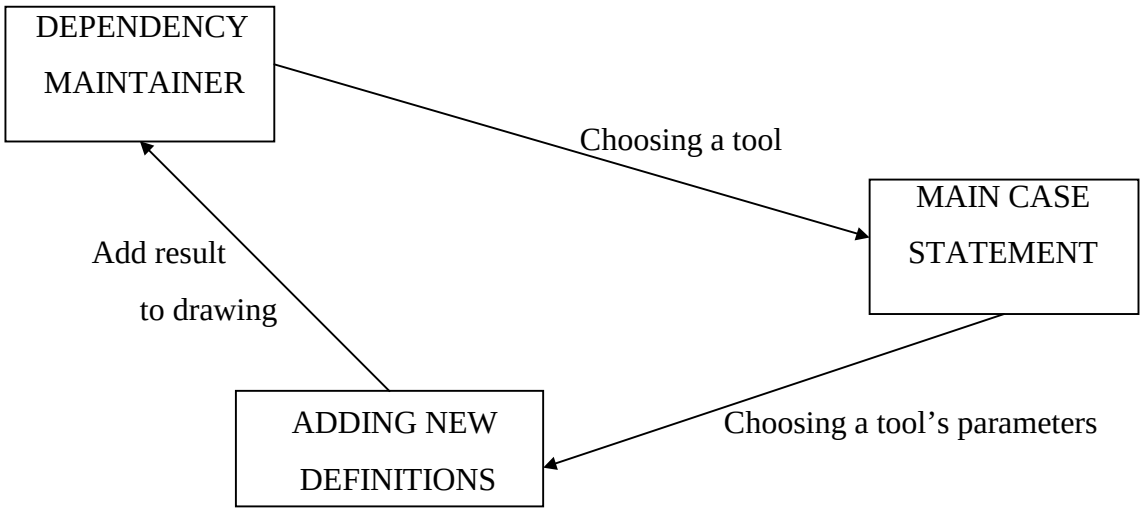

Fig 6-7 : Interlinking script organisation in the coordinate geometry model

EDEN is a mix of definitive principles and procedural programming giving the modeller the power of two contrasting paradigms to choose the appropriate type at the appropriate time. The coordinate geometry model shows this mix off to good effect. I will describe how the triangular linking of the three major components works.

i) The dependency maintainer - The definition manager underlying the development environment supports all the activity that takes place on the coordinate grid itself. The positions of the lines, axes and labels are all dependencies defined in terms of the bottom left and top right coordinates of the grid and so whenever these are changed the grid is automatically updated. All the entities (points, lines, circles, rectangles, labels) on the grid are defined in terms of their key points (e.g. centre point and circumference point for circle) and whenever these change the circle will change and update on the grid. As a modeller I set up these dependencies but I never need concern myself with making sure they are up to date, the representation on the grid will always be correct. Moving any point on the grid using the mouse will cause all the relevant entities to be updated, irrespective of how many there are. This is also handled automatically by the system, as is the redrawing of the display. This has saved an enormous amount of time in the development of the model. I have not needed to consider how best to store the entities so that they change when parts of them change, or to make sure that the grid is not overwritten by entities moving around the grid.

ii) The main case statement - This is held within a procedure triggered by a click or motion of the mouse on the coordinate grid. It contains the contexts within which the mouse click can be taken and handles the specifying of parameters to those contexts. To add a new tool to the current ones an entry into this case statement needs to be made as well as any associated routines that it may require.

iii) Adding new definitions - The parameters specified by the previous statement are turned into a definitive statement for that entity which is then added to the system. This triggers the display to be updated and any other relevant changes to be made. The new definition is then handled automatically by the system and changes whenever any entity defined in its formula changes.

6.3.5 Tools for the coordinate geometry model

The most important functionality decisions that need to be made regard the tools that the system should provide for the user to explore geometrical concepts. They can be broadly grouped into two categories, those that add entities to the diagram and those that perform transformations on the diagram. In the time available I have implemented as many tools as possible, but given the enormous scope in this domain there are many more tools which have not been written in to the system. I decided to put emphasis on the transformation tools rather than on construction tools. The following is a list of the tools provided by the system grouped into these two categories :-

I) Adding entities to the grid -

i) points - a single point at a location (x,y) in the Cartesian coordinate plane.

ii) lines - a line between two points in the plane.

iii) circles - a circle with one point as its centre and the other as a point on the circumference.

iv) rectangles - a rectangle signified by two points representing opposite corners of the rectangle.

v) labels - a string specified by one point which is at the centre of the text.

II) Transformations or operations on shapes -

i) Select - Allows the user to pick up and move a point around the screen and drop it somewhere else.

ii) Translation - translates the shape being edited by a specified Cartesian distance (x,y).

iii) Rotation - rotates the shape being edited around a point by a specified number of degrees.

iv) Enlargement - enlarges the shape being edited by a given scale factor around a specified point.

v) Reflection - reflects the shape being edited through a specified mirror line.

vi) Measure - measures a line on the current shape and displays its length on the line.

viii) Angle - measures an angle between two lines which is displayed around the angle centre.

More information about the tools can be found in the user manual in Appendix A on page 102.

There are a number of tools which I believe are fundamental to this model that have not been implemented as yet. The reason for this is due to time constraints, but the general design of the model is such to allow another modeller to add new tools to the system fairly easily. The sequence of events necessary to add a new tool is as follows :-

i) Create a new window/icon for the tool on the button bar and its help window entry.

ii) Add a trigger to change the tool selected to the name of that tool when its window is clicked on.

iii) Add its entry to the case statement, together with any parameters that it needs (e.g. a user clicking on two points to specify a mirror line)

iv) Add to the script the outcome (i.e. new definitions or redefinitions) of using the tool.

This is described in more detail in the developers manual in Appendix B on page 119.

The following is a short (honest!) list of the tools that I believe are required before the model is used in the classroom :-

i) Construction tools - midpoint of a line, intersection point, parallel lines, perpendicular lines and bisectors, angle bisectors, fixing points to lines and circles. Shapes such as ellipses, parabolas, arcs and regular polygons.

ii) File handling tools - saving and loading of drawings and parts of drawings as DoNaLD script. One useful extension to the model would be to have output as a pure DoNaLD drawing, without the Cartesian representation of the points, so that these scripts could be incorporated into users models with no modification. The environment could then be used as a DoNaLD resource editor.

iii) Operation tools - area of a shape, perimeter of a shape (these both appear on the interface but due to difficulties in deciding how to calculate the areas and perimeters of self-intersecting shapes were left for the present time), tessellation tool, equations tool.

All of these functions could be added to the model fairly easily at an abstract level, although implementation of individual functions may prove to be more difficult due to

their internal complexity. This was certainly the case with the area and perimeter tools that are very easy to calculate in the simple case. Deciding on a conceptually correct manner of handling arbitrary self-intersecting shapes made me decide to postpone their implementation. These functions are all based in geometry. The power of the Empirical Modelling environment means that we can use the model to investigate different phenomena with the addition of suitable tools. The following is one example. There are many others that can be conceived of, simply think of a domain in which grids are used mathematically. The model linking scenario (described later in this chapter) describes a process by which a graph of speed against time is produced. It would be nice then to have graphing tools or statistical operations to provide children with the ability to calculate area under the graph (which could lead to some nice activities concerning the approximation of the integral under a curve, Simpson's rule and the trapezium rule), to find means, modes and medians and to calculate derivatives at points in the graph, to for example calculate the acceleration at a point in time.

This is equivalent to one level of abstraction away from the preconceived use of the model. There is a possibility that the model could be used for a purpose that is another level of abstraction away, using the grid simply as a visualisation of the model itself. For example having a situation with a 3x3 grid could then lead the modeller to believe that they were playing a game of OXO, as in that simulation. Other board games could also be visualised in this manner as could phenomena such as cellular automata.

6.3.6 The coordinate geometry model in a distributed system

There are certainly possibilities for the use of this model in a distributed environment. I would envisage a situation where a teacher could operate at the server and the class being clients operating the coordinate geometry model. The teacher could send the children a predrawn diagram with an investigation embedded within it and the class could investigate it at their workstations. The teacher would be able to monitor all of the class at the server, and be able to have more than one child's work on the screen, each in a separate window on their machine, using a windowing environment that looks familiar.

They can bring up a window with a child's work in and also close that window to make more space for looking at other children's work. The teacher could then send advice to those pupils who were struggling or the pupils could request information from the teacher if they were unsure. This also leads to the possibility of autonomous monitoring with the server monitoring the child's progress and deciding, according to some heuristic, whether they needed assistance and the type they required. These ideas could form the basis of some interesting extension work on the model, investigating the best methods of communication and how they compare, in terms of productivity and fairness, to the conventional classroom situation where the teacher circulates amongst the students giving help where needed.

6.3.7 Screenshots of the coordinate geometry model

To illustrate the ideas discussed in this section the following pages have a number of screenshots of the coordinate geometry model.

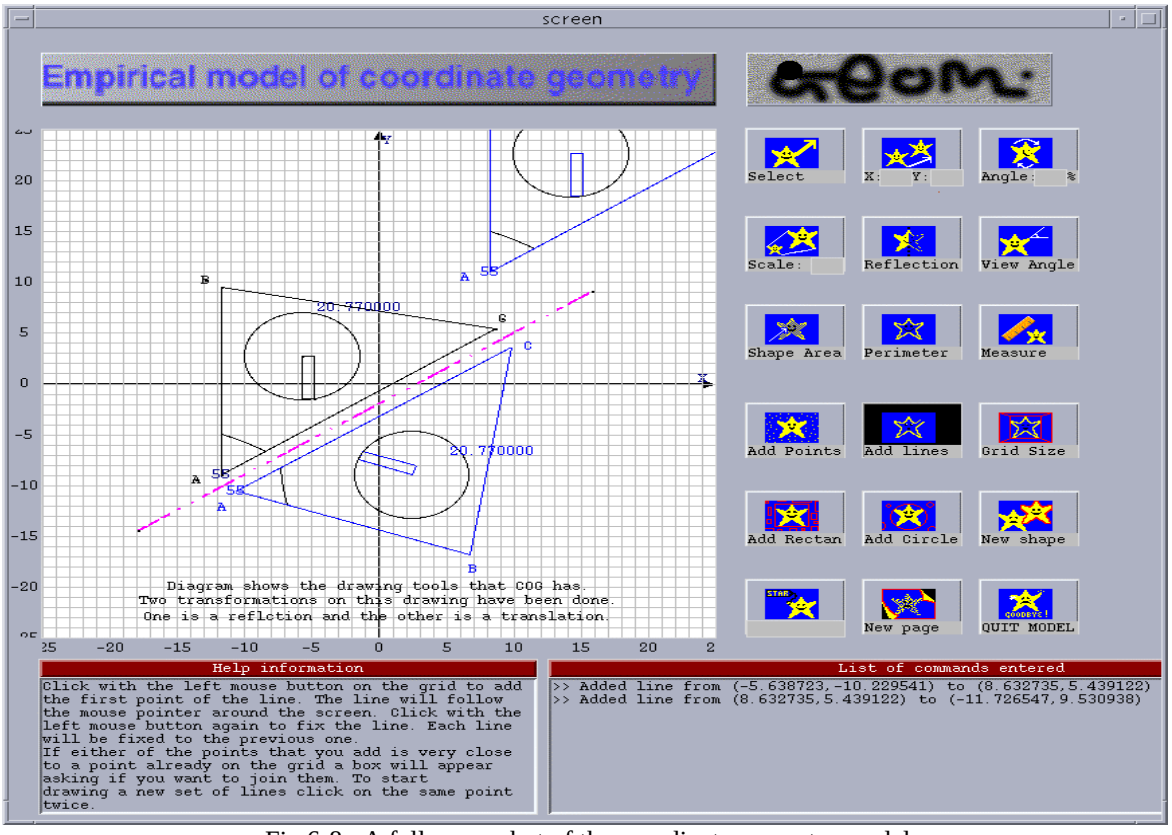

Fig 6-8 : A full screenshot of the coordinate geometry model

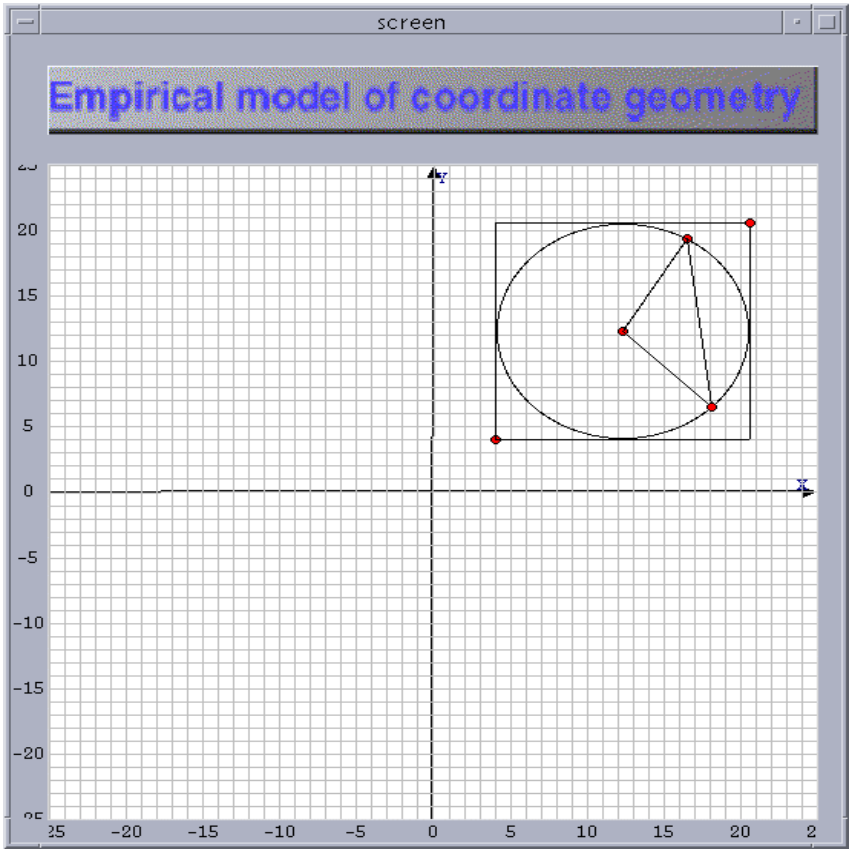

Fig 6-9 : A screenshot of some of the drawing tools in the coordinate geometry model

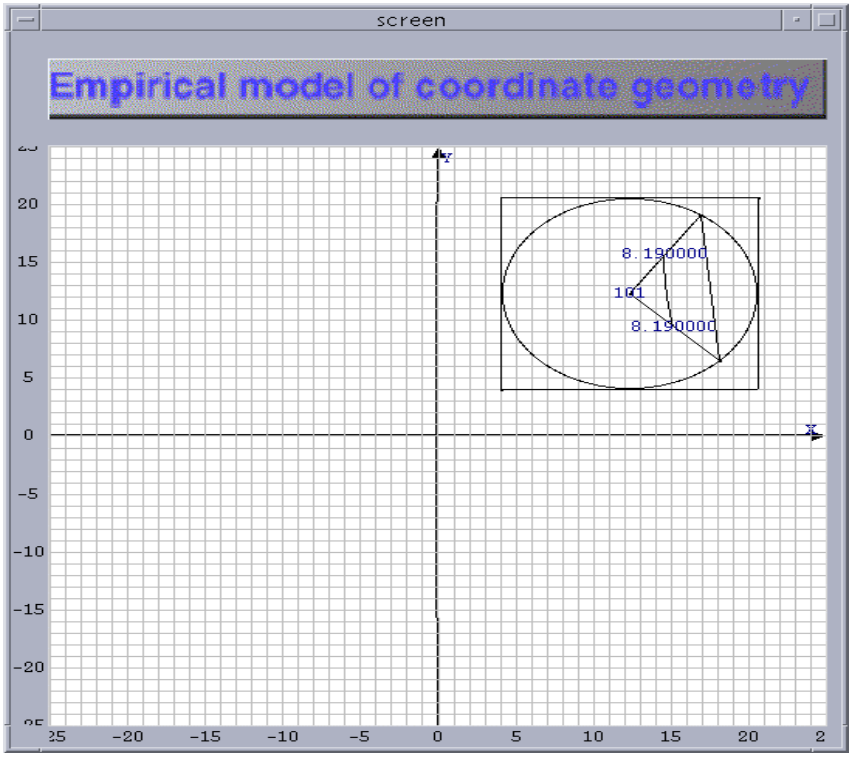

Fig 6-10 : A screenshot of the measuring tools in the coordinate geometry model

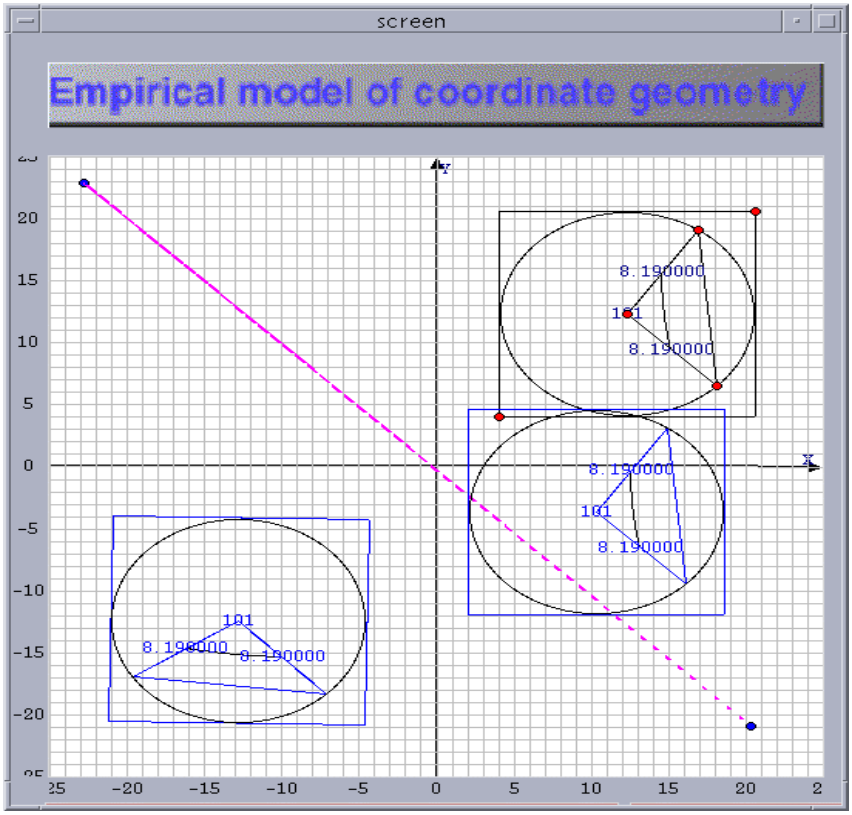

Fig 6-11 : A screenshot of some of the transformation tools in the coordinate geometry model

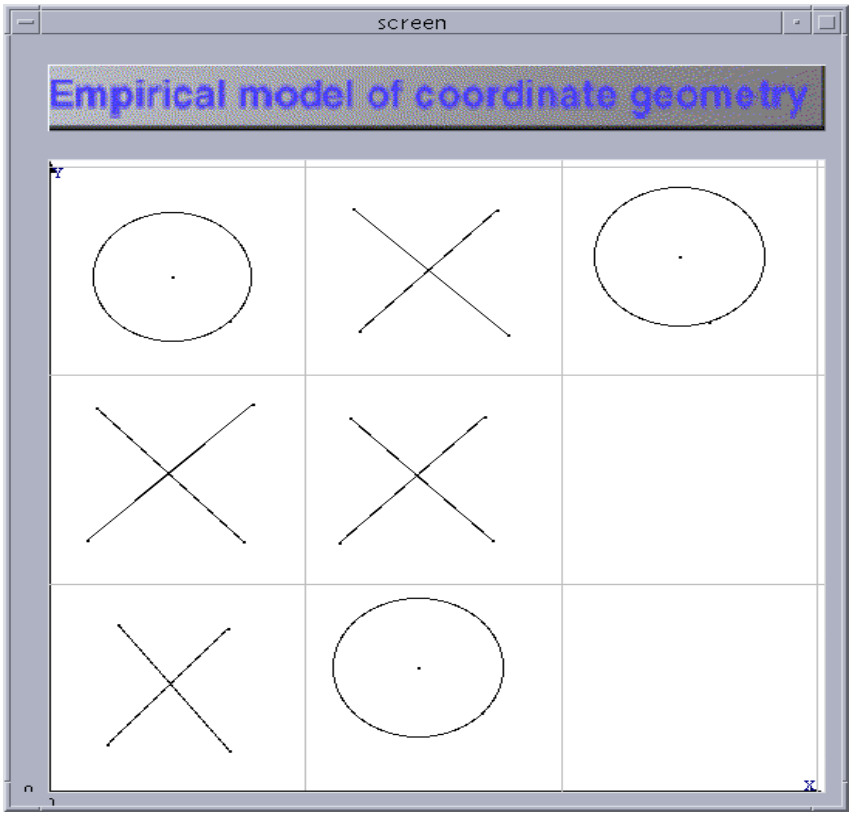

Fig 6-12 : A game of noughts and crosses in the coordinate geometry model

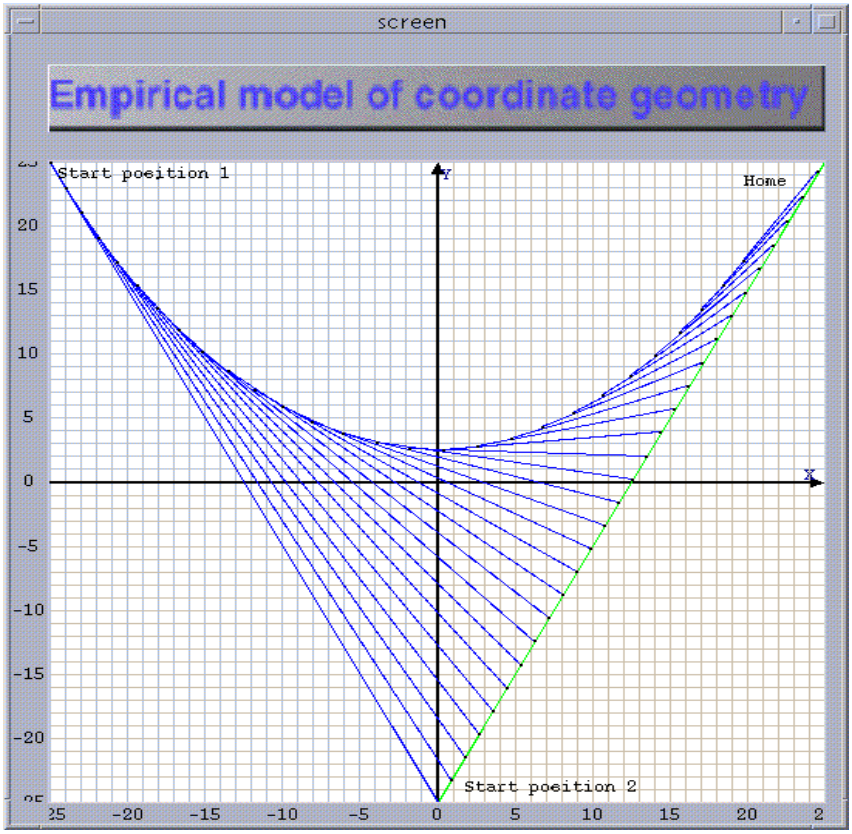

Fig 6-13 : The curves of pursuit investigation in the coordinate geometry model

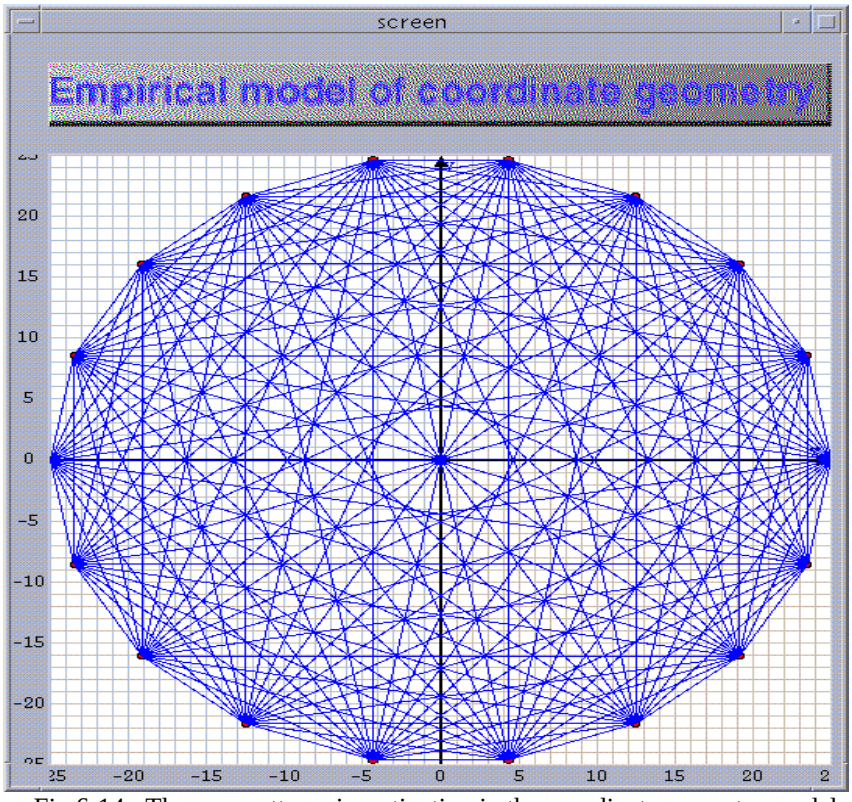

Fig 6-14 : The rose patterns investigation in the coordinate geometry model

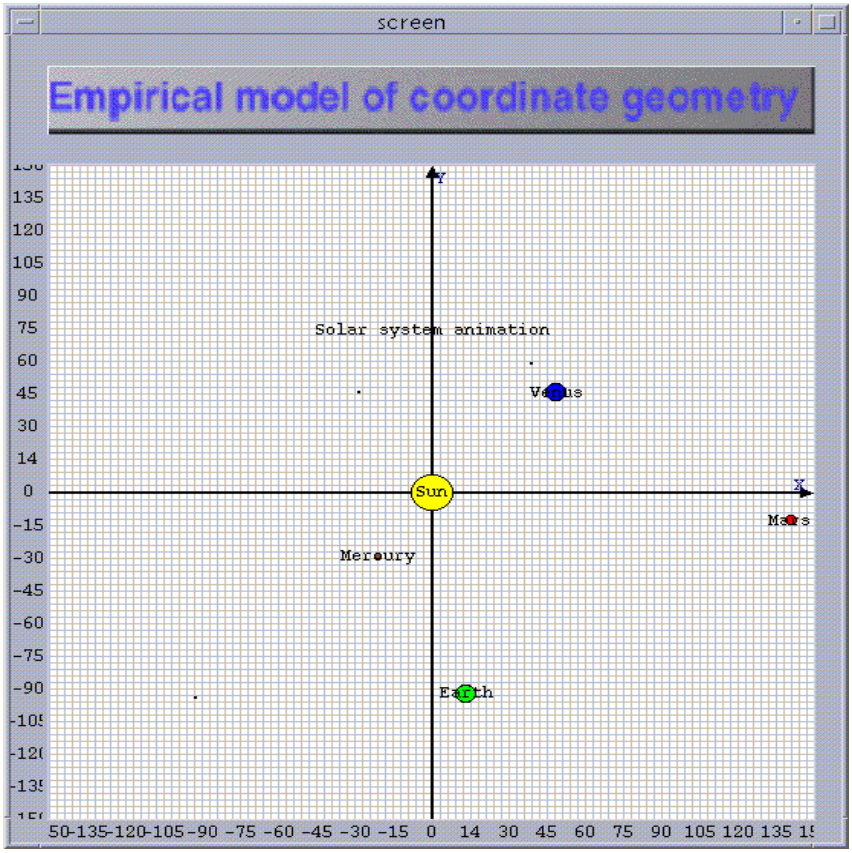

Fig 6-15 : The solar system animation in the coordinate geometry model

6.4 Model Linking

The third strand of practical work was to investigate the possibilities for the sharing of information between models in a meaningful way, linking them together using the *dtkeden* environment and using data from one model as input to another. The two models that I chose to try to link were very closely related to diSessa's idea of using a simulation in conjunction with a graphing tool. This has been achieved in the Agentsheets environment discussed in section 4.4. They linked together a program called SimCalc and an Agentsheet which simulated the effects of a contagious disease on a population. The sheet has a doctor agent who can move around and cure sick people when he comes in contact with them. Each time a person with a disease moves onto a person without the disease the healthy person becomes infected as well. A screenshot is shown on the next page.

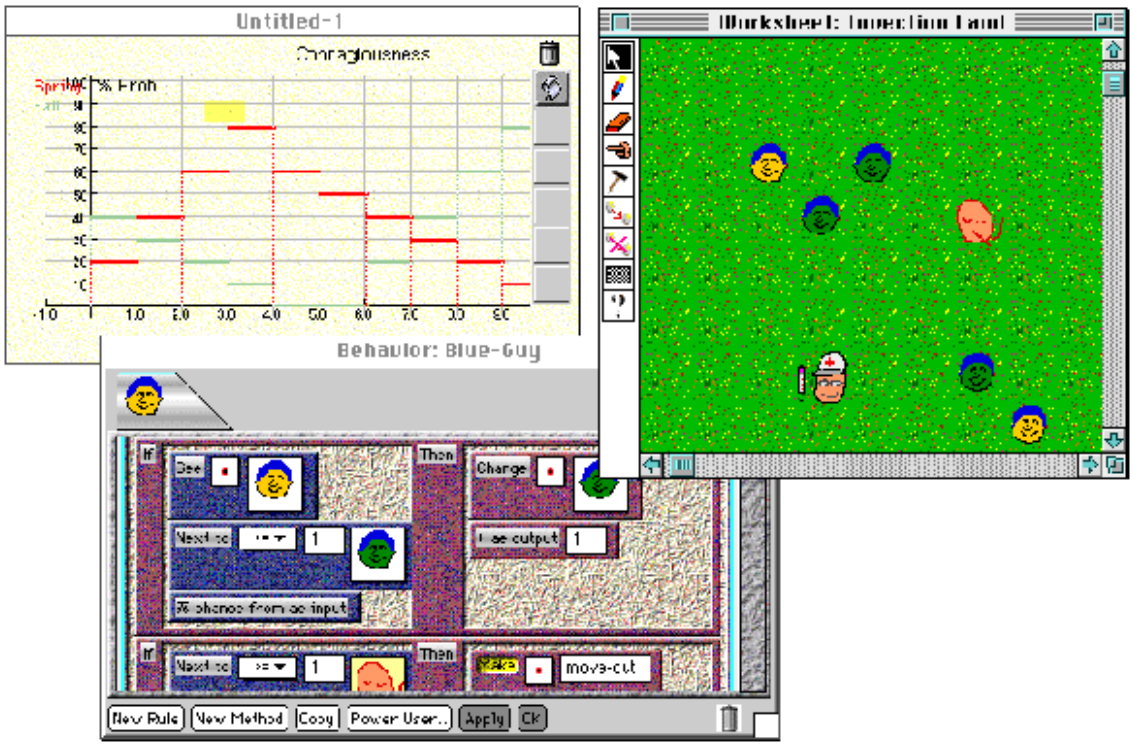

Fig 6-16 : An Agentsheet linked with the SimCalc program

This also gives me the chance to show one of the many ways in which the coordinate geometry model could be used in a new domains. Since the coordinate geometry model utilises a grid structure and a graph is usually represented on squared paper the screen layout of the model need not be changed, except the grid may be resized and repositioned to show the graph on a better scale. The vehicle cruise control simulator is one of the most well used models that the group possesses and I felt it was the best model to use as the simulation part of this exercise. The aim was to start the vehicle and the clock on the simulation and at every clock cycle (on the vehicle simulation, not the computer!) send the time and the current speed of the vehicle to the coordinate geometry model. It would then output the speed against time for the vehicle as a bar graph.

This would require two extra definitive scripts to be added, one to each model, to handle the communications and also to generate the graph in the coordinate geometry model. The set-up of the architecture is shown on the next page.

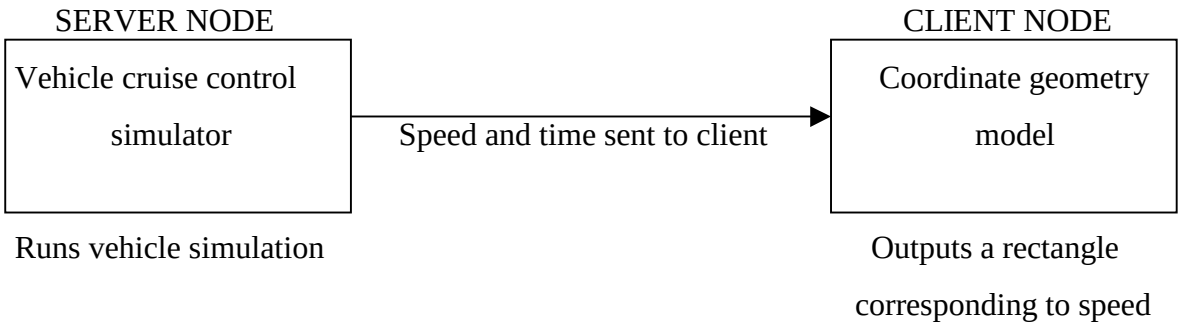

Fig 6-17 : Architecture for model linking in *dtkeden*

The code required to achieve this linking is shown in Appendix D but I wanted to reproduce it in the main text to show exactly the difficulty that this data sharing presents with the Empirical Modelling tools.

```
/* File used for model linking between COG and VCCS at the client (running COG)*/

%eden

proc dographit : c { /* triggered by a change in the clock */
        currentcolor = "red"; /* set colour of graph */
        pointx1 = c-1; /* horizontal start position of bar */
        pointy1 = 0; /* vertical start position of bar */
        pointx2 = c; /* horizontal end position of  bar */
        pointy2 = sp; /* vertical end position of bar (height of speed) */
        addarectangletoashape(currentshape,currentshapeno,pointx1,pointy1,pointx
2,pointy2,FALSE,FALSE); /* add the rectangle representing this quantity */
        eager(); /* forces update of definitions */
}
```

/* File used for model linking between COG and VCCS at the server (running VCCS)*/

%eden

proc Senddatatoclient : iClock { /* each time iClock changes trigger this procedure */

if (iClock % update2 == 0) { /* only send every 100 clock cycles, equal to clock on VCCS model */

sendClient("chris","sp = _curSpeed;"); /* send speed definition */

sendClient("chris","c = iClock/100;"); /* send clock definition */

sendClient("chris","eager();"); /* send update immediately definition */

eager();

}

}

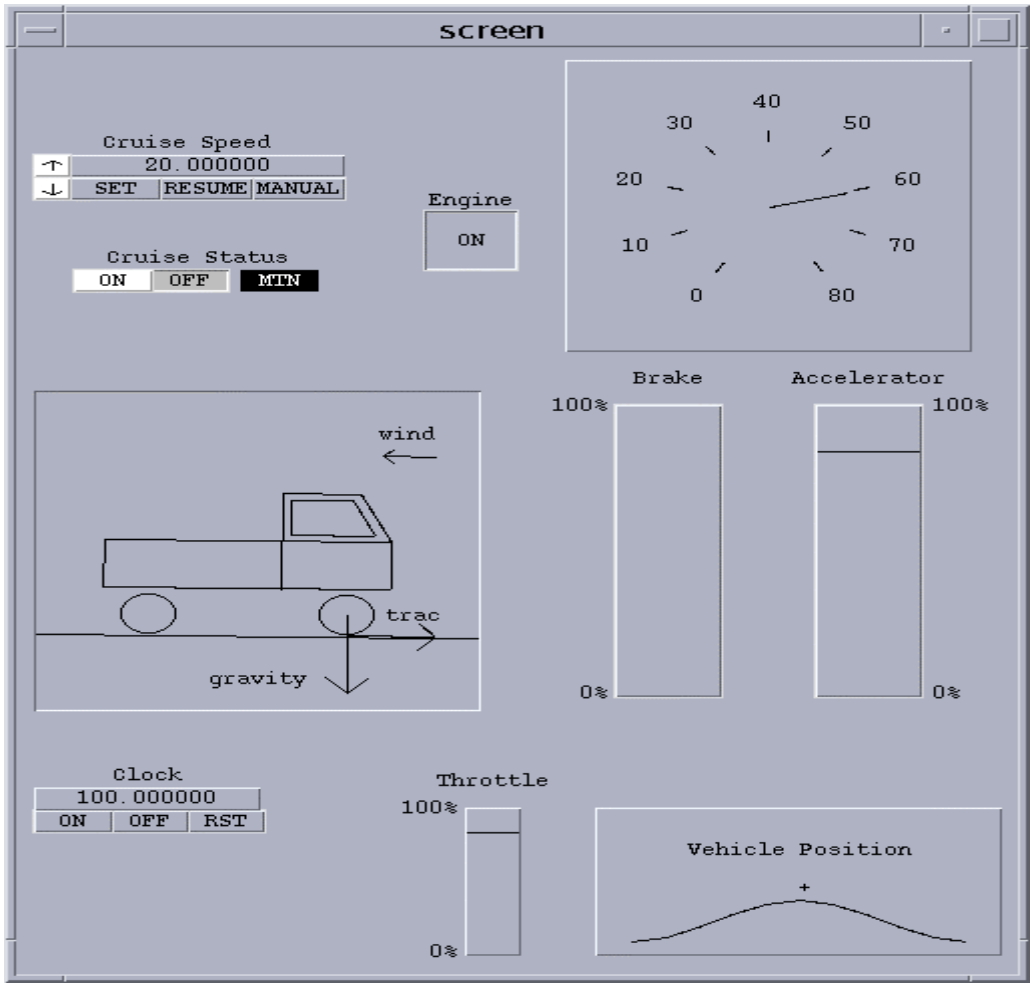

Fig 6-18 : A screenshot of the vehicle cruise control simulator during the model linking

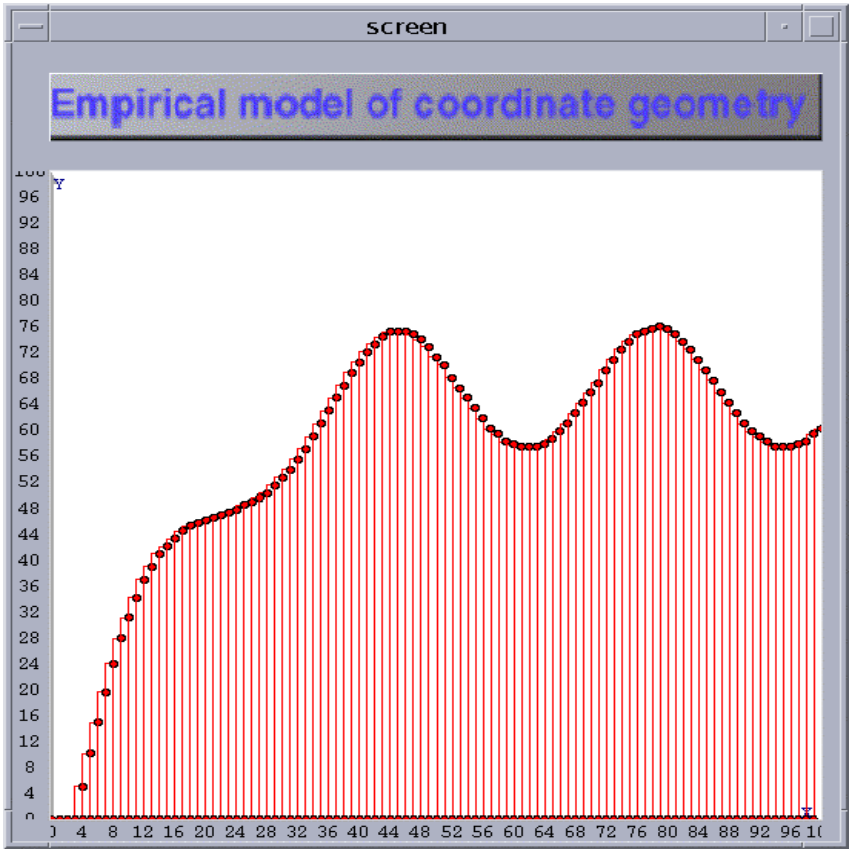

Fig 6-19 : A screenshot of the coordinate geometry model during the model linking

There are still problems with the implementation as it stands, most obviously the need to stop the clock on the cruise controller before the coordinate geometry model updates its graph (due to the number of redefinitions required and the frequency of new definitions that are sent by the server). This exercise has shown it is possible and can be the subject of future work.

6.5 Massively parallel simulations using *dtkeden*

The final strand of practical work, trying to generate simulations that exhibited decentralised phenomena in an attempt to give an environment where children could explore with the notion of distributed constructionism was shelved due to time limitations. However I did have ideas about how these simulations could be achieved and noting them here could provide the stimulus for some future work with massively parallel simulations using Empirical Modelling tools.

The architecture for such a simulation would need to use as many workstations as possible to reduce the computational load at any node. At present *dtkeden* only allows us one server connected to many clients. This arrangement is shown below.

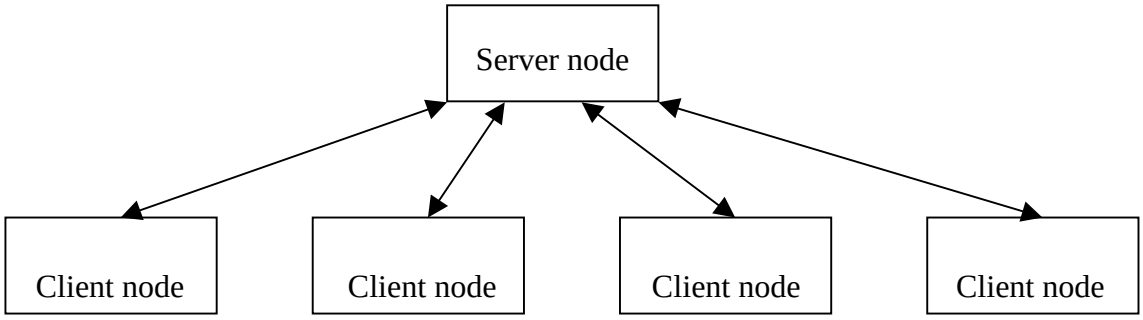

Fig 6-20 : The client-server architecture for massively parallel simulations in *dtkeden*

All of Resnick's simulations [44,pp 49-117] have similarities in that large scale patterns arise from local interactions, either with other members of the community (e.g. traffic jams) or with the environment (e.g. ants foraging for food). Therefore in the arrangement above we have to be able to communicate both between two agents and between one agent and the environment. There are two ways of doing this that seem plausible. One is to let the server contain the environment information for the whole world, be it levels of pheromone or any other quantity. Then when an agent needs to enquire about the state of a particular 'patch' in the world it can send a query to the server asking for that information to be sent to it. The thousands of agents would be distributed evenly amongst the number of workstations being used for the simulation. Alternatively the server could send out the entire world information at regular intervals and then all computations could be computed locally. This would work in simulations where the individuals interact through their environment and not with one another. This set-up does not allow for two agents to communicate directly. A diagram showing this is shown on the next page.

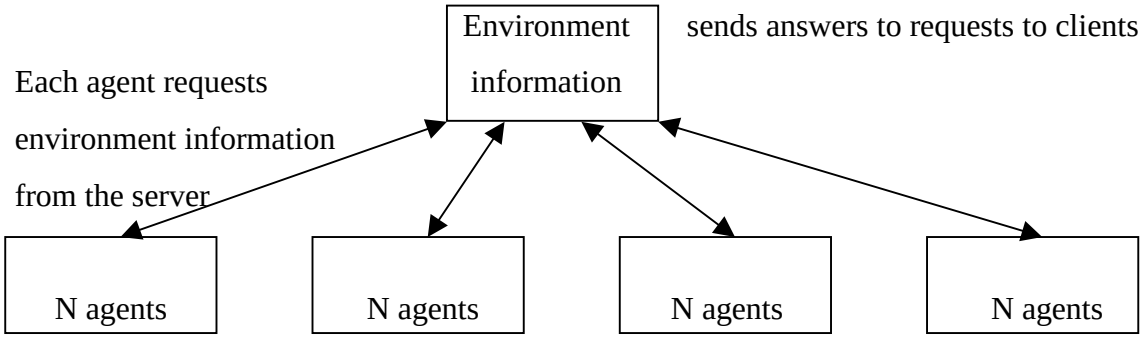

Fig 6-21 : First option for an organisation for massively parallel simulations using *dtkeden*

Another way of organising this kind of computation is to split the world into patches and distribute to each workstation an equal number of patches. An individual agent that is on a patch will reside on the processor that the patch is associated with. The server would be left free to handle communications between clients. Interactions between individual agents could then be computed locally at each processor and if an agent moves out of a patch held by a processor its information is sent to the server which forwards it to the required processor.

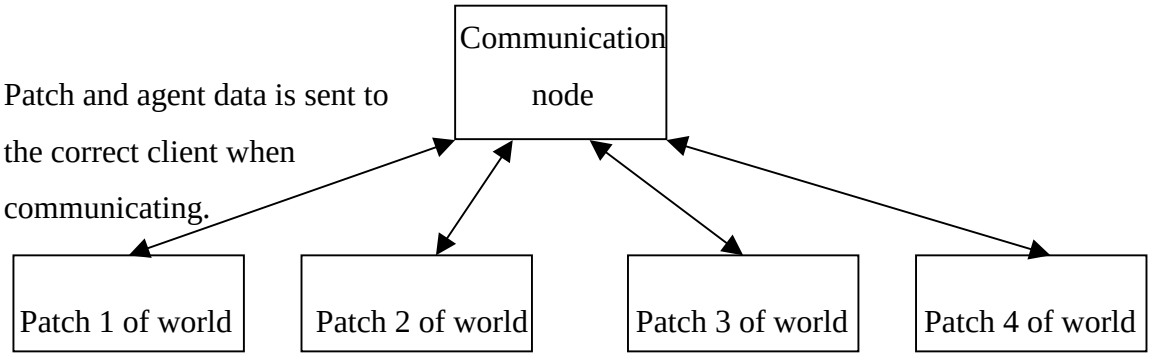

Fig 6-22 : Second option for an organisation for massively parallel simulations using *dtkeden*

It would be interesting to see whether the *dtkeden* tool is capable of supporting many thousands of agents acting in parallel and to see whether massively parallel simulations running this way benefit from the open-ended interactions and experimentation that Empirical Modelling provides.