Chapter 7

Cadence and Empirical Modelling

With the Cadence framework being identified in chapter 6 and a prototype with examples given in chapters 4 and 5, it is now possible to analyse Cadence through comparison and independent use. In doing this two additional prototype systems were developed to compare Cadence with EDEN and EM. The first is a new custom notation for EDEN that attempts to implement the Cadence semi-structured OD-net in EDEN. The second is a hybrid tool of EDEN and Cadence to see how they complement each other. Using these tools and the Cadence of chapter 4, students have developed various models for coursework which can be used to evaluate the Cadence concepts. Whereas the models in chapter 5 were developed in conjunction with the development of Cadence for testing purposes, the models in this chapter have been developed post development of Cadence specifically to compare and provisionally evaluate. Additionally, myself and Beynon have also developed smaller models to help compare and contrast EDEN and Cadence. This chapter outlines the two new Cadence systems, how Cadence has been taught to students and how they have used and understood Cadence for EM. The intention being to illustrate and justify the framework given in chapter 6.

7.1 Cadence-in-Eden

Previous developers of the EDEN tool added the ability to define custom definitive notations using an Agent-Oriented-Parser (AOP) [Harfield, 2006a, 2003]. Since definitive notations are the intended way to extend EDEN it seems appropriate to attempt to add the Cadence framework, or some subset of it, to EDEN by making use of a custom definitive notation. The purpose of doing this is to help bridge between existing EM tools and Cadence for the benefit of those familiar with EDEN and for the benefit of existing EM models in EDEN. It also enables a better critique to be given, asking why a new notation is not good enough. The notation is called Cadence-In-Eden (CINE). Key aims for the notation are:

1. Provide an object like structure for organising observables.
2. Enable context switching.
3. Allow for the cloning of objects to generate larger models from prototypes.
4. Reduce type restrictions enforced by the Eden notation itself.
5. See how well the notation can link with other notations.
6. Check out the performance and complexity of such an approach.

All of the above will be used to evaluate the approach. Whilst dynamical definitions could be implemented in Eden by having a new observable for each instant in time, this would in practice lead to an exceptionally large number of observables being generated. As a result implementing dynamical definitions in CINE is unrealistic and indicates that adding a new notation is not the ultimate solution. However, seeing how EDEN copes with object concepts is important.

CINE is similar to the DASM notation described in chapter 4 so it will not be covered in depth here[1]. Internally the parser attempts to generate object-like structures

[1]The most significant differences being the use of *dot* instead of a space between labels and the requirement of a semi-colon at the end of each line. Both are required by the AOP.

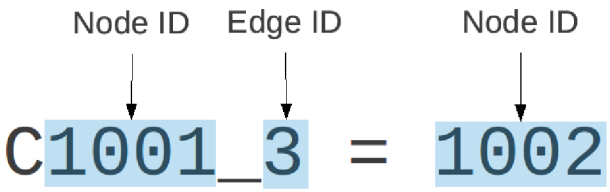

Figure 7.1: An example of an automatically generated Eden observable name using the various ID components. Note that a node id and edge id are actually the same kind of id.

on top of the flat Eden name-space by generating observable names which have two components, a node number and edge number as shown in figure 7.1. These numbers can also be stored in these observables as values and hence it corresponds to the ϕ of eq. 6.7. There is then a mapping from these node numbers to strings and integers using a lookup table (cf. figure 7.2), needed to transcend the limited EDEN type system. An example of the CINE notation is given in listing 7.1 along with its subsequent translation into Eden code in listing 7.2. Figure 7.2 shows the translation mechanism as well as the *names* table used to convert to id numbers.

```
root.table = <1001>;
root.table.sides = <1002>;
root.table.width = 300;
root.table.height = 250;
```

Listing 7.1: Cadence Notation Example.

```
c1_2 = 1001
c1001_3 = 1002
c1001_4 = 5
c1001_6 = 7
```

Listing 7.2: Translation of listing 7.1 into Eden.

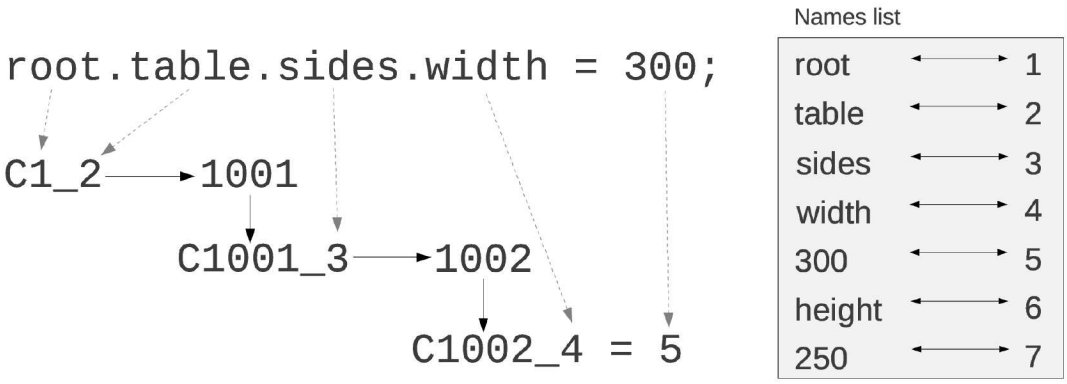

Figure 7.2: An example of the CINE to Eden translation process which involves navigating an existing structure. The names table is also given which corresponds to the examples given.

The short example given in listing 7.1 is an extract from a model of a room adapted from a previous model that used the Donald notation [Yung, 1989]. The room model contains clearly identifiable objects such as a table, light, door and the room itself. Donald provides some concept of structure, much like Scout (cf. §3.3.3), but this is not well mapped into Eden and is certainly not generic enough for use across other notations. As a consequence, the objects in the original model cannot be clearly identified from within Eden. One of the major drawbacks of EDEN is that all communication and connection between different notations must go back to the underlying Eden translation of that notation (cf. issue E1 in §3.3.3 and §6.3.1). The CINE notation is also translated into Eden, so to reduce the impact of this on inter-notation communication, several utility functions have been provided in Eden for the CINE notation as otherwise the modeller would need to understand the automatically generated observable names given in listing 7.2. The use of a selection of these functions is illustrated in table 7.1.

Function 4 in table 7.1 enables the result of a path to be queried, with the path being given as a list of strings where the strings label nodes and edges. The first element of the list is the starting node and the rest label edges to follow from that. If numbers are given instead of strings then they are interpreted directly as node ids. The last example

Table 7.1: CINE notation and the equivalent Eden expressions

root.a = 5	‘observable("root","a")‘ = convertname("5");
root.c.d = 5	define(query(["root","c"]),"d","5");
root.b **is** root.table.height	define("root","b",["root","table","height"]);
?root.table.width	query(["root","table","width"]);
... **is** root.table.height	b **is** navigate("b", ["root","table","height"]);

in table 7.1, *navigate*, is the same as *query* except that it will also add dependencies to an observable so that if any part of the path were to change that observable would be marked as out-of-date. In the example the observable *b* is being defined and so the first parameter to the navigate function is the string form of that observable name so as to identify the observable to add the dependencies to. Such an approach seems convoluted but this is due to EDEN not adding dependencies on observables that are used inside functions (cf. issue D1 in §3.3.2) and so a manual approach is required that makes use of a previously little used feature of Eden[2].

Within the CINE notation itself it is possible to write definitions that are similar to those in the real Cadence tool[3]. Internally definitions translate to the navigate example given in table 7.1. Listing 7.3 is an example of a definition written in CINE with its Eden translation shown in listing 7.4.

```
root.a is root.table.width;
```

Listing 7.3: Cadence notation definition example.

More complex nested paths may be used in definitions by using parentheses in CINE and nested list structures in the Eden form. One good example of this is for

[2]The feature used is $\sim>$ which allows a function to explicitly add dependencies. Unfortunately it was found that this feature did not work correctly and so the EDEN source needed to be patched before the CINE notation could be used (tkeden-1.72).

[3]Not including dynamical definitions.

```
c1_8 is navigate("c1_8", ["root","table","width"]);
```

Listing 7.4: Translation of listing 7.3 into Eden.

conditional statements where the condition is used to select an edge in another object. Listing 7.5 shows an *if*-statement in the CINE notation[4].

```
root.a = true
root.b = 2
root.c = 3

<1000>.true is root.b
<1000>.false is root.c
root.d is <1000>.(root.a)
```

Listing 7.5: An if-statement in CINE showing nested queries.

```
c1_10 is navigate("c1_10",[1001,["root","a"]]);
```

Listing 7.6: Translation of listing 7.5 into Eden.

It is clear that to a large extent the power of object structures and the navigation style definitions in the Cadence framework can be added on top of the EDEN environment as a new notation, but the question remains of how useful it is in this role. If any structures described in CINE are then flattened into unintelligible observables in Eden it seems that all benefits are lost[5]. One final capability of this notation is that of being able to clone an object and therefore automatically generate large numbers of Eden observables. Unfortunately the observables being generated are of the kind in figure 7.1 so is unhelpful to other notations (or the modeller). To take advantage of CINE (cloning

[4]Note how in CINE there is no syntactic sugar for if-statements as there is in DASM.

[5]It becomes nearly impossible for the modeller to interpret the observables as structure without assistance and also requires the modeller to understand the translation mechanism in figure 7.2.

etc) the other notations would need to translate to it and work directly with it, and in doing this the Eden language would be made redundant as the meta-domain language, CINE would take its place.

In order to test the real capabilities and limitations of the new notation before moving on to the next approach (cf. §7.2), a few different types of model were constructed, some of which will be outlined in the following subsections. Each of these models is able to illustrate the benefits of having the Cadence framework applied to EDEN, however they also show that CINE *as a notation* cannot be fully exploited.

7.1.1 Cloning for Timetable and Bubble Sort

One such model developed was an attempt to fix and improve an existing timetabling model (cf. figure 7.12). In this model there are many almost identical display elements for each slot and originally these needed to be manually copied. The author of the original model had attempted to make a form of object cloning to achieve this, however, it has become too complex to understand and adapt. To try out the new CINE notation an attempt was made at performing this cloning activity using CINE.

Due to the complexity of the timetable model a simpler model of bubble sort was constructed first which included similar box like visual elements as the timetable. The first step was to develop a representation of the display elements inside the CINE notation. A prototype point, line and box was developed, as shown in listing 7.7. The tilde operator in CINE means make a clone of the object on the right-hand-side[6]. The last line of listing 7.7 shows one of many definitions relating the individual corner points of the box to the boxes overall position using simple (built-in) arithmetic operations.

With the prototype in place all of the individual boxes to be displayed, representing the cells in an array, can now be cloned. The first cell is cloned from the prototype box (cf. listing 7.8) but all subsequent cells are cloned from cell 1 or 2 to further simplify the script (cf. listing 7.9).

[6]Cloning *null* creates a new empty object.

```
...
proto.line ~ null
proto.line.p1 ~ proto.point
proto.line.p2 ~ proto.point
proto.line.width = 1
proto.line.colour ~ proto.colour
proto.line.type = line

proto.box ~ null
proto.box.type = shape
proto.box.N ~ proto.line
proto.box.S ~ proto.line
proto.box.E ~ proto.line
proto.box.W ~ proto.line
proto.box.x = 0
proto.box.y = 0
proto.box.width = 0
proto.box.height = 0
proto.box.linewidth = 0
proto.box.colour ~ proto.colour
proto.box.N.p1.x is this.parent.parent.x.sub.
    (this.parent.parent.width.div.2)
...
```

Listing 7.7: Line and box prototypes in CINE.

```
bubble.array.1 ~ proto.box
bubble.array.1.width is bubble.array.bwidth
bubble.array.1.height is bubble.array.bheight
bubble.array.1.x = 100
bubble.array.1.y = 100
bubble.array.1.linewidth = 1
```

Listing 7.8: First cell in bubble sort array.

```
bubble.array.2 ~ bubble.array.1
bubble.array.2.prev = bubble.array.1
bubble.array.2.x is this.prev.x.add.(this.prev.width).add.
    (bubble.array.space)

bubble.array.3 ~ bubble.array.2
bubble.array.3.prev = bubble.array.2
```

Listing 7.9: Second and third cells as clones.

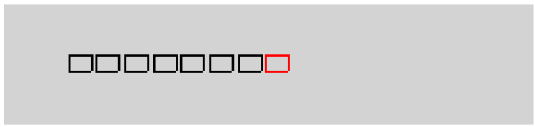

Figure 7.3: Bubble cells from CINE script. The last cell on the right is red.

These structures now exist within the CINE notation and have been translated into the flat Eden name-space[7]. To visualise these boxes it is necessary to utilise the existing Donald notation. There are two ways to achieve this: 1) link the CINE observables by dependency into a Donald script, or 2) automatically generate (via an agent) a Donald script from the CINE structure. The first approach involves entirely replicating the CINE structure in Donald form and then connecting with dependency[8]. This completely removes all benefit of cloning gained and involves the storage of duplicate representations of the same thing. The second approach is far less interactive since any change in CINE will not be immediately visualised until a new script is generated and executed. Neither approach is desirable. However, automatic script generation has been attempted and results in the bubble array cells being drawn as in figure 7.3.

To change the colour of a cell (as with the last cell of figure 7.3) the change in listing 7.10 must first be done in CINE and then a new Donald script generated using the command in listing 7.11 (an Eden procedure).

[7] The actual representation is always in EDEN, CINE is only a view of EDEN.
[8] An example of inter-notation communication difficulties and notation inconsistencies (cf. §3.3.3).

```
bubble.array.8.colour.R = 255;
```

Listing 7.10: Changing a cells colour in CINE.

```
makescript(_cadence_query(["bubble","array"]));
```

Listing 7.11: Updating the Donald display.

Due to the infeasibility of this script generation approach and the complexity of the Eden translations for CINE, the use of CINE for the timetable model was abandoned. The cloning mechanism and the way of structuring the model is faithful to Cadence and works well. However, it is the inter-notation communication problem that prevents it working in practice. Having to generate scripts goes against the liveness principle (cf. §2.1.1) which is fundamentally important for Empirical Modelling. Despite attempts at getting the script to be generated automatically upon changes to CINE, it was deemed to be too slow and inefficient for practical use.

7.1.2 Boolean Lattice Model

The CINE notation may be used for certain kinds of model largely independent of other notations and hence the associated translation problems. A model of a boolean lattice is well suited to being represented in the graph-like way of Cadence. Figure 7.4 shows a visualisation of a boolean lattice that has been described in CINE[9], with the right-hand image being a particular sublattice extracted as a subgraph in CINE.

The lattice can now be manipulated and observed in different ways either by changing the structure in CINE or through procedural actions using the Eden functions provided for interaction with CINE[10] (cf. table 7.1). These kinds of models are difficult

[9] There is some connection to Donald for this visualisation, making use of the script generation technique described above

[10] The need to use special functions in Eden for manipulating CINE structures is an example of issue E4 in §3.3.3

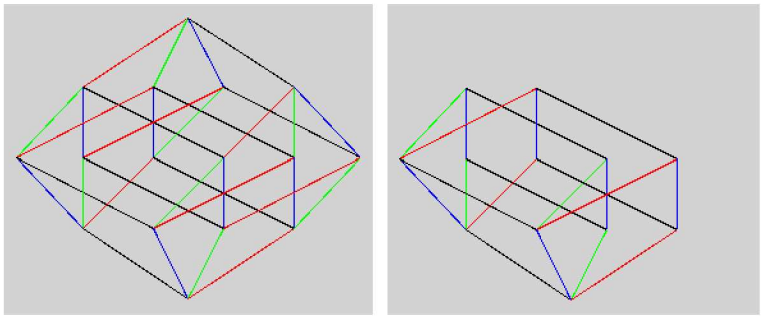

Figure 7.4: Boolean lattice described in CINE. Left is full lattice, right is a subset of the lattice.

when just using Eden, however, a notation called ARCA does already exist[11] for graphs of this kind.

An important feature of the CINE construal of the Boolean lattice is that it allows the modeller to refer to the lattice, and to manipulate it, in ways that reflect the perspective of a mathematician. Specifically, it is easy to reconfigure the structural definition using the Cadence-in-EDEN notation in order to describe constructions that have mathematical interest. These include:

- sublattices, such as may be specified by selecting a subset of the generators;
- quotient lattices, such as may be specified by identifying certain subsets of generators;
- decreasing subsets of the lattice, comprising all those elements that are less than or equal than at least one of a set of non-comparable elements of the lattice.

Decreasing subsets of the Boolean lattice depicted in figure 7.4 feature in the construction of the free distributive lattice on 4 generators, as illustrated in the EDEN model [Beynon, 2003]. The use of CINE leads to a much simpler and more elegant construction than was developed in the EDEN model.

[11] Although not available in newer versions of EDEN.

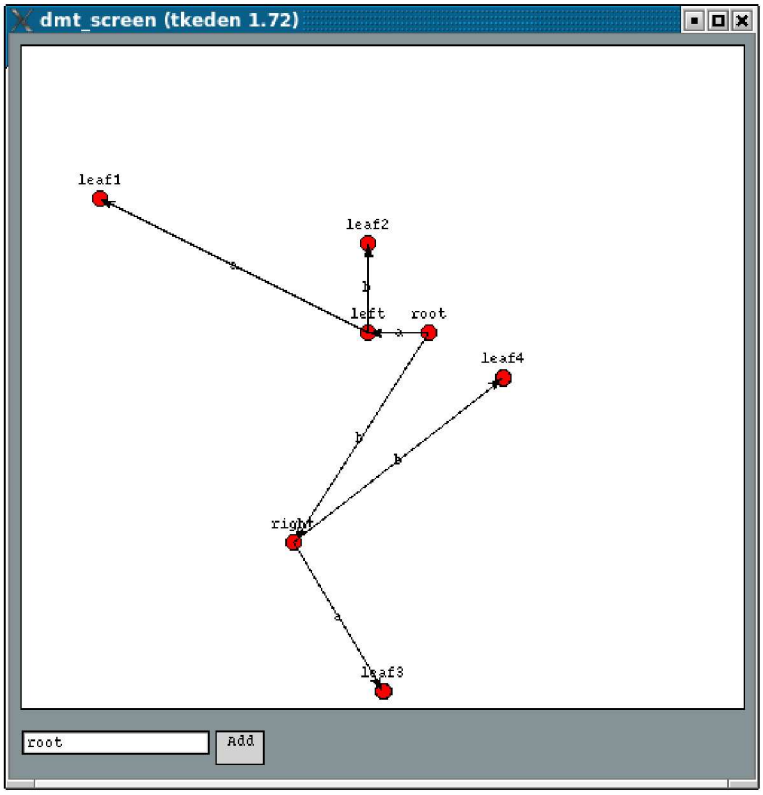

Figure 7.5: DMT visualisation of a CINE structure.

7.1.3 Visualisation using the DMT

Taking advantage of, and adapting, an existing EDEN model (the DMT [Harfield, 2006b]), a graph visualisation of any CINE structure can be generated. Originally the DMT was used to visualise EDEN dependency relationships as a graph but with only minor alterations it can traverse the CINE observables in EDEN to show structural relationships. Figure 7.5 shows the DMT visualisation of the CINE structure given in listing 7.12.

Visualisation of this kind is not included in the Cadence prototype of chapter 4 but shows how Cadence graphs can be automatically visualised. More significantly here though, it is an illustration of custom mediator agents (EDEN procedures) observing ϕ and developing one possible, highly generic, representation of it.

```
root.a = left
root.b = right
right.a = leaf3
right.b = leaf4
left.a = leaf1
left.b = leaf2
```

Listing 7.12: Test binary tree in CINE (cf. figure 7.5).

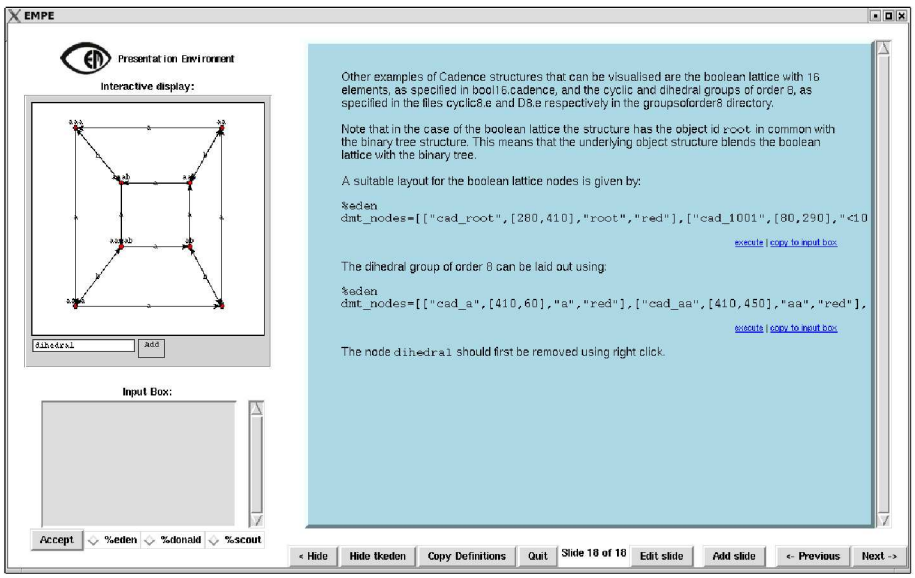

Figure 7.6: Dihedral group of order 8 in CINE, visualised using DMT and embeded into a presentation.

Taking advantage of the DMT visualisation of CINE structure, a model of the dihedral group of order 8 was created and then visualised (cf. figure 7.6). Unlike the boolean lattice model in §7.1.2 where a custom visualisation was developed, here the generic DMT was used. This illustrates how the modelling of structure is a generically useful capability in its own right, rather than only being useful for particular visualisations[12].

It is interesting to compare the use of CINE in the construal of the dihedral group of order 8 with EDEN construals of group structures that have been developed using the ARCA notation (cf. [Beynon, 2003] and the EDEN model of Schubert's Erlkoenig [Beynon, 2006a] discussed in [Beynon, 2006b; Beynon et al., 2006b]). ARCA [Beynon, 1983, 1986a] was the first definitive notation to be conceived, and is unusual in that it incorporates ways of manipulating references to group elements as values. By moving away from the conventional association of value with identifier such as is typical of traditional programming languages and definitive scripts, Cadence is able not only to emulate this feature of ARCA, but to support much richer and more general modes of reference and manipulation.

So structure is useful for certain kinds of model and CINE works well when there is minimal inter-notation communication. However, for cloning, visualisation and general model management it is not practicable to have it only as a notation on top of EDEN. Putting the Cadence framework above EDEN fails to resolve the key issues identified in §3.3[13] and, therefore, the Cadence framework must form the foundations of an EM tool in order to benefit fully from it. The ϕ (cf. eq. 6.7) function needs to be considered, it is argued here, as the most primitive mode of representation.

[12] Previously structure was used in Donald and Scout but was not often utilised as a core part of a model.

[13] Some can be resolved but at the expense of exacerbating others.

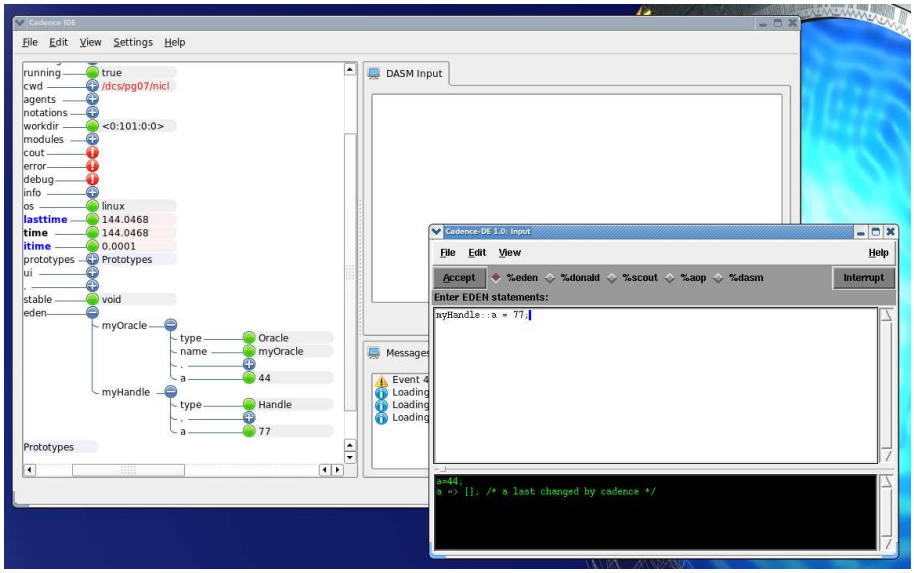

Figure 7.7: Screenshot showing EDEN running inside Cadence and communication between the two tools.

7.2 Eden-with-Cadence

An alternative approach to constructing a new notation within EDEN is to develop a hybrid where both the Eden and Cadence tools are combined and given mechanisms for inter-tool communication. The full benefit of both environments is then available and given sufficiently well developed communication mechanisms the powers of each can be utilised together to produce new kinds of model that might previously have been too complex or impossible. The issues of translation that exist with CINE are no longer relevant and the modeller is free to customise the communication. Whilst Cadence should be capable of all that EDEN is capable of in principle, in practice it is a far less well developed prototype and lacks the extensive library of features and past projects that EDEN has. The benefits of CINE also apply here, Cadence can be used to enhance existing models in EDEN.

The hybrid was achieved using the Cadence module mechanism. EDEN was converted into a Cadence module that could then be loaded dynamically into an ac-

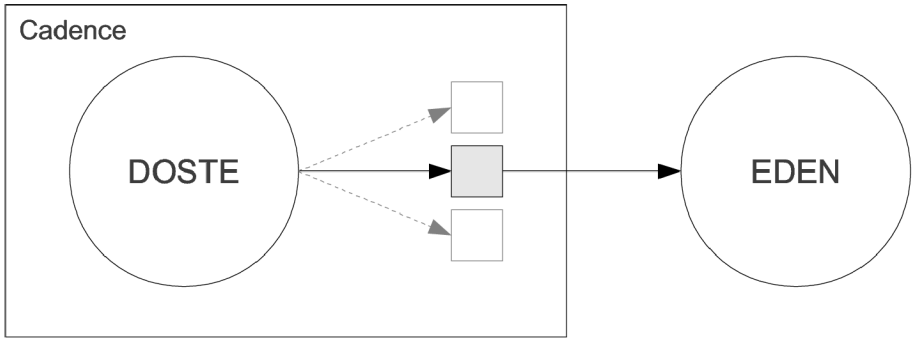

Figure 7.8: Architecture of Eden-with-Cadence hybrid.

tive Cadence environment as desired. This involved minor modifications to the main run-loop of EDEN so that it became driven by Cadence rather than being its own application. Additional changes and simplifications were required to support inter-tool communication[14]. The diagram in figure 7.8 illustrates this architecture.

7.2.1 Inter-Tool Communication

The module (shaded grey in figure 7.8) is a C++ file implementing a communication mechanism between the two tools, and is approximately 200 lines. Two mechanisms were developed to enable the two tools to communicate with each other. Both of the approaches involved the sharing of observables but did this in different ways:

1. Mapping the EDEN symbol table directly into the Cadence graph (cf. figure 7.9).
2. Using an LSD style agent mechanism based upon oracles and handles (cf. figure 7.10).

The first approach would have enabled an almost seamless means of accessing all EDEN observables from within the Cadence graph. A Cadence handler was developed that routed events from Cadence destined for EDEN observables into the EDEN tool where they retrieved or modified the relevant observables. Ultimately this approach failed

[14]The existing virtual agent mechanism was adapted to provide name-spaces that could be used as contexts and mapped into Cadence.

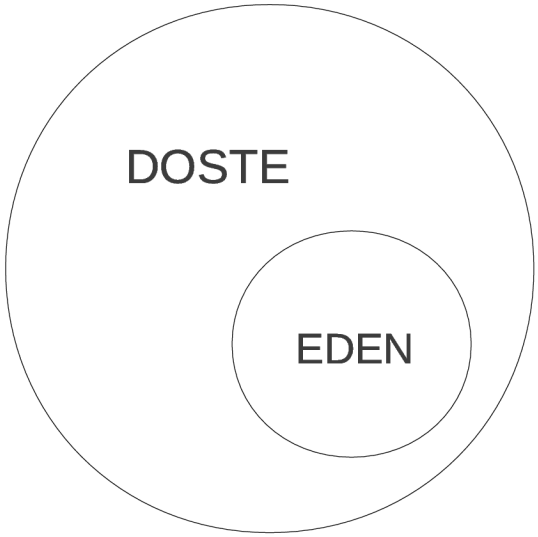

Figure 7.9: The EDEN symbol table is mapped into the DOSTE graph to merge the two tools at the lowest level.

due to the disparity between type systems and the definition evaluation mechanisms. Since an EDEN observable could be given both an EDEN definition and a Cadence definition there was potential for serious conflicts to develop. Also, EDEN observables could not contain objects and so broke when such things were attempted from within Cadence. This low-level merging was eventually abandoned in favour of the second approach.

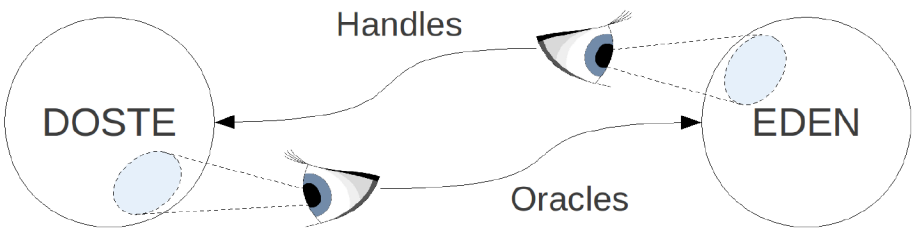

Figure 7.10: Oracle and Handle approach to inter-tool communication. Handles observe EDEN and change DOSTE whilst Oracles observe DOSTE and change EDEN.

With the second approach agents were created in Cadence which monitor certain nodes in the Cadence graph and certain name-spaces in Eden (a feature enhanced as a part of this work). When changes are noticed by these agents they then perform a corresponding change in the other tool. They act as intermediaries that provide Eden with oracles and handles to the Cadence environment. An oracle is something an agent can observe, whilst a handle is something an agent can change[15]. Whilst some typing issues still remain they are easier to resolve using this more relaxed approach to communication.

The *oracle* and *handle* agents are specified inside Cadence as shown in the examples of listings 7.13 and 7.14. Both types of agent must be placed inside an *eden* context so that the EDEN module can create the appropriate agents.

```
.eden myHandle = (new
    type = Handle
);
```

Listing 7.13: EDEN handle for Cadence specified in DASM.

```
.eden myOracle = (new
    type = Oracle
    a = 1
);
```

Listing 7.14: EDEN oracle for Cadence specified in DASM.

Inside Eden it is now possible to create and change observables inside a myHandle namespace as well as observe changes to observables in a myOracle namespace. Listings 7.15 and 7.16 give Eden statements corresponding to the handles and oracles defined in listings 7.13 and 7.14.

[15]Both oracle and handle are terms borrowed from LSD.

```
myHandle::a = 2;
```

Listing 7.15: Eden script changing a Cadence handle.

```
?myOracle::a;
```

Listing 7.16: Eden script observing a Cadence oracle.

By separating observables into oracles and handles rather than mixing the two it creates read-only and write-only observables. As a consequence the problem of multiple definitions on a single observable disappears since changing a read-only oracle in Eden by giving it a definition will have no effect upon Cadence and such a definition will be removed by Cadence when it changes that oracle. The opposite it also true for handles.

The following sections discuss a few of the models developed using the hybrid tool and communication mechanism discussed here. Table 7.2 lists features that are supported by EDEN and Cadence. From the table it is clear that neither tool is comprehensive (at present) and so the models discussed illustrate ways in which each can be taken advantage of.

7.2.2 Traffic Light System

Originally this model was developed by James McHugh, an MEng student, as a part of his coursework for the Empirical Modelling module at Warwick. It used an older version of the Cadence-EDEN hybrid that still utilised the first inter-tool communication approach of merging symbol tables. More recently McHugh's model has been updated to make use of the newer oracle-handle approach to connecting the tools together. The model itself is of a T-junction with traffic lights and shows cars and queues to see the effect of altering the traffic light sequence. McHugh's objectives were to improve visualisation, realism and increase the complexity of related older traffic models. For this reason

[16] Using SASAMI notation but is far more basic than the game library of Cadence

Feature	Cadence	EDEN
Structural relations	✓	X
Latent relations	✓	✓
Dynamical relations	✓	X
Custom Agency (live)	X	✓
Custom Functions (live)	-	✓
2D line drawing	X	✓
3D models	✓	✓[16]

Table 7.2: Comparison between EDEN and Cadence

Cadence was used for visualisation and animation using dynamical relations.

It is possible to construct such a traffic model entirely in Cadence, however, McHugh gives a reasonable argument for making use of the hybrid. His argument is that a real traffic light system would likely be controlled by a procedural program of sorts and so he wished to write that aspect of the model in a procedural way. Custom procedural agents were therefore required and so this feature of EDEN was to be used. For the rest of the model he concluded that Cadence, with its dynamic definitions and objects (with cloning), would be more suitable for representing cars and their motion. For the purposes of this section the use of the hybrid features will be focused upon.

Listing 7.17 gives a script extract from the newly updated McHugh model[17] which links an observable in Cadence with an observable in EDEN by dependency.

In the model there is procedural Eden code linked to a timer that calculates the light sequence (cf. listing 7.18). There are 3 observables, state1, state2 and state3 which correspond to each of the traffic lights at a 3 way junction. Each light can be in one of 4 states which correspond to which lights should be active (the observables can have the values 0,1,2 and 3). In Cadence handles and oracles have been set up in an 'eden' context and so a definition has been used to connect the EDEN state observable to

[17] Update done by myself in December 2010.

Figure 7.11: James McHugh traffic light model

```
@display %deep light1 = (new union (@tlight)
    x = 339
    y = 330
    stage is { @root eden handles state1 }
);
```

Listing 7.17: DASM script connecting Cadence and EDEN in the Traffic model.

the corresponding state observable for that light in Cadence (cf. listing 7.17). Other observables have been used in Eden as oracles which allows Cadence to control aspects of the Eden code.

What this shows is the successful integration of a procedural script written in Eden with the Cadence OD-net. Agents can control and observe specific observables. It is a seamless integration of two paradigms and two tools using dependency. It also shows that having just a single paradigm, as Cadence does at the time of writing, is not always desired and that allowing for a mixture of paradigms is the way forward. In this respect it highlights some limitations of the current Cadence implementation (cf. chapter 8). At the same time it also justifies the existence of dynamic definitions and object structures as well as better integration with more sophisticated graphics libraries and so on. Without these newer capabilities such a model would be a challenge to construct in this more realistic way.

7.2.3 Timetable Revisited

In the late 90's a group of EM researchers, led by Beynon, started a project that involved developing a timetabling model in EDEN which the department secretaries could then use to organise project orals [Keen, 2000; Beynon et al., 2000b]. As can be seen in figure 7.12, this model involved a large number of almost identical cells and other components. Ordinarily each one of these cells would need to be modelled, most likely involving a lot of copy paste operations in a script. Due to the scale Allan Wong decided against doing this as it was inflexible to change. Instead he developed a mechanism which allowed him to effectively clone some existing default observables and automatically construct all the required observables by iterating a procedure. The implementation of this has since proven to be very complex and difficult to understand and is not generic in nature.

The model is an obvious candidate for trying out the object and cloning characteristics of Cadence, and following the failed attempt at using CINE it seemed appropriate to try again with the hybrid. Beynon and myself have partially developed a new version

```
proc Tlight1:iclock {
   if (state1==4){
      if (state2==4 && state3==4 && turn==0){
         state1=3;
         p1=4;
      }
   } else if (state1 == 3) {
    oticks1++;
    if (oticks1 > otime1) {
        if (p1 == 2) {
            state1 = 4;
            turn=1;
        } else {
            state1 = 2;
        }
        oticks1 = 0;
    }
   } else{
      gticks1++;
      if (gticks1>gtime1){
         state1=3;
         p1 = 2;
         gticks1 = 0;
      }
   }
}
```

Listing 7.18: EDEN agent controlling a traffic light. *state1* is a handle observable for Cadence (cf. listing 7.17).

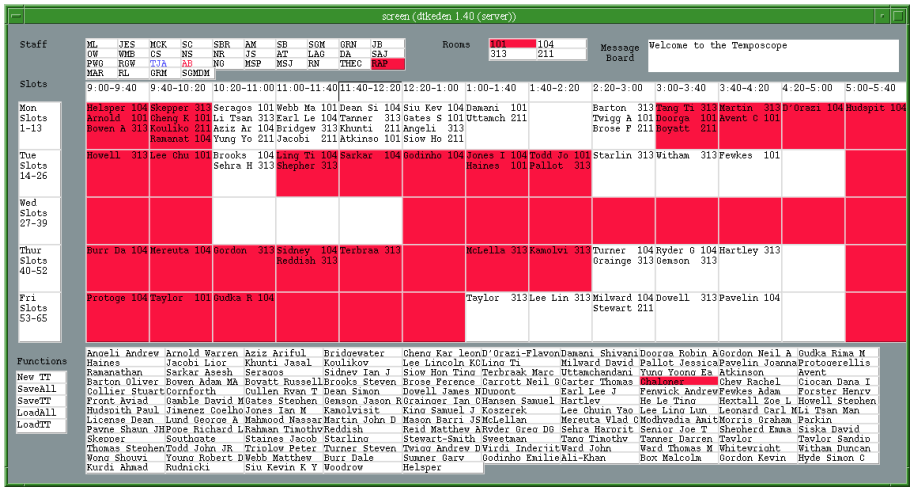

Figure 7.12: Original Eden timetable model (Chris Keen version)

of the timetable model where the cells are represented in Cadence but the visualisation and interface still make use of Donald in EDEN. The intention here was to be able to clone cells and have an automatic mechanism by which these cloned Cadence objects get converted into the required Donald observables in Eden.

```
@slotswin slotnum is {@root eden output sn};

.eden input = (new
    type = Oracle
    x is {@timetable slotsTable (@day (@slotswin daynum))
                                (@slotswin slotnum) x}
    ...
);
```

Listing 7.19: DASM context selection for cells.

To generate the Donald visualisation, context switching was used in Cadence to cycle through all the cell objects (cf. listing 7.19). The currently selected context (controlled by Eden, as shown in listing 7.20), or cell, has its various attributes connected

```
winix = 0;
sn is winix % 13;
output::sn is sn;
```

Listing 7.20: EDEN selects the context with a handle observable.

```
x is input::x;
wloc is "\tbox␣:␣" // "[{" // str(x) // "," ...;
```

Listing 7.21: Generating Donald script from Cadence oracles.

to an Eden Oracle agent so that when a context changes Eden will be able to observe that new cell. In Eden, various observables with dependencies on these Oracles build up a Donald script using string operations and finally a triggered procedure is used to execute this generated Donald script (cf. listing 7.21). The result is a considerably more elegant and generic way of generating large numbers of similar display objects.

Whilst this approach to generating Donald scripts from Cadence structures is interesting, it suffers from several problems that, as with CINE, mean that the approach is less than satisfactory:

1. Oracle observables need to be manually constructed, although this could in principle be automated by some other means.

2. Inter-tool communication is not responsive enough. There were problems with changing observables too quickly and for certain cells to be missed. Other models have also demonstrated this problem[18].

3. No direct dependency links. If something changed in Cadence then EDEN would not be aware of it until a new Donald script was manually regenerated.

4. Need to flatten the Cadence structures and rebuild object-like structures of Donald.

[18] A hybrid (Cadence-EDEN) traffic model by James Brotherton-Radcliffe skipped updates seemingly randomly, causing the logic to break.

The conclusion that can be drawn from this is that using EDEN for visualising Cadence suffers from the same problems as CINE. Either everything needs to be fundamentally based upon Cadence to take advantage of its structural characteristics, or a far more sophisticated hybrid is required. However, the use of EDEN agency to control and manipulate Cadence has potential for simpler models.

7.3 Cadence in the EM MSc Module

Having seen how both the CINE notation and the hybrid fail to resolve the issues of §3.3, and that EDEN cannot be modified to support the Cadence framework of chapter 6, it is now time to revisit the Cadence prototype of chapter 4 as a standalone environment. The question is: *Can Cadence be used for Empirical Modelling?* To answer the question, this section explores how Cadence has been used in the teaching of Empirical Modelling.

Both the 2009 and 2010 class of MEng and MSc students doing the CS405 Empirical Modelling module have been introduced to Cadence and the Cadence-EDEN hybrid tools. Each year has seen approximately 40 students so a total of 80 students have used Cadence and the hybrid, along with other 3rd year project students. For the module the students had roughly 8 lab sessions of 2 hours each and around half of these were devoted to Cadence or the hybrid tool. Also, several of the lectures discussed Cadence and EM tools generally to explain the conceptual aspects of them. The nature of the labs and coursework will be given here along with some feedback and example models from the students.

7.3.1 Teaching Cadence

The lectures for the module primarily focused on the conceptual aspects of Empirical Modelling in general. However, a few lectures took a look at Cadence in an attempt to explain how it relates to the EM conceptual framework and how it should be used[19].

[19] These lectures were given prior to the full development of the Cadence framework given in chapter 6.

The following Cadence related topics were covered by the lectures:

- Multiple paradigms, including: prototype-based objects, data-flow, reactive and functional.
- Graph terminology and computation as navigation concept. Material for this was taken from chapter 4 of this thesis.
- Merging of requirements, design, implementation and testing into a single environment. Not entirely specific to Cadence but was used to explain the intention of Cadence.
- Dealing with different contexts. Switching between models or parts of models better supported in Cadence due to the object mechanisms.
- Dynamic definitions. The benefits and difficulties of having process observables.
- Characteristics of the Stargate model. The different agent views and the way state has been modelled.

7.3.2 Lab Sheets

For 2010 there were 5 labs based around the Cadence tool, as outlined below [Pope and Beynon, 2010a]. The labs are not assessed but are used to teach the students how to use tools they will later require for their coursework. It was decided by Beynon, the module organiser, that Cadence provided a more up-to-date and exciting tool for students to use despite its prototypical nature. One of the purposes for using Cadence was to see how well the students took to it and what kinds of models they would eventually be able to construct using it. In addition, whilst the students were working with the tool it allowed myself to develop better interfaces for them to work with and find bugs that had not previously been seen.

1. **Introduction to the EM tools** The first lab of 2010 introduced the Cadence tool using the Stargate model discussed in chapter 5. Students were given exercises

that involved observing and modifying the model interactively, becoming familiar with the DASM syntax and the nature of the tool. Some of the semantics of the different definition types was explained.

2. **Lift Exercise** Lab 2 asked the students to construct their own model from scratch in Cadence. The exercises gave examples of how to clone an image object, customise it and put it on the screen. Using these skills they built up a model of a lift which they went on to animate, making it move between floors when buttons were clicked using the dynamical definitions in Cadence. The final exercise was open-ended, asking the student to extend the model in various ways.

6. **Using Cadence and EDEN together** Having introduced EDEN in a previous lab, the Cadence-EDEN hybrid was introduced to the students as a way of supporting agency. The main focus of this lab was on demonstrating inter-tool communication using oracles and handles. They were tasked with constructing a word game model that made use of both Eden and Cadence and involved communication between the two.

7. **Cadence DIY introduction** Although mostly a coursework preparation lab, a question and answer session on Cadence was also given which went over more difficult topics with examples. Questions asked related to the semantics of definitions and how to use the deep cloning feature of DASM.

8. **Networking with Cadence** The final lab of the year asked students to work as pairs on the same model. Similar to the lab on "Using Eden and Cadence together", this lab asked them to use two machines running Cadence to communicate via the networking module. It also made use of EDEN and they were tasked with getting observables of a model in EDEN to connect with Cadence, shared with another Cadence using XNet and finally to have them appear in another instance of EDEN on that remote machine.

7.3.3 Coursework Models

Some of the models developed by students have already been introduced (cf. §7.2.2), however, there are two particular pure Cadence models that are worth mentioning. A SCUBA diving model by William Dangerfield and a Calculator model by David Evans. The calculator model is of particular interest because it demonstrates, to an exceptional degree, the use of Cadence characteristics not found in existing EM tools. This section will explore these two models.

Calculator

The calculator model developed by Evans implements a calculator from binary logic gates (cf. figure 7.13). The logic gates are also modelled in Cadence using the boolean operators as given in listing 4.17 and figure 4.8[20]. Of course, the boolean operators are themselves modelled as structural relations.

What is remarkable about the calculator model is the purity and scale of it. There are over 440k observables[21] involved in the model, with many of those having latent or dynamical dependency definitions, and no use of agency beyond the game library for visualisation[22]. The entire model has been described within the OD-net (Φ). It is an example of a program that has fully transitioned from a construal, although it could be argued that it was program-like from the beginning since there was always a clear objective and little ambiguity or subjective decision making since it is a well understood artefact. However, Evans did not understand all of the components involved prior to development and so it was initially a construal to him. Despite it being program-like from the start, it does show how Cadence can support artefacts that can be considered as programs whilst still remaining flexible. Evans himself states that:

> *"[Cadence] allows any user with some deeper knowledge of the model to directly modify any part of the calculator, for example by causing a bit to*

[20] Evans also extended the existing boolean operators to include *xor*.
[21] The largest EDEN model to date only contains 5k observables so this is almost a 100 fold increase.
[22] Even the LCD screen has been modelled down to the individual segments

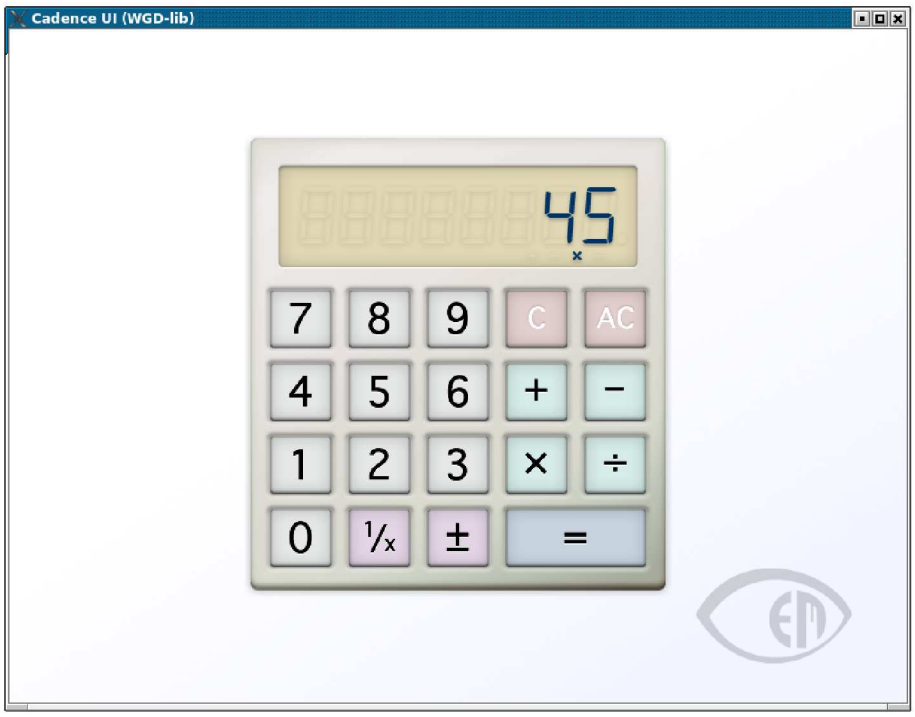

Figure 7.13: David Evans' Calculator Model 2011

be stuck on, or modifying the behaviour of certain buttons to simulate a hardware glitch. The potential ways in which these bit operations might fail can be better understood in this way. Similarly, new functionality can be added..." [Evans, 2011]

The modifications of which he speaks are done live via the Cadence IDE. To manage, comprehend and construct a model containing 440k+ observables requires the use of structural organisation. Evans makes extensive use of Cadence's structural relations and cloning capabilities by, in a hierarchical fashion, constructing individual components such as bit comparison constructs (cf. listing 7.22) which are then combined to produce a byte comparison construct (cf. listing 7.23). The approach taken here is similar to that used by Subtext as described in §2.1.3. Each of these components were tested independently via the IDE and refined from an initially provisional prototype to a functional component. The artefact was decomposed into smaller more manageable models, something that is more difficult to achieve in EDEN (although not impossible). Figure 7.14 shows the prototype hierarchy found in the calculator model.

```
.binaryStuff bit_nequals = (new union(.binaryStuff bit)
    input1 = null
    input2 = null
    b is {.input1 b xor (.input2 b) or (.prev b)}
);
```

Listing 7.22: Bit inequality "function" for calculator model.

Dynamical definitions were also used in the calculator to observe events such as button presses and to store state in register like constructs. This use of dynamical definitions for registers relates strongly to attempts in the EDEN Logic Simulator [Lee, 2007], discussed in §3.3.2, where similar structures could not be constructed without resorting to using agency in an unprincipled way to arbitrarily break cycles.

Evans gives a small critique of Cadence at the end of his coursework paper

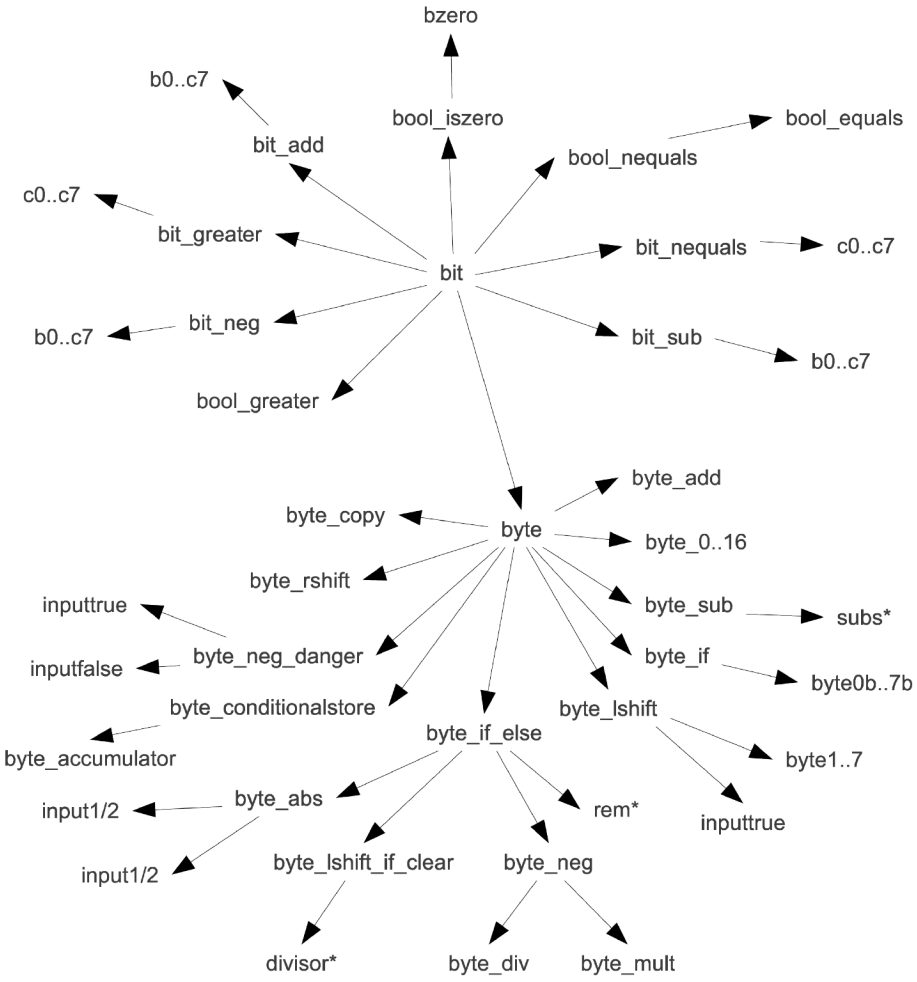

Figure 7.14: Partial prototype hierarchy for the Calculator model.

```
.binaryStuff bool_nequals = (new union(.binaryStuff bit)
    # returns input1 != input2
    input1 = null
    input2 = null
    %deep c0 = (new union(.binaryStuff bit_nequals)
        input1 is {..input1 b0}
        input2 is {..input2 b0}
    )
    %deep c1 = (new union(.binaryStuff bit_nequals)
        prev is {..c0}
        input1 is {..input1 b1}
        input2 is {..input2 b1}
    )
    ...
    %deep c7 = (new union(.binaryStuff bit_nequals)
        prev is {..c6}
        input1 is {..input1 b7}
        input2 is {..input2 b7}
    )
    cLast is {.c7}
    b is {.cLast b or (.input1 negative xor
            (.input2 negative))}
);
```

Listing 7.23: Byte comparison construct using individual bit comparisons. Input bytes are given at the top and the result is the value of *b* which is indivisibly calculated by dependency.

where he states that there appear to be two major limitations. 1) the lack of iteration for generating large numbers of instances and 2) performance when more than 500k definitions are involved. The first problem can be resolved with richer support for meta-relations and/or richer forms of custom agency (cf. §§8.2.5 and 8.2.6). The second issue is a consequence of the Cadence tool as implemented in chapter 4 being only an unoptimised prototype. Whilst a calculator is a relatively small program, even if implemented from logic gate foundations, this model does show that Cadence supports flexible programs.

SCUBA Diving Model

A model of decompression when SCUBA diving was constructed by William Dangerfield which showed how a diver responds in certain scenarios (cf. figure 7.15). The model is another example of a construal that has evolved to become a program. The "program" can now be used visually and interactively to demonstrate to novice divers the problems of diving and surfacing too quickly[23].

The user of the model can make the diver dive to different depths at different rates and surface again. The model will then calculate nitrogen pressures in different parts of the body ("compartments") and display these to the user, highlighting them orange and red if they are too high. The model could, as Dangerfield comments in [Dangerfield, 2011], be extended to include other gas mixtures and additional compartments.

Whilst the model does, like the calculator, make extensive use of cloning, it is the use of dynamical definitions that is more significant here. The model is inherently dynamical in nature with gas concentrations continually changing over time without any particular reference to agency causing this change.

Dangerfield was not familiar with decompression models prior to starting the development process. He came to understand these models by experimentally and in-

[23] Dangerfield has shown the model to instructors who have commented that it would be useful.

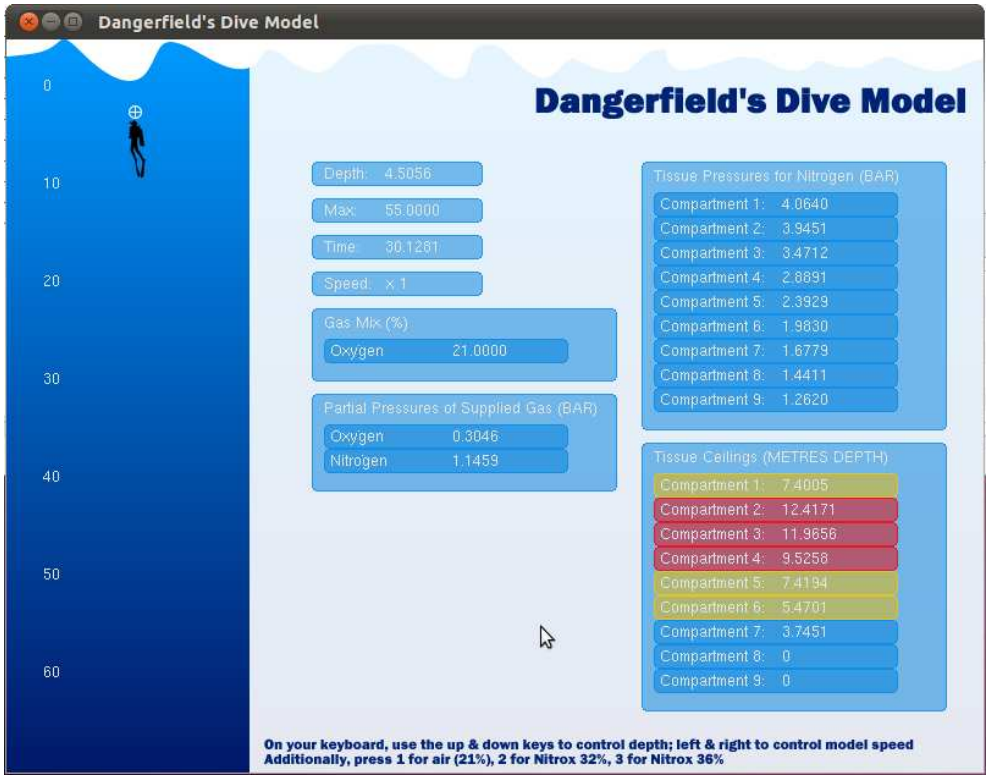

Figure 7.15: William Dangerfield's SCUBA Diving Model

crementally developing this diving model. In other words, the model started life as a provisional and private learning exercise which solidified into the more generic and public artefact presented in figure 7.15.

7.3.4 Student Feedback

Some feedback was gathered from the students after they had completed the coursework by using a web form questionnaire (cf. appendix C). For the most part the students say they understood the concepts of "computation by navigation" and "dynamical relations" (56% and 89% respectively replied yes when asked). The "friendliness" of the interface and DASM notation were also judged to be average to good, although 22% did suggest that DASM could be simpler and the comments relating to this were about confusion over the use of brackets and operator precedence.

For the students who did not choose to use Cadence for their coursework assignment, they were asked why. The response was overwhelmingly (56%) that there was not enough documentation and a lack of confidence in the tool (22%). The lack of confidence was often associated with a lack of documentation and example models[24].

Other criticisms and suggestions include the following:

- A lack of formal technical syntax (for DASM).
- Better (and earlier) explanation of navigating a tree (graph).
- Better file handling[25].
- A means of clearing state and variables to reset a model.
- Interface was unstable and crashed regularly.
- Performance in updating observables in larger models.

[24] Specifically it was the lack of documentation of the game library and %deep cloning mechanisms that were problematic

[25] The file handling facilities have since been improved by searching in customisable default locations.

- Missing math functions and no way to specify them[26]

Largely these problems are due to the experimental and prototype nature of the Cadence tool they were using. Documentation had not been written as the tool itself was still being developed. The other issues such as "clearing state" and "missing math functions" have already been identified in chapter 6 as needing further work and are discussed in chapter 8. The success of the models of those students who did choose to use Cadence is without question. Compare, for example, the calculator and SCUBA diving models with previous EM models in EDEN that have similar characteristics (e.g. Timetable [Beynon et al., 2000b; Keer, 2010]). The internal structure of the model is much more clearly and elegantly captured, and the interface to the models benefits from access to observables through the Cadence GUI and much superior visualisation of current state.

[26] Generic definitions were not shown to them, but nor are they sufficiently well implemented.