Chapter 3

Enabling Plastic Applications

Within this chapter the exact nature of the problem being addressed by this work is explored. In chapter 1 the concept of *plastic applications* was introduced and identified key problems that need to be overcome on a grand scale. Chapter 2 then introduced Empirical Modelling along with EUD and other technologies that are of considerable interest with respect to achieving the *plastic applications* aim. This work will focus on adapting the existing Empirical Modelling concepts and tools in order to fulfil that goal and to do this it is necessary to give a critique of EM. Once specific problems have been identified the focus will turn towards industry and EUD approaches to see how they may resolve the problems found in EM. The result of this will be a specific set of questions and ideas that are to be the focus of the remaining chapters in this thesis. It is in this chapter that the problems with EM are identified with solutions proposed and the specific research questions being expressed.

3.1 Empirical Modelling and Plastic Applications

Plastic applications have been introduced by this work to classify applications which can be freely moulded by an end-user whilst also being capable of solidifying into what might appear to be a more traditionally developed application. Perhaps another phrase

is *flexible functionality* programs as opposed to *fixed functionality* programs[1]. Empirical Modelling has developed a similar concept in its notion of "program" which is something that has evolved from a construal and has become "constrained" through *ritualised* and restricted interactions. An EM "program" is, as already discussed in §2.2.1, not the same as a traditional program in that it supports *flexible functionality*. In this work the EM notion of "program", when it has a particular resulting functional objective, will be taken to be the same as the concept of a plastic application. The EM idea of "program" has only become clear recently as a result of this work on plastic applications. Previously to get to a program from an EM model it was assumed that a translation process was required (cf. figure 2.10). As a consequence there has been little to no prior research into how to transition from an EM model to a plastic application without a translation step. Whilst the EM conceptual framework has easily (and appropriately) been adapted to include this idea, there is little support in the EM tools for it and there are likely to be conceptual consequences yet to be identified. Despite this the Empirical Modelling conceptual framework is to be explored as a means of supporting plastic application development. In the remainder of the thesis the meaning of *program* will, unless otherwise stated, be the EM concept of "program" instead of the traditional interpretation.

In Empirical Modelling the migration from *construal* to program is considered as a change of context with a "different family of pre-engineered interactions and interpretations" [Beynon, 2011] but is always represented as "a net of observables and dependencies". This net will be known as the *Observable Dependency Network* (OD-net). The OD-net is developed over time from experience and so its meaning remains directly grounded in experience, consequently the program remains meaningful so long as the OD-net remains. So to enable plastic applications it is necessary to constrain interactions and interpretations of the OD-net by human and non-human agents but keep the OD-net in place at all times, allowing any restrictions to be relaxed. A tradi-

[1]Fixed and flexible do not refer to any run-time modularity mechanisms or similar techniques, but are much more radical in nature and directly involves end-users.

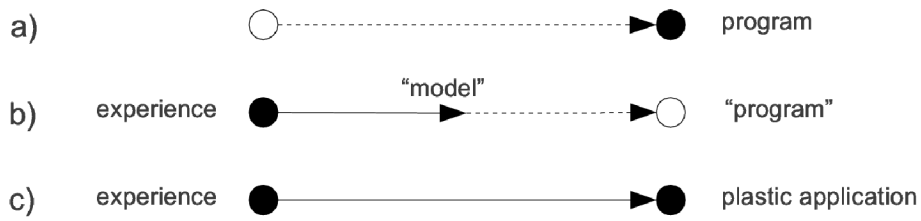

Figure 3.1: Transition from personal to public. a) shows traditional programs that have no such on-line transition. b) gives Empirical Modellings attempt and c) shows the ideal plastic applications result.

tional program does not have an OD-net, or any similar concept, and as a consequence remains isolated from experience which is perhaps why there are such difficulties in dealing with changes to requirements and also why formal approaches need to be taken to provide semantics. Looking back at figure 2.2 a traditional program is removed from the refinement curve by an abstraction process that is performed as a separate step. This step is usually performed to translate from requirements and design into an actual, and optimised, implementation on a machine. Such a step is contrary to the first EUD principle given in §2.1.1, that of being a *gentle-slope* system. It is also unnecessary since Self [Hölzle and Ungar, 1994] has shown how interactive applications can be optimised as needed and EM has given extensive evidence that rich models can be developed without abstraction being centrally important[2].

The Empirical Modelling process purports to support the full transition from experience to program, a reasonably recent development in itself. The reality is more like that shown in figure 3.1 where EM only partially supports the transition in practice. Much of the work on enabling EM tools and concepts to be used as a plastic software environment will relate to enabling the transition from "model" to plastic application.

[2]For example, colour sudoku [Beynon and Harfield, 2010; Harfield, 2007b], the temposcope [Beynon et al., 2000b] and the ant navigation model [Keer, 2010].

3.2 Dimensions of Refinement

To get a better handle on what is needed to support a plastic application it is necessary to revisit the dimensions of refinement first introduced in §2.2.1. The refinement process, as illustrated in figure 2.2, has at least four dimensions which together are a measure of the refinement of a software artefact. These dimensions are:

1. Personal → Public
2. Subjective → Objective
3. Provisional → Assured
4. Specific → Generic

It is the right-hand-side, the public, objective, assured and generic, which are the plastic application end of the refinement process and which need the most attention. Each of these dimensions has certain characteristic properties associated with them which give a form of specification for what any plastic software environment must enable.

Public A public entity needs to be communicated and shared among many people. Any experientially-mediated associations and observables in the artefact are to be understood by others in that they can also identify the same associations in experience. To share an experience there needs to be a common understanding, a common interpretation, perhaps by following certain conventions or by "embedding" specific interpretations into the artefact.

Objective The artefact must become independent from the individual modeller's emotions and thoughts by being fully realised as an actual artefact. It needs to be complete in the sense that no part of the model can remain only in the mind of the modeller, and also specific rather than fuzzy about what it is.

Assured A fixed, precise and perhaps formal interpretation is required. There may need to be certain guarantees, either by formal proof or by extensive testing, to show

that the application fulfils some goal. Again restrictions on interaction become important, as do security concerns if multiple users are involved. Resilience and robustness are also important.

Generic Instead of being about a specific concrete scenario the application becomes abstract and adaptable to more generic situations. The application can then be used for a broader set of problems than the original one which was used to gain better understanding but is now understood sufficiently well to be applied elsewhere.

What is perhaps unclear is the true role of abstraction and formal representations. Beynon claims that EM is complementary to formal approaches and that "it is possible to situate a formal representation within a context moulded from experientially-mediated associations" [Beynon, 2011]. Formality comes when stable patterns of interaction and interpretation can be appreciated *universally*. In EM, abstractions and formal representations have been given an auxiliary role which has resulted in such issues being under-explored in the tools, and in the framework. How in practice abstractions, restrictions and interpretations are to be given to an OD-net universally *at the level of a program* is unclear. The LSD notation [Beynon, 1986b] has been developed as a way of accounting for stable patterns of interaction and it was the intention of the ADM tool to *animate* the LSD accounts [Slade, 1990]. However, LSD was never intended to provide a "formal operational semantics" and "an LSD account does not lead directly to an executable model" [Beynon, 1997b]. What it does provide is a step towards abstraction and formality that needs to be capitalised upon far more than it is in current tools.

It is necessary to revisit the importance of the observational context (cf. figure 3.2). As stated by Beynon in [Beynon, 2011], the classifications of "construal", "model" and "program" are blurred and do not involve any real change to the OD-net but are instead a shift in observational context. It was highlighted earlier (cf. §2.2.2) that observational contexts are not well developed in the tools and so this is perhaps another concern to focus on to enable plastic applications. Without an adequate concept of

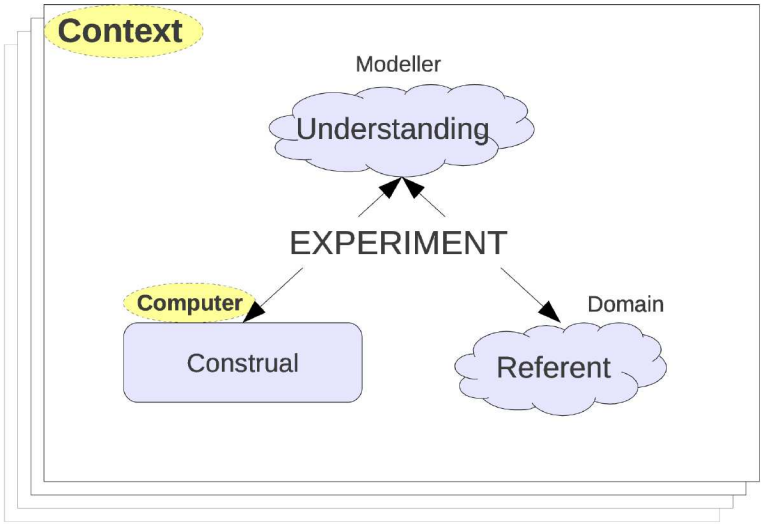

Figure 3.2: A need to improve support for *context* and for *construals* implemented on a computer

observational context it is difficult to see how the high degree of refinement in any of the four dimensions can be achieved. This relates to LSD in that an LSD account describes a particular observational context, one where particular patterns of interaction have been identified. An LSD account should not, however, be the only possible account and should not, as made clear above, have any effect upon the underlying OD-net so that other contexts may exist. Existing implementations of LSD do not obey this, partly because LSD itself encourages the OD-net to be developed with respect to its entities.

It is important to remember that there should be a smooth transition from one end to the other during the refinement process, and that critically such refinement can be reversed. The other end of the refinement process could also be improved. To be truly personal, subjective and provisional it is important that as many restrictions on interaction and interpretation are removed as is possible with a computer implementation, to allow for complete freedom. These improvements along with the ability to smoothly transition involve looking more closely at how a *construal* is supported on a computer (cf. figure 3.2).

3.3 Limitations of EM Tools and Concepts

Whilst EM is attempting to address the gulf between the informal world of experience and the formal world of programs, it has still been unable to fully bridge the gap in practice. The conceptual framework is believed to bridge this gap in principle, however, the tools that currently enable EM cannot realise this. Figure 3.1 shows the gaps between experience (informal) and programs (formal). EM has only been able to achieve a partial migration from construal to plastic application, due perhaps to its focus on the personal and experiential aspects.

To identify common problems a small survey of 20 WEB-EM[3] papers and models was conducted, and combined with my own knowledge/experience of many other projects and comments by previous PhD and MSc students who did not produce WEB-EM papers. A total of 20 relevant technical deficiencies were identified in the *tkeden* tool which relate to usability concerns and the concerns discussed in the previous section. These problems are discussed in this section.

3.3.1 Richness of Observables

In order to support construals and be faithful to the personal, subjective, provisional and specific nature of the software artefact in the early stages, observables need to remain as unrestricted as possible. What this means is that static types for observables are inappropriate and even the concept of type is too restrictive. EDEN is dynamically typed but with a limited range of types: integers, floats, strings and lists. These type restrictions have originated from the C language upon which EDEN is based. The EDEN list type is perhaps the most flexible in that it can be of any size, can contain heterogeneous dynamic types and does not name any of its components (all are accessed by integer index or list operations). These type restrictions do mean that more complex and non-standard kinds of observable (e.g. a shadow or bed) are difficult to represent

[3]Warwick Electronic Bulletin for Empirical Modelling, coursework for an MSc/MEng module at the University of Warwick.

in EDEN without resorting to abusing the list structure.

At the same time as saying that type restrictions are unwanted to better support construals, it needs to be recognised that structures and types do exist and that they are perhaps necessary to support the migration to a plastic application where abstraction and restriction are required. The EM solution is to use definitive notations and make it the job of the agents to observe specific types in a duck-typing[4] fashion, but to leave the underlying OD-net as flat, unstructured and un-typed as possible. Unfortunately to make a model on a computer a certain degree of abstraction is required to approximate experience in binary form. Reconciling all of these issues is the challenge faced when implementing tools for Empirical Modelling and plastic applications.

From the survey and from personal experience there are 9 key problems identified with the *tkeden* approach to the representation of observables on a computer. These problems are split into two categories, names and types:

Naming

A1. Requirement for C syntax observable names There are times where the C syntax restrictions prevent giving an observable the name it really should have.

A2. Observable aliasing problems Having a single flat observable space is hugely problematic since each observable name must be unique. This is particularly problematic when combining models.

A3. Arbitrary choice of naming conventions Each modeller for each model is free to choose a naming convention. Such conventions are needed due to problems A1 and A2 above but are often difficult for others to understand.

Typing

B1. Observables hard to manipulate and search As the EDEN environment is unaware of certain relationships and structure there is little support for copying or

[4] If it looks like a duck, sounds like a duck and acts like a duck then it is a duck.

sophisticated search of observables in the flat space[5].

B2. Not scalable to larger complex models Difficulty with automating the construction of large models means that either tedious manual construction or inflexible automated construction is used.

B3. Primitive types inadequate Lists become unmanageably complex as a way of representing structures that are required.

B4. Ineffective reuse of existing models Components in models are not clearly separable and so it is difficult to reuse only parts of a model. This is also related to problem A2.

B5. Little support for shifting focus and contexts Switching between groups of observables depending on the current situation is difficult[6]. It is also not possible to move to a higher, more abstract focus where the individual observables are no longer important.

B6. No "embedded" interpretation Most of the interpretation of a model remains in the mind of the modeller and so is not easily transferred to others looking at the model[7].

All of these problems can be seen in and illustrated with existing models developed in EDEN. Problems A1, A2 and A3 can all be seen in Beynon's Lines model [Beynon, 1991] where observable naming has been a real difficulty. The examples in listings 3.1 and 3.2 show some of the observable names and definitions found in that model and it is clear that some syntactically restricted naming convention has been used to generate unique observable names for thousands of observables. What this convention is remains

[5]Using the DMT [Wong, 2003, p.181] it is possible to navigate the dependency graph to search for observables.

[6]Achieved by loading in other script files to make and undo bulk changes.

[7]The purpose of LSD is to describe protocols for interaction which gives the intended interpretation [Beynon, 1997c], however this is not used or well supported in EDEN.

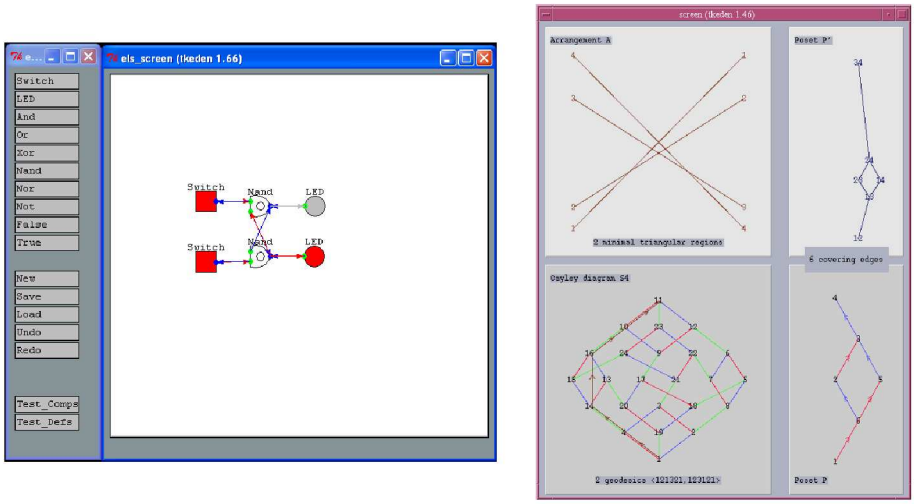

Figure 3.3: Left: Eden Logic Simulator. Right: Lines Model.

a mystery, even to Beynon at times[8], and is an example of problem B6 [Rungrattanaubol, 2002, p.189]. The syntax restrictions of A1 also go against the EUD guideline of making syntax errors hard (cf. 1 in table 2.2). The Lines model is not the only example of these problems as similar problems exist in almost all models of a significant size.

```
l1334
g2434
pbai_v
eqabbai_2
```

Listing 3.1: Lines model observable names

```
upbai_4 is check_int(if_assign(i_eq(pbai[4][4],UNDEF),
        one_int(1), one_int(0)),upbai_4_kind)
```

Listing 3.2: Lines model definition

[8]Although he would claim to be able to recover an understanding through interaction with the model [Rungrattanaubol, 2002, p.189], something which EM tools, such as EDEN, are particularly good at supporting.

To interpret the observables and definitions in 3.1 and 3.2 requires some detailed explanation of the convention used and even then having to manually search through thousands of observables to figure out what the model is doing is obviously extremely challenging. This relates to problems B1, B2 and B3. It should be noted that the definition given in listing 3.2 is actually part of the Eden translation from the ARCA notation so is not directly written by the modeller. The three problems, B1, B2 and B3, also feature prominently in the Timetable model [Beynon et al., 2000b] as well as many student coursework projects such as the ELS (Eden Logic Simulator) [Lee, 2007]. Both of these models contain large numbers of similar *components*, such as cells in the timetable or logic components in a circuit, and have attempted to automate the construction of these components using agents. In each case a different approach has been taken and they have proven to be very inflexible. Not only are they inflexible and complex but they also automatically generate thousands of observables following a particular convention and exacerbate the problems of A1, A2 and A3. The fact that they had to use different approaches for essentially the same problem is also an example of problem B4.

The *personal* focus of Empirical Modelling is partly why problem A3, that of personal naming conventions, has come about. Such conventions do not help in communicating an artefact to others, i.e. making an artefact *public*. The same *personal* focus leads to B1 due to an assumption that the individual modeller will have some mental model so knows what observables there are. Obviously such mental models remain in the mind and are not transferred to others when the artefact is. To some extent this relates to the EUD guideline of providing incremental disclosure (cf. 8 in table 2.2). Some solutions have been proposed by Rungrattanaubol, for example using dynamic annotations of scripts (described by other scripts) which helps to identify the meaning of observables (also helps with B6) [Rungrattanaubol, 2002, chapter 7].

Similarly the lack of structure and support for types (B3,B5 and B6) is a result of focusing on the *provisional* nature of a model rather than an *assured* artefact. It is open

to fluid change but cannot be solidified into specific identifiable entities. Problem B4 is an example of where the focus on *specific* (concrete) has led to models that cannot be made *generic* enough for reuse in other models. This has proven to be a significant limiting factor for EM. Many models end up reinventing the wheel with their code as modellers are unable to adapt models to their needs. Most models can only act as inspiration rather than a real resource to be used and a considerable number of students have made this point in the coursework reports. It is clear that this problem has its roots in many of the other problems which help to make such reuse difficult. Often it is this lack of modularity combined with B2 that students use as an argument for using Object-Oriented languages instead of EM.

The final point above highlights another aspect of the refinement process, that of non-linearity. Parts, or components, of a model may be refined at different rates with certain components becoming public, assured and generic whilst the rest of the model using those components is still personal, provisional and specific. This indicates a need to not only support plastic applications as a whole but plastic components of any size that may be combined in a hierarchical fashion to varying degrees of refinement (cf. B2 and B5). Again, looking back to EUD guidelines this problem of components matches with a need to support decomposable test units (cf. 7 in table 2.2).

The lack of support for context shifting (cf. B5) is often not mentioned explicitly in student models but does have an impact upon the choice of model constructed by students. For example, Beynon has attempted to develop models that require the kind of context switching abilities of B5. One such model involves a need to specify the car of a mother-in-law's husband where the car, the mother-in-law and husband could change. It is difficult to achieve this kind of switching without resorting to the crass approach of loading script files to make bulk changes. In the end these models are abandoned or adapted to avoid such problems.

3.3.2 Analogue and Process Dependencies

The concept of dependency is well developed in the EM conceptual framework. Since the focus in EM is on developing a rich model of state with which agents may interact to experiment, these dependency relationship concepts are passive in that they respond to change to maintain these relationships but do not themselves cause change. All change comes from agent interaction with this model of present state. Figure 2.4 highlights how dependency is only used to support the model of state and plays no direct part in describing processes and behaviours. Consequently the OD-net only gives a static representation of the current state of a model with the *experientially-mediated* associations [Beynon, 2011] being *latent* and only coming into action to deal with agent initiated change.

What is being missed by this is the potential for using dependency to describe processes[9], where there are *experientially-mediated* associations [Beynon, 2011] which are not about dealing indivisibly with change but are across time and so make change visible. Indivisibility is about being coherent in the presence of change so it is still important that there be coherence between and within these states[10]. The purpose of dependency in the EM conceptual framework is to enable experimentation (by maintaining the integrity of state) to take place in order to gain understanding and help develop an artefact. To this end EM has focused on providing a rich model of state that supports such experimentation (the OD-net). Agents, on the other hand, are there to do the experimenting by interacting and observing the artefact. What if, however, the artefact and experiment is about a process or involves associations across time? Often the most difficult systems to understand are dynamical systems and so computer support for developing dynamical artefacts that can be experimented with is important. Faraday's experiments to develop the electric motor involved a dynamic process and he used what Gooding calls *dimensional enhancement* when a "causal explanation is

[9] Processes should be included as *particular* things which can be observed [Smith, 1996, p.118].

[10] Changes in the current state are still propagated indivisibly and the change from one discrete state to the next is also indivisible to an observer.

sought" [Gooding, 2001]. Dimensional enhancement is the addition of a third and possibly fourth (time) dimension to a diagram or actual model, something which is done only when the simpler *state-as-experienced* has been grasped (through the use of *dimensional reduction*). The artefact itself is given a temporal component that can then be experimented with, making it a dynamical artefact. Today in chemistry, physics, biology and other fields it is becoming increasingly important to recognise the dynamic nature of the world around us.

The concept of coherent observable current state still plays a vital role even though the current state may be unstable. The view that there ever exists a stable concept of current state relates to the now debunked notion of natural systems being self-regulating and stable, that there is, for example, a "balance of nature". The role of dependency, it seems, is to rebalance the artefact into a stable state after each change has occurred. However, it is widely acknowledged by ecologists and others that "the natural environment... is in a constant state of flux" [Jacobs, 2007], and to abstract to a static conception of state is to ignore something vitally important. Brian Cantwell Smith also identifies a necessary move "away from treating the world in terms of static entities instantiating properties and standing in relation, and towards a view that is much more intrinsically dynamic and active" [Smith, 1996, p.36]. With Empirical Modelling's current focus on state such dynamical processes require the use of agents which can be argued goes beyond their remit of experimental interaction and observation. The artefact itself, the OD-net, should embody the dynamism found in the referent and become a *dynamical artefact*. Both ways of construing the world are valid, with agents or dynamical processes, and Empirical Modelling should allow for either approach to be taken depending upon the personal preference of the modeller.

Whilst the EDEN implementation of dependency has proven successful and is faithful to the EM concept of dependency, it does suffer from numerous problems that have come to light through years of modelling activity, and which draws attention to the need for a different conceptualisation of the role and nature of dependency. There

are 6 key problems identified by the survey and personal experience. These problems fall into two categories: conceptual (C) and technical (D):

Conceptual

C1. Feedback not supported Dynamical dependencies cannot be specified which prevents 2-way relationships and feedback situations from being easily described.

C2. Animation requires use of clock ticks All animation requires either procedural actions or dependence upon a clock observable.

C3. Unable to observe events Sometimes a change needs to be observed rather than just the value itself and this cannot be done with definitions.

Technical

D1. Not monitoring all dependencies Any observables used inside functions are not added as dependencies and so do not trigger updates.

D2. Contains procedural elements These elements allow for side-effects which causes concurrency and conceptual issues.

D3. Dynamic dependencies not maintained If the dependencies a definition describes change then the system does not track those new dependencies.

The first three conceptual problems are the ones which do need addressing, whereas the last three are only deficiencies in the implementation rather than a conceptual problem. Examples of C1, C2 and sometimes C3 have been encountered in many student projects. Going back to the digital logic simulator referred to earlier [Lee, 2007], the author of the model commented that he was unable to use dependency for the wires connecting components together because sometimes these connections were cyclic in nature. To resolve this he needed to resort to using the EDEN clock mechanism to tick through discrete instants and also maintain a concept of previous state. In this way the

next state was continually updated based upon previous state but all this was achieved using agency and so lost most of the benefits of using EDEN, lost the benefits of dependency. What was being attempted was the development of a dynamical artefact where a simple and static conception of state is inadequate. Another example is a model of the Water cycle and infrastructure in Singapore [Beng, 2006]. The water cycle is, as the name would suggest, cyclic in nature and again the author of the model commented on how dependency could be used everywhere except in one, randomly chosen, point because the cycle needed to be broken. Other such cyclic models include: Agent Based Bridges [UNK, 2005] and a Neural Networks Notation [Hammond, 2006]. Beynon has also played with this problem in the form of two blocks connected by a string [Beynon, 1989]. Pulling one of the blocks will move the other if the string is taught and vice-versa. Such dependency is 2-way and hence not possible in EDEN as it is cyclic.

Problem C1 seems to be the bigger issue, however, C2 is more common. Typically models need to be animated in some way and currently this involves some clock agent incrementing a tick observable that all other definitions depend upon. Whilst this works in many cases there is a clear need for a better conception of time. Animation is an example of an instability in current state and represents a form of dynamical system. Looking back to Gooding's work on Faraday, he describes how construals and physical artefacts were built which were changing without the intervention of the experimenter and how Faraday was observing these processes and influencing them through interactions [Gooding, 1990, p.149]. Such "animation" is not conceptually attributable to an agent, or at least it is not especially meaningful to think of it as such.

The final conceptual problem, C3, does not occur often in student projects but has been noted by Beynon in a train model where a conductor whistles. Observables are needed to know if the conductor is whistling or has whistled and this involves the observation of an event. Another example would perhaps be the observation of a button click by watching for both a press and release event from the mouse. Such observations would need a concept of previous state, however EDEN and the OD-net concept only

allow for the present state to exist in an artefact.

With regards to plastic applications it is important to allow for dynamical systems, but also by moving away from agency towards dependency it is hopefully easier to generate a more *assured* and formal description. Dependency is functional in character, as opposed to the imperative nature of agency. As a consequence it reduces the difficulties of orchestrating side-effects (cf. Coherence in §2.1.3) and can draw on the mathematics of functions (and data-flow systems [Wadge and Ashcroft, 1985]). EDEN as it stands, and EM in general, requires an excessive use of agency which is detrimental to this goal of becoming *assured*[11].

From a technical point of view problems D1 and D2 can be resolved by following particular conventions or using special features within EDEN. For D1 simply pass all required dependencies as parameters to the function or if this is not possible then use a special feature that allows dependencies to be manually added. Ultimately this should not be necessary. For D2 the modeller can avoid using side-effects inside functions used by definitions. More difficult to resolve but also quite uncommon is D3. Ashley Ward has discussed this in his thesis [Ward, 2004] and at one time decided to deprecate the ability to dynamically change the dependencies within a definition, however, this broke some models. The problem lies in the ability to dynamically generate observable names from strings inside a definition. If these observable names change for some reason then EDEN does not keep track of the new dependencies for those new observables.

3.3.3 Notations and Agents

In the EM framework, agents interact with, observe and construct the artefact. The artefact is represented as an unstructured OD-net. To help with this, task-specific *definitive notations* are used to provide appropriate structures and operations for agents to use with the OD-net. Task-specific notations[12] are nothing new and have been

[11] Especially in concurrent systems where concurrent dependency maintenance is a far easier problem than concurrent agency.

[12] Originally the term *notation* was used in EM to mean an incomplete (non-Turing complete) language. There is no need for a specialised language to be complete. The term should also include visual notations

explored a great deal by the EUD community. The benefits are obvious, the end-user may work with a notation that relates to the task at hand and their domain of interest, rather than being more generic but complex[13]. One of the EUD guidelines is to use domain-oriented languages where possible (cf. item 4 in table 2.2). However, there is an important difference in motives between the EUD use of task-specific languages and the use of definitive notations: EM is trying to avoid being grounded in a foundational way in any particular representation. EUD, on the other hand, seems only concerned with usability rather than foundational issues. Concerns about foundations are discussed in depth by Smith [Smith, 1996, p.81]. No single approach to representation can do justice to the extraordinary diversity of things (in experience) that need to be represented.

It is, however, well acknowledged that task-specific languages suffer from a collection of problems: the difficulty of creating many different languages, inconsistencies between languages and knowing what any particular language should and should not contain [Nardi, 1993, p.50]. Each of these problems can be found in the survey of EDEN. The issues identified by Nardi are further exacerbated by the nature of the EM refinement process. Initially all interaction needs to remain free from unwanted restrictions that specific notations will tend to cause. To be free and yet use definitive notations may require unrealistic notational flexibility. At the other end of the process these notations may need to be refined to become far more specific. Nardi's third problem of deciding exactly what needs to be in a particular notation becomes a part of the refinement process of an EM artefact and so may be specific to a particular artefact, meaning that many different unique notations are required which is "expensive". Without this notational flexibility the artefact will be unduly constrained by the existing notations.

A partial (and perhaps unsatisfactory) solution to this, as suggested in Repenning and Ioannidou's guidelines (cf. §2.1.4), is to have a meta-domain language to connect

as well as textual.

[13]Turing tar-pit: "Beware of the Turing tar-pit in which everything is possible but nothing of interest is easy" [Perlis, 1982].

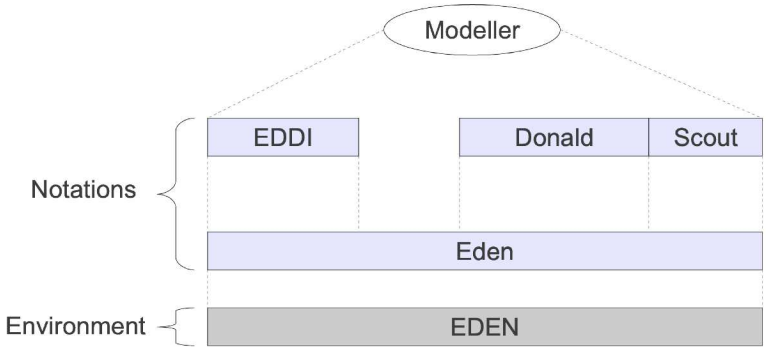

Figure 3.4: Notation layering in EDEN

the other languages together and allow for generic situations where a specific language does not exist. In EDEN this is the Eden language. The Eden language is in fact a generic representation of the OD-net so it is the OD-net that fundamentally provides the meta-domain link between notations. These problems and others have been identified in EDEN by the survey:

Notations

E1. Inter-notation communication difficult All notations translate in a unique way to the underlying Eden notation and so users must understand all these conventions. Connections also have to take place in the domain neutral Eden notation.

E2. Inconsistencies between different notations Major syntactic differences exist between each notation and this can be confusing, especially if large numbers of domain specific notations are eventually supported.

E3. Custom notations for each different domain It is not practical to have a custom notation for each domain of interest and so having to fit with one of the existing specialised notations or the entirely flat world of Eden is a major cause of frustration.

E4. Focus on textual notations only The notation mechanism involves translating

DoNaLD	Eden Definition	Eden Value

within desk {
line E
point NE, SE
E = [NE, SE]
}
	_desk_E is
	line(_desk_NE,
	_desk_SE);
		['L',['C',365,465],
		['C',365,115]]

Table 3.1: Illustrating DoNaLD to Eden translation

to the Eden notation and so a textual approach is encouraged. This discards the possibility of a visual notation and is less direct and interactive than desired.

E5. Notations only work for modeller and not other agents All agents have to be written in the domain neutral Eden notation and cannot take advantage of types, structures and operators available in other notations. Related to problem E1.

The first problem (E1) is best shown by an example of how definitive notations translate down to the underlying Eden notation. Table 3.1 shows how a simple DoNaLD definition of a line is converted into Eden and what the resulting value of the observable is. This example shows how the list type in Eden has to be used to represent the concept of a line and how this could be difficult for end-users to interpret. The problem gets far worse for more complex types such as Scout windows (cf. listing 3.3), largely because these representations are not *self-describing*[14] once translated (cf. listing 3.4).

With many of the existing definitive notations, including EDDI, Sasami, DoNaLD and Scout, there are object-like concepts that require translation into Eden list structures. The prevalence of such object concepts would indicate a need for the underlying and integrating notation to better support such concepts. In other words, the OD-net

[14]A term often used with semi-structured representations, such as XML, where structural descriptions are included with the content because generic descriptions do not exist.

```
%scout
window A1 = {
    type: TEXT
    frame: ([{A1_X1, A1_Y1}, {A1_X1+gridsquare_width.c,
                A1_Y1+gridsquare_height.r}])
    font: A1_font
    bgcolor: A1_bgcolour
    fgcolor: A1_fgcolour
    bdcolor: A1_bdcolour
    border: A1_border
    relief: A1_relief
    alignment: CENTRE
    sensitive: ON+ENTER+LEAVE
    string: a1
};
```

Listing 3.3: SCOUT window example

```
A1 is [0, [formbox([A1_X1,A1_Y1],[A1_X1 +
    column(gridsquare_width), A1_Y1 +
    row(gridsquare_height)])], a1,[0,0,100,100], "pict1",
    DFxmin, DFymin, DFxmax, DFymax, A1_bgcolour, A1_fgcolour,
    A1_border, 3 ,1.0 + 4.0 + 8.0, A1_bdcolour, A1_font,
    A1_relief, "A1"];
```

Listing 3.4: SCOUT window translated to Eden

needs to support these kinds of structures to make inter-notation communication a little less daunting.

The second most contentious issue is E4 where definitive scripts are far too static to deal with the dynamic nature of artefacts. This is strongly connected with the previous argument in §3.3.2 and the motivations behind the creation of Subtext (cf. §2.1.3) which is trying to move away from paper-centric approaches to programming. If dynamical systems are to be supported then the foundational role of definitive scripts is no longer practical since a fixed textual representation of state cannot do justice to the dynamic nature of the artefact[15]. Even without the introduction of dynamic concepts a script is not as interactive and observable as the system actually is internally, and could also be in violation of the EUD principle of directness since scripts rely on indirect textual manipulation of an entity.

Problem E3 has already been discussed and is a well known problem. E5 illustrates a technical deficiency in the way in which EDEN has chosen to implement automated agency. Instead of allowing agents to act above and outside of the artefact and its corresponding notations, EDEN has implemented agents directly in the "foundational" language Eden which means they are forced to interact and observe the OD-net directly with no alternative views or interpretations being possible[16]. Agency needs to be fundamentally removed from the artefact itself and allowed to operate at an entirely different level. Automated agents need to be given the same role as the modeller. There is much more to say on the nature of agents and how they should be implemented, but for this thesis the focus will stay with the OD-net. The observation made by E5, with regards to agency, is all that will be said since it does impact upon the nature of the OD-net.

[15] This is not to say that task-specific notations are not important, they still have a role to play for agent refinement.

[16] It is possible for Eden agents to use an *execute* command to run code written in other notations, however, this has proven to be exceptionally complex to write and subsequently understand as it involves manipulating strings of scripts.

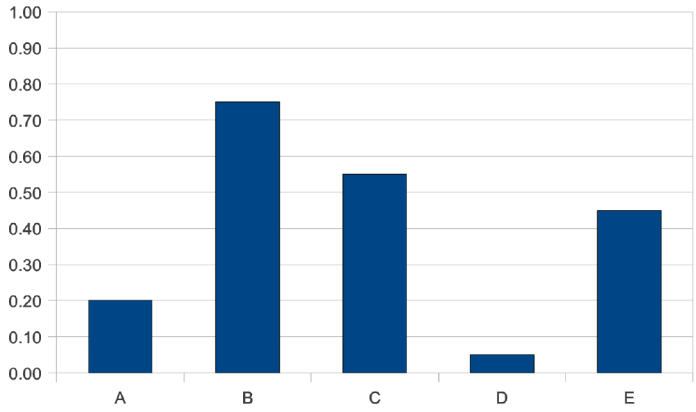

Figure 3.5: Fraction of student models that explicitly or implicitly mentioned problems in each category

3.3.4 Summary of Survey

This section has identified a total of 20 problems with both EM concepts and their current tool EDEN. The problems have been listed together in table 3.2. These problems form the basis for the questions asked by this work and in the next section of this chapter solutions are proposed, followed by specific approaches to be taken by the rest of this thesis.

As a part of the analysis involved in discovering these problems, 20 Web-EM coursework papers were reviewed to see how often each problem came up. The graph in figure 3.5 shows the results grouped into the categories of problems identified. Figure 3.6 breaks this down further into the individual problems. This analysis is limited in that most students are directed to explore particular kinds of models that avoid certain technical issues and often they do not report problems they had in the written paper. Also, these are individual and personal projects so do not identify issues with scaling up, which is where my additional EM experience and that of others has also played a part in identifying the problems. Despite this the analysis clearly shows what the most common

Table 3.2: 20 problems of EM and its tools

A1	Requirement for C syntax observable names
A2	Observable aliasing problems
A3	Arbitrary choice of naming conventions
B1	Observables hard to manipulate and search
B2	Not scalable to larger complex models
B3	Primitive types inadequate
B4	Ineffective reuse of existing models
B5	No support for shifting focus and contexts
B6	No embedded interpretation
C1	Feedback not supported
C2	Animation requires use of clock ticks
C3	Unable to observe events
D1	Not monitoring all dependencies
D2	Contains procedural elements
D3	Dynamic dependencies not maintained
E1	Inter-notation communication difficult
E2	Inconsistencies between different notations
E3	Custom notations for each different domain
E4	Focus on textual notations only
E5	Notations only work for modeller and not other agents

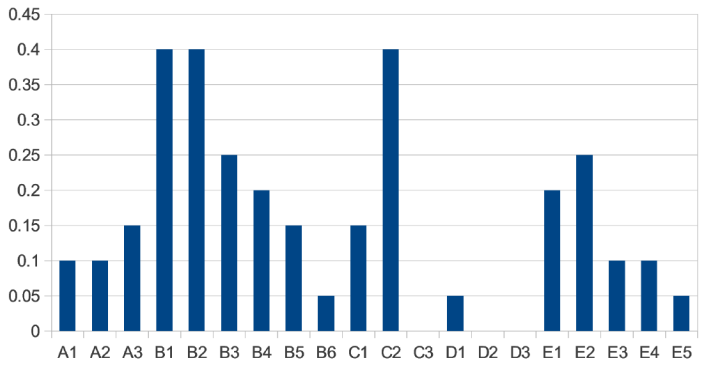

Figure 3.6: Fraction of student models that explicitly or implicitly mentioned specific problems

problems are (of those included in the non-exhaustive list in table 3.2). It seems that type issues such as the need for objects and some need for active forms of dependency are most common and so will be focused upon.

3.4 Looking for Solutions

For most of the issues identified with EDEN and the EM concepts there are suggestions of solutions to be found in other research areas and industry. Chapter 2 introduced End-User Development and three of its tools, along with other XML related technologies. These approaches are to be explored here as possible ways of resolving many of the technical and perhaps conceptual problems with EDEN and EM. For now the conceptual consequences of using these practical solutions will mostly be put aside to be revisited later (chapters 6 and 7), but since one of the founding objectives of EM is to develop "theoretical frameworks that do justice to [the] practice" [Smith, 1987] there is a need to understand how these EUD and industry technologies fit within Empirical Modelling and how they can be appropriately and beneficially moulded into its framework. Due to needing to constrain this work the issues relating to agency and the refinement of

their interactions will not be focussed upon at present, leaving the ideas behind the LSD notation as an appropriate solution for the time being. The focus instead will be on the OD-net itself and how to better support the notion of *context* and *construal* on a computer (cf. figure 3.2) with regards to the ability to transition to the more refined plastic applications level.

3.4.1 Richer Types and Semi-Structure

Many of the problems identified in §3.3.1 on naming and typing of observables are related to the motivations behind the development of structured and semi-structured representations [Buneman, 1997], including object-oriented languages. These problems highlight the need for giving groups of observables a collective identity and for developing structures and categories to enable interpretations, manipulations, communication and reasoning. The motivations for this do not need to be stated as they are well known from object-oriented programming, mathematics, the arts and philosophy, in other words practice shows it is important. What is problematic is the lack on an ontological theory and that "traditional ontological categories ... are both too brittle and too restrictive" [Smith, 1996, p.45] as well as being overly committing. As Brian Cantwell Smith puts it in "On The Origin of Objects", we want:

> *"notions of objects that are fluid, dynamic, negotiated, ambiguous and context-dependent ... rather than the black-and-white models inherited from logic and model-theory."* [Smith, 1996, p.46]

Plastic applications and Empirical Modelling are striving to allow for exceptional flexibility which requires exceptional representational flexibility. It is argued, correctly, in EM that preconceiving structures and classifications for observables is not possible in an exploratory, experimental and experiential process that starts life as a personal, subjective, provisional and specific construal. To be provisional and subjective means that fluid representations are vital. To this end the issue of structure has largely been

avoided, with EDEN's flat observable space being an example of this. Instead *definitive notations* are used to provide task-specific views which includes the grouping and classification of observables that relate to the task at hand but that in principle is intended as only one view of the otherwise unstructured OD-net.

Unfortunately it is clear from §3.3.1 and §3.3.3 that the use of notations to provide representations is inadequate and that richer but fluid representations need to exist within the OD-net itself[17]. The ontological issue cannot be ignored in the foundations of Empirical Modelling, especially if the transition to a program is being sought.

Interestingly there has been some suggestion of enriching the OD-net with object-oriented characteristics before. Edward Yung, the original developer of EDEN, suggested in the future research section of his Masters thesis on EDEN that object-oriented concepts should be considered to allow for structured data types and inheritance [Yung, 1990, p.101]. Similarly, Allan Wong proposed ways of organising definitions into *containers* and proceeded to develop new tools[18] and interfaces based upon this idea [Wong, 2003, p.167]. What these comments and attempts fail to fully appreciate is the inflexibility of traditional OO approaches and how significant and fundamental the issue of representation really is.

The answer may lie in the ideas of semi-structured data, as already indicated, and those of prototype-based languages. In both of these approaches a degree of flexibility is possible and accepted as necessary. Subtext (cf. §2.1.3) takes advantage of a tree structure, as does XML. However, the most unrestricted form of structure is perhaps the edge-labelled directed graph which can be related to the most basic algebraic structural concept, that of a *magma* [Rosenfeld, 1968, p.90][19]. Without any additional object-oriented concepts such as message passing, or any additional complexity as present in

[17] Especially since some inflexible representations in the form of types are already in the OD-net causing difficulties.

[18] WING (WINdowing and Graphics tool) [Wong, 1998] and EME (Empirical Modelling Environment) [Wong, 2001] which formed the basis for the DMT [Wong, 2003, p.181].

[19] A magma may also be known as a *groupoid* and contains a single set and a single binary operation closed over that set.

XML descriptions, the idea of using a pure graph to provide structure is perhaps a viable approach for EM and plastic applications as a way of structuring state. The idea being to semi-structure the OD-net in an attempt to resolve the issues identified previously without becoming too inflexible in the presence of change. The concept of *schema* could then be taken from XML and research on semi-structure as a means of developing refinements on interpretations for agents as the artefact moves from construal to a plastic application. Schema may be a new way of interpreting *definitive-notations*. Capability-based security[20] may also be applied to the graph as a way of restricting agent actions and observations through privileges.

The idea of a semi-structured OD-net is rich with possibilities. Some possible consequences are that observables could be manipulated using cloning of sub-graphs, that richer hierarchical types can be supported, that components can be isolated and refined at different rates. There are even greater consequences that will become apparent through this thesis[21]. The main argument against this is that it may still not be fluid and dynamic enough in that even with a semi-structured approach there is some need to commit fairly early to particular representations. Albeit considerably later than fully structured approaches and with a greater possibility for subsequent re-factoring if it proved to be inappropriate. How beneficial a semi-structured OD-net would be, and how problematic, is unknown.

3.4.2 Taking Advantage of Dependency

A need to support dynamical artefacts was identified in §3.3.2. The basic problem is that cycles over time do exist and that some processes are not appropriately described using agents. There is a clear need to consider artefacts which are not simply static entities. A fundamental aspect of EM is to enable experimental interaction and observation of an

[20] Capability-based security is an alternative to Access-Control-Lists in operating systems [Levy, 1984; Miller et al., 2003]. It involves particular users and processes (or agents in our case) being given tokens to act not only as object identifiers (or node references in a graph) but also to say what permissions they have. This is a decentralised approach to security that has been explored in the CapROS operating system (successor to EROS).

[21] Computation as graph navigation being the main one.

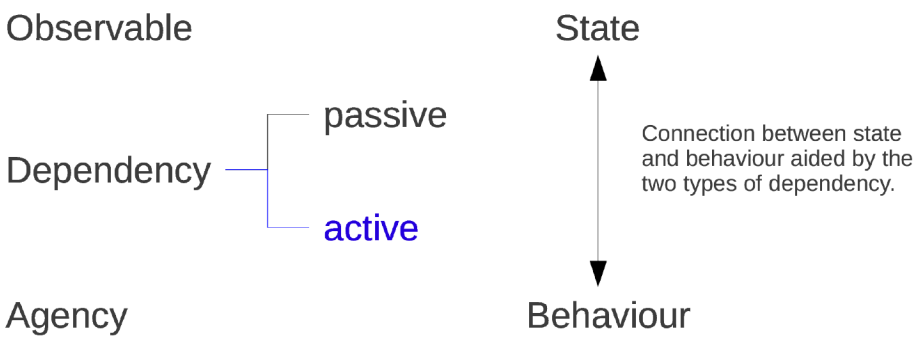

Figure 3.7: ODA with active dependency.

artefact in order to evolve it. The goal of plastic applications is to evolve it sufficiently far that it becomes like a program. One way to enable experimentation with an artefact is to use explicit dependency to maintain a coherent state in the presence of change, this has been the EM approach to date. So to enable process-like artefacts and animation, is it possible to also use dependency? Can dependency be used to not only support rich models of state but to also support processes (cf. figure 3.7)?

The advantage of the EM notion of dependency is its indivisible nature that prevents observation and interaction by agents from occurring before any previous change has been fully propagated. This enables agents to observe only coherent state rather than incoherent state that is still being computed[22]. This characteristic of indivisibility must be maintained with dynamical artefacts. To retain this indivisibility there must remain a notion of coherent current state, somewhat like a snapshot that agents may observe. This snapshot notion is in fact well aligned to scientific experiment where the experiments typically take such snapshots at discrete points in time for later analysis[23]. How frequently such snapshots can be taken depends on the complexity and sophistication of the experimental apparatus, which in the case of a computer artefact is very frequently indeed. So indivisibility must exist within a snapshot and between snapshots.

[22] In traditional imperative programs it is difficult to identify such globally coherent state so meaningful observation and experiment are all but impossible.

[23] As should be obvious, experiments are often left unassisted between such snapshots which is why a notion of unassisted process is needed for EM artefacts.

Dynamical systems are described by evolution functions which give a trajectory through time to a particular property [Weisstein]. These systems can be discrete and if multiple properties are involved they can be synchronised, which gives a degree of coherence. Such evolution functions could in fact be described as dependency definitions by allowing for dependencies on previous values of observables. If appropriately implemented this would enable indivisibility between states and, with a special case where a definition only depends upon present values, within individual snapshots of current state as well. The result is a discrete set of snapshots that record the history of an artefact and, through active-dependencies, describe a trajectory into the future.

Back in chapter 2 the EUD environment Forms/3 was introduced (cf. §2.1.3). Forms/3 implemented a similar concept for spreadsheet style formulas and cells where each cell has a temporal vector to store its history. The formulas can then refer to previous values of a cell and this enables animation. The concept works well and relates well to the proposal here. The challenge then is to integrate this notion of active-dependency with the now semi-structured OD-net to provide a dynamical artefact that can be refined in an EM manner. The additional benefits of this regarding refinement to a plastic application have already been covered in §3.3.2.

3.5 A New Tool?

What the proposed solutions in §3.4 describe is a dynamical[24] semi-structured OD-net that supports the Empirical Modelling process. There is no existing interactive tool that supports this concept and the existing EM tool EDEN cannot be adapted this radically (cf. §7.1). Therefore, a new prototype will need to be constructed to test out the ideas. There are four key questions to be answered in the following chapters of this thesis:

1. Is it possible to implement a dynamical semi-structured OD-net as an EM tool?

[24] Dynamical is used as opposed to dynamic because it indicates that the system changes itself rather than that it is capable of being changed.

2. Does the idea actually resolve the problems identified with EM for plastic applications?

3. What is now possible and does this help move EM towards its notion of program?

4. Are there any conceptual consequences of making these changes and how can they be dealt with?

What has not been mentioned, and has been deliberately left out, is the role and nature of agency. It is not the concern of this prototype to resolve the issues with agency and so it is free to choose any implementation. One thing EM tools have suffered from is the lack of modularity and lack of integration with existing technologies. Dependency-injection, which traditionally uses XML, would seem to fit well with the new semi-structured approach and so could make use of the semi-structured OD-net as *glue* for external components, which can be conceived of as agents in the new tool (Cadence). In this way existing technologies of almost any kind could be flexibly linked with the OD-net. This is perhaps one, less radical, means of bringing EM and plastic application principles to existing programs and gives a practical framework with which to enable end-user development of applications using EM principles. Although developing applications in a radically EM and plastic way from the start is the aim, this allows for a more incremental transition away from traditional approaches which may enable EM ideas to be accepted by the software industry.