Chapter 8

Conclusions and Further Work

At the beginning of this thesis the question of "*how to bring extreme plasticity to software in a way suitable for everyday users?*" was asked and framed with reference to using the Empirical Modelling framework as a starting point. A more specific question asked how EM concepts and tools could be improved to support the notion of *plastic applications*. It was argued that EM concepts and tools were not at that time capable of scaling up to supporting plastic applications. The title of this thesis states the problem concisely, EM is a framework for modelling construals (on a computer) but it does not give adequate support for migrating from informal construals to formalisable programs without a translation step. So the goal of the work was to see if enhancing support for both construals and programs in a single environment allowed for such a migration. This migration is then an approach to enabling extreme plasticity in software.

First, in chapter 2, a brief summary of EM was given along with existing End-User Development research which is perhaps the closest research area in attempting to support end-user developed plastic applications. The notion of *refinement* from experience to program was introduced in chapter 2, along with various *dimensions* that have guided this work. The principles of EUD identified here should also be followed in the development of any new EUD environment. Chapter 3 then identified specific problems with the way current EM tools had been implemented with regards to supporting

programs and scaling up to realistic applications, but also with regards to supporting construals. From this a few specific suggestions for improvements were given which would ameliorate or resolve the identified issues. These suggestions were based upon existing technologies that had proven successful for traditional software in achieving increased flexibility. They were: 1) to semi-structure the observable dependency network and 2) to allow for dynamical dependencies. Following from these suggested improvements a new tool (Cadence) was developed which aimed to maintain the EUD principles and stick with the ODA framework of EM whilst implementing the improvements. Many examples were then given to show how Cadence was still able to support Empirical Modelling but also how the new concepts could radically improve upon existing EM models enabled by EDEN.

Having developed Cadence the thesis then goes on to discuss a new framework based upon an idealisation of the prototype tool. The framework gives an account of Cadence that is both informal and formal in order to provide both an informal and formal semantics. Such an account is needed to show how Cadence can support both construals and programs. A discussion is then given of how Cadence does improve support for both construals and programs in comparison to EDEN. Finally, chapter 7 of the thesis goes back to Empirical Modelling to demonstrate that Cadence concepts need to be fundamental in any EM tool in order to take full advantage of them. It also highlights further limitations of the Cadence implementation of chapter 4. Student coursework models are then used to show that Cadence does support the migration from construal to program, much more so than existing tools.

There are many limitations with this work, the primary one being the scale and complexity of the resulting plastic applications. However, the aim of the thesis was not to show large scale and complex applications but to show a way of supporting end-user developed applications that start life as construals and migrate to being considered as programs. The aim has been to improve EM tool support with this objective in mind. As Beynon states at the end of Beynon [2011]: "[Imagine] what might be achieved by

investing as much effort in developing effective tools for developing construals as has been dedicated to tools for developing programs". If a true Cadence environment were to be implemented, as an operating system, then it could radically alter the way in which computers are used by the everyday user.

8.1 Contributions

The work contained within this thesis has made several contributions to both Empirical Modelling research and to the broader computer science community. Specific contributions are summarised below:

- The development of a new tool for Empirical Modelling and programming. The tool has already proven useful in the teaching of Empirical Modelling and has much potential to be improved to the point where it could become the primary EM tool. Outside of EM the tool demonstrates how EM concepts can be applied to more traditional problems in managing software, where dependency can be used as *glue* between components, as shown in the Stargate model.

- Enhancing the EM conceptual framework by removing the focus from EDEN. Previously the EDEN implementation of EM has had an unjustified hidden impact upon the conceptual framework, particularly relating to the nature of observables and dependency. Whilst the ODA concepts themselves remain the same, the interpretations of them in EDEN terms have led to much confusion. For example: the role of functions, notations, time and structures (moding issues). The discussion of the Cadence framework in chapter 6 has highlighted these issues by reinterpreting the EM framework in Cadence terms which reveals the original bias.

- Improved support for construals on computers. The addition of structural relations has allowed for the removal of types and specific algebras from the underlying representation of construals. As a consequence of this work it is now possible to

represent richer and more varied kinds of construal than were previously possible with EDEN. The inflexibility of definitive notations has been identified as a problem and somewhat resolved through the Cadence framework.

- Demonstrating that a transition from informal modelling of construals to formalisable programs is in principle possible using the EM framework, without a translation step. This has been illustrated with various models developed in Cadence both by the author of this thesis and other students.

- Empirical Modelling has in the past been somewhat distanced from more traditional software development processes and technologies. The discussion of EM given in this thesis, along with the updating of its tools using recent technologies helps to relate EM to the traditional software world. In doing this the community at large can better understand the purpose and use of EM thinking which has previously been rather difficult to get across.

- The ideas initially explored by Subtext and Forms/3 have been taken further in this thesis. Whilst Subtext was a practical attempt at becoming less paper-centric by programming a tree structure that contains what we term dependency, Cadence has looked more deeply at the conceptual issues by linking with Empirical Modelling and also at scaling up the use of similar structures and dependency. The thesis has further demonstrated the potential of the Subtext concept through far richer examples, tools and concepts. Similarly, Cadence also contains the notion of temporal formula found in Forms/3 which has been used extensively in relatively complex models.

8.2 Further Work

This thesis introduces new tools and concepts which, as has been made clear throughout the thesis, require much further work. A brief summary of the major areas of further work already highlighted is given here. In some cases considerable work has already been

done by the author in these areas that has not been elaborated upon in the thesis. In others the need for further work reflects time and resource constraints.

8.2.1 Larger Project and Complex Applications

All the examples given within this thesis have been relatively small scale applications. Whilst the scale of the applications is still considerably larger than that supported by EDEN, it still falls far short of many real world applications. To a large extent the reason for this is the prototype nature of the Cadence tool developed in this thesis which is too unstable and lacking in features to scale appropriately. These issues are discussed as other forms of further work, but there will be a need to see just how complex and large applications within a Cadence-like environment can be.

8.2.2 Interactive Development Environment

In §6.1.1 a need to move away from using DASM scripts was identified, with the suggestion being to take a more visual approach. Taking a visual approach to manipulating the Cadence OD-net matches well with the EUD visual programming attempts and the need for *directness*. The existing interface was one area that was criticised by students using Cadence with some of them suggesting possible improvements.

The interface currently includes a tree representation of the OD-net which is somewhat inadequate for representing a graph. Ideas have been explored for generic ways of representing and exploring objects besides the use of a tree. One other consideration is that any particular structure can be interpreted in many ways and so it should be possible to interact with and visualise a structure differently through applying different mediator agents. The direct manipulation of Forms/3 is one approach. The style of programming in Subtext is another (cf. §2.1.3). Recent work by Abi-Antoun et al. on visualising runtime object structure is also particularly relevant as a way of visualising the OD-net [Abi-Antoun and Selitsky, 2010]. Self with its Morphic interface can also provide inspiration for ways of developing an OD-net.

8.2.3 Collaboration and Distribution

A vision for Cadence is for the OD-net to be distributed globally so that dependencies and relationships could exist between observables all over the world. An early version of this vision involved being able to share hardware devices across machines in a transparent manner. For example, the mouse on one machine could by dependency be linked to the mouse on another to allow for remote control. Such connections could be set up with ease. There has been a suggestion of global distribution with EM previously in connection with F-Rep shape modelling [Cartwright et al., 2005], something that may well benefit from using Cadence.

Network distribution was explored in §5.3. To scale up and work efficiently it will be necessary to develop caching mechanisms, scheduling algorithms, security and a global OID allocation mechanism. Beyond these technical concerns it will also be vital to see how construals and applications can be refined collaboratively by potentially hundreds or thousands of individuals. It may be appropriate to regard the entire globally distributed OD-net as a single system in which case there would potentially be billions of people involved in constructing it. Without doubt, this would need substantial further work and may not even be realistic.

8.2.4 Histories and Persistence

Although this has not been documented within the main body of this thesis, there has been considerable effort made in finding a way to make the Cadence OD-net persistent between sessions. This involves finding an effective way of storing the entire OD-net to disk to be reloaded when the environment is restarted. Related to this is the possibility of caching parts of the OD-net in memory whilst leaving the rest on disk so that larger nets may be supported. The benefit of a persistent OD-net is that reliance on DASM scripts could be reduced or removed entirely (subject to developing appropriate alternative interfaces).

Various compression and caching mechanisms were developed but not fully im-

plemented in the Cadence prototype. One particular concern is the ability to undo damaging changes to a model effectively. While in EDEN undoing changes involves reverting a definition to its previous form, in Cadence - due to dynamical definitions - there may be more to undo. A form of version control of OD-net history would be vital in achieving this since a persistent net cannot be reverted to an original state as easily as an OD-net constructed from scripts.

Some of the compression ideas included the automatic classifications of structures in the OD-net to move content descriptions common to many instances into classes. Such mechanisms would be hidden from the user who would still be completely free to change structures. Also, history data can be stored as it is in classic version control, by storing only the differences. In Cadence it would be adequate to store only agent changes since every other change could be recalculated. Storing the occasional snapshot would help improve performance.

8.2.5 Custom Agency: Security and Schema

At present there is little support for custom agency as can be found in EDEN. Agency in Cadence currently requires the modeller to write C++ modules which reduces the liveness of the environment. There has been one attempt to remedy this by providing simple conditional actors that can perform single assignments when a condition is satisfied. These actors can be chained together to act sequentially or they can act in parallel to make changes. This solution is not particularly elegant or sufficiently rich for many tasks (i.e. looping to generate large numbers of new definitions). The hybrid described in §7.2 was constructed to overcome this lack of agency.

In addition to richer custom agency, an account of security and other forms of restriction is required. There has been some suggestion previously in the thesis that capability-based security would be ideally suited for use in the OD-net. Agents have capabilities based upon the kind of OID they have for manipulating a particular object. This approach would allow for distributed security in a collaborative and distributed

environment, as well as providing a mechanism for restricting agent actions to help solidify the artefact into a program. No work has yet been done on implementing such a security mechanism.

8.2.6 Support for Meta-Relations

As pointed out when discussing latent relations in §6.1.2, the Cadence prototype only supports single variable meta-dependency. A story for how meta-relations relate to programs and construals needs to be developed further than it has in this thesis before deciding whether Cadence should support multi-variable meta-dependency and meta-structure. How to implement an environment that does support the full range of OD-net meta-relations is unclear as it would be, or so it appears at the moment, rather complex to mix with the more concrete enumeration approach currently supported.

With full support for meta-relations it would be possible to develop a comprehensive library of generic functions for use in Cadence. Currently it is difficult to specify many kinds of functions without resorting to oracles, which limits the kinds of models that Cadence can practically implement. The ability to develop generic functions would be vital in scaling up to more substantial construals and programs.

8.2.7 Optimisations and Concurrency

The implementation of Cadence described in chapter 4 has undergone many revisions and has been developed to allow for experimentation with its architecture. As a consequence in many respects it is far from an optimised implementation. Specifically the storage of objects and representations used for definitions are entirely unoptimised which leads to increased use of memory and reduced execution efficiency. Techniques such as automatic classification of structures could reduce the memory footprint of a model as well as improve lookup performance. Definitions could be JIT compiled in ways that are similar to how methods are optimised in Self [Hölzle and Ungar, 1994]. Despite the lack of optimisation it still performs substantially better than EDEN (cf. Calculator model in

§7.3.3).

It is also conceivable that definitions could be updated in parallel. The existing implementation did at one time support the concurrent processing of events which did increase performance on multi-core machines[1]. However, the implementation of this concurrency was poor and there are synchronisation issues which need further exploration.

8.2.8 Formal Account

Finally, the formal account of Cadence given in chapter 6 is only a provisional first attempt that has not been rigorously critiqued or put to use. There is a need to show the universality of Cadence and elaborate on agency as well as relating it to other concepts. Being able to prove things about Cadence artefacts would go a long way towards it being used for serious and critical applications, which would perhaps be necessary if Cadence were to be considered as an operating system.

8.3 Limitations of the Approach

Beyond any practical limitations due to time constraints, there are also certain areas where this approach to plastic applications is not appropriate. Any applications that are to be used in safety critical situations may require a degree of formal proof that is perhaps not achievable with the framework discussed in this thesis. In which case a translation step might be required, going via a more formal specification. Alternatively a specification may be developed to which a Cadence model must be matched.

Although the potential for just-in-time (JIT) optimisation has been touched upon in this thesis, it is still unlikely that performance would be good enough for the most demanding applications. With high-performance applications a translation step may also be required to optimise for specific hardware. The problem here is that hardware is

[1] Performance improved by around 50% on dual core machines for models with large numbers of definitions.

typically imperative in character and so imperative programs are better suited to it. If specific hardware were to be developed for Cadence then such performance issues may become less sigificant[2].

Finally, many existing problems, algorithms and applications can be adequately developed using traditional programs. There is little benefit in using the framework in this thesis to develop these. Instead problems of a pre-theoretical nature or those involving end-user development would benefit the most.

8.4 Looking Forward

There has been considerable focus in this thesis on Empirical Modelling. However, the original and future vision for Cadence is largely independent of Empirical Modelling, with the intention being to develop it for more widespread use. Empirical Modelling has been used here to provide an underlying framework for Cadence, but with this now largely in place it is possible to focus on more practical concerns such as network distribution, concurrency, persistence and moving towards the plastic operating system vision stated in chapter one. Recent developments by Google, for example with their Chrome OS, are in many respects remarkably similar to how Cadence as an operating system would function. Cloud computing, where almost everything is on-line, is increasingly becoming a reality and there is a need for better operating system support (moving away from Linux). Various problems that Chrome OS suffers from could be dealt with elegantly by Cadence if some of the further work suggested above is carried out. In particular the transparency of a shared OD-net, caching potential and use of dependency could overcome limitations such as dependence on an internet connection for applications in Chrome OS. The richness and ease with which Cadence applications can be developed, adapted and potentially transparently shared matches well with these recent developments by Google, Apple and Microsoft.

[2]There has been some consideration of developing specific hardware for Cadence but it remains unknown as to whether this is possible or practical.