Emergent Properties Do Not Refine

Fiona Polack, Susan Stepney [1,2]

Department of Computer Science
University of York
Heslington, York, YO10 5DD, UK.

Abstract

Refinement is conventionally used to effect a demonstrably-correct development from an abstract specification. Emergent systems present an interesting challenge in terms of demonstrably-correct development, because there is a discontinuity between the global and local system descriptions. This is a position paper, exploring some aspects of the challenge in relation to the traditional model of refinement.

Key words: Emergence, Refinement, Systems Engineering.

1 Introduction

Our interest in the refinement of emergent systems relates to research into the engineering and predictability of emergence. From a specification of the desired system (displaying emergent properties), we want to determine a set of similarly-specified low-level components that can reliably generate the desired system.

In this position paper, we explore how the development of emergent systems might relate to the classical notion of refinement. The engineering of emergent systems is likely, ultimately, to be inexact, requiring probabilistic and argumentation techniques as well as conventional incremental development. These issues are not covered here.

We use the example of a cellular automaton (CA) producing gliders. A CA is perhaps the simplest of the commonly-studied emergent systems, in that its operational environment is completely determined by its representation and a single update rule. This provides a starting point for consideration of development issues, from which our position can be developed.

The paper starts with a brief review of classical refinement. We then describe a model of emergence, illustrated by a CA; we highlight issues relating to

[1] Email: fiona@cs.york.ac.uk, susan@cs.york.ac.uk

[2] This work is part of the TUNA feasibility study, EPSRC grant EP/C516966/1.

This is a preliminary version. The final version will be published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

the specification of emergent systems. On this basis, we examine meanings of refinement for emergent systems, before exploring reasons for the mismatches between classical refinement and emergent systems development. This allows us to examine how emergent systems development might proceed, and where it might exploit at least some of the ideas of classical refinement.

2 The Conventional Model of Refinement

The conventional, or relational, model of refinement is a formal expression of one incremental step in a development. The relational refinement is cast in terms of a many-many relation between an initial and final global state. Refinement is a relationship between an abstract program expressed in some abstract world that captures this initial-to-final relation, and an equivalent concrete program expressed in the concrete world [7]. Conventionally, the proof of such a refinement is discharged using a "simulation"; this reduces the global proof obligation (expressed over general sequences of operations) to proofs of the refinements of the initialisation, finalisation, and single system operations, making use of a retrieve relation, figure 1. The refinement proof demonstrates that functional properties in the abstract model are preserved in the concrete model. Non-functional properties such as security [9] are not necessarily preserved by such refinement.

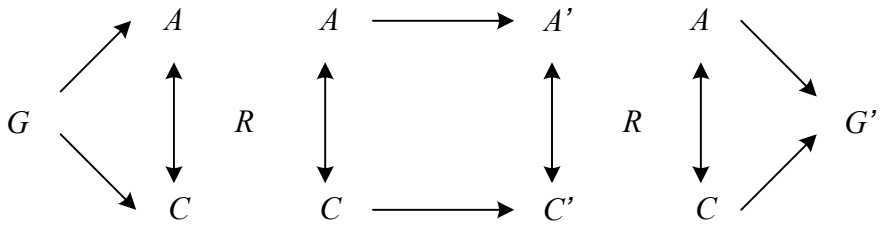

Fig. 1. The relational model of refinement, with simulations

For the purposes of this paper, we highlight a number of implicit conventions in the relational refinement model.

The *language* in which the model elements are written comprises concepts that are common to all levels and steps; the abstract specification uses the same base language concepts as the concrete specification and the retrieve relations. Note that, we are not referring here to a particular description language (such as Z), but rather to a common set of concepts needed to express the desired properties of the specification, such as sets or sequences of states.

The *refinement* is proved using a simulation, captured in a (relatively simple) retrieve relation. The simplicity is a pragmatic constraint, to enhance provability. If the refinement requires a more complicated retrieve, the proof process can be simplified by breaking the refinement into a sequence of steps with intermediate specifications linked by simpler retrieves (as in [15,16]). Ultimately, the steps can be made so small that they are captured in a refinement

calculus (eg [11]). A combination of "backward" and "forward" simulations is *sufficient* to prove any such refinement expressed in these terms [7].

The *initialisation* and the *finalisation* of the system are similarly simple relations between the global state and the respective abstract and concrete states (including the inputs and outputs of the operations).

3 A Model of Emergence

An emergent system is a system the ultimate behaviour of which is discontinuous with the behaviour observed at lower levels. Often-cited examples of emergent systems include network navigation by ants (real or simulated), construction by termites, swarming and flocking, for example by birds or their simulated equivalent, boids, and CAs.

In describing a system, low level component agents have behaviours and interactions expressed in some language C; the high level global behaviour is expressed in some language A. The system displays emergence if there is a fundamental discontinuity between the languages A and C (a similar definition is made by eg. [13]). For example, each agent in a flock might be described as moving towards a locally perceived centroid, whereas the global behaviour of interest is the velocity and size of the entire flock [12]. It is this discontinuity of description language that offers a challenge to conventional refinement.

3.1 Cellular Automata

One of the simplest emergent systems to study is a CA. In this paper, we model a CA in separate components, namely a cell (with a state, update rules and a set of neighbours) and the representation of the collection of cells. This has the advantage of moving the CA closer to other forms of emergent system, in which the representation (of a 2 or 3 dimensional space of various form and content) is clearly separable from the active entities (ants, termites, boids).

A CA cell is a finite state automaton; its update rule determines the next state of the cell from the current states of the cell and some collection of other cells, referred to as its neighbours. Such a cell can be programmed very easily, and is utterly uninteresting; the value of the state fluctuates, and at some point may stop fluctuating.

The cell becomes part of a CA (a cellular array of finite state automata) when it and its neighbours are located in some discretised space. Conventionally, the state of each cell is represented by colour or shading. At this point, some visual patterns may be apparent, but nothing is happening.

The emergent features appear only when the update program for the cells, located in space, is allowed to run repeatedly over a period of time, such that the visual representation of the cell states forms moving patterns[3].

[3] It is hard to demonstrate a CA adequately on a static sheet of paper: there are many online CA demonstrations, such as the simulation of Conway's Game of Life automaton at

The criticality of the space and time elements can be demonstrated, for CAs, by setting up equivalent cell models on different spatial representations. For example, [8] takes the same neighbourhood definition and update rule, and applies these to a 2-D regular grid and a Penrose kite-and-dart grid; they find that most characteristics of the observed CA behaviour are quite different on the two representations. Equally, the same specification starting from different initial states produces wildly different patterns. Furthermore, some CA configurations are fleeting, with all cells reaching quiescence in a small number of time steps, whilst others engage in seemingly complicated interactions over prolonged time periods. (For discussion and classification of CAs, including other representations and forms of CA, see eg. [18,19,20].)

3.2 Specification Requirements

If we were specifying and refining an emergent system, we would want to be able to specify the characteristics of the final system, then deduce some low-level components. We would then wish to prove (at some confidence level) that the emergent properties are established by the low-level components.

In the CA context, we might, for example, specify a requirement for some gliders, as in figure 2. We would expect the system to produce at least this pattern of behaviour reliably and predictably.

> The system shall comprise a region, across which a small repeated pattern, or "glider", passes.
> When two gliders meet they shall produce a single glider moving in the direction of one of the incident gliders.

Fig. 2. The glider requirement specification

It is conventional to design and program a CA as a grid of cells, and then to observe the behaviour of the CA over a series of synchronised update steps. However, when starting from an abstract specification of the desired behaviour, this could make an unjustifiable assumption about the implementation. The specification says nothing about grids; motion across a region such as a screen window simply requires some point of relative reference (eg. a boundary). The representation, as well as the state designation and update rules, are areas requiring analysis and refinement.

4 Comparing Features of Refinable and Emergent Systems

In this section, we examine some common meanings of classical refinement to illuminate why it does not naturally encompass development of emergent

http://www.math.com/students/wonders/life/life.html

systems.

The specification of the glider requirement in figure 2 is quite precise. It is non-deterministic in that we have left open the choice of representation (format, state designation, time steps etc) — indeed, it does not even state that the system must be built using a CA. In common with a classical refinement, we need to clarify these aspects of the implementation through a systematic, verifiable development process.

As in a classical refinement, we wish to represent the specification formally, in a way that allows refinement, according to what we know about the system and the target medium. Thus, we might start by deciding that CAs make a sensible target medium for this system. A naive development plan based on our knowledge of classical refinement is,

- to express the specification in a suitable formalism;
- to express the CA in a suitable formalism;
- to express a retrieve relation that maps between the two formalisms.

To express the specification, we need to define the glider (which has an area and a movement vector), the conditions for a collision of two gliders, and the effect of the collision.

To express the CA, we need to formalise the cell (its state, update rule and set of neighbours), and then formalise the representation of the collection of cells and the synchronised firing of all the update rules.

4.1 Refining to Reduce Non-Determinism

A refinement from abstract to concrete reduces non-determinism.

The non-determinism of the glider specification is reduced by refining the region in a way that facilitates representation of movement, ie by introducing something relative to which motion would be perceived. A common approach is to use a discretised spatial representation that allows quantification of location. For example, a regular or irregular grid allows some form of co-ordinates to be used to record the shapes and relative motion of gliders.

Having determined the representation for the glider in its window, we now explore how this relates to the cell and cell-collection model. We observe that, rather than finding a more deterministic implementation goal, the cell models appear to be *less* deterministic than the discretised spatial representation:

- the model of the cell alone merely requires a fixed number of related (neighbour) states, in order to calculate the next state of the cell;
- the model of the collection of cells has tied down the identity of neighbourhood to a specific group of cells from which a particular cell will always calculate its next state — some non-determinism has been removed;
- now, we need to tie down the neighbourhood to a specific spatial layout, so that neighbourliness is equated (in some way to be determined) to spatial

contiguousness — more non-determinism has been removed.

The intermediate level to which the retrieves takes us appears to be at a lower abstraction level than both the target medium and the original glider specification — in contrast to the classical refinement, where an intermediate level is at an intermediate abstraction level.

4.2 Data and Operation Refinement

A simple, abstract datum can be systematically refined to a more elaborate structure. The abstract datum can always be extracted via the retrieve relation, which is necessary for the developer to demonstrate that the output of the system (suitably filtered at the interface, or *finalised*) is the same as that specified abstractly. Similarly, an abstract operation can be refined, essentially by any transformations on the internals of the operation that maintains the interface of the operation unchanged (again, suitably filtered).

When refining the glider specification, represented as an emergent result of a CA, the first problem is to determine what abstract data could be refined. The second problem is determining what abstract operations could be refined.

The data in the glider requirement, figure 2 would seem to be *window* and *glider*. We have already noted that we could reduce the non-determinism by using a discretised spatial representation; in data refinement terms, we could refine *window* and *glider* in terms of this representation.

Refining a window to a representation of a grid (even an irregular or, within limits, a multidimensional one) seems conventional enough. However, a brief study of known gliders reveals that they cannot be refined to a spatial definition alone. Furthermore, the proposed reduction of non-determinism requires introduction of a time element as well as the discretised spatial representation.

For example, Conway's Game of Life [6] on a 2-D regular grid produces a well-known glider (figure 3) that can be detected only over five time steps (the period of the glider) across a space which is one (specific) row and column greater than the neighbourhood of the central cell of the initial configuration [4]. Moreover, the initial configuration is critical: there must be "clear space" (here, unshaded grid locations) "ahead" of the glider, and the "on cells" (here, shaded) must exactly match one of the four phases. No other initial configuration can produce this glider's behaviour.

The operations in the glider specification might be identified as glider movement, glider collision, glider resolution. In the abstract, these are simple operations. However, even at the level of the refined representation, it is hard to express these operations. Glider movement can be represented as the movement of a bounding box on a group of grid locations with a particular representation, but even this is non-intuitive: we are looking to produce vector

[4] If we allow general transformations, the glider is detectable in three steps, since steps three and four are reflections of steps one and two, and the movement space is suitably symmetric.

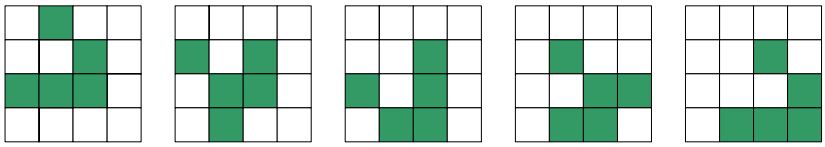

Fig. 3. A complete cycle of the simple Game of Life glider

movement, but it is only the cumulative movement of a glider that defines this vector — over four time steps, the bounding box of the Game of Life glider (1) moves one location in one direction, (2) does not move, (3) moves one location in the orthogonal direction, (4) does not move. Collision and resolution are no easier.

Indeed, trying to refine the glider movement into a concrete operation is misguided, since the movement, and the consequent collision and resolution, are emergent properties of the CA, not things that we wish to program into the solution. This is the key point. In an emergent system, the "abstract" operations do not have any corresponding "concrete" operations. There is no simple relationship between the abstract and concrete that can be captured in a refinements proof statement.

5 Refinement Concepts Re-examined

In section 2, we make various observations on the classical relational refinement. We now revisit these observations in the context of emergent systems.

5.1 Language

Two aspects of language are considered here: the form of the language and the extent of the language.

The components of the classical refinement are all expressed in a common form of language. The concrete language names refined data structures, but the concepts that the names represent are expressible in terms of the abstract language — that is why the retrieve relation is formally expressible; it converts one model of named components into another model of named components by transformations in the common language of the two models.

In the emergent system, there is a different language for describing the abstract (emergent) system and the concrete system. As shown in the discussion of refinement possibilities, the glider specification, figure 2, uses a language of relative motion — passing across a region implies concepts of spatially and temporally extended existence and velocity. The specification of a cell, figure 4, by contrast, refers to state and operation — a vocabulary typical of computer systems and computer programming. So, in developing emergent systems, we are trying to relate a specification in the language of relative motion (gliders), to a specification in the language of computation (cells).

If we turn to the representational aspect, and extend the cell specification

> The cell shall comprise a state, and an update operation. The update operation shall calculate the next state of the cell, based on the values of its (fixed) set of neighbours.

Fig. 4. The CA cell specification

to that of a collection of cells (figure 5), we find a vocabulary that makes more explicit the notion of state from the cell specification (that state-values have distinct representation). The notion of neighbour is also made more explicit, but this requires a change of vocabulary to that of spatial layout. This is reminiscent of the vocabulary of the glider specification. The collection specification also has explicit reference to time-steps, a concept that was implicit in the glider specification. However, there is nothing in the cell or collection specifications that could relate to the key concept of the glider specification, motion. Cells of a CA quite definitely do not move.

> A cellular automaton shall be a representation of a collection of cells. The representation shall represent each cell according to its internal state. For every cell, its neighbouring cells shall be a fixed pattern of contiguous cells. The cellular automaton system shall represent the synchronous update of all cells by replacing the representation of all current cell states with the representation of the next state of all cells.

Fig. 5. The specification of a CA as a collection of cells

Turning to the extent of the language, we observe that a classical refinement relates two specific computations. The fact that we could refine the abstract system in many ways is irrelevant; the refinement relation determines one specific concrete transformation of the abstract specification. Indeed, even the part of the language used to express the abstract state is restricted — in Z, we omit all concurrent and temporal aspects of systems; in CSP we omit much of the detail of the state of the system; in Circus, we ignore non-functional aspects such as system performance and security.

In emergence, as if the language discontinuity were not sufficient problem, we find that many CAs are Turing Complete languages in themselves (see [14,3]). The language of the abstract system is also of much wider scope than the formal languages used in classical refinement; in particular, the specification of any emergent system is likely to require the physical concepts of time, space, and relative motion as well as the notion of detectable patterns across time and space. Even if there is a path to a common language definition, the potential for refinement in it must be seriously jeopardised by the size of the search space.

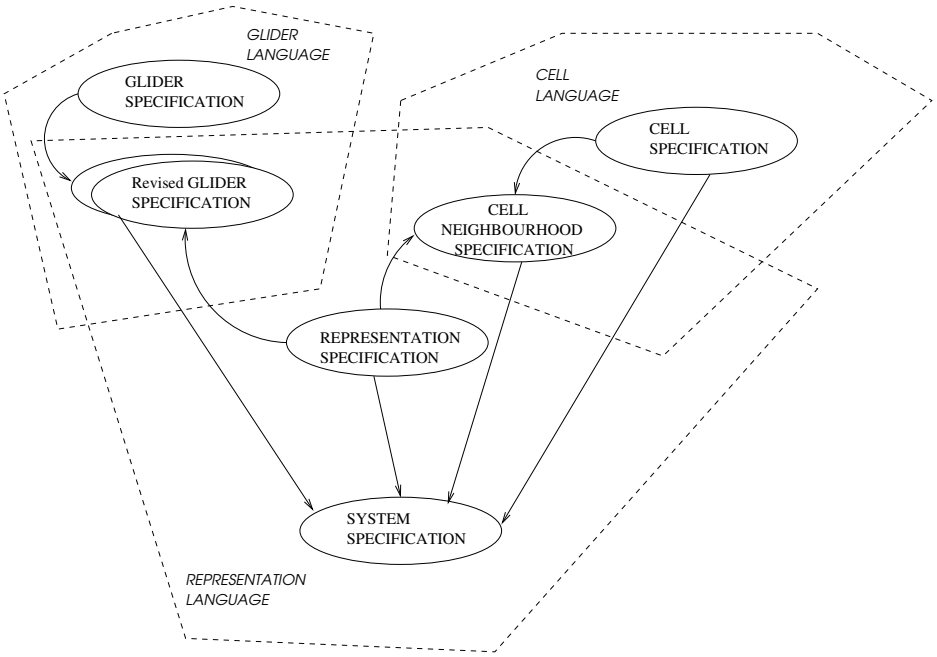

Fig. 6. Suggested elements in the development of the glider system

5.2 Refinement Relation

In the classical case, there is typically a single refinement relation (it may be composed of several steps). Even large industrial refinements such as the Mondex smart-card development [15,16] have been possible using refinement via only a single intermediate state.

In emergent systems, the consideration of refinement language suggests that there would rarely be a refinement that could be represented as one step. Even in this very simple case, the language of the glider, cell and cell collection suggests that the refinement of this system would have a number of steps some of which have partial intersections of concepts. The inter-related elements might be sketched as in figure 6. Some possibilities are apparent — we may be able to refine the cell to a cell on a representation, and the glider to a glider on the same representation, and then use the physics of state and motion to link the specifications. However, it is clear that there is no single refinement relation.

5.3 Initialisation

In the classical refinable systems, initialisation assigns each element of the system state a starting value that conforms to the state invariants. Formalists invariably seek a simple initialisation, typically setting each set to empty and each numerical value to 0 whenever this is a legal state of the system. It is supposed that the system state can then be constructed via the system operations (though the reachability of any system state from its initialisation is rarely considered). In refinable systems, the abstract system initialisation

is refined just as any other operation (figure 1).

Initialisation of the emergent system is not so straightforward; we have a problem with what to initialise. The most abstract state (the glider requirements) could be initialised in a system where the movement of discrete elements could be expressed. For example, the specification of gliders in general might be realised on a billiard table, where the initialisation is the setting in motion of one or more billiard balls. However, an abstract (implementation independent) initialisation is difficult to imagine, since the definition of the glider operations is dependent on the definition of the chosen representation.

For the CA, the specification permits a conventional initialisation of the states of a cell and its neighbours; the representation is initialised when the cell and its neighbours are translated to the representation, their initialisations determining the initial shading. Alternatively, the representation could be initialised with a particular pattern of shading, and the cells initialised from this when they are located on the representation.

This initialisation does not, of itself, guarantee the emergence of even one glider. The only way we can guarantee to initialise the collection of cells to produce a glider is by "rigging" the initialisation to one of the phase-layouts of a known glider for the particular representation, and ensuring that there is "enough clear space" around the glider for it to persist "long enough" for whatever detects gliders to register its presence.

We need some way of initialising the whole system (the cells, the collection, and the discretised spatial representation of the collection), because the CA behaviour is critically determined by the starting state. However, neither the specification of the CA, nor the specification of the glider system, gives any way of finding this starting state from the components identified here.

5.4 Finalisation

In a classical system, finalisation determines what is (or should be) *observed* as output from the system. Finalisation strips away the additional structure of the concrete data, so that the system returns just the data required in the global specification. (See [4] for discussion of finalisation and its limitations in describing what is observable, even in the classical case.)

Finalisation is the end-state of the system. An emergent system is, in general, a non-halting system, for which finalisation is not conventionally defined. However, where a series of outputs is non-interfering, finalisation could be taken as the "limit" of a sequence of outputs of a non-halting system.

There are a number of terminating sub-computations within the continuous emergent system. Taking an interpretation that sees finalisation as the mechanism that provides something from a component to the whole (by analogy with classical subsystems), finalisation can be applied to each sub-computation. For example, a cell updates at a time step. The finalisation ensures that the new state is passed to the representation. The finalisation of the representation

at the completion of one (collective) update ensures that the whole changed representation is revealed instantaneously.

These appear to be acceptable instances of finalisation, particularly if the cell, cell collection, and representation specifications are linkable by a classical-style refinement. However, this finalisation is incapable of revealing whether the system has produced the required high level behaviour, namely the production and interaction of gliders.

The nearest thing to a finalisation of the whole glider system would be an observation of the representational aspect of the system over a sufficient spatial and temporal duration that the required glider behaviour could be observed. The single abstract glider operation (movement) is thus detected as the result of a series of concrete glider operations (movement steps).

Considering system "finalisation" as a specific observation of the system leads us to another element that is needed to realise the whole emergent system. The movement that is fundamental to the glider must be detectable by some monitoring operation. The detection of the required gliders requires:

- a monitor that can identify the spatio-temporal signature of any glider
- a monitor that can identify the spatio-temporal convergence of two gliders; a side effect (not necessarily revealed to the observer) must be to establish the direction and velocity of the two converging gliders
- a monitor that looks for the emergence of a single glider from a collision, and identifies that it is indeed incident with one of its progenitors

The monitor in the emergent system can be thought of as a finalisation (a perspective of observation) with duration. Note that none of the observations does anything except monitor the spatio-temporal characteristics of the representation. Even the side-effects of the second monitor do not affect the CA or its representation; the rules of the individual cells determine what happens in the time-step following the first "collision interaction".

5.5 Observations on the Nature of Levels in CAs

In expressing the components of a CA, we note that each level (system, representation, cell) expresses, in some sense, a derivative of the next. Thus, a cell is one-dimensional; the representation introduces a multi-dimensional space; and the system introduces the time dimension. In the representation, each cell gains a location; in the system, blocks of cells appear to have velocity.

In changing the language of the levels of a CA, furthermore, we observe different identities emerging:

- a cell is simply an atom; it is distinguishable from any other cell only by measuring its state; there may be many cells with the same state; there is no sense of identity
- in order to link the cell and its neighbours to the representation, the cells acquire identity; the identity is (or can be derived from) the discrete lo-

cation of the cell on the representation; it is this identity that allows the neighbourhood of one cell to become the cell neighbourhood for the CA

- when the system runs, the observer (the watcher of the simulation, or, as above, the monitoring finalisation operations) identifies groups of cells with the same state designation on the representation — in the glider system, the monitor is seeking the characteristic vector movements of gliders, over extended time periods; monitoring, and thus recognition of emergence, is crucially related to the identification of groups of "state" that persist, with some periodicity and possibly some change of location.

The last point is the key for recognition of emergence (here, patterns in space that persist over time). Some CAs produce dramatic amounts of activity over many time steps — short-lived gliders, oscillators, and other CA structures, become momentarily visible and are then absorbed by less-coherent structures; sometimes, wave patterns, or Brownian motions, appear in the CA, perhaps resolving to a few oscillating residuals. How groups of cells are identified and monitored over time is crucial to the recognition of a required emergent feature of the CA. Recognition may not be achievable at the level of the representation — to detect the smoothed movements of the system over time, the monitor may need to be external, at yet another discontinuous level (like movement tracking in video).

5.6 A Reality Check

The specifications explored here are for one of the clearest emergent properties of one of the simplest of all computer simulations of emergence. CAs are used for computation; applications include conceptual work on massively-parallel computation, simulation of gas diffusion, ferromagnetism, percolation, crystallisation, forest fire propagation and urban development, as well as graphics generation. Toffoli uses a CA as an alternative to differential equation [5]. In such cases, CA requirements are significantly more complicated. We might, for example, require a *glider gun*, a key component of a CA-based model of conventional computer circuits. A possible specification is given in figure 7.

> A glider gun shall emit a stream of gliders which travel on the same path at a regular separation. The glider gun shall produce a such gliders until terminated by an external event; there should be no internal interruption to emission.

Fig. 7. The specification of a single glider gun

Figure 8 shows an initial state of a Game of Life glider gun on a 2-D regular grid, that is, one of the phases of the gun itself, before production of a glider;

[5] From http://www.rennard.org/alife/english/acintrogb02.html, accessed 17/12/04

the CA behaviour comprises a complex pattern of movement that oscillates between the pair of four-cell "blocks" at either side of the grid. Figure 9 shows a different phase of the gun (the two marginal blocks are as before), with one emerging glider and three already-fired gliders.

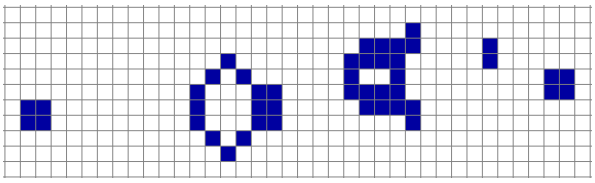

Fig. 8. Game of Life Glider Gun: the initial set-up

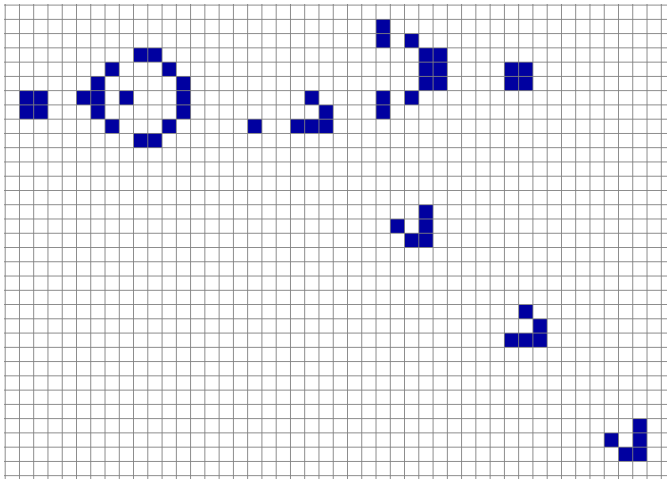

Fig. 9. Game of Life Glider Gun: snapshot with four gliders

We leave the development by refinement of such a system as an exercise for the ambitious reader.

6 Designing Emergent Systems

Currently, emergent systems are discovered by accident or by trial and error. Systematic development is possible using genetic algorithms (GAs) to "breed" the right rule set and initial conditions for a particular representation, using the closeness to the desired outcome (or emergent property) as the fitness criterion for breeding ([10] reviews work in this area). However, there is no engineering guidance for GAs, either — GA development of CAs relies on trial-and-error on, for example, the measure of closeness of representation.

The main research in making CA design more rigorous starts from known CA patterns, rather than refining down from (emergent) system specifications. For example, [5] builds on a range of work on the categorisation of CA and emergent systems, developing a computational mechanics of CAs to focus the evolution of a GA. Whilst the work is mathematically interesting, for systems development it seems to equate to working out how to characterise

all possible computer programs in a particular paradigm, in order to find out how to implement the solution to a particular problem.

The sole advantage of evolving CAs over accident and human trial-and-error is that it stands a better chance of converging automatically on an acceptable solution in an acceptable amount of time.

However, classical refinement is also a process of trial and error (though some formalists try to convince us otherwise). Despite neat published illustrations, the reality is an iterated guess-work, guided crucially by the experience of the guessers. Text books are notorious for tweaking the abstract specification so that refinement is clean. Retrenchment [2] acknowledge that in reality clean refinement is rarely possible, whilst forward-and-backward refinement to an invented intermediate state provides a trial-and-error approach when there is no easy direct route from the (real) abstract specification to the required concrete model. The authors have also investigated (to various levels) refinement patterns [17], template-based refinement [1], and evolutionary approaches to proof and refinement.

The guesswork of classical refinement appears so much less problematic than that of emergent systems development, because the scope for guessing is constrained by the language of specification, and by the (relative) simplicity of the problems tackled. Can we devise similar ways to constrain the distinct languages of the abstract and concrete descriptions of emergent systems, without losing the power to express the system concepts?

6.1 Some Tentative Design Guidelines for Emergent Systems

From the analysis above, a number of guidelines emerge. These give pointers to (a) how the languages of emergent system specification might be constrained or organised to support development and (b) the extent to which conventional refinement might be applicable.

In considering what refinement means for an emergent system, section 5, we arrived at a conceptual breakdown of the CA under consideration. It would seem that CA systems are simpler than other forms of emergent system because of, firstly, the simplicity of cell state and operations, and, secondly, the simplicity of the environment in which cells interact, as a visual representation that has no direct influence on the operations performed by the cells.

This model generalises, as shown in figure 10. We propose that certain links (dual-headed arrows) might generally have the potential for classical (or at least rigorous) refinement. Single-headed arrows bridge the linguistic discontinuities of the system elements. Such an element-wise decomposition has the potential to minimise the effect and scope of the linguistic discontinuities.

So our first two guidelines are to **identify the three key elements of the emergent system** — required system specification, the functional component (or instance) specification, and the specification of the integration representation; and to **identify elements with common vocabulary**. Following on

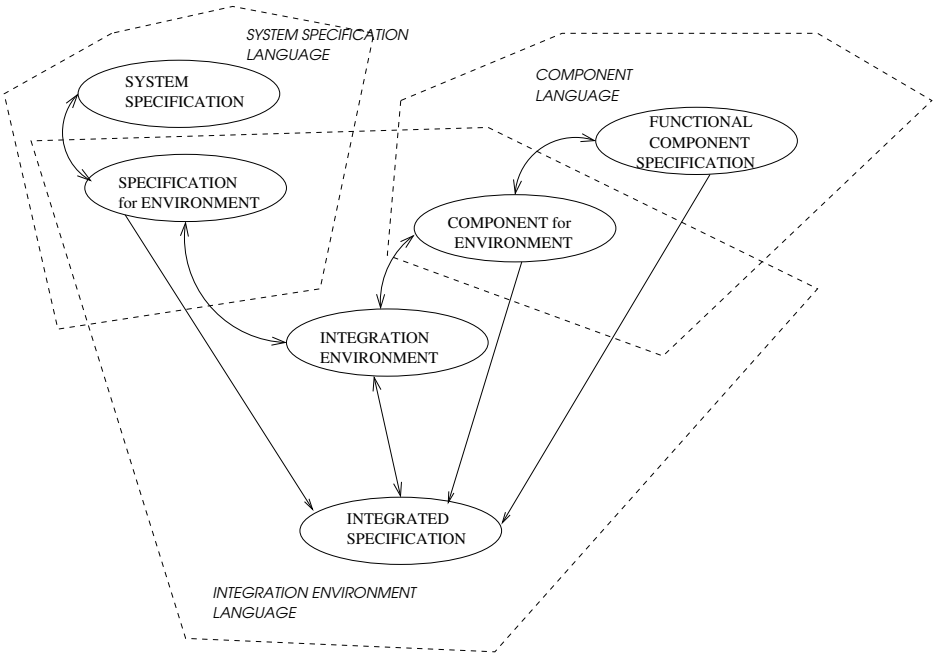

Fig. 10. A generalisation of figure 6

from these, **identify intermediate elements** — elaborations, possibly refinements, of the abstract system specification and the functional component specification, in terms of the integration environment.

Insight into achieving the non-refinable links of figure 10 comes from the consideration of finalisation in section 5. We propose the guideline, **establish how emergence is detected**. That requires the developer to consider how the emergent system will be monitored and what behaviours must be detected. We have not studied this part of the problem, but the initial impression is that, because emergence depends on a change in the frame of reference from the system to the observer, it is likely to be difficult to specify and refine monitors; monitoring perhaps captures the problem of linguistic discontinuity between elements and the whole system.

The only guideline relating to the initialisation of an emergent system is identical to that for a refinable system — **establish the initial state of each component of the system**. Some components in most systems are easy to initialise (because they are simple operational components). However, some components (eg the start state of the CA) can not be generalised; a start state known to produce the desired goal must be imposed. Emergent systems which have negative feedback or construction rules, may be easier to initialise, since the required behaviour occurs in the dominant (or sole) basin of attraction, to which the system converges from any starting state. However, it is also likely that the more precise the requirement for the system behaviour is, the harder the initialisation is to achieve. Thus, any boid simulation might flock, but only certain start states will create flocks of a particular size, direction and speed.

In reality, the situation is bad; there is strong evidence from biology that a complex system cannot be "initialised" in a state that results in a desired form, but has to be "grown". For example, it is rarely possible to construct a complex ecosystem by simply dropping in a bunch of species. It would seem that the part of the phase space occupied by "stable" emergent systems is so vanishingly small that these are unlikely to be discovered by chance, or by mathematical refinement.

7 Conclusion

We present a position on the development of emergent systems, using insights gained from classical refinement. We tentatively develop our position from an analysis of a system specification that can be realised by a CA. Our work is at an early stage, but seems to be producing a generalisable approach to the "architectural" refinement of emergent systems, that takes account of the linguistic discontinuities and variations of abstraction level among the elements of such systems.

Our approach differs from other attempts to determine the development of emergent systems because, rather than build up emergent systems from known components, we start from the specification of the required system, and then try to systematically determine a component and integration environment that is capable of forming the required properties as emergent effects.

It is likely that all attempts to design for emergence will rely on "rigging" somewhere; we anticipate catalogues of patterns of systems that can produce particular emergent behaviours; the linguistic discontinuity gaps in the development would be bridged by instantiating appropriate patterns. We need to minimise the gap so that the required patterns are as nearly determined by the form of the gap as possible, thereby avoiding the specialisms of, for example CA computation approaches.

References

[1] N. Amálio, S. Stepney, and F. Polack. Modular UML semantics: Interpretations in Z based on templates and generics. In *Workshop on Formal Aspects of Component Software, FME2003, Pisa, Italy*, 2003.

[2] R. Banach and M. Poppleton. Retrenchment, refinement and simulation. In *ZB2000*, volume 1878 of *LNCS*, pages 304–323. Springer, 2000.

[3] E. R. Berlekamp, J. H. Conway, and R. K. Guy. *Winning Ways for Your Mathematical Plays. Volume 2: games in particular*, chapter 25. Academic Press, 1982.

[4] J. A. Clark, S. Stepney, and H. Chivers. Breaking the model: finalisation and a taxonomy of security attacks. Technical Report YCS-2004-371, Dept of Computer Science, University of York, 2004. To appear in Refine'05, ENTCS.

[5] J. P. Crutchfield, M. Mitchell, and R. Das. The evolutionary design of collective computation in cellular automata. In J. P. Crutchfield and P. K. Schuster, editors, *Evolutionary Dynamics. Exploring the Interplay of Selection, Neutrality, Accident, and Function*. Oxford University Press, 2002.

[6] M. Gardner. Mathematical games: The fantastic combinations of John Conway's new solitaire game "life". *Scientific American*, 223:120–123, 1970.

[7] Jifeng He, C. A. R. Hoare, and J. W. Sanders. Data refinement refined (resumé). In *ESOP'86*, volume 213 of *LNCS*, pages 187–196. Springer, 1986.

[8] M. Hill, S. Stepney, and F. Wan. The Game of Life on irregular topologies. in preparation, 2005.

[9] J. Jacob. Basic theorems about security. *Journal of Computer Security*, 1(4):385–411, 1992.

[10] M. Mitchell, J. P. Crutchfield, and R. Das. Evolving cellular automata with genetic algorithms: A review of recent work. In *EvCA'96*. Russian Academy of Sciences, 1996.

[11] C. Morgan. *Programming from Specifications*. Prentice Hall, 2nd edition, 1994.

[12] C. W. Reynolds. Flocks, herds, and schools: A distributed behavioral model (SIGGRAPH '87). *Computer Graphics*, 21(4):25–34, 1987.

[13] E. M. A. Ronald, M. Sipper, and M. S. Capcarrère. Design, observation, surprise! a test of emergence. *Artificial Life*, 5(3):225–239, 1999.

[14] A. R. Smith. Simple computation-universal cellular spaces. *Journal of the ACM*, 18(3):339–353, 1971.

[15] S. Stepney and D. Cooper. Formal methods for industrial products. In *ZB2000*, volume 1878 of *LNCS*, pages 374–393. Springer, 2000.

[16] S. Stepney, D. Cooper, and J. C. P. Woodcock. An electronic purse: Specification, refinement, and proof. Technical Monograph PRG-126, Oxford University Computing Laboratory, 2000.

[17] S. Stepney, F. Polack, and I. Toyn. An outline pattern language for Z: five illustrations and two tables. In *ZB2003*, volume 2651 of *LNCS*, pages 2–19. Springer, 2003.

[18] S. Wolfram. Statistical mechanics of cellular automata. *Reviews of Modern Physics*, 55:601–644, 1983.

[19] S. Wolfram. Computation theory of cellular automata. *Communications in Mathematical Physics*, 96:15–57, 1984.

[20] A. Wuensche. Classifying cellular automata automatically: Finding gliders, filtering, and relating space-time patterns, attractor basins, and the z parameter. *Complexity*, 4(3):47–66, 1999.