A Formal Approach to the Engineering of Emergence and its Recurrence

Martin Randles[(1)], Hong Zhu[(2)] and A. Taleb-Bendiab[(1)]

[(1)]*School of Computing and Mathematical, Liverpool John Moores University, Liverpool UK.*
[(2)]*Department of Computing, Oxford Brookes University, Oxford, UK.*
[(1)]*{cmsmrand a.talebbendiab}@livjm.ac.uk* [(2)]*hzhu@brookes.ac.uk*

Abstract

Emergent behaviours are often characterised by the recurrent and recognizable events observable in a system's macro-scale environment, which result from simple local interactions between system components. Such interactions are governed by simple rule sets, which lead to complex higher-level global behaviour. Hence, engineering emergence is a necessarily subtle process subject to the impacts of system evolution with minor changes at micro-scale giving completely different outcomes, to those envisaged, at the global system level. Thus, to harness the self-organisation and emergence as a design principle for the build and management of assured and trusted systems requires a principled approach to specification and reasoning on emergence. In this paper the authors advocate the use of a formal approach/method to specify component interactions, system evolution, and runtime global states. The proposed method provides a formal specification for the engineering of known emergence and runtime deliberation on the emergence observable in the global system. A process is detailed where a model with a formal specification of emergent behaviour is given in the SLABS language. The implementation is endowed with an observer system that reasons on novel emergent features with a consequent update of the formal system model. The process is evaluated by means of a newly created simulation that shows these ideas in operation.

1. Introduction

In self-organising systems, consisting of complex interconnected systems of systems (agents/components /particles), global behaviour, or events, arise from the simple interactions of participating entities, and are often observed through the system's global environment. However globally observable events cannot easily be seen or predicted through the study of the low-level situated components alone. As the term emergence stems from the Latin word *emergere* – to rise out of water, which is the opposite of *mergere* – to be submerged, emergence is taken to mean some event or occurrence that becomes apparent rising out of a containing system. The key feature is that the occurrence becomes apparent or observable. So emergence may be taken as a process that results in some phenomenon that might be detected by an observer. This means that an observer is a vital part of any system seeking to engineer, or make use of, system emergence.

This is equally valid for large-scale complex computing systems such as: P2P networks, grids or self-organising overlay networks, in which the prediction of the global system behaviour based on the study of individual autonomous agents is notoriously difficult, if not impossible. For such a class of systems, there is a general lack of underpinning theory and techniques to facilitate the specification of the individual components, at design-time, to explain/allow-for the accurate prediction of runtime states of the global whole-system behaviour. These predictions need not necessarily be totally accurate, as some level of error is allowed, but they ought to be adaptable/defeasible, by the system itself, throughout the runtime. Thus, for runtime autonomic detection/characterisation of emergence and/or monitoring of systems' global behaviour, observer agents must be included that are capable of data acquisition from the system and subsequent deliberation on the global system state using this data. Traditionally multi-agent systems handle emergence through local data and communications, as global data is most often not readily available to the local components. Thus global data is almost certainly incomplete, due to the scale of the system. Therefore a model of self-organising emergence systems is proposed that is based on dedicated observers, termed the Observer Model [1, 2].

The observer is required to base its reasoning on likelihood rather than complete certainty and scenarios need to be specified to identify likely emergent behaviour or characterise self-organising systems.

Work has already been reported on a Scenario Calculus [3] to reason about emergent behaviour at design time. Its approach to the formal specification of emergent behaviour can be used for expressing global behaviours recognisable to an observer type entity. The recurrence properties for this emergence, reachability, stability or convergence for instance, can be proved as properties of the scenario. However this approach still relies on the exact pre-definition of each specialised instance of a self-organising behaviour. To address this problem, formal models of systems are highly desirable; as it is only through a formal logic that adequate deliberation can be achieved. A propositional account of the systems operations gives provably correct specifications whilst a suitably powerful logical formalism can endow formally modelled observer agents with inferential and deductive capabilities to reason on observable emergent properties using higher level abstractions.

It is for this reason that this paper focuses on the formal specification of self-organising systems through the Observer Model in an agent-based specification language – SLABS [4], with the situation calculus [5] to reason and gain a good representation of the forces and flow in a dynamical system for deliberation. This includes both supporting the detection/characterisation of known models of self-organisation, via observer detectable signatures, and identifying and evaluating, in the context of the system's operation, new signatures via signal groundings.

This paper details a formal approach to the engineering of self-organising systems, which is based on a provably correct formal model of the systems operation that is directly implemented through the efficient logical specification. Also composed into the system, as a separate concern, is a hierarchical cognitive observation system [2]. The observation system is capable of reasoning on the observed emergence either through metric evaluation or the inferring of new groundings for signals. For certain novel events, where no logical inference can be made from the existing knowledge base, it will be necessary for this deliberation to culminate in human level intervention. Additionally, where system adaptation and/or evolution has occurred through the run time, the formal model can be periodically informed and checked for its logical cohesion, through consistency and completeness, for instance.

The paper is thus structured as follows: The next section, 2, gives an overview of the proposed research agenda with a model of the associated procedures, tools and languages that will be used to support the process. Section 3 presents a simple scenario out of which arises an interesting global event, not apparent at participant level. The formalism and process is applied to this example with a formal specification of the system, an analysis of the emergence and a translation into executable code. The simulation results are evaluated and the paper continues, in section 4, with a setting of this work in the context of currently related works. A comparison to existing work is given with a discussion of the advantages and problems in this work. The paper concludes with a brief summary in section 5.

2. Proposed Observer Model of Self-Organising Systems

In this section an overview is given of the proposed approach to engineering emergence in self-organising systems.

2.1 The Observer Model: Self-Organising Systems

In existing system engineering methods a system can be engineered to recognise and ascribe meaning to a finite set of known signals (events). However, for self-organising or emergent behaviour, such signals cannot be a priori defined. Therefore signals may be grounded only by using existing signals to infer the intrinsic meaning of new (composite) signals or by assessing the future state of a system following the observation of a novel occurrence. The latter can be a specific action set taking-place in a specific context (situation, scenario, etc.). The assessment of the future state may either take place as the system runs or be performed in a modelling environment based on past performance. In the end signals can only be grounded via a cognitive deliberative observer [2], which can be performed via human level input or through an observational meta-system attached to the system to reason on any ensuing novelty. Such a process may occur as detailed in Figure 1.

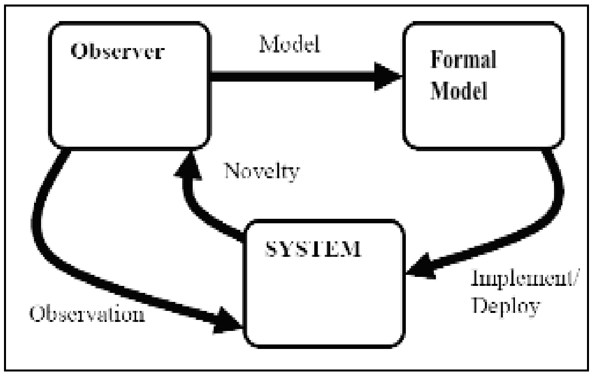


Figure 1. The overall structure of the Observer Model

Following recognition reoccurrence needs to be taken into consideration. For a signal to be recognised or grounded requires more than one manifestation of the observed occurrence. At the detection of a certain signal the particular conditions that held can be analysed for consideration of the likelihood of reoccurrence. For an immediate "piece of knowledge" to use in the run time system this analysis can be done through a formal model based deliberative observer. So upon an observer/monitor detecting some system feature with associated conditions or action history the manifestation of this feature may be mapped to the conditional set or history and the formal model used to establish the likely occurrence of the conditions in the future.

This is beneficial because the rules governing behaviour in an evolving system may be formalised and used for a number of purposes such as to improve future performance, to apply to another part of the system or to reuse as best practice in new system design. This is different to the previously specified technique of reinforcement learning because the rules in reinforcement learning are evaluated only through their application in the running system through reward or penalty.

In [1] a deliberative, logical, observer system was demonstrated using system imperatives to derive global system behaviour by deliberating on the emergent features resulting from the bounded autonomous behavioural norms of the system members. Norm governed safe, predictable behaviour is performed by the low-level components through their interactions and beliefs, whilst the global system properties, emerge from these low level interactions, to be assessed through the deliberative observer, as shown in Figure 2.

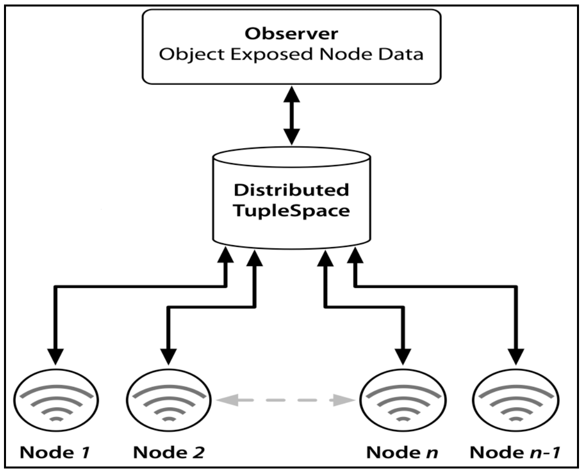


Figure 2. The observer model

The hierarchical nature of this system is apparent in that an entire system may have very many observer modules meaning that each of the N nodes in Figure 2 may itself be an observer system.

2.2 Dealing with Emergence: The Process Elements

This work is based on a dual approach to the formal methods employed in the system design and detection and deliberation on emergent system features. An efficient implementation method is also available to take the formalism into a fully coded application. The process starts with a formal specification of the overall structure of the system and an appropriate class of emergent behaviours observed, for example, those common in World Wide Web type systems and the overall structure of the system. This is followed by formal reasoning regarding the alternative choices for the defining of suitable rules targeted to the related emergent features. An executable system may then be derived from the formal specification for actual development or simulation. Subsequently deliberation on the run time system further informs the initial formal specification and run time system evolution/adaptation.

2.2.1 Formal Specification of System and Emergence Behaviour

The formal specification of the system is achieved using SLABS, which stands for Specification Language for Agent-Based Systems [4, 6].

In SLABS, the agents in a multi agent system (MAS) are classified by a partially ordered set of castes, which are the templates for agents which define a set of structures, behaviours and environmental features [7]. A specification of a MAS in SLABS consists of a set of definitions of castes in the following form.

$$\begin{array}{lll}
\multicolumn{3}{l}{\text{CASTE } C \text{ <= } C_1, C_2, \ldots, C_n;} \\
& \multicolumn{2}{l}{\text{ENVIRONMENT } EC_1, \ldots, EC_w;} \\
& \text{VAR} & {*v_1}:T_1, \ldots, {*v_m}:T_m;\ u_1:S_1, \ldots, u_l:S_l; \\
& \text{ACTION} & {*A_1}(p_{1,1}, \ldots, p_{1,n1}), \ldots, {*A_s}(p_{s,1}, \ldots, p_{s,ns}); \\
& & \quad B_1(q_{1,1}, \ldots, q_{1,m1}), \ldots, B_t(q_{t,1}, \ldots, q_{t,mt}); \\
& \text{RULES} & R_1, R_2, \ldots, R_h \\
\multicolumn{3}{l}{\text{END C.}}
\end{array}$$

where $C_1, C_2, \ldots, C_n$ are caste names. The clause 'C <= $C_1, \ldots, C_k$' specifies that caste C inherits the structures, behaviours and environments of castes $C_1, \ldots, C_k$.

The state space of an agent in the caste is described by a set of variables with keyword VAR. The set of actions is described by a set of identifiers with keyword ACTION. An action can have a number of parameters. An asterisk before an identifier indicates that the variable or action is invisible.

An agent's environment is a subset of agents in the system whose visible variables and actions can be perceived by the agent. It can be explicitly specified by clauses in the following forms. (a) 'agent name' indicates a specific agent; (b) 'All: caste-name' means all the agents of the caste; (c) "identifier: class-name" is a variable that an agent in the caste can be assigned to.
Agents' behaviours are defined by transition rules in the following form.

```
Behaviour-rule ::=
   [<rule-name>] pattern |[ prob]–>
                  event, [If Scenario] [where pre-cond];
```

where the *pattern* describes the pattern of the agent's previous behaviour and its current state. The scenario describes the situation in the environment. The event is the action to be taken when the scenario happens and the pre-condition is true, which is given in the where-clause. An agent may have a non-deterministic behaviour if multiple rules are applicable. The expression *prob* defines the probability for the agent to take the specified action on the scenario. It can be omitted so that the choices are non-deterministic.

A pattern describes the behaviour of an agent by a sequence of observable state changes and observable actions. It is written in the form of $[p_1, p_2, ..., p_n]$, $n\geq 0$, which means the previous sequence of events taken by the agent matches $p_1, ..., p_n$, where p_i can be in the form given Table 1.

Table 1. Meanings of the patterns

Pattern	Meaning
$	The *wild card*, which matches with all actions
τ	*Silence*
X	*Action variable, which* matches an action
Act $(a_1, ...a_k)$	An action *Act* that takes place with parameters match $(a_1, ...a_k)$
!Pred	*Pred* is a predicate that contains the agent's state variable. The agent's state matches *!Pred* if the predicate is true.

A scenario is a combination of a set of agents' behaviours and states that describe a global situation in the execution of the system. Table 2 gives the format and semantics of scenario descriptions in SLABS.

As shown in [3, 8] and will be also demonstrated later in the paper, SLABS can be used for the formal specification of agents of MAS using castes as well as the global emergence behaviour using scenario descriptions. The recurrence properties of the emergence behaviour can be proved using Scenario Calculus as properties of the scenarios in the context of system's formal specification [3].

Table 2. Semantics of scenario descriptions

Scenario	Meaning
Predicate	The state of the agents satisfies the predicate
$A=B$	The identifiers *A* and *B* refer to the same agent
$A\in C$	Agent *A* is in the caste *C*
$A{:}P$	Agent *A*'s behaviour matches pattern *P*
$\forall X\in C.Sc$	The scenario $Sc[X/A]$ is true for all agents *A* in caste *C*.
$\exists_{[m]}X\in C.Sc$	There are *m* agents in caste *C* such that $Sc[X/A]$ is true, where the default value of the optional expression *m* is 1.
S_1 & S_2	Both scenario S_1 and scenario S_2 are true
$S_1 \vee S_2$	Either scenario S_1 or S_2 or both are true
$\neg S$	Scenario *S* is not true

2.2.2 The Description of the Observer's Knowledge

The formalism, for the system deliberation via the observer system, follows a propositional account taking situations as a history of the previous actions. In these experiments a variation of the Situation Calculus [5] is used. In the initial situation (S_0), before any actions or events have completed, fluent values are described and subsequently vary according to effect axioms for each action or event that occurs. Successor state axioms that define the system variables complete the specification taken with action precondition axioms and the initial situation. In the first instance an action, a, changes the initial situation from S_0 to do(a, S_0) with the next action, a_1 say, changing the situation to do(a_1 do(a,S_0)) with a_2 giving do(a_2,do(a_1,do(a,S_0))) and so on. The comprised set of successor state and action precondition axioms show the changes in value of the fluents and the possibility of completing an action in each situation accordingly. In other words a successor state axiom for a fluent is TRUE in the next situation if and only if an action occurred to make it TRUE or it is TRUE in the current situation and no action occurred to make it FALSE, with precondition axiom *poss*(a, s) meaning it is possible to perform action a in situation s. The pieces of knowledge, or agent belief sets, are represented in the situation calculus by equating each world state with an action history or situation with the concept of accessible situations [9]. So if s_1 and s_2 are situations then $(s_1\ s_2) \in K_i$ means that in situation s_2 agent i considers s_1 a possible situation with K_i the accessibility relation for agent i. So that $K_i(s_1\ s_2)$ means that all fluents known to hold in situation s_2 also hold in s_1. So the accessibility fluent may be specified: $K_i(s_1\ s_2)$ denotes the belief state that in situation s_2 agent i thinks s_1 could also be the actual situation.

So knowledge for agent i ($knows_i$) can be formulated in a situation as:

$$knows_i(\phi, s) \equiv \forall s_1(K_i(s_1, s) \rightarrow \phi(s_1))$$
$$[\text{alternatively } \forall s_1\ (\neg\ K_i(s_1, s) \vee \phi(s_1)\ )\ ]$$

Thus knowledge dynamics, represented in the situation calculus, become available for deliberation through logical operations. However through this representation there is a distinction to be made between actual system actions and knowledge producing actions: If the action that occurred, to change the situation to its successor, was the perception of the value of a fluent then a sensing action to produce knowledge occurred. So the change was a change in the knowledge base of the agent. Thus it is necessary to distinguish sensing actions by writing $SR(sense_\phi,s)$ to record that the action produced a resulting value for ϕ. $SR(sense_\phi, s) = r$ = value of ϕ in s. Thus the successor state axiom for K follows:

$$\begin{aligned} K(s_2, do(a, s)) \Leftrightarrow \exists\ s_1(\ & s_2=do(a, s_1) \\ & \wedge K(s_1, s) \wedge poss(a, s_1) \\ & \wedge SR(a, s) = SR(a, s_1)) \end{aligned}$$

This formalism allows the representation of knowledge in partially observable domains, where only incomplete data is available. This is a vital prerequisite, for the approach presented in this paper, to reason on detectable emergent signals and infer the presence of new signals. For example suppose the following action history is specified:

$do(a,do(a_1,do(a,s)))$ with
$SR(a,s) \neq SR(a, do(a_1,do(a,s)))$
where $a=sense_f$ for some fluent f and a_1 is some deterministic action.

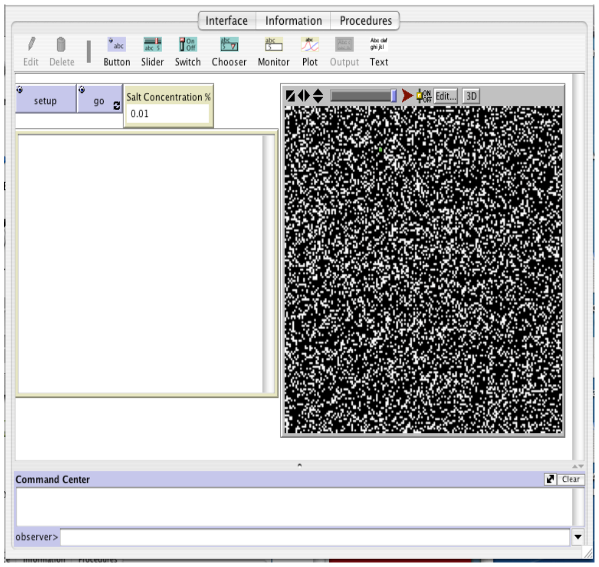
This then describes a process whereby a new prediction for the results of action a_1 may be inferred, where the values of other fluents in situation s form the action precondition axioms for a_1 as a context. So an action a_1, occurring in the context of situation s, grounds the signal, represented by the monitoring indicator f.

2.3 The Development Process

Using the formalisms described in the previous subsections the major elements of the engineering for emergence approach are derived. The known likely candidate scenarios that exhibit emergent outcomes are pre-defined and handled through the design time formalism whilst a logical dynamical representation, of the deliberative observer system, permits an efficient implementation with attendant reasoning functionality. The known signatures for emergent outcomes are part of the initial specification so may easily be recognised, when they occur, with additional tools, in the scenario calculus, to address the properties of reoccurrence and thus determine the relative importance to the system of this signal detection. Subsequently, through run time, the observer system assesses novel sources of system variable change based on the action history that preceded the variability in the fluent. Additionally the known emergent signatures are monitored and actions may be taken accordingly. At determined points through the operational lifecycle the snapshot run time system state can be synchronised with the overall formal model, as originally composed, and assessed for its logical properties and implications for future operation.

3. Example Evaluation: Salt World

In this section, we report on an experimental evaluation of the proposed method.



Salt Concentration %
0.01

Figure 3: Initial situation

3.1 The Salt World

The Salt World implements a very simple set of rules for the actors that results in an emergent global outcome. In the initial situation grains of salt (white) are randomly distributed across a 2-D world that also contains randomly distributed salt carriers (green) to move the grains, as shown in figure 3, a screen shot of the implementation in Netlogo [10].

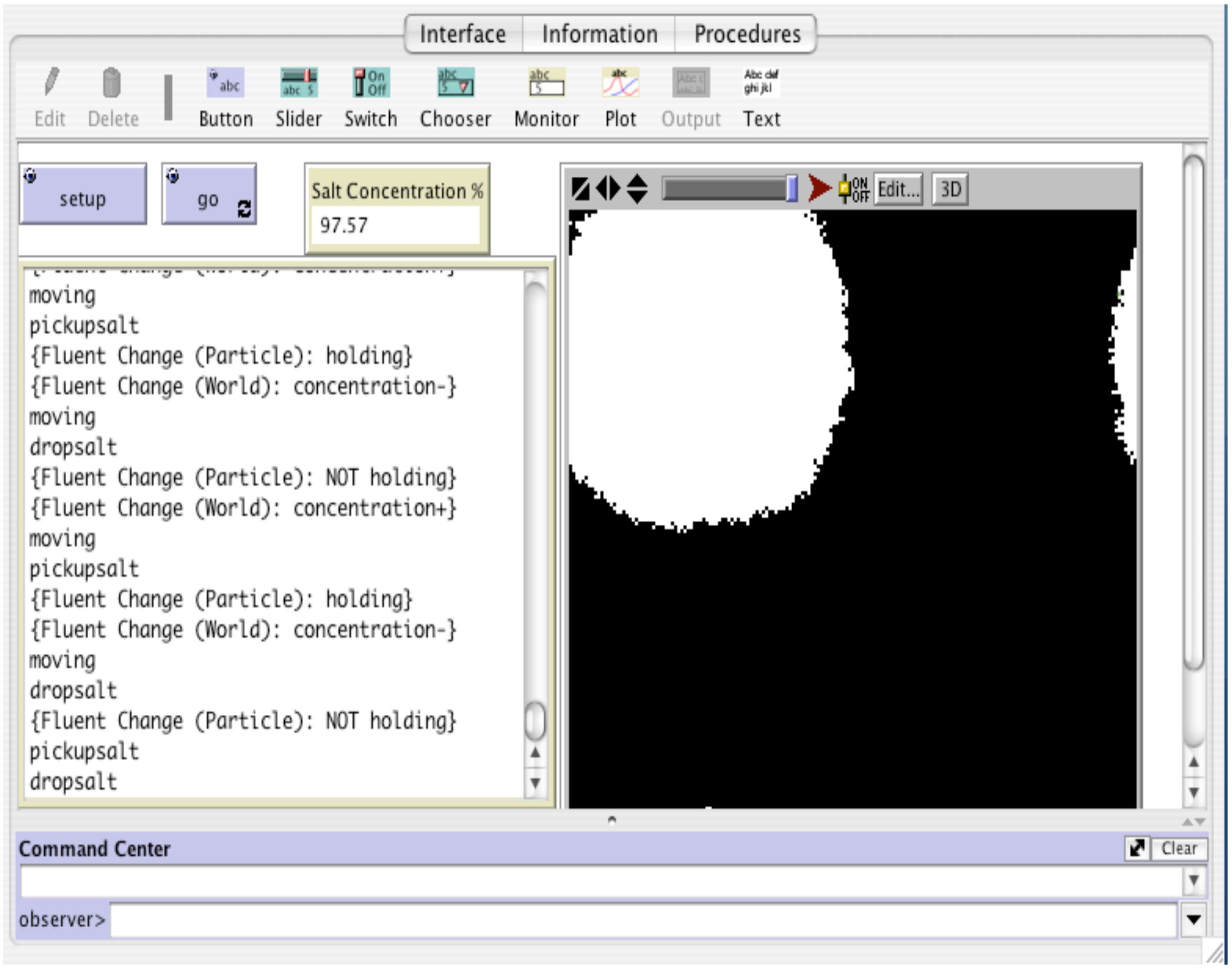
The salt carriers move randomly, through the world, picking up salt grains and dropping them at the nearest empty space when another salt grain is encountered. As shown in figure 4 this behaviour causes the formation of a highly concentrated completely connected salt pile.

The output to left of the screen shows the action history for a specific general salt carrier. The moving action merely abbreviates a sequence of move actions. The concentration- and concentration+ means concentration is decreasing or increasing respectively. The actual fluent value is recorded but not output for readability. The fluent changes are also output to this window and the distinction is made between the local (salt carrier) fluents and the global (world properties) fluents.

3.2.1 The SLABS Specification

The specification in SLABS provides the design time formal model through which the actual implementation can be easily produced. This is shown in Figure 5.

It may be seen from this, that certain recurrence properties may be deduced at design time. The emergent state of high concentration (state 2 in Figure 5) is not always reachable. For instance if the number of salt grains is low or widely dispersed the emergent state (2) will not always result. Additionally, from Figure 5, the emergent states (1) and (2) are reached with high probability when the number of salt grains is high. Furthermore, these states are stable. The system remains in either of these states once the states have been entered. Emergent state (3), however, is not stable as the search may result in a new find.



Salt Concentration %
97.57

```
moving
pickupsalt
{Fluent Change (Particle): holding}
{Fluent Change (World): concentration-}
moving
dropsalt
{Fluent Change (Particle): NOT holding}
{Fluent Change (World): concentration+}
moving
pickupsalt
{Fluent Change (Particle): holding}
{Fluent Change (World): concentration-}
moving
dropsalt
{Fluent Change (Particle): NOT holding}
pickupsalt
dropsalt
```

Figure 4: Emergence of single "clump"

```
Specification of the system
Caste Turtle;
Environment field: Field;
Var
    Position: Integer x Integer;             (* The position of the turtle is at the coordinate *)
    State: {FindSalt; FindPile, GetAway};    (* The mental state of the turtle *)
Action:
    Move;                                    (* Make a random move in the field *)
    PickSalt(x: Integer, y: Integer);        (*Pick a grain of salt at coordinate (x, y) *)
    DropSalt(x: Integer, y: Integer);        (*Drop a grain of salt at coordinate (x,y) *)
Behaviour:
<Move>:       (* Keep moving when the field has no salt at the current position, if the turtle is finding salt *)
    [!State = FindSalt & Position=(x,y)]
        |–>Move!Position=(x',y'); if field: [!HasSalt(x, y)=False]; where (x', y') ∈ Neighbour(x, y)
<Pick>:       (* Pickup a grain of salt and change the state    into finding a pile, when the turtle is finding salt*)
    [!State = FindSalt & Position=(x, y)] |–>PickSalt(x,y)!State=FindPile; if field: [!HasSalt(x, y)=True];
<Search>:     (*Keep moving if the turtle is in the state of finding a pile of salt until a pile is found *)
    [!State = FindPile & Position=(x,y)]
        |–>Move!Position=(x',y'); if field: [!~OnPile(x, y)];    where (x',y')∈Neighbour(x, y)
<Drop>:       (* Drop a grain of salt when the turtle found a pile while search for a pile *)
    [!State = FindPile & Position=(x,y)] |–>DropSalt(x, y)!State=GetAway; if field: [!OnPile(x, y)];
<GetAway>:    (*The turtle moves to a random place where has no salt and start a new cycle of searching of salt grains *)
    [!State = getaway & Position=(x,y)]
        |–> Move ! (Position=(x',y') & State = FindSalt);   if field:[!~OnPile(x',y') & ~HasSalt(x', y')]; where 0<x', y'≤100
End Turtle

Caste Field;
Environment All: Turtle;
Var      HasSalt[1..100, 1..100]: Bool;  OnPile[1..100, 1..100]: Bool;
Action  Update(x, y: Integer);
Behaviour:
<WhenPicked>:     (*When a salt grain is picked up by a turtle, it is removed from the field*)
    [!HasSalt(x,y)=True] |–> Update(x,y) !HasSalt(x,y)=False; if ∃turtle∈Turtle:[PickSalt(x,y)];
<WhenDropped>:  (*when a salt grain is dropped down by a turtle, it is placed on the field*)
    [!HasSalt(x,y)=False] |–> Update(x,y) !HasSalt(x,y)=True;  if ∃turtle∈Turtle:[DropSalt(x,y)];
<Update>:     (*When after a salt is dropped down or picked up by a turtle, the condition for a position in the field about
              whether it is on a pile of salt is updated*)
    [Update(x,y)] |–> !(∀(i, j)∈Neighbour(x,y).(OnPile(i,j)=True ⇔ ||{(x',y')∈Neighbour(i, j) | HasSalt(x', y')}||>5);
                                    where Neighbour(x,y)={(i, j) | x–1≤ i ≤ x+1 & 0< i ≤ 100 & y–1≤ j ≤ y+1 & 0 < j ≤ 100 }
End Field.

Agent field: Field.

Specification of the emergent states:
(1) Every salt grain is on a pile:
      field:[!∀x,y.(0<x, y≤100 & HasSalt(x,y) ⇒OnPile(x,y))]
(2) At least 95% of salt grains are on piles:
      field:[ || {(x,y) | 0<x, y≤100 & HasSalt(x,y)&OnPile(x,y)} || / || {(x,y) | 0<x, y≤100 & HasSalt(x,y) }|| >95%]
(3) All turtles are idle:
      ∀turtle∈Turtle:[!State = FindSalt]
```

Figure 5. System Specification in SLABS

3.2.2 The Observer's Perspective

For the observer system a dynamic action history is required, where the sensed emergent outcome can be related to the action history of the system. The underlying participant successor state axioms are:

```
holding(do(a,s)) ⇔ (holding(s) ∧
            a≠dropsalt) ∨ a=pickupsalt
```

```
foundpile(do(a,s))⇔( foundpile(s)
                          ∧ a≠dropsalt) ∨
  ( holding(s)∧ a= sensecolor(white))

atcolorwhite(do(a,s)) ⇔
                (atcolorwhite(s) ∧
           a≠move)∨a=sensecolor(white)

atcolorblack(do(a,s)) ⇔
                   (atcolorblack(s) ∧
           a≠move)∨a=sensecolor(black)
```

with action precondition axioms:

```
poss(dropsalt, s) ⇒ holding(s) ∧
       atcolorblack(s) ∧ foundpile(s)
poss(pickupsalt,s) ⇒ atcolorwhite(s)
poss(sensecolor,s) ⇒true
```

[the *sensecolor(X) (X=black or white)* action is shorthand for the formalism detailed in previous works:

$SR(sense_{\phi}, s) = r$ = value of ϕ in s. where ϕ is color and r is X (black or white)

So that accessible situations can be defined by the successor state axiom:

$K(s_2, do(a, s)) \Leftrightarrow \exists s_1(s_2=do(a, s_1) \wedge K(s_1, s) \wedge poss(a, s_1) \wedge SR(a, s)= SR(a, s_1))$

meaning knowledge within situation calculus may be defined thus:

$knows(\phi, s) \equiv \forall s_1(K(s_1, s) \rightarrow \phi(s_1))$]

3.2 Formal System Specification

As detailed earlier, this work relies on two approaches for a formal statement of the system. The outcome of emergence through correlation with known signatures is dealt with as a separate concern to that of the observation system, where logical entailment is used to extract new emergent features. Additionally the newly derived properties evaluated within the running system may then be assessed in the context of the design time formal model.

These rules alone cause the emergent behaviour that the observer detects via grounding using the *~saltconcentration* fluent. A distinction is made between the fluents that are monitored to assess global emergence and those which are local to the micro-scale participants. The global fluents arise from the operations represented by the fluents specific to the particle salt carriers. In this representation global fluents are flagged with a preceding "~" to enforce this distinction.

```
~saltconcentration(do(a,s))=n⇔
           [~saltconcentration(s)=n ∧
        (a≠dropsalt ∨ a≠pickupsalt)]∨
                 a=senseconcentration(n)
```

Then the high salt concentration appears as the emergent behaviour:

```
~highconcentration(do(a,s))⇔
              (~highconcentration(s)∧
          a≠senseconcentration(<97))∨
            a=senseconcentration(>97)
```

3.3 Translation into Executable Code

The formal representation, being an efficient logical specification readily transfers into code; so the basic actions of the participants are coded in Netlogo as:

```
to search-for-salt
ifelse pcolor = white
    [ set pcolor black
      set color white
      output-print "pickupsalt"
      output-print  "{Fluent  Change
(Particle): holding}"]
      [ moveon
      search-for-salt ]

to put-down-salt
ifelse pcolor = black
   [set pcolor white
    set color green
   output-print "dropsalt"
   output-print    "{Fluent    Change
(Particle): NOT holding}"]
      [rt random-float 360
    fd 1
    put-down-salt]
```

Whilst the emergent behaviour is detected by the Netlogo code:

```
to calculate-saltconcentration
ask patches [if pcolor = white
[ifelse count neighbors with [pcolor
= white] > 6
[set surrounded? true]
[set surrounded? false]]

set  saltconcentration  count  patches
with [pcolor = white]
if saltconcentration > 97
[output-print    "{Fluent    Change:
highconcentration}"]
```

4 Emergence in Multi-Agent Systems

In [3] a framework was presented to specify and prove emergent behaviours of multi-agent systems using SLABS. Thus, given a sufficiently detailed model of a general system, emergent properties can be proved for the scenario.

4.1.1 Related Works

Other approaches have tended to focus on emergence of role for the participating actors [11] or on seeking merely to define emergence [12]. There is also work on emergence in particular types of multi-agent systems, in narrow areas of application [13]. Work on emergence within multi-agent systems follows either a biologically inspired approach [14] or focuses on emergence from social interactions [15]. In [16] various techniques for engineering emergence are discussed. The proposed algorithms provide an assurance of convergence for an eventual tangible global state that is stable to perturbation from the micro-level. There has also been considerable interest in stigmergy, based on the indirect interactions of system participants, where messages are deposited through the environment to be picked up by subsequent encountering entities. There have been a number of specific applications of stigmergic techniques, [17,18] for example. However there is little work on a related general formal specification and there is no proof of convergences to an emergent state. Of some relevance to the work presented in this paper are the issues surrounding self-organisation in agents based on behaviour adaptation according to reinforcement. The model of adaptive agents [19] dynamically adapts logical relations between different behaviours. For the observer model, proposed in this paper, this strategy could be utilised to monitor the recurrence of recognised emergence. Cooperation is frequently used, where desired collective behaviour emerges to provide the system's functionality. AMAS theory [20] specifies that each cooperative agent is able to rearrange its local interactions dependent on its knowledge of the emerging system function. It is also possible to model systems based on meta-models of agent organisation. The PROSA architecture [21], for example, involves a holonic hierarchy model. Agents participate in holons, forming holonic structures with self-organisation occurring by adapting the holonic hierarchy to environmental perturbations.

A model where there is direct interaction between system participants allows the engineering of emergence through previously observed outcomes. However, as mentioned in [16], this is only useful for simple global equilibrium states modelled in a strictly linear manner. Stigmergy type scenarios have the added benefit of giving an implementation from an observed calibrated simulation. This gives some ideal solutions for specific application domains but, as mentioned earlier, does not permit a general formal explanation. Cooperation behaviour requires an exhaustive enumeration of cooperative states and adaptations, which is not always possible for large systems.

4.1.2 Engineering Emergence through a Formal Specification and Observer System

A fully general formal treatment for the detection and subsequent rigorous analysis of observable emergent behaviour, from and in a run time system, has not been addressed adequately. The work presented in this paper shows an approach based on using the formal language of SLABS to provide a model of the system, which is subsequently updated via the deliberative observation system specified through a dynamical logic of situations (action histories). The course of events (action history) that led to a specific emergent feature may be captured and used (or rejected) in the run time system. In the end the new emergent behaviour can be analysed through SLABS and incorporated into the formal system model, or not as the case may be according to logical constraints.

The great advantage of this approach is the ability of the system to continue running in the light of some novel entailed emergence, even making use of the inferred results. At a suitable time the behavioural model can be assessed for inclusion in the formal system model to inform future design practices. At present a problem may be the huge memory requirements required to ground novel signals. The illustrated example is intended to show a small part, at a specific hierarchical level, of the monitoring system that grounds the observed phenomenon. Even this is extremely costly in terms of storage space and search techniques. Further research is at a reasonably advanced stage to provide an algorithm for maintaining the situation space through methods that allow deletion of less important data without necessarily having knowledge of the data itself. Work is also underway to further improve this approach and demonstrate its efficacy through more complex simulations and scenarios. It is hoped to streamline the processes and provide a more logically integrated approach to the problems, allowing full run time adaptation and evolution of the design time model. In this way recognisability and recurrence can be analysed and proven, for use by the system, without any interruption.

5. Conclusions

This paper has presented a process for the formal engineering, evaluation and handling of emergence in multi-agent type systems. There remains a general lack of support for formal reasoning about emergent behaviour in multi-agent systems. This work proposes a described process, for multi-agent systems, where the known types of emergent behaviour may be formally specified and proved. The run time system also incorporates a required cognitive observer system to deliberate on known and unknown emergent behaviour, influencing, evolving and adapting the system accordingly. These changes may then be evaluated for a desired logical structure through the derived updated formal model. The process was illustrated and evaluated for deployment by means of a simple scenario, implemented in Netlogo.

References

[1] Martin Randles, A. Taleb-Bendiab, Philip Miseldine, "Harnessing Complexity: A Logical Approach to Engineering and Controlling Self-Organizing Systems", International Transactions on Systems Science and Applications Volume 2, Number 1 (2006), pp:11-20

[2] M. Randles, A. Taleb-Bendiab, P. Miseldine, "Addressing the Signal Grounding Problem for Autonomic Systems". Proceedings of International Conference on Autonomic and Autonomous Systems, 2006 (ICAS06), pp: 21,Santa Clara, USA, July 19-21, 2006

[3] H. Zhu, "Formal Reasoning about Emergent Behaviour in MAS, Proceedings of SEKE'05, July 14~16, 2005. Taipei, pp280-285

[4] H. Zhu, "SLABS: A Formal Specification Language for Agent-Based Systems", International Journal of Software Engineering and Knowledge Engineering, Vol. 11. No. 5, pp529~558. Nov. 2001

[5] H J Levesque, F Pirri and R Reiter, "Foundations for the Situation Calculus". Linköping Electronic Articles in Computer and Information Science, Vol. 3 No. 18. http://www.ep.liu.se/ea/cis/1998/018/, 1998.

[6] H. Zhu, "A formal Specification Language for Agent-Oriented Software Engineering", in Proc. of AAMAS'2003, July, 2003, Melbourne, Australia, pp1174 – 1175.

[7] H. Zhu, "The Role of Caste in Formal Specification of MAS", in Proc. of PRIMA'2001, LNCS 2132, Springer, 1-15.

[8] H. Zhu, "Formal Specification of Evolutionary Software Agents", Proc. ICFEM'2002, Shanghai, China, Oct. 2002, Springer LNCS 2495, Formal Methods and Software Engineering, George, C. and Miao, H., (eds), 2002, pp249~261.

[9] R.C. Moore, "Reasoning about Knowledge and Action", Technical Report, SRI International, 1980

[10] U. Wilensky, NetLogo Simulation Software http://ccl.northwestern.edu/netlogo Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL.

[11] D. Hales, B. Edmonds, "Evolving Social Rationality for MAS using 'Tags'", in Proc. of the 2nd Int. Joint Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS 2003). Melbourne, Australia: ACM Press, 497-503.

[12] C. Gillett, "The Varieties of Emergence: Their Purposes, Obligations and Importance", Grazer Philosophische Studien , vol:65 pp:89-115, 2002

[13] R. Axtell, J.M. Epstein, H.P. Young, "The Emergence of Classes in a Multi-Agent Bargaining Model", in Social Dynamics (eds: Durlauf and Young) pp:191-212 The MIT Press, USA, 2001

[14] J-P. Mano, C. Bourjot, G. Leopardo, P. Glize. "Bio-inspired Mechanisms for Artificial Self-Organised Systems". Informatica Vol. 30(1) pp:55-62, Ljubljana, Slovenia, 2006

[15] D.J. Watts, S.H. Strogatz, "Collective Dynamics of 'Small-world' Networks", Nature 393, pp:440-442, 1998

[16] F. Zambonelli, M.-P. Gleizes, M. Mamei, and R. Tolksdorf. "Spray Computers: Frontiers of Self-Organisation for Pervasive Computing". Second International Workshop on Theory and Practice of Open Computational Systems (TAPOCS 2004) in 13th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'04), pp. 397-402. Los Alamitos, USA, 2004.

[17] H. Karuna P. Valckenaers, B. Saint-Germain, P. Verstraete, C. B. Zamfirescu, H. Van Brussels, "Emergent Forecasting using a Stigmergy Approach in Manufacturing Coordination and Control". Engineering Self-Organising Systems. S. Brueckner et al. (Eds), Lecture Notes in Artificial Intelligence, volume 3464, pp. 210-226, Springer-Verlag, Berlin, 2005.

[18] N. Foukia. "IDReAM: Intrusion Detection and Response executed with Agent Mobility". The International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'05), pp 264-270, Utrecht, The Netherlands, 2005.

[19] D. Weyns, K. Schelfthout, T. Holvoet, and O. Glorieux. "Role based model for adaptive Agents". Fourth Symposium on Adaptive Agents and Multi-agent Systems at the AISB '04 Convention, 2004.

[20] M.-P. Gleizes, V. Camps, and P. Glize. "A Theory of Emergent Computation Based on Cooperative Self-Organisation for Adaptive Artificial Systems". Fourth European Congress of Systems Science. Valencia,1999.

[21] L Bongaerts. "Integration of Scheduling and Control in Holonic Manufacturing Systems", PhD Thesis, Katholieke Universiteit Leuven, 1998