Proceedings of the 2008 Workshop on
Complex Systems Modelling and Simulation

CoSMoS 2008

Susan Stepney, Fiona Polack, Peter Welch,
Editors

CoSMoS 2008

Luniver Press
2008

Published by Luniver Press
Frome BA11 6TT United Kingdom

British Library Cataloguing-in-Publication Data
A catalogue record for this book is available from the British Library

CoSMoS 2008

ISBN-10: 1-905986-17-3
ISBN-13: 978-1-905986-17-0

Preface

We are pleased to be running the first CoSMoS workshop in association with the thirty-first Communicating Process Architectures Conference (CPA 2008), in York, UK. Complex Systems often involve a large number of agents, which can be modelled as processes, communicating and interacting, resulting in emergent properties. As such, the architectures of complex systems simulators fit well with the scope of the CPA series of conferences.

The genesis of this workshop is the similarly-named CoSMoS research project, a four year EPSRC funded research project at the Universities of York and Kent. The project aims are stated as:

> The project will build capacity in generic modelling tools and simulation techniques for complex systems, to support the modelling, analysis and prediction of complex systems, and to help design and validate complex systems. Drawing on our state-of-the-art expertise in many aspects of computer systems engineering, we will develop CoSMoS, a modelling and simulation process and infrastructure specifically designed to allow complex systems to be explored, analysed, and designed within a uniform framework.

As part of the project, we are running annual workshops, to disseminate best practice in Complex Systems modelling and simulation. To allow authors the space to describe their systems in depth we put no stringent page limit on the submissions.

We did not want these workshops to be focussed on our own project; rather, we wanted to embrace the wider complex systems community, so that we could all benefit from the whole community's experiences and best practices. So we are delighted at this first workshop to have papers from a range of contributors, from a range of backgrounds including computer science, clinical medicine, immunology, and plant biology.

Ognen Paunovski, George Eleftherakis, and Tony Cowling have a detailed description of their multi-agent simulation framework to investigate emergence. They include an interesting blend of formal modelling, simulation, and validation and verification steps. They apply their approach to a small case study of animal herding.

Robert Alexander, Ruth Alexander-Bown, and Tim Kelly explore some of the problems that arise when engineering safety critical complex systems, and in particular, how one might argue the validity of simulation data in a safety case analysis.

Philip Garnett, Susan Stepney, and Ottoline Leyser apply the CoSMoS project's modelling approach outside the project itself, for the purposes of building a detailed simulation of a complex biological system: auxin transport canalisation in plant stems. Here the focus is very much on the biological process itself, rather than on more generic systems concerns, yet generic principles can still be extracted.

Finally, some members of the CoSMoS team itself, Paul Andrews, Fiona Polack, Adam Sampson, and Jon Timmis, and their biological collaborators Lisa Scott and Mark Coles, delve into questions that arise from using the initial CoSMoS toolset and method to simulate an immunological process. In particular, how can the simulation be validated, be argued to have any relevance to reality?

It is interesting to see some common themes emerging already in this very first workshop, in particular the emphasis on validation.

Our thanks go to all the contributors for their hard work in getting these papers prepared and revised, and to our programme committee for their prompt, extensive and in-depth reviews of all the papers submitted. We hope that readers will enjoy this set of papers, and come away with insight on the state of the art, and some understanding of current progress in Complex Systems Modelling and Simulation.

Susan Stepney, Fiona Polack (University of York)
Peter Welch (University of Kent)
August 2008

Programme Committee

Rob Alexander, University of York, UK
Paul Andrews, University of York, UK
Fred Barnes, University of Kent, UK
Richard Paige, University of York, UK
Fiona Polack, University of York, UK
Adam Sampson, University of Kent, UK
Susan Stepney, University of York, UK
Jon Timmis, University of York, UK
Peter Welch, University of Kent, UK

VIII

Table of Contents

CoSMoS 2008

X

Framework for Empirical Exploration of Emergence using Multi-Agent Simulation

Ognen Paunovski[1], George Eleftherakis[2], and Tony Cowling[3]

[1] South East European Research Centre (SEERC), 17, Mitropoleos St., 54624 Thessaloniki, Greece ogpaunovski@seerc.org
[2] City College, 13, Tsimiski St., 54624 Thessaloniki, Greece eleftherakis@city.academic.gr
[3] University of Sheffield, Regent Court, 211 Portobello St., Sheffield, S1 4DP. UK. a.cowling@dcs.shef.ac.uk

Abstract. In recent years the concept of emergence has gained much attention as computer systems have started exhibiting properties usually associated with complex systems. Although emergence creates many problems for engineering complex computerized systems by introducing undesired behaviour, it also offers many possibilities for advancements in the area of adaptive self-organizing systems. However, at the moment, the inability to predict and control emergent phenomena prevents us from exploring its full potential or avoiding problems in existing complex systems. Towards this end, this paper proposes a framework for structured empirical study of complex systems exhibiting emergence. The framework relies on agent-oriented modelling and simulation as a tool for examination of specific manifestations of emergence. The main idea is to use an iterative simulation process in order to build a coarse taxonomy of causal relations between the micro and the macro layers. In addition to detailed description of the framework, the paper also elaborates different aspects of ongoing work on herd dynamics case study.

1 Introduction

There is a variety of systems which are perceived as complex. In the natural world, cells, immune systems, nervous systems, ant colonies and many others can be viewed as complex systems. Similarly in the human world, a wide range of cultural and social systems like families, political parties,

companies, scientific communities, economical markets and many others are also complex systems. However in recent years the study of complex systems attracted considerable interest in the field of computer science. One of the main reasons for this trend is the increase in the complexity of computer based systems, which are becoming complicated, open and distributed [1].

The importance of emergent behaviours in computer systems can be viewed from two aspects. From an engineering perspective mastering the control of the emergent phenomena can be very useful. Emergence is responsible for self-organization, self-optimization, adaptation and other beneficial properties encountered in complex systems. The utilization of these emergent behaviours in an information system can benefit the development and performance of the system by making it highly available, scalable and robust. However on the other hand, a greater concern concern for computer scientists and software engineers is the appearance of undesired emergent behaviour or so-called "misbehaviour". Emergent misbehaviour can be viewed as an unexpected behaviour which may have an undesired effect on the system. An example presented in [2] shows that complete degradation of the service can occur in a fairly simple multi-tiered distributed application due to a small increase in database server latency. Examples like this show that it is vital to prevent the appearance of emergent misbehaviour in computer based systems. However since the phenomenon of emergence is inevitably linked with the complexity of the information systems, it cannot be simply avoided. Consequently there is a need to devise means for the development of correct systems which will guarantee (to some extent) that there will be no undesired emergent behaviour at runtime.

Nevertheless given the stochastic nature of emergence, in practice it is infeasible to formally verify the appearance of emergent behaviour [3]. Therefore in recent years agent-oriented modelling and simulation has been suggested as a tool which can shed light on the problem [4, 5, 6]. The idea is to model the components of a complex system as agents and use them in simulation-based experiments. Nevertheless so far there is no study which deals with the practicalities of constructing a framework for analysis of emergent formations. In this paper, we are addressing this issue by proposing a structured two-phase framework for empirical exploration of emergent behaviour through multi-agent modelling and simulation. The initial phase of the framework addresses the verification and validation of the multi-agent model, while the second phase is an experimental process aimed at determining the causal relations between the micro-level interaction and the visible effects of emergence at the macro-level. The end goal of this process is to address the problem of

analysing emergent behaviour in complex systems through a structured set of well defined activities and practices. As an ongoing work the paper also addresses the application of the proposed process and activities to a case study dealing with formation and dynamics of herds in animals.

The rest of the paper is structured as follows. Section 2 offers a introductory discussion on emergence. The application of the multi-agent paradigm for modelling of complex systems is elaborated in section 3. Section 4 provides a structured overview of the proposed framework followed by a detailed discussion of the practices envisioned as part of the framework. The ongoing work on the herd formation case study is described in section 5. Finally conclusions and future work are summarized in section 6.

2 Emergence and types of emergence

The basic idea behind emergence was popularized by Anderson in [7], where he argued that a simple component interaction can give rise to complex phenomena which are more than a simple sum of the components' behaviours. A simple example of this is the liquidity in water. There is nothing to suggest, in a single H_2O molecule, that many molecules at room temperature have the property of liquidity. Another example is the interaction among local weather patterns, which influence emergent formations like hurricanes, tornados, temperature inversions and other weather phenomena. Similarly stock market crashes can be viewed as emergent phenomena based on the interaction between traders on the stock market. The phenomena of consciousness in the human brain follows the same basic principle of emergence. While one neuron is a relatively simple entity, the collective interplay of millions of neurons in the human brain can result in something much more than a simple sum of the neuron's properties and behaviour.

If we examine these examples in more detail, we can come to the conclusion that emergence describes a system where a global phenomenon arises from the local interactions between the individual (micro level) components of the system. However due to the diversity and complexity of emergent phenomena, in natural as well as social systems, different sciences have focused on different aspects in the investigation of emergence. Consequently there is a variety of definitions [8, 9, 10, 11, 12, 13, 14] used to describe emergence, but none is generally accepted. Nevertheless in order to continue the discussion on emergence, there is a need for a working definition within the field of computer science. The authors of this paper adopted the definition proposed by Wolf and Holvet in [6], where they view emergence as part of the system:

> "...when there are coherent emergents at the macro-level that dynamically arise from the interactions between the parts at the micro-level. Such emergents are novel w.r.t the individual parts of the system."

In this context, "coherent emergents" denotes orderly (logically or aesthetically) consistent effects (properties, behaviours, structures, patterns) which are products of the process of emergence at the macro (system) level, caused by interactions at the micro (individual, elementary) level. While this definition describes the basic principle behind emergence, it gives almost no insight into the particular manifestations of emergence. Therefore in order to understand the forms of emergence there is a need to differentiate and classify different types of emergent phenomena.

Very often the concepts of weak and strong emergence are used in order to differentiate between emergent phenomena [9]. However this classification, although relevant for philosophical discussions, is too general in order to be useful in the field of computer science. Consequently there is a need for a more clearly defined classification structure like the one proposed by Fromm in [15], where he builds upon the classification for cellular automata proposed by Wolfram [16]. He distinguishes four primary classes (types I-IV) based on the causal relations of the phenomena. Furthermore the classification follows a gradation in complexity. Class I contains the simplest emergent phenomena with a single feed-forward relation which can be found in engineered systems (e.g. intentional design of a machine like a clock, computer program etc.) and systems exhibiting aggregated emergent phenomena (e.g. wave front in water, avalanches, cascades). Classes III and IV have the highest complexity. Class III phenomena have multiple feedbacks, both positive and negative. This type is common in open systems with high complexity and it is usually associated with activator-inhibitor systems (e.g. patterns in biological entities, stock market rush, prisoner's dilemma) as well as evolutionary and adaptive systems (evolution of ecosystems, sudden scientific or mental revolutions and so on). Class IV on the other hand, contains emergence of completely new complex systems (culture, life). From an engineering perspective a particularly interesting case is type II emergence which encompasses systems exhibiting self-organization and other useful properties (type IIA), as well as emergent phenomena which are based on imitation and self-amplification (type IIB). The latter subtype is responsible for the so called negative emergent behaviours like crashes and bubbles in the stock market, explosions of social unrest, buzz in the news and so on. In principle these are the same phenomena perceived as misbehaviour in computer systems. Consequently the

framework proposed in this paper is primarily concentrated on emergent phenomena of this type (II).

3 Multi-agent modelling of complex systems exhibiting emergence

Many researchers [4, 5, 8, 17, 18] dealing with the problems posed by emergence agree that an initial approach in understanding emergence should be done through Multi-Agent Systems (MAS). There are several main reasons behind such claims. First of all there is a natural correspondence between the structure of complex systems and MAS. They both rely on many individual components (agents) in order to achieve their goals. Each agent in an MAS is autonomous and able to interact in a stochastic manner with other agents. Moreover there is no limitation on the interaction scenarios, which means that an agent is able to communicate indirectly on multiple levels by modifying the local environment, which arguably is the most common approach used for communication in natural complex systems. For example ants communicate indirectly to each other by dropping pheromone which modifies the environment and serves as a guide towards food. Another correspondence between MAS and complex systems is the level of complexity. An MAS can achieve almost any level of complexity and thus a multi-agent system of specific complexity is essentially a complex system.

Nevertheless this does not mean that any MAS is complex or exhibits emergent behaviour by default. There are several properties common in MAS exhibiting emergence [19]:

- **Agent mobility, or visible states for fixed systems** - e.g. spatial repositioning for mobile agents, .
- **Ability to influence the environment** - e.g. chemotaxis, self-replication and other approaches for modifying the environment.
- **Ability to distinguish groups and individuals** - e.g. flocking of birds as a model composed of individual agents and groups (flocks).

In addition, it can be argued that the basic multi-agent modelling concepts, of abstraction, decomposition and organization [19], match the requirements for modelling complex systems. The ability to *abstract* allows the designer to simplify the representation of the system by hiding unnecessary complexity. Some properties of the model are emphasized while others are suppressed. This is very important when dealing with a system whose entities may be complex systems themselves. Otherwise it would be very difficult (if not impossible) to develop a complete model of the system.

The idea behind *decomposition* is to divide a complex problem into several smaller, more manageable components. Thus each component could be examined and analysed in relative isolation. Nevertheless the application of decomposition in systems exhibiting emergence is a very delicate issue. Decomposition of a system might diminish the emergent phenomena. For example dividing a living entity into parts could result in bunch of dead pieces. This is because "life", as an emergent phenomenon, relies on interaction between different components in the system and might not be a property of the component itself. Consequently it cannot be examined in isolated components. Nevertheless this does not mean that decomposition is useless in the case of emergence, but that it is important to devise an appropriate decomposition strategy which will not influence the phenomenon.

The third concept discussed by Jennings and Woodridge in [19] is *organization*. Although the development of an individual agent is a relatively straightforward process, the intentional design of the organizational structures of agents is extremely difficult. This is primarily due to the dynamicity and unpredictability of interaction patterns within the system. This unpredictability is a big problem from an engineering perspective since it diminishes the predictability of the system. However in the case where the agent system is used as a model in a simulation study, the uncertainty of the runtime dynamics is not a disadvantage nor a problem for that matter. In fact it offers the unique possibility for the investigator to gain insight into the possible behaviour that might be exhibited by the modelled system under certain conditions.

Although at the moment MAS may be the best tool for modelling complex systems, the modelling of complex systems exhibiting emergence faces several difficulties. One of the fundamental problems is the inability to capture the emergence with a model of the system. It is visible only during runtime operation of the system. Furthermore by its definition emergence introduces novelty at the system level which cannot be deduced from the properties of the individual components. So the developer of the MAS model cannot simply design a synthesis rule, which will encapsulate how combination of the elementary components gives rise to emergent properties. When emergence is concerned things simply do not add up in the way they are supposed to.

4 Description of framework for exploration of emergent behaviours

The framework proposed in the following sections is devised in order to provide a general framework for empirical examination of emergent

phenomena through multi-agent simulation. The fundamental idea is based on incremental increase of the knowledge about the causes and effects of the emergent phenomenon under study.

4.1 General Overview

The main object of the study is a multi-agent model of the specific system which exhibits emergence. The animation of this model is the primary method of achieving synthesis of the elementary behaviours into a macro-level emergent behaviour. In addition, the animation of the model generates the data which can be used as basis for the top-down delineation and traceability of cause and effects. In this manner, the proposed framework incorporates the two-way experimental method proposed by Edmonds and Bryson [4, 5]. The overall process can be divided into two phases: model verification and experimentation.

- The initial phase encompasses the development and validation of the model in respect to the expected behaviour. The idea is to use iterative refinement to the model in order to bring it closer to the desired behaviour. Thus building confidence about the model operation. This is an important step, which aims to provide an alternative to formal verification and validation, since it cannot be achieved for complex systems with stochastic interactions [4]. Once the model is validated the process can move to the second phase.
- The second phase of the investigation is an "experimental" stage. It is essentially an analytical process aimed at detection of invariants, interaction patterns, local properties and other elements that have influence on the emergent formations. The goal of this process is to detect the possible causes and their impact on the observable emergent behaviours. The mechanism proposed in the framework is essentially based on the experimental scientific method. The investigator forms a testable hypothesis and puts it to a test through modification of the model and animation (simulation). The observation of the model execution and the analysis of the gathered data should support or refute the hypothesis, thus increasing the understanding of the specific emergent phenomenon.

A phase, as defined in the framework, can contain several iterative cycles. Each cycle is composed out of predefined arrangement of activities (steps) and transitions. An activity defines the tasks (operations) which need be performed at a particular point, while the transitions define the output from one activity and the input to the next one. The section that follows contains a detailed overview of the activities envisioned as part

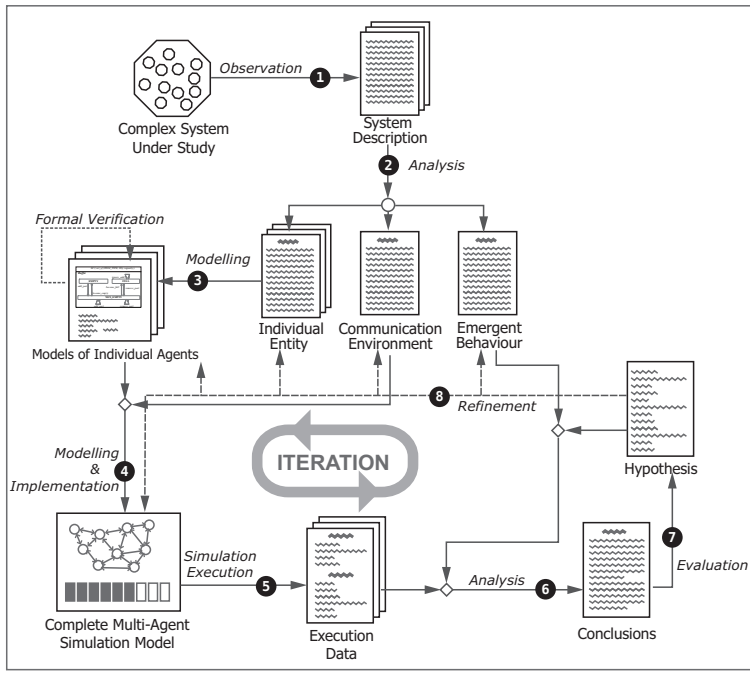

Fig. 1. Artifacts and transitions in the proposed framework

of the framework. Examples related to herd dynamics case study (elaborated in section 5) are used, where appropriate, in order to illustrate certain concepts in an activity.

4.2 Activities and Transitions

Figure 4.2 presents a detailed overview of the activities and transitions envisioned in a cycle. However it has to be noted that the first two steps are performed only in the initial cycle of the first phase. The steps deal with the system description and model specification. The steps 3-8 represent the elements of a single iteration. The description of the activities, transitions and the corresponding artifacts is summarized as follows:

– Activity 1: The initial transition is a jump from the real system to its theoretical description. Depending on the system being studied the most appropriate person to perform the description is an expert

in the field of study. For example in the case of a natural system, like the herd formation in animals [26, 33], this transition should be performed by biologists.

- Activity 2: The second activity focuses on the analysis of the system description from the initial step, in order to derive the major properties of the multi-agent model. The overall properties of the model should capture three aspects:
  - The properties of each type (if more than one) of agent in the model. In the case of herd formation the document should provide the specification of the properties of an individual animal.
  - The properties of the environment and types of communication used. This could include an explicit communication protocol or some indirect way of communication between the agents, or interaction between the agent and the environment. In the herd formation scenario example the communication is achieved through modification of the environment by repositioning of the animals.
  - The properties of the emergent phenomena. The major focus should be on clear (quantifiable, if possible) definition of a particular manifestation of emergence. The characteristics of the emergent phenomenon could be defined through macroscopic variables [6] or other indicators. In the herd scenario such an indicator could be the level of herd cohesion.
- Activity 3: Based on the specification developed in the previous activity, an appropriate model for each type of agent needs to be developed. A good practice is to use a formal modelling language, so as to be able to verify the properties of the model in order to determine that there are no undesired discrepancies and errors.
- Activity 4: The models of the individual agent, developed in the previous activity, should be combined with the appropriate representations (models) of the environment and communication in order to form the complete multi-agent model of the system. An important issue in this activity is the relationship between the individual agent and the environment, which is supposed to enable non-deterministic multilevel interaction.
- Activity 5: At this point the complete model should be transformed (implemented) in an environment which should allow animation of the model. The selection of simulation environment may vary depending on the system under study. A visual animation of the model may prove very useful in cases like the herd formation. Although animation is a bottom-up process, all of the data required for the top-down delineation needs to be generated during the animation process. Therefore it is essential that parameters which define the model's behaviour, both at micro and macro levels, are recorded

in sufficient details for later analysis. This data should present two views on the executed simulation.
  - First it should contain quantitative measurements on the state of the model at a particular time frame, offering traceability in terms of continuous model execution.
  - The second type of data should provide insight into the properties of the macroscopic level. This could be achieved through a set of global variables. For example, in the herd formation scenario such global variable could be "herd cohesion" which will indicate the strength of the herd.
- Activity 6: This step involves analysis of the simulation data gathered in the previous step. The goal of the analysis process is different in the two phases.
  - In the initial phase (development and verification) the analysis of the data should be done in order to evaluate the current state of the model in respect to the desired model. Parts of the data can be used in order to trace possible errors in the model.
  - In the experimental phase the analysis of the gathered data should be compared with the expected outcomes in order to test the hypothesis. The analysis should focus on evaluation of the stability of emergent behaviour in the model and detection of patterns by means of statistical and correlation analysis as well as detection of invariants. Based on the findings the investigator can derive conclusions about the behaviour of the model in the particular simulation run.
- Activity 7: This step involves evaluation of the simulation data in order to quantify the behaviour of the model and emergent phenomena.
  - In the first phase, the goal of this activity is to identify elements of the model which cause discrepancies in the model behaviour from the desired one. Thus to be able to define (if any) the required changes to the model in order to bring it closer to the expected behaviour.
  - In the second phase the main focus is on formulation of a hypothesis which will be examined in the next iteration. The hypothesis should be empirically testable by making changes to the model. In addition, the investigator should define criteria according to which the hypothesis will be supported or refuted.
- Activity 8: The final stage of the cycle can be viewed as an initial stage of the next iteration and involves modification (refinement) of the model in order to test the hypothesis set. As can be seen from the diagram, the modification can be done on all aspects of the model including the individual agent, the environment and communication.

Additionally changes in the data reporting routines of the simulation environment may be needed in order to gather additional data.

The discussed steps represent a single cycle in an iterative process, while the number of iterations is not fixed it should be sufficient to support or refute the hypothesis being examined. In this manner the proposed framework adopts an experimental approach to exploring emergence in already existing systems.

However since the object of the study is a model of the real system rather than the system itself, there is a major concern that the developed model might not possess sufficient details, i.e. it might omit important factors or make wrong assumptions about the system. If it did, the model would be potentially useless in respect to the goals of the study. This is the main reason why the verification and validation of the model is a very important part for the success of the study.

4.3 Verification and validation of the model

The verification and validation of the model is incorporated as an initial phase of the proposed framework. The main goal of the tasks in this phase is to minimize the potential discrepancies between the real system and the model. In respect to this, several issues concerning validation and verification need to be addressed.

Figure 4.3, taken from Sargent's work in [20], presents overview of the modelling process in relation to the validation and verification steps. The problem entity denotes the "real" system which is the object of the study, while the conceptual model is an abstract representation of the system which is developed during the modelling process. The validation of the conceptual model should determine whether the model corresponds to the real system for the intended purpose. Although the practical details of various validation techniques are beyond the scope of this discussion (for more information see [20]), there are several issues that need to be considered. In this context perhaps the most important issue is the selection of a representation technique. There are several aspects that need to be taken into account when selecting a representation technique [21]:

- The expressive power of the representation technique. Can the model be fully captured with the particular technique?
- The technical knowledge of the people to whom this model will be communicated. Other people, involved in the study, should be able to understand the notation used to describe the model.
- The application of formal analysis and verification of the model in the particular form.

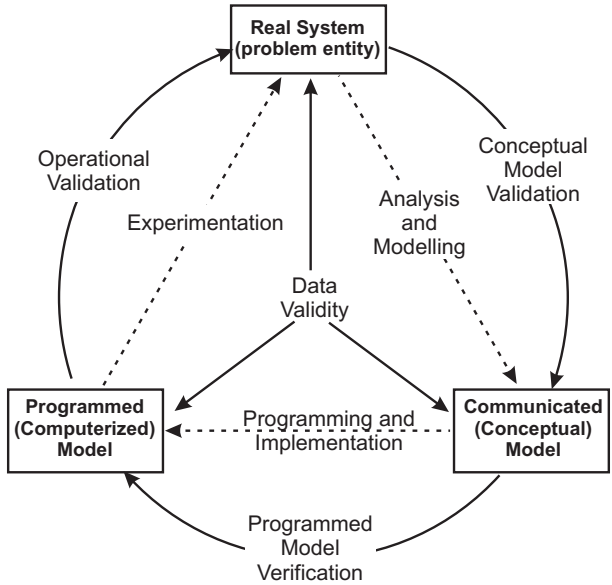

Fig. 2. Overview of verification and validation steps in a modelling process, taken from [20].

– Automation of the transformation, from communicative to programmed model. How can the conceptual model be transformed (implemented) into a form which can be animated and used in a simulation study?

Given the variety of representation techniques, there is no generic solution to all of these problems. The success of a particular representation schema depends on the system under study as well as the environment and people involved in the study. Therefore we avoid proposing a particular technique as part of the framework. Nevertheless we strongly believe that a formal representation should be used in order to develop the model of the individual agent. By using formal methods the investigator can verify that the model of the individual agent is correct and can therefore focus its attention on the validation of the communication and interaction between the agents. However not all formal methods are suitable for modelling agents. In general, in order for a formal method to be useful for modelling agents, it should satisfy the following criteria [22]:

– to model both the data and the internal changes of an agent,

- to model separately the behaviours of an agent and the ways in which the behaviours interact with each other,
- to be intuitive, practical and effective towards the implementation of an agent,
- to facilitate the development of correct agents.

Although the use of formal techniques may be applicable to an individual agent, in practice it is infeasible to formally verify a complex multi-agent system with stochastic interaction [4]. Consequently in this case the most appropriate solution is to informally validate the interacting multi-agent model by means of simulation. However in order to do so the conceptual model needs to be implemented (programmed) in a form which can be executed. The transition from the conceptual to the programmed model is the second step in the validation and verification process presented in figure 4.3. The primary concern here is to ensure correctness and correspondence of the implementation to the conceptual model design. In the best case this transformation could be done using a tool (which has been extensively tested) which will automate the transformation process and ensure correctness. However depending on the problem and the representation technique used, such a tool might not be available. In this case the transformation needs to be performed manually. In this context, verification techniques which are used in software engineering could be applied here [23]. Detailed analysis of various testing techniques is beyond the scope of this discussion, for a comprehensive overview see [24].

At a point when this transition from the conceptual to the programmed model is complete, the investigator has the means to validate the complete multi-agent model of the system. This is the final verification and validation step in figure 4.3 (operational validity) which aims to validate the correspondence between the behaviour exhibited by a programmed model and the real system. The final behaviour of the programmed model must have a reasonable accuracy in respect to the real system and exhibit the intended applicability to be used in the experimental stage. According to Sargent [20] a general division of the operational validation process suggests two main approaches:

- Objective approaches, which usually rely on statistical or mathematical proof of the correspondence between the model and the real system. This approach is much more difficult (compared to the second one) in terms of effort and time, but it has much higher credibility. Nevertheless it is not always applicable, especially in the case of complex systems.
- Subjective approaches, which basically rely on techniques where the final decision of the performed comparison (between the model and

the real system) is made by the investigator. These kinds of approaches are usually used to ensure operational validity of complex systems, since it is in practice impossible to this in a formal way.

The idea that we propagate through the framework is to use mathematical validation where applicable, however since this is rarely the case with complex systems, we suggest an iterative process which will enable the investigator to gradually build confidence in the behaviour of the model.

4.4 Two-way experimental approach

The experimental phase follows the validation phase and can commence once the investigator has confirmed that the model is valid. As previously discussed the main aim in this stage is to explore the causal relation between the micro and the macro levels through an iterative experimental approach. In each experiment the role of the investigator is to formulate a hypothesis and then test it by executing an appropriate simulation. In order to do so, the simulation conditions or even the model itself can be modified. This process in essence resembles the so called "general scientific method", where by testing a hypothesis the knowledge about the system is gradually increased.

The investigation of the data from the simulation execution as envisioned in the proposed framework is addressed in a manner similar to the two-way approach proposed by Conte and Castelfranchi in [25]. It encompasses both bottom-up and top-down processes in order to analyse micro-macro connections.

The **bottom-up process** facilitates attaining collective behaviour from the individual agents. It offers insight on how their behaviour is combined and aggregated. The role of the investigator during this process is to identify interactions that have an immediately visible result at the system level. Additionally through observation of the model animation the investigator has the opportunity to gain insight into systems' operation. This knowledge can be very useful in the top-down analysis.

The **top-down process** is concentrated on analysis of the data gathered during the simulation. The analysis process should address several issues:

- *Behaviours of micro entities*
  Deduce the behaviour of the individual element (agent) from the global behaviour of the system. This includes identification of how the micro elements behave at a given time instance, also what behaviour should be visible at the micro level given the behaviour of

the overall system, as well as define how the behaviour of the system imposes restrictions on the behaviour of an agent (top-down feedback).

- *Behaviour of emergent phenomenon*
  Define a set of global variables which indicate different aspects of the observed phenomenon. Where applicable avoid specifying binary variables. If possible devise a metric for each of the variables. Define values for the variables for each time instance. Compare changes in the variable values (if more than one variable). Correlate the changes in a variable with micro level events.
- *Associate roles and states*
  Identify the possible roles and role transitions for the micro level entities (agents) in terms of responsibilities, permissions and activities. Identify the possible states and state transitions for the system at the macro-level. Determine the possible roles of the micro level components for a particular state at the macro-level. Identify (if possible) how state changes at the macro-level influence the role changes at the micro-level. Associate the role changes at the micro level with the state transitions at the macro-level.
- *Communication and conflicts*
  Identify the micro level communication (interaction) and coordination mechanisms and if possible determine tolerable conflicts and inconsistencies. This means identification of the type of communication (interaction) exhibited by the agents as well as the reason for the initiation of communication. Additionally it is important to determine if there are repetitive interaction patterns and how a particular interaction pattern in the micro level yields an observable system behaviour and the macro-level.

It has to be noted that both processes (bottom-up and top-down) are complementary to each other and share a common goal. Therefore the findings of the bottom-up observation and top-down analysis need to complement each other. Any contradictory findings need to be further examined either by re-examining the data or by repeating the simulation in an iterative manner until the conflict is resolved. The findings need to be consistent in order to determine the micro level factors which have observable influence on the emergent behaviour at the macro-level.

5 Case Study: Herd Dynamics

The herd dynamics model was developed on the basis of the work done by Gueron et al. [26]. The model follows a Lagrangian modelling approach, which avoids continuum constraints in favour of discrete individual based

modelling. The basic idea behind this approach is that group formation and dynamics is a result of sequence of decisions made by individual entities. This kind of approach is more appropriate [27] when dealing with large bodied animals, compared to Eulerian modelling approach where the individuals are expressed through units of volume. Furthermore an individual based modelling approach is more suitable for studying emergent behaviour, due to the fact that it gives opportunity to correlate the collective behaviour to the individual decisions.

The model description and specification (derived in activities 1 and 2) in terms of the properties of the individual animal, communication and environment as well as the emergent behaviour exhibited are presented in section 5.2. The developed models of the individual animal, the multi-agent model (corresponding to activities 3 and 4) and the simulation environment are presented in section 5.3.

5.1 Herd formation as emergent behaviour

Group formation is a common characteristic for many social animals. Shaw [28] examined the grouping in fish, where numerous individuals form a social aggregation known as a school. Similar aggregations can be found in birds forming flocks, as well as various species of mammals which manage to form herds [29]. Most observations made towards understanding the dynamism of a herd suggest that both small and large groups rely on local coordination between the individuals to form a herd [26]. Consequently one can argue that a herd, as an aggregation of the individuals, can be viewed as a global pattern which emerges from actions of individuals in their interaction with neighbours.

Nevertheless there are arguments, like the one put forward in [30], where the authors claim that flocking (and presumably herding as similar phenomena) is not an emergent behaviour since it lacks the element of surprise due to widespread use. Thus the authors argue that throughout the years (since 1987 when Reynolds [31] created the famous "boids" simulation), we have become aware of the underlying principles and thus the appearance of the phenomena is no longer a surprise and therefore not emergence. This view overemphasizes the observer's surprise as vital for emergence. While this approach comes close to Rosen's [32] view of emergence, it obscures the micro-macro relation as significant for appearance of the phenomenon. Thus it moves the view on emergence towards the appearance of "mystery" in the eye of the observer. Although the relation of the observer and the emergence is very intriguing (and by no means should be discarded), the authors of the paper adopt the view that emergence is a property of the system. The formation of a herd has

a functional significance to the system by decreasing the risk of predators and increasing the possibility for mating [33], which in turn affects the persistence of the herd. In addition the formation of a herd is novel in respect to the individual animals which comprise the herd. There is nothing to suggest in a single animal that when in large numbers these animals would form a herd. Therefore we argue that a herd is an emergent phenomenon due to its functional significance to the system and introduction of novelty in respect to the properties of an individual animal.

5.2 Model Description and Specification

An animal in the model is represented by an agent with two main parameters: movement speed and movement direction. In order to simplify the model only three speeds (slow, normal and fast) were taken into account. While having the same general direction, the animals have the ability to move laterally left or right. Additionally it is assumed that the model is composed out of homogeneous entities with equal body sizes and movement speeds. Therefore the movement speed and direction at time t for animal A are dependent on *A's* speed, direction and the position of *A's* neighbours at time *t-1*. While there is no explicit message exchange between the animals, they do interact with each other through modification of the environment by means of spatial repositioning. The environment is represented by a two dimensional grid-like space with uniform fields. At the particular time instance only one agent can occupy a particular space in the grid. Therefore based on the position of the neighbours an animal is faced with a set of constraints where it is able to move. Furthermore the position of the neighbouring animals is vital for the animal's decision where to move next. The approach followed in [26] defines the decision making process through evaluation of influence zones. An influence zone is a spatial area on a predefined distance relative to the individual in question. This approach assumes three zones (stress, neutral and attraction), which are depicted in figure 3. A brief explanation of the zones follows:

- The **stress** or **personal** zone, is of highest importance (compared to the other zones) and it is the primary factor which determines the actions taken by the animal. The need for individual space causes individuals to be repelled by their neighbours when their personal space is invaded [26]. Thus depending on the position of the neighbours in the stress zone the animal moves in opposite direction or increases/decreases its speed.

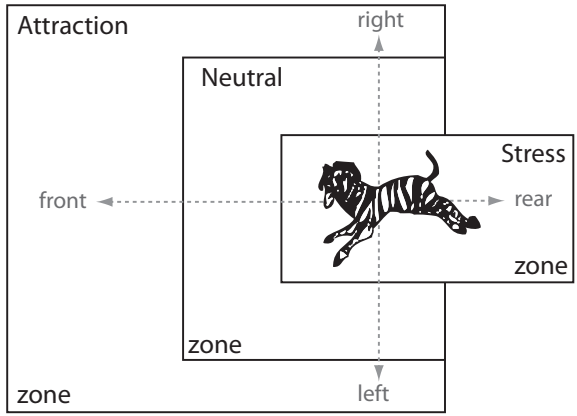

Fig. 3. Overview of the neighbour influence zones, based on [26]

- The **neutral** zone is an intermediate zone with no rear dimension. If the neighbours are in the neutral zone the animal does not react. However when all of neighbours are on the same side the animal moves toward them. This kind of behaviour according to Hamilton [33] is a display of "selfishness" as the animal is moving towards the centre of the herd in order to reduce the chance of being attacked by predators.
- The **attraction** zone influences change in the agent's direction and speed. The presence of neighbours in the attraction zone means that an animal is on the "edge" of the herd. Consequently the instinctive response for an animal is to move towards the group in the attraction zone.

5.3 Model of the individual animal and the multi-agent model

Based on the description elaborated in the previous section, the model of an individual animal was developed using X-machine formal notation. An X-machine is a general computational machine introduced by Eilenberg [34] and extended by Holcombe [35]. In many ways it looks like a Finite State Machine (FSM), however it is extended with memory. A deterministic stream X-machine [47] is an 8-tuple

$$X = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$$

where:

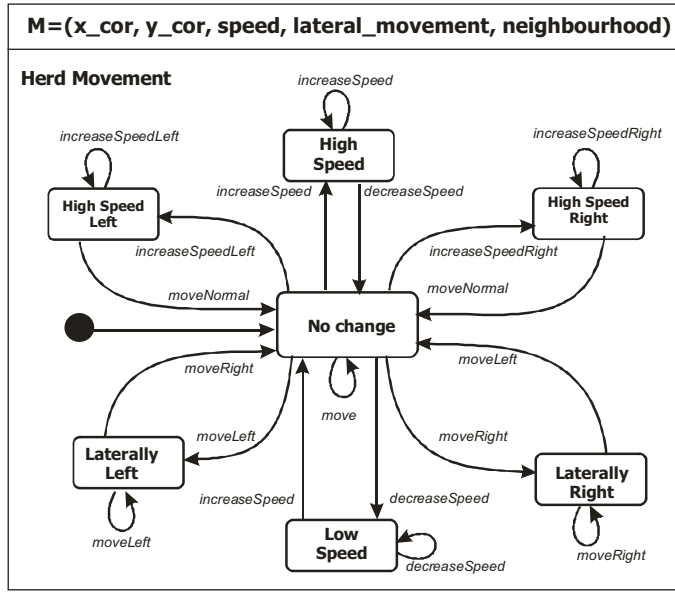

Fig. 4. Diagrammatic X-machine model of an animal in the study.

- Σ and Γ are the input and output alphabets respectively.
- Q is the finite set of states.
- M is the (possibly) infinite set called memory.
- Φ, the *type* of the machine X, is a set of partial functions φ that map an input and a memory state to an output and a possibly different memory state, $\varphi : \Sigma \times M \rightarrow \Gamma \times M$.
- F is the next state partial function, $F : Q \times \Phi \rightarrow Q$, which given a state and a function from the type Φ determines the next state. F is often described as a state transition diagram.
- q_0 and m_0 are the initial state and initial memory respectively.

A diagrammatic representation of the X-machine model of an individual animal is presented in figure 4. The model does not focus directly on the influence zones, but rather captures the internal decision making structure of the animal. The states represent the movement direction and speed at a particular time instance, while the transitions represent a change in the animal's movement speed and/or direction.

Using formal stream X-machines notation, the definition of the animal from figure 4 is the following:

- The set of inputs is $\Sigma = \texttt{zone} \times \texttt{direction}$ where

```
zone = {empty, stress, neutral, attraction}
```

and

```
direction = {none, front, left, right, rear,
  left_front, left_rear, left_right, right_front,
  right_rear, front_rear, left_front_rear,
  left_front_right, left_rear_right, right_front_rear}
```

- The set of outputs is Γ = `MESSAGES`, where

```
MESSAGES = {noChange, movingFast, movingSlow,
  movingLeft, movingRight,
  movingFastRight, movingFastLeft}
```

- The set of states is Q = {`noChange`, `highSpeed`, `lowSpeed`, `movingLeft`, `movingRight`, `highSpeedRight`, `highSpeedLeft`}.
- The memory contains the position of the animal, its speed and direction as well as the position of its neighbours at time $t-1$. Consequently M = (`x_cor`, `y_cor`, `speed`, `moving_direction`, `neighborhood`), where `x_cor` and `y_cor` are real numbers, while the `speed` = {5, 10, 15} (denoting slow, normal and fast movement speeds) and `moving_direction` = {`left`, `normal`, `right`}. Also the `neighborhood` = `zone` × `direction` which holds the zone and neighbour position.
- The type of the machine is Φ = {`increaseSpeedLeft`, `moveNormal`, `move`, ...} the labels in the transitions in figure 4.

The functions $\varphi \in \Phi$ of the model are presented in X-Machines Description Language (XMDL) [36]. Using XMDL as the modelling language, X-System (an XMDL interpreter) allows animation of X-Machine models. This enables the designer to get insight into the behaviour of the model and determine possible problems and/or test different modelling alternatives. In order to demonstrate XMDL, we present an example function `increaseSpeedLeft`, which is defined in XMDL as:

```
#fun increaseSpeedLeft( (attraction, ?direction),
                          (?x, ?y, ?s, ?d, ?n) ) =
  if ?s >= 10 and ?direction belongs {left, front_left} and
  ?d belongs {normal, left} then ((movingFastLeft),
     (?newX, ?newY, ?s, left, (attraction, ?direction)))
  where ?newY <- ?y  + 15 and ?newX <- ?x - 5.
```

Nevertheless in many cases there is a need to formally prove certain properties of the model. The formal verification technique for X-machine models enables the model designer to formally verify the developed model against temporal logic formulas which express the properties

that the system should have. For this purpose an extended version of temporal logic, named XmCTL [37], can be used. Different properties of the systems are expressed in XmCTL and then verified against the X-machine model through model checking (for more information see [49]). An example of XmCTL properties of the individual animal are presented below:

– This formula implies that there is no memory instance where the speed of the animal is less or equal to 0, or in other words that the agent is always moving.

$$\mathbf{AG}\,\mathbf{M_x}(speed > 0)$$

– There exists a computational path where in all states of this path the speed of the agent is 10 (normal) until it changes to 15 (high_speed).

$$\mathbf{E}[\mathbf{M_x}(speed = 10)\ \mathbf{U}\ \mathbf{M_x}(speed = 15)]$$

– For all computational paths when the agent is in "noChange" state and there is a neighbour in the stress zone coming from the "front" direction, the agent always reduces its speed in the next state (i.e moves to "lowSpeed" state) .

$$\mathbf{AG}\,[\mathbf{EF}\,\mathbf{M_x}(M(moving\_direction) = normal \wedge speed = 10 \wedge neighbourhood = (stress, front)) \Rightarrow \mathbf{AX}\,\mathbf{M_x}(speed = 5)]$$

Although X-machines offer intuitive modelling of an individual animal, in a case of a complete complex system the computational model will need too many states, making the entire model incomprehensible. Furthermore, depending on the complexity, in some cases the task of developing such a model seems nearly impossible. To address this shortcoming, the paradigm was further developed in order to facilitate communication between individual X-machines through Communicating X-machines [38, 39]. The Communicating X-Machines model consists of a number of single X-machine models, which are able to exchange messages. Thus the communicating approach uses decomposition in order to separate the problem into smaller more manageable components and then synthesizes the overall behaviour by specifying the communication pathways between the components. There are several variations how this is achieved; for a detailed overview refer to [40]. The main disadvantage of the Communicating X-Machines paradigm is inability to deal with dynamic restructuring of communication pathways as well as efficiently deal with non-deterministic communication. Although the approach followed in [41] promises to resolve these issues, by combining the Communicating X-Machines with ideas from P-Systems [48], at the moment

tools supporting the practical application of the approach are still under development.

Consequently for the purpose of modelling the multi-agent system as well as execution of the simulation experiments, NetLogo [42] was selected as the most appropriate platform. NetLogo allows modelling and animation of agent like entities in a simulation environment supported by a scripting language, visual animator and data output mechanisms. Besides the visual animator which is an excellent tool for observation of the model behaviour (especially in this case, when dealing with herd formation), another argument for choosing this platform is the correspondence between X-Machines model encoded in XMDL and the NetLogo scripting language. For example the "increaseSpeedLeft" function presented in XMDL above, has similar structure in NetLogo as presented below.

```
to-report increaseSpeedLeft [zone? direction?]
  if zone? = "attraction" [
  ifelse s? >= 10 and member? direction? ["left" "front_left"]
   and member? d? ["normal" "left"]
          [report (list true "movingFastLeft" (list (xcor + 15)
           (ycor + speed-lateral) 15 "left") ((zone?) (direction?)))]
          [report (list false)]  ]
  report (list false)
end
```

Nevertheless a simple translation of the individual model from XMDL to NetLogo is not sufficient in order to create the complete multi-agent model. The individual models need to be connected in order to interact together. In order to achieve this, we have developed a simulation mechanism (implemented in NetLogo) which is able to use the decision-making architecture of an animal by feeding appropriate inputs to it (based on the state of the environment) and then modifying the environment according to the output of the animal's reasoning process. Thus by utilizing indirect communication through the environment we are able to achieve interaction between multiple animals. While the developed simulation mechanism is tailored for the herd formation study, we are working on constructing a generic simulation mechanism coupled with a tool for automatic translation of XMDL to NetLogo. This will decrease the time and effort required for implementing the model in NetLogo in the case of modifying the model and reduce the number of errors in the process.

5.4 Animation and data analysis

The animation (execution) of the model is vital in synthesising the emergent properties from the individual behaviour of the animals. However in

order for this process to be useful in the context of the study, appropriate data needs to be gathered. The data needs to address the behaviour of the individuals and their interaction (the micro level) as well as the emergent phenomena (the macro-level) in both quantitative and qualitative manner. In the case study in question, gathering data about the behaviour of an individual animal is pretty straightforward as their input and output can be easily recorded for later analysis. However gathering data about the properties of the herd formations is much more complicated. In fact in order to achieve this, there is a need for a mechanism that will continuously detect (and evaluate) groups of animals and their dynamics as the model is being animated.

We believe that the most natural and straight forward way to detect herd formations is through visual inspection of the spatial distribution of the animals. While other approaches like meta-modelling may be applicable, this approach has a major advantage of being independent from the specific modelling technique and the language in which the model is encoded. In this manner the detection of herds is essentially a spatial clustering problem, where the goal is to group similar objects into clusters, so that the elements of the cluster have similar properties (ie. they spatially close). When dealing with a clustering problem, in most of the cases there are clearly defined criteria which allow differentiation of clusters. However in the case of herd formation, the criteria for differentiating between different groups of animals are far from clearly defined. The main problem is the inability to arrive at a formally quantifiable definition of a herd which will be generally accepted in different contexts by all possible observers. For example one might say a group of animals needs to be reasonably close together and have sufficient members in order to form a herd. However these criteria guided by "reasonably" and "sufficient" quantities are unclear and ambiguous. This is the main reason why fuzzy reasoning approach is a perfect candidate for developing the herd detection mechanism. Furthermore fuzzy reasoning allows utilization of inherently inexact concepts in the way humans differentiate herds, within an automated computer based process.

It has to be noted that our approach is quite different from classic fuzzy clustering approaches (for more information on fuzzy clustering see [43, 44]). We apply concepts from fuzzy set theory in order to deal with the imprecise nature of the herd clustering criteria rather than using it to express fuzzy membership to multiple clusters. In the developed recognition scheme each entity is assigned to a single cluster and does not possess membership levels to multiple clusters. Furthermore, since in our study the emphasis is on herd formation as an emergent phenomenon, our approach combines bottom-up and top-down directions as part of

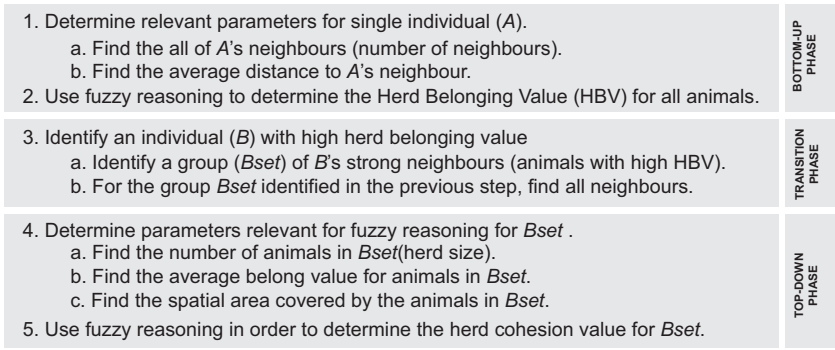

Fig. 5. The developed reasoning algorithm for automated herd detection

a two-way reasoning process. A brief overview of the developed reasoning algorithm is presented in figure 5. Although the operations in the reasoning process are sequentially interconnected, they can be logically divided into three major phases.

- The first phase is the bottom-up reasoning phase, which aims to evaluate the preference of an individual animal to be part of the herd. This reasoning process is depicted by steps 1 and 2 in figure 5, and it is primarily concerned with identification of the individual preferences for an animal to be part of a herd. Towards this end, two major factors (properties) of an animal are taken into account: the average neighbour distance and the number of neighbours. The result of the reasoning process, called "Herd Belonging Value" (HBV), denotes the animal's preference to be part of a herd.
- The second phase of the reasoning process, depicted by step 3 in figure 5, is the transition from the evaluation of the individual animal, towards reasoning about a group of animals. The identification of the group is primarily depended on the animal's preference to be part of the herd (HBV). The HBV is used to determine a set of animals which form the core of the herd. Once the core group is identified all neighbouring animals in the area are added to form a group which will be evaluated in the next phase.
- Once the groups potentially forming herds are identified, the reasoning process continues with the evaluation of the group coherence. This is in fact the top-down reasoning phase which is depicted by steps 4 and 5 in figure 5. In addition to the group's average HBV, also the group size and the spatial area occupied are taken into account as input parameters into the reasoning process. The reasoning

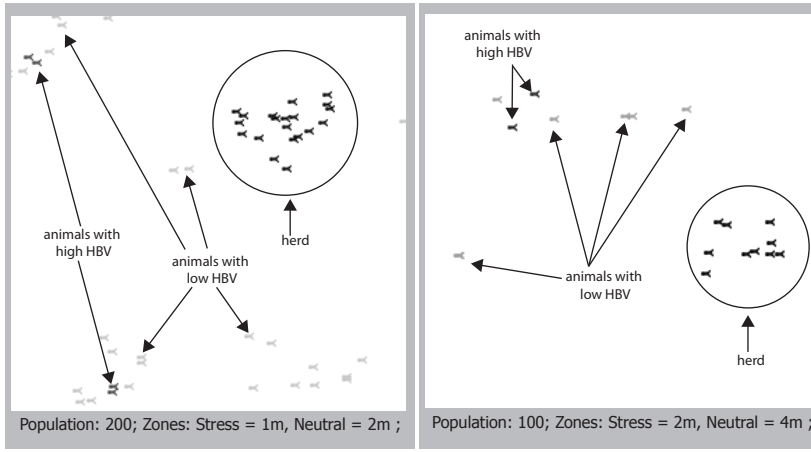

Fig. 6. Partial screen capture of the herd recognition during model execution. Labels and arrows are added for presentation purpose.

output variable called herd cohesion is an indication of the strength of the herd. If the value of this variable is above a threshold defined by the investigator the group is visualized as a herd.

The discussed design of the fuzzy reasoner was implemented as a Java-based extension to the Netlogo simulation model. It uses the FuzzyJ libraries [45] in order to support the fuzzy reasoning process. The evaluation of the reasoner was done in two steps. First of all, the implementation of the reasoner was tested in order to evaluate the influence of a single input variable on the output variable for both herd belonging value and herd cohesion variables. The results of this testing led to revisions in the fuzzy set distribution for certain input variables.

The second step of the evaluation was focused on the reasoner's ability to detect herd formations during the execution of the model. A major constraint in this process was the fact that clear-cut "correct" output could not be determined due to the nature of the herd detection problem. Consequently the intuition of the investigation team on herd formation patterns was used as basis for comparison. Thus the process was based on an estimate of how closely the developed reasoner matched what the investigators considered to be a herd in a given context.

In simulation scenarios with near optimal values the reasoner performed well. This is shown in the screen captures presented in figure 6. It managed to clearly differentiate between loose groups and herd for-

mations. However, several simulation scenarios revealed problems when there was an extreme increase or decrease in the animal population. In order to resolve the issue there is a need to incorporate a population density function as part of the reasoning process. To this end there is ongoing work to express the scales of the fuzzy variables through a function of the population density.

In addition to the herd detection mechanism, there is also ongoing research in developing methods for automatic invariant detection in the simulation data. An invariant is an internal property of a particular software program, model, or a design that is true at the particular point (or points) during the program is execution [46]. Invariants can be very useful as part of software engineering, however at the same time they can be used in order to gain insight into hidden parts of a simulation model design. By finding relations between different events occurring during model execution, a dynamic invariant detector can be a useful tool in finding correlations between micro and macro behaviours.

The two discussed tools (for automated detection of herd formations and dynamic invariant detection), are vital in carrying out the data analysis process.

6 Conclusions

In recent years the phenomenon of emergence, as one of the fundamental properties of complex systems, managed to capture significant attention from the scientific community. There are several reasons behind this development. First of all, emergence seems to be everywhere in nature. It appears in different forms and shapes in a variety of systems from simple to the most complex. It is responsible for a variety of fascinating properties and behaviours. The ability to engineer emergent phenomena can be very beneficial in many areas of science and technology. For example, from computer science perspective, utilization of emergent phenomena like self-organization and adaptation can significantly advance the development of computer systems. On the other hand, emergence can also be viewed as negative phenomena, it can significantly infringe the functional performance of engineered systems. This prospect is especially concerning since there is an ongoing trend in engineering open distributed systems with growing complexity. Nevertheless at the moment there are no studies dealing with the practicalities of constructing a framework (comprised of well defined process supported by a set of practices and tools) which will guide the analysis of emergent formations in existing systems.

We address this issue by proposing a framework for empirical exploration of emergent formations. The core idea is to offer a structured

approach which utilizes iterative multi-agent simulation as means for experimental examination of emergent manifestations. The goal of the process is to gradually increase the understanding of the causal relations between the individual (micro) and emergent (macro) levels of the system under study.

Nevertheless such a study can only yield beneficial results when it is applied on a "correct" model of the system. Therefore ensuring the correctness of the model is of paramount importance. In our view, the best way to deal with this issue is to use formal techniques to verify the correctness of the individual components (agents). The formal validation of the complete multi-agent model on the other hand is often impossible (due to complexity) or too expensive in terms in time and effort. Consequently we propose to validate the model through iterative simulation and refinement. The second phase is focused on experimental analysis of the emergent behaviour. It is essentially an analytical process aimed at detection of causal relations in the model, through evaluation of hypothesis about the expected behaviour of the model under certain conditions. Thus the main idea is to incrementally identify the causal relations between the behaviour of the agents at the micro level and the emergent phenomenon at the macro-level.

In addition to the proposed framework, this paper also discusses an ongoing work in a herd dynamics study. In this context the paper elaborates the theoretical basis for the model, the specification and the corresponding models of an individual animal as well as the multi-agent model and simulation environment. Furthermore a solution to the problem of automatic herd detection is presented as a required initial step in the analysis of the model behaviour.

Since the presented case study is work in progress, the immediate research focus in the upcoming period will be to address issues leading to its completion. To this end, there are several ongoing tasks:

- Revision of the herd detection mechanism in order to incorporate population density functions.
- Development of a tool for automatic translation of XMDL to NetLogo.
- Development of methods for automated invariant detection.

Once this work is completed, the study will continue with the validation and verification phase of the study. During this process the model will be refined until it posses the required level of detail and exhibits the expected behaviour. After that the study will be moved to the second phase where the casual relations in the model will be examined as described in the second phase of the framework.

Upon the completion of the study the proposed framework will be evaluated and revised accordingly. Following this, future work will also include development of a second case study, exhibiting more complex emergent phenomena. While the herd dynamics case study was primarily used as a simple example of emergence where the framework could be developed and revised based on the practical experience, the aim of the second case study will be to test the ability of the proposed process and practices to guide the process of detecting emergent misbehaviour in a distributed computer based systems.

References

[1] Bullock S. and Cliff D., "Complexity and emergent behaviour in ICT systems," Hewlett Packard, Tech. Rep. HPL-2004-187, 2004.
[2] Mogul J. C., "Emergent (Mis)behavior vs. Complex Software Systems," HP Laboratories Palo Alto, Tech Rep. HPL-2006-2, 2006.
[3] Wegner P., "Why interaction is more powerful than algorithms", in Communications of ACM, 1997, vol. 40(5), pp. 80-91.
[4] Edmonds B. and Bryson J., "The insufficiency of formal design methods - the necessity of an experimental approach - for the understanding and control of complex MAS", In: Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems, 2004, pp. 938-945.
[5] Edmonds B., "Using the experimental method to produce reliable self-organised systems", In: Proceedings of the 2nd International Workshop on Engineering Self-Organising Applications (ESOA 2004), 2004, pp. 84-99,.
[6] De Wolf T. and Holvoet T., "Towards a methodology for engineering self-organising emergent systems", In: Self-Organization and Autonomic Informatics (I), H. Czap, R. Unland, C. Branki, and H. Tianfield (ed.), Frontiers in Artificial Intelligence and Applications, 2005, vol.135, pp.18-34.
[7] Anderson P. W., "More is different," Science New Series, 1972, vol. 177(4047), pp. 393-396.
[8] Bar-Yam Y., "Dynamics of Complex Systems," Perseus Books, 1997.
[9] Chalmers D.J., "Varieties of Emergence," Department of Philosophy, University of Arizona, USA, Tech. rep./preprint, 2002.
[10] Simon H. A., "The Architecture of Complexity: Hierarchic Systems," In: Proceedings of the American Philosophical Society, 1962, vol. 106, pp.467-482.
[11] Bedau M., "Weak emergence," Philosophical Perspectives: Mind, Causation, and World, 1997, vol. 11, pp. 375-399.
[12] Holland, J. H., "Emergence: From Chaos to Order," Reading, MA: Addison-Wesley, 1998.
[13] Jones S., "Organizing Relations and Emergence," In: Proceedings of the Artificial Life VIII: The 8th International Conference on the Simulation and Synthesis of Living Systems, 2002, pp. 418-422.

[14] Goldstein, J., "Emergence as a Construct: History and Issues," Emergence: Complexity and Organization, 1999, vol. 1, pp. 49-72, .
[15] Fromm J., "Types and forms of emergence," Complexity Digest, vol. 25(3), 2005.
[16] Wolfram S., "Universality and complexity in cellular automata," Physica D: Nonlinear Phenomena, 1984, vol. 10(1-2), pp. 1-35.
[17] Conte R. and Castelfranchi C., "Simulating Multi-Agent Interdependencies: A Two-Way Approach to the Micro-Macro Link," In: Social Science Microsimulation, Troitzsch K. G., Mueller U., Gilbert N. and Doran J. (eds.), Springer-Verlag Berlin, 1996, pp. 394-415.
[18] Fromm J., "Ten questions about emergence," Complexity Digest, vol. 40(15), 2005.
[19] Jennings N. R. and Wooldridge M., "Intelligent agents: theory and practice," Knowledge Engineering Review, 1995, vol. 10(2), pp. 115-152.
[20] Sargent R. G., "Verification, validation, and accreditation of simulation models," In: Proceedings of the 2000 Winter Simulation Conference, 2000, pp. 50-59.
[21] Balci O., "Guidelines for successful simulation studies," In: Proceedings of the Winter Simulation Conference, 1990, pp. 25-32.
[22] Eleftherakis G., Kefalas P., Sotiriadou A. and Kehris E., "Modelling biology inspired reactive agents using X-machines," In: Proceedings of the International Conference on Computational Intelligence (ICCI04), 2004.
[23] Kehris E., Eleftherakis G. and Kefalas P., "Using X-machines to Model and Test Discrete Event Simulation Programs," Systems and Control: Theory and Applications, N. Mastorakis (ed.), World Scientific and Engineering Society Press, 2000, pp. 163-168.
[24] Whitner R. B.and Balci O., "Guidelines for selecting and using simulation models verification techniques," Department of Computer Science, Virginia Polytechnic Institute and State University, Virginia, Tech. Rep. TR-89-17, 1989.
[25] Conte R. and Castelfranchi C., "Simulating multi-agent interdependencies. A two-way approach to the micro-macro link," In: Social science microsimulation, 1995, pp. 394-415.
[26] Gueron S., Levin A. and Rubenstein D. I., "The Dynamics of Herds: From Individuals to Aggregations," Journal Theoretical Biology, 1996, vol. 182, pp. 85-98.
[27] Okubo A., "Dynamical aspects of animal grouping: Swarms, schools, flocks and herds," Advanced Biophysics, 1986, vol. 22, pp.1-4.
[28] Shaw E., "Schooling fishes," American Scientist, 1978, vol. 66, pp. 166-175.
[29] Underwood R., "Vigilance behaviour in grazing African ungulates," Animal Behaviour, 1982, vol. 79, pp. 82-107.
[30] Ronald M.A. E., Sipper M. and Capcarrere M. S., "Design, Observation, Surprise! A test of Emergence," Artificial Life, 1999, vol. 5(3), pp. 225-239.
[31] Reynolds C.W. , "Flocks, herds, and schools: A distributed behavioral model," In: Computer Graphics, pp. 25-34, 1987.

[32] Rosen R. , "Anticipatory systems," Pergamon Press, New York, 1985.
[33] Hamilton W. D., "Geometry for the selfish herd,", Journal of Theoretical Biology, 1971, vol. 31, pp. 295-311.
[34] Eilenberg S., "Automata, Machines and Languages," Academic Press, New York, 1974.
[35] Holcombe M., "X-machines as a Basis for Dynamic System Specification," Software Engineering Journal, 1988, vol. 3(2), pp. 69-76.
[36] Kefalas P., "XMDL user manual: version 1.6.," Dept. of Computer Science, CITY College, TR-CS07/00, 2000.
[37] Eleftherakis G., Kefalas P., and Sotiriadou A., "XmCTL: Extending Temporal Logic to Facilitate Formal Verification of X-machines Models", Analele Universitatii Bucuresti, Matematica-Informatica, 2001, vol. 50, pp. 79-95.
[38] Kefalas P., Eleftherakis G. and Kehris E., "Modular System Specification using Communicating X-machines," Dept. of Computer Science, CITY College, TR-CS11/00, 2000.
[39] Kefalas P., Eleftherakis G., and Kehris E., "Communicating X-machines: from theory to practice", In: Advances in Informatics, Y.Manolopoulos, S.Evripidou, A.Kakas (Eds), LNCS 2563, 2003, pp.316-335.
[40] Aguado J. and Cowling A., "Systems of Communicating X-machines for Specifying Distributed Systems," University of Sheffield, Research Report CS-02-07, 2002.
[41] Stamatopoulou I., Gheorghe M. and Kefalas P., "Modelling of Dynamic Configuration of Biology-Inspired Multi-Agent Systems with Communicating X-machies and P Systems," In: Proceedings of the 5th International Workshop in Membrane Computing (WMC'04), Milan, Italy, June 14-16, 2004.
[42] Wilensky U., "NetLogo 3.1.3 User Manual," Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL, 1999.
[43] Bezdek J., "Pattern Recognition with Fuzzy Objective Function Algorithms," Plenum Press, 1981.
[44] Baraldi A. and Blonda P., "A survey of fuzzy clustering algorithms for pattern recognition," International Computer Science Institute, Berkeley, CA, TR-98-038.
[45] National Research Council of Canada's Institute for Information Technology, "Java library for Building Fuzzy Systems (FuzzyJ Toolkit)," Available online at: `http://iit-iti.nrc-cnrc.gc.ca/projects-projets/fuzzyj_e.html`.
[46] Ernst M. D., Perkins J. H., Guo P. J., McCamant S. , Pacheco C., Tschantz M. S. and Xiao C. , "The Daikon system for dynamic detection of likely invariants," Science of Computer Programming, 2007.
[47] Ipate F. and Holcombe M. , "Specification and testing using generalised machines: a presentation and a case study," Software Testing, Verification and Reliability, pp. 61-81, 1998.
[48] Păun Gh., "P systems with active membranes: attacking NP compete problems," Automata, Languages and Combinatorics, 2001, vol. 6(1), pp. 7590.

[49] Eleftherakis G., "Formal verification of X-Machine models: Towards formal development of computer based systems", PhD Thesis, Department of Computer Science, University of Sheffield, UK, 2003.

Engineering Safety-Critical Complex Systems

Robert Alexander[1], Ruth Alexander-Bown[2], Tim Kelly[1]

[1] Department of Computer Science
University of York, York, YO10 5DD, UK
{robert.alexander, tim.kelly}@cs.york.ac.uk
[2] Royal United Hospital, Bath
ruth@alexander-bown.com

Abstract. Some of the complex systems with which the CoSMoS project is concerned are safety-critical, and if such systems are ever to be built and operated then they will need to be certified safe to operate. By looking at how conventional safety-critical systems are developed, we can find basic principles for safety-critical complex systems – this may be harder or easier than non-safety-specialists expect. In this paper, we outline current safety engineering methods and illustrate them using an artificial platelet case study. We also summarise our previous work on using simulation in safety engineering, and make some observations about applying simulation to very small systems.

1 Introduction

The CoSMoS project [6] is concerned with the engineering of complex systems. Some proposed complex systems are safety-critical – they have the potential to cause human injury or loss of life. Engineers developing safety-critical systems must meet a range of legal requirements, and to achieve this they must use considerable safety-specific technical knowledge. To outsiders (used to the engineering, particularly software engineering, of non-safety-critical systems) the methods used in safety-critical systems engineering may seem arcane, and in places even primitive or backwards. Nevertheless, failing to follow these methods may lead to the development of systems that can never be used because of safety concerns.

Sections 2 and 3 of this paper provide a primer in current safety-critical systems thinking for complex system researchers and developers.

This is not a complete solution to safety concerns, or a cookbook approach to safety engineering. It does, however, provide the non-safety-expert with a starting point for appreciating the specific challenges that safety-critical complex systems pose. Equally, it should provide some insight into what is *not* needed under current safety regimes, and thereby open up avenues that may have seemed impractical before.

In Section 4, we have taken the Artificial Platelet (AP) system used as a case study by the TUNA project [30] and performed an initial hazard identification on the concept. We then use this to illustrate a number of the safety engineering issues raised in the previous section, and to highlight some of the problems that complex systems present.

To supplement the above, and with particular relevance to the simulation aspects of the CoSMoS project, Section 5 reviews previous work by some of the authors on simulation-based analysis of complex Systems of Systems (SoS). This illustrates how simulation techniques can move engineered systems from the unmanageable to the potentially tractable. Some of the extant challenges faced by this work will be equally (or more) salient to the CoSMoS project.

Finally, Section 6 discusses some of the unique aspects of engineering and simulation for very small artefacts.

The emphasis throughout is on the practical implications for complex system engineering (in terms of practical needs) and research (in terms of research objectives).

2 The Need for Safety Engineering

Safety engineering is relevant to a great many endeavours; specifically, any situation where human injury or death is possible. In the UK, the legal requirement to perform safety activities stems from the Health and Safety Executive and the 1974 Health and Safety at Work act [9]. For example, the HSE imposes a duty of care on employers with respect to the safety of their staff.

Certain industries, such as air transport and nuclear energy, are explicitly *regulated*, and specific legal duties are imposed on manufacturers and operators. These go beyond the common requirements imposed by the HSE; for the most part, the regulations are concerned with preventing major accidents. A recent overview of the relevant safety regulations for a number of different industries can be found in Jones [10]. A common theme is that before an installation or product can be operated or sold, it must be *certified* – an independent safety assessor must agree that it appears to be adequately safe.

Safety in regulated industries is governed by *safety standards* – documents that lay down what procedures must be followed in order to claim adequate safety (and thereby achieve certification). These standards are (generally) industry specific. For example, DO-178B [27] covers the safety of aircraft software, while Def Stan 00-56 [32] sets safety requirements for all UK military equipment.

Traditionally, most standards were *prescriptive*; they laid down a set of procedures to follow and processes to perform. If a product developer followed the process, they could claim an adequate safety effort. For example, a software-related standard might mandate code reviews, MC/DC test coverage, and the use of a programming language subset designed to eliminate common coding errors. Examples of prescriptive, process-based standards are DO-178B and the obsolete Def Stan 00-56 Issue 2 [31].

Prescriptive standards are problematic for two reasons. First, they present a safety problem: there is not particularly strong reason to believe that merely following any given process to the letter will necessarily lead to a safe system. Second, they strongly restrict the technologies (e.g. algorithms and hardware) that can be certified as safe. This second problem would prevent the certification of the complex systems with which the CoSMoS project is concerned.

Because of the problems with prescriptive standards, there has been an increasing move towards *goal-based* safety standards. Such standards define the measure of safety that is required (for example, expected lives lost per operating hour), and the evidence of safety that must be supplied, then require system developers to present a structured argument that their system will meet the required level of safety. Rather than mandating processes or technology choices, goal-based standards set the bar for safety achievement and put the onus on system developers to provide evidence of safety. Provision of such evidence for complex systems is of obvious relevance to the CoSMoS project. Examples of goal-based standards include Def Stan 00-56 Issue 4 [32], which is particularly clear and straightforward.

Safety thinking, particularly in its goal-based form, is making inroads into other domains. For example, the ISO is developing an Assurance Case standard which will lay down standards for arguing non-safety properties of software systems. In the long term, it may be that standards for arguing dependability (the umbrella term that includes safety, security and other attributes – see Despotou in [8]) become widespread across many engineering domains.

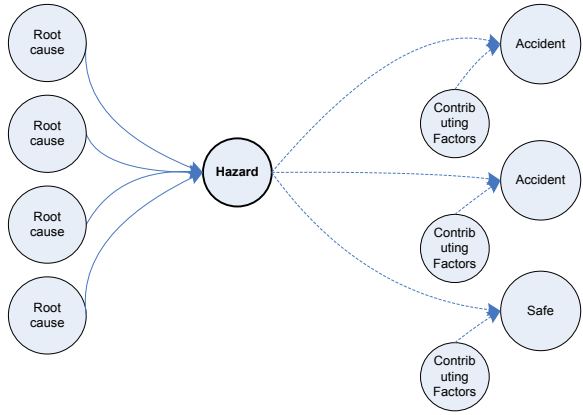

Fig. 1. Causes, Hazards and Accidents (reproduced from [24])

3 The Basics of Safety Engineering

In this section, we give an overview of how a system developer can perform safety engineering and end up with a system that is safe to operate. It is written with the assumption that there is a goal-based safety standard in place that the developer must meet, but we feel that this overall approach is a strong one even where there is no legal requirement to follow it. We can only give a very general outline here – anyone performing safety engineering will need to use (or develop) a lot of other domain-specific guidance.

3.1 Hazards and Risk

At the very core of safety engineering is concept of the *hazard*. In its simplest sense, a hazard is a state which it is dangerous for the system to be in. Def Stan 00-56 defines a hazard as *"A physical situation or state of a system, often following from some initiating event, that may lead to an accident."* [32].

An example hazard for a car is *"Loss of steering control"*. An example for a chemical plant is *"Tank temperature exceeds flash point of contents"*. There is no hard rule for when one hazard is sufficiently distinct from another – hazards are a view that we impose on a system in order in order to manage the huge numbers of possible accidents and causes of those accidents (see Figure 1 for an illustration of this).

The process of identifying hazards is called *hazard identification*, and the process of deriving causes and consequences (accidents) for hazards

is known as *hazard analysis*. These activities need to happen early in any safety-critical engineering project, and should be updated throughout the lifecycle (even during operation). A variety of techniques exist for guiding engineers through this process, such as Functional Failure Analysis (FFA) (see Pumfrey in [24]) and HAZOP (see Kletz in [13]).

3.2 Predictive Analysis

Once hazards have been identified, we need to determine how often they will occur and what (how severe) the consequences will be if they do. The severity of each hazard needs to be measured in some standard way – several classification systems exist, but many can be seen in terms of mathematical expectation of loss of life (or a combination of that and the expectation of monetary cost). The combination of probability and severity is known as the *risk* posed by the hazard, and the combined risk from all hazards provides the *total system risk*.

A convincing prediction of hazard probability and severity is critical for safety certification. As noted in Section 2, certification requires that a developer show that the risk posed by the system is acceptable, and prediction of hazard risk is the centre of that.

3.3 Safety Requirements

In many cases, the inherent hazards in a system will make it unacceptably dangerous in its "naked man" (no safety measures) form. From hazard analysis, engineers will have to derive a number of additional system requirements that, if met, will reduce the risk posed by a hazard. These *safety requirements* then drive later safety engineering activity, and obviously have knock-on effects as they interact with requirements from other sources (e.g. the system's basic functional and performance requirements).

Safety requirements may be qualitative (*"If a wheel locks, then the anti-lock braking system must engage"*) or may be quantitative (*"The probability of a pressure relief valve jamming shut must be no more than* 1×10^{-4} *per operating hour"*).

In a goal-based safety regime, how safety requirements are met is up to the developer. The key, however, is that the developer must provide evidence that the requirements are adequate (that they really will provide the desired increase in safety) and that they have actually been achieved in the implemented system. The level of confidence needed will depend on the level of risk posed by the system's identified hazards – more dangerous systems require correspondingly better evidence.

Even if a requirement is actually adequate and is actually met (from the perspective of some hypothetical all-seeing observer), it may still be that the system developer cannot convincingly show this. This is particularly likely for novel systems and novel technologies. The simulation and analysis methods proposed by the CoSMoS project are clearly relevant here.

3.4 The Safety Case

A safety case is a document that presents a compelling structured argument that a system is safe, for some explicit definition of "safe". It is on the basis of the safety case that a regulator (or other independent safety assessor) approves a system for safety certification. For example, the required standard of safety might be expressed in terms of "Expected loss of life per operating hour".

There are several ways to structure and express a safety case. A plain natural language presentation can be used, or a tabular format, although there is increasing use of explicit argument structure notations such as the Goal Structuring Notation (GSN) (see Kelly in [11]).

Below the level of notation, a safety case must be organised in some systematic way so that the completeness and adequacy of the argument can be assessed. Several different *safety argument patterns* are presented in [11]. Perhaps the simplest structure is a breakdown by identified hazards – such a case would argue that all hazards had been identified, that all hazards had been adequately mitigated, and that the combined risk from all mitigated hazards was acceptable. The rest of this paper will assume such a structure.

The standard of argument in a safety case is *informal*, in the manner described by Toulmin in [29]. Formal notations and formal proof are emphatically *not* required – indeed, we are a very long way from a formal language that could adequately express all the diverse aspects of a safety case. Rather than proof, a safety case seeks to establish adequate *confidence* that the safety argument is sound. Specifically, it needs to provide evidence that the safety requirements derived from hazard analysis are met.

The level of confidence required for any particular claim depends on the system and the claim. Def Stan 00-56 makes this explicit: the required confidence depends on the risk posed by the system, its complexity and the degree of novelty (either in terms of technology or application). Specifically, it states: *"The quantity and quality of the evidence shall be commensurate with the potential risk posed by the system and the complexity of the system"* [32] – the novelty issue is presented in an additional diagram (the "McDermid Square"). "Complexity" here is not

in the CoSMoS sense – rather, it means merely "complicatedness". (In any case, this is contingent on the developer's ability to understand and manipulate the system; it is in some sense "analysability".)

One part of any safety case will be the claim that the derived safety requirements are adequately complete. Typically, the developer will assert that all hazards have been identified, and that the set of requirements derived from each is adequate. Such claims can be difficult because they involve assertions about the (absence of) the unknown. They are, nevertheless, absolutely essential – if there is a missing hazard (or a missing cause of a known hazard) then the estimated total system risk could be very wrong.

In particular, a simple claim of adequate *effort* in hazard analysis is not sufficient – if the system being argued about is extremely complicated, and uses technologies that are poorly understood, then it may not be possible to make an adequate completeness claim. It may, therefore, not be possibly to certify the system – the developer may have to go back to the drawing board, changing the design or concept until it is tractable to analysis.

3.5 Operational Safety Management

Once a system has a safety case, has been certified, and is in active use, it still needs active management if it is to remain safe. The safety level of the system needs to be monitored and maintained. This *operational safety management* is critical because this is the point where prior safety-related modelling and analysis encounters the reality of the system in its real operating environment. The practical evidence coming from real operation has far higher confidence than the evidence from prior models or pre-operation tests.

System operators can and should track the accident and incident rates for the system. This can be used to estimate the actual safety risk of the system. Such *safety performance management* allows us to assess whether the risk levels claimed in the safety case are accurate. When a discrepancy is found, the developer must re-assess the safety of the overall system in the light of this change, and may need to change the design or operational procedures to restore an adequate level of safety. When this happens, the safety case should be updated; a safety case should be a *living* document.

3.6 Conservatism

A common theme throughout safety engineering (and indeed, throughout all safety-critical industries) is *conservatism*. Safety-critical systems

are built using well-established technologies, methods and tools, often after similar non-safety developers have moved on. System developers are often extremely reluctant to change their development processes, architectural assumptions or even programming languages. In some sectors it is difficult or impossible to certify systems using technologies that have been accepted for decades in non-safety domains (for example, it is currently difficult to use software control in a UK nuclear energy system because the regulators will not accept it).

Conservatism in terms of technology is often motivated by the practicality of convincing analysis or by perceived protection from human error. Determinism is also a factor – if it is hard for an engineer to predict the precise behaviour of a system component at a critical time, then it is hard for them to claim that it meets any associated safety requirement.

As an example of conservatism and its motivation, consider memory management in software. Many software developers working in non-safety industries use languages such as Java and platforms such as the Java Virtual Machine that provide garbage collection, and therefore automatically reclaim dynamically-allocated memory. By contrast, software developers who are working on safety-critical software are generally unable to dynamically allocate memory at all (it is prohibited by the MISRA C subset [19] and simply not supported by the SPARK language [4].) They are, in a sense, two whole paradigms behind.

It is *not* true that dynamic memory allocation is fundamentally incompatible with safety. It is, however, a notorious source of programmer errors and very difficult to statically analyse. In particular, it is difficult to assert that the memory needed by the software will never exceed the amount that is available. By contrast, if a software program only uses static memory allocation, it is easy to determine how much memory is required.

Garbage collection provides some protection from programmer error (the classic case of losing all pointers to an allocated memory block before releasing it, thereby causing an unrecoverable "memory leak") but does nothing to bound the worst-case memory usage of a program. In addition to this, most garbage collection schemes are non-deterministic; it is impossible to predict exactly when the garbage collector will be called or how long it will take.

Conservatism will be a major obstacle to the creation of safety-critical complex systems. This will be both in its intuitive, "gut-feeling" form (complex systems are strange architectures implemented using strange algorithms on strange substrates) and in its well-reasoned form (if we can't predict the worst case behaviour of a particular emergent algo-

rithm, then we may be forced to eschew it in favour of a conventional, centralised algorithm that we *can* understand – even if the average performance is much worse).

4 Safety of Complex Systems

To illustrate some safety activities in the context of a complex system, we will use the Artificial Platelet (AP) system. The AP system was used as a case study by the TUNA project (see [28] and [26]). It provides a set of artificial platelets that can be injected into a patient's blood stream to speed up wound clotting (for example after a serious accident). Obviously, this has inherent safety risk, so safety engineering is essential.

Section 4.1 gives an initial identification of accidents and hazards for the AP system concept, and Section 4.2 discusses the architectural implications of some of these hazards. Section 4.3 then shows how a safety case could be developed from this starting point.

4.1 Initial Accident and Hazard Identification for the AP Concept

As noted in Section 3.4, it is critical that safety engineers identify all the hazards that a system can exhibit. The set of hazards is unique to the particular system being developed – it may not be shared with other systems using the same technologies or approaches.

For well-established types of systems (such as cars and chemical plants) the types of accidents that can occur are well understood (e.g. crashes and explosive reactions). We can therefore move straight to indentifying hazards. The AP system is highly novel, however, and the set of possible accidents is *not* obvious. We must, therefore, identify a set of accident scenarios before we can identify all hazards.

Tables 1 and 2 show the output of an initial accident and hazard identification for the AP concept, developed by brainstorming between the authors (two academics specialising in safety engineering and a medical doctor). This method (safety engineers brainstorming with domain experts) is a common and important way to start the identification of hazards.

The accident scenarios presented in Table 1 are events occuring at a relatively large scale, and are events that we care about directly (they are events that have very direct consequences for the survival and future health of patients). The hazards in Table 2 are events at the level of the AP system that are not necessarily of immediate concern, but which

ID	Description	Notes / Causes & Consequences
A1	AP fail to form clot	If not announced may delay other treatment.
A2	Clot is too short-lived	Human platelet transfusions have a very limited lifespan and are often used in people who have low platelets who are actively bleeding, or alternatively, immediately prior to an operation. Temporary clotting is of little value and AP would need to at least match the current human platelet lifespan.
A3	Clot blocks blood vessel	Could cause stroke, coronary thrombosis or potentially a pulmonary embolism. Announcement required at *inter-platelet/multi-agent* level.
A4	AP cause Disseminated Intravascular Coagulation (DIC) – widespread clots form in multiple vessels	Usually fatal. Very poorly understood phenomena – root cause is not known, but platelets are involved in the mechanism. Unregulated clotting occurs, which leads to haemorrhage as clotting factors and platelets are subsequently used up.
A5	Allergic reaction to AP	May be immediate or delayed. Effect could range from minor symptom (e.g. rash) to fatal anaphylaxis.
A6	AP act as infection vector	Particularly dangerous given that patient is already in poor health.
A7	AP damage permeable membrane	AP may interact differently in response to the body's permeable membranes such as in the kidneys filtering system. It could cause an obstruction or pass through the kidney filtering system inappropriately. This could damage the filtering apparatus and lead to renal impairment or even organ failure.
A8	AP prevent secondary haemostasis	May prevent formation of stable, long-term clot.

Table 1. Identified AP Accident Scenarios

could lead to one or more of the identified accident scenarios. In particular, they could lead to accidents *without anything else going wrong*. Note how one hazard may lead to multiple accident scenarios. Indeed, it is often possible to group many similar accidents under a smaller number of hazards; this may help to keep the safety case manageable (this was shown diagramatically in Figure 1).

A note on immune response: Hazard H2 and accident scenario A5 may be caused by an immune response, and because the immune system

ID	Description	Notes / Causes & Consequences
H1	AP go somewhere they shouldn't	e.g. crossing the blood-brain barrier. May cause A3 or A7.
H2	AP destroyed by immune system	May cause A1 or A2. Detection and announcement may be difficult.
H3	AP lifespan too short	May cause A2.
H4	AP form oversized clot	May cause A3.
H5	AP contaminated with infectious agent	May cause A6. May depend on platelet storage arrangement – might need to be stored at or near body temperature. Bacterial infection most likely in this scenario.
H6	AP have unexpected interaction with permeable membrane	i.e. their interaction is unlike that of natural platelets. May cause A7.
H7	AP fail to release mediators for secondary haemostasis	May cause A8 (see clotting cascade diagram in Figure 2).

Table 2. Identified AP Hazards

learns this may become worse over time. If an immune response did occur, the first application of AP to a given patient would have no visible ill effects, while the second might have very severe ones.

It is important to remember that safety engineers aren't very worried about hazards that *always* happen – such hazards will become obvious before long. Their prime concern is with hazards that will rarely manifest, but not so rarely as to be of no concern. A common baseline (in process plants and aviation) is that all hazards combined should cause a fatality no more than once every 10^6 years, but this varies widely.

One important variant of this is finding hazards that will manifest often under certain circumstances – in the AP case, this might mean only for some patients, only when administered in combinations with some drugs, or only for patients with some other medical condition.

H5 (AP contaminated with infectious agent) shows how performing safety analysis early can save effort (and maybe save your engineering project). If you know that preventing bacterial growth on AP (or in their storage medium) is critical, then you can take steps to achieve this. If you *don't* know that this is critical until late in the project, then you might have made design decisions that conflict with this. For example, your AP might have to be stored at human body temperature (which is ideal for bacterial growth).

It is important to remember that you don't have to argue safety *in all possible situations*, provided that you can argue that you know when

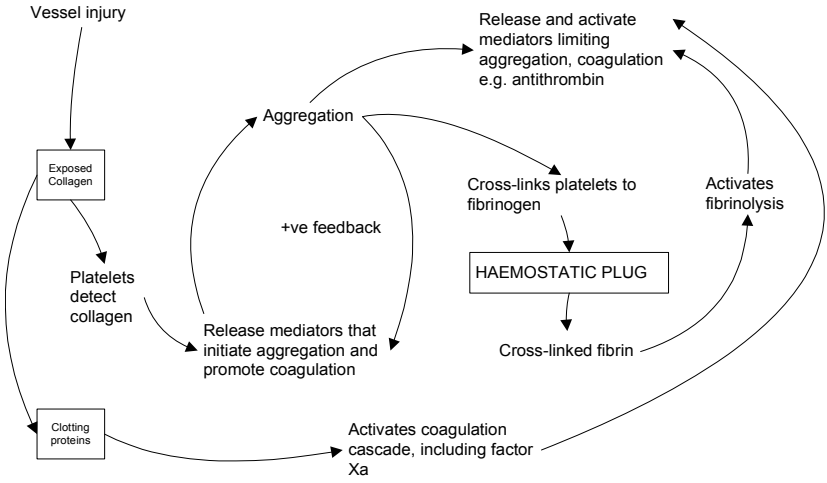

Fig. 2. Clotting Cascade (after [7])

your system is *not* safe to use. Taking A3 as an example, the AP system might not be safe to use on a patient with a cardiac stent, because the stent may be a likely site for an unwanted artificial clot. If you know this, you can explicitly exclude this situation from the safety case (except to note that this restriction will be clearly documented).

Tables 1 and 2 are, of course, only a starting point, but they illustrates the typical form and character of a high-level hazard analysis. It is likely that there are possible accidents that are not covered by Table 1. There are certainly ways that those accidents could occur that are not covered by the hazards in Table 2 (for example, no hazard is identified that could lead to accident scenario A4). During the development of the AP system, the hazard list would be expanded and additional detail would be added. Further brainstorming with other domain experts (and more specialised experts) would be valuable (the Oxford Handbook of Acute Medicine [25] gives "abnormal platelet behaviour" as a possible cause for many different conditions, and various platelet-related disorders are discussed in [15]).

As more sophisticated models became available, engineers would use a variety of hazard analysis techniques that work over these models. Some general-purpose methods were identified in Section 3.1, and Section 5 outlines a possible simulation-based technique. For example, the HAZOP technique (see [13]) is widely used for analysing process diagrams in chemical plants. It could be adapted for use on biological

process diagrams, such as the clotting cascade shown in Figure 2. (It should be noted that Figure 2 is a simplified representation of the cascade, only touching on the secondary (non-platelet) part; more detailed representations exist.)

One corollary of the above hazard analysis is that the safety of the AP system has a tremendous dependency on the behaviour of the patient's immune system. Any simulation-based attempt to address these hazards will need to be paired with a reasonable immune system model. Most likely, this environment model will be built up over time (e.g. through concept, design, animal trials and human trials). Scope control is crucial here – developers may need to restrict the use of simulation to where it is explicitly needed (e.g. for "emergent hazards").

In some ways, the AP system is an easy example because we already have natural platelets to learn from. If we were considering an entirely novel bloodstream-dwelling nanomachine, we'd have a more difficult job. For example, consider whether the hazards presented by an artificial heart are the same as those of a natural heart – there is probably significant overlap (particular those hazards concerned with the blood vessels around the heart) but the artificial heart has a very different internal mechanism.

4.2 Individual and Multi-agent Hazards

H1 (AP in wrong location) could potentially be detected by an *individual* AP – the AP to detect that the surrounding tissue is of the wrong type. By contrast, hazard H4 (AP form oversized clot) is a state at the *inter-platelet* (i.e. *multi-agent*) level, and can be detected only by something that has an overall awareness of multiple APs.

This is an architectural distinction – it is much easier to argue that a hazard is mitigated if it can be resolved at the individual platelet level. For example, an AP could be built to detect whether it was in or near a blood vessel, and to self-destruct if it was not. Alternatively, it might be possible to argue that the lifespan of an AP, although long enough to be useful for clotting, is not long enough to cause problems in other tissue. On a theoretical level, these kinds of arguments are straightforward – they're much like the ones we use in existing safety-critical systems.

By contrast, it is difficult to achieve and argue mitigation for multi-agent hazards. Most likely, an engineer would need to argue that the platelet would be able to detect that the AP system had formed a clot where it was not needed, and then disassemble it. This is an extra requirement for emergent behaviour (extra to those needed to provide useful clotting in the first place). If this approach was taken, this would be an example of a *safety requirement* as introduced in Section 3.3.

(One alternative might be to detect unwanted clots through external imaging, but this would raise a variety of issues including imaging equipment availability, interaction of imaging actions with other treatment of a critically injured patient, and human factors such as ability to see small developing clots in obscure locations).

'Detect' in the above paragraphs could have a number of meanings. One is that the AP has some kind of explicit knowledge of its state (and can therefore take an explicit action) – this is the norm for conventional systems. Another is that the AP is engineered so as to only be stable under a narrow set of 'safe' conditions. For example, it might be that the AP could be designed only to be stable in the physical and chemical environment of a suitable blood vessel – if it was in any other location, it would break up and become inert. For ultra-simple (e.g. nanoscale-assembled) agents the latter approach may be the only viable one.

Nanoscale devices may be novel in that they are built on a single substrate, without the traditional divisions between e.g. software and hardware in a conventional vehicle or robot. They may be built from a single material, both logic and actuators, making them closer to a clockwork device than an electronic robot. In safety engineering, we often rely on the software-hardware divide to manage and contain the software contribution to hazards. More generally, this illustrates how novel technology may break existing safety techniques.

4.3 Building a Safety Case for the AP System

A common and practical approach to building safety cases is to argue that all hazards presented by the system present an acceptably low risk. Figure 4 shows the top level of a possible safety case for the AP system, presented in the Goal Structuring Notation (GSN).

Arguments in GSN have a hierarchical structure. At the top of the structure is a *goal* which takes the form of a particular claim about the system (in Figure 4 this is the node RisksTolerable). The level below that breaks this down into multiple child goals. Each of these child goals then broken down in turn, until we reach a point where the goals can be supported by ("solved by") explicit *evidence* (evidence is shown in a GSN diagram as a "Solution" node). Between any goal and its child goals a *strategy* may be used; this explains how the breakdown is being performed (in Figure 4, ArgOverHazards shows that we are claiming that the risk from all hazards is tolerable on the basis that the risk associated with each identified hazard is tolerable). Figure 3 gives a key to the symbols used in GSN; for a more comprehensive description of the notation, see [11] or [12].

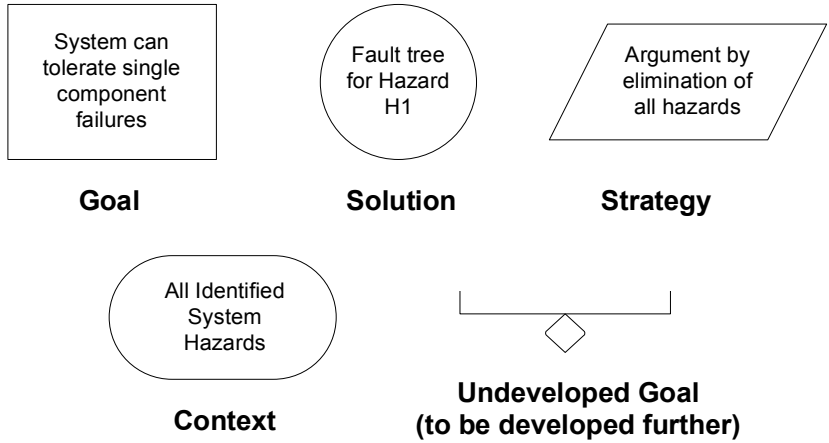

Fig. 3. Principal Elements of the Goal Structuring Notation (from [12])

(Note: The strategy ArgOverHazards breaks down the argument in terms of the top-level hazards that were identified in Table 2).

The context node DefTolerable is very important – it defines the term 'tolerable' that is used in the rest of the argument. The definition used here, "expectation of less than 10^{-6} accidents per use" is arbitrary, although not untypical of existing safety arguments. If we knew what the regulator in this case would require as a standard of safety, then we can change our definition (although we might then need to change our argument structure in order to meet it).

It is important to note that the definition of 'tolerable' provided by DefTolerable applies to the whole AP system active within a particular patient; it sets a probablistic tolerability for accidents at the macro level. This may eventually be converted into probablistic requirements on behaviour at the *micro* level in terms of individual AP. The value of the probabilites used at that level would depend on the macro-level tolerability, the number of AP likely to be given in a single treatment (for AP, that could be billions) and the ability of the overall AP system to mitigate undesired behaviour by a small number of AP. No simple mapping from macro-level accident rates to micro-level failure or hazard rates is possible.

The big advantage of arguing over hazards, compared to other ways of structuring a safety case, is that we can adopt a unique argument structure for claiming that each hazard is tolerable. Figure 6 illustrates this for hazard H6 – we argue over the various membranes that the AP

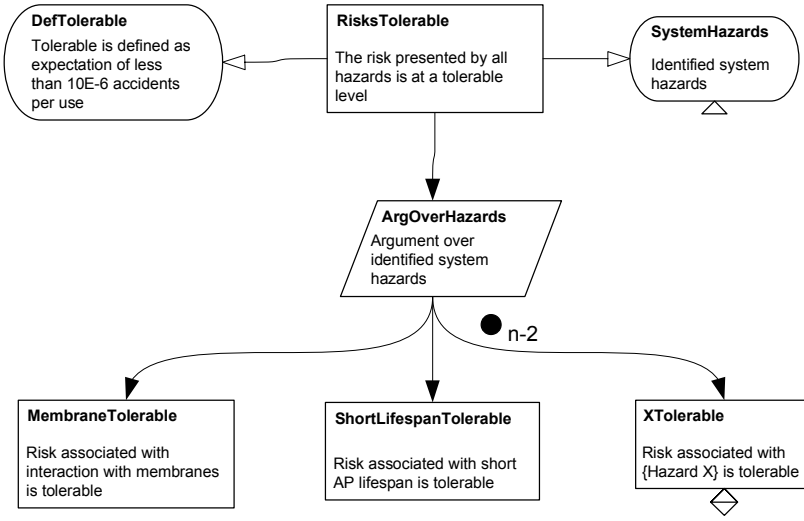

Fig. 4. Top Level of AP Safety Case

could encounter, attempting to show that the interaction with each will be safe. Similarly, Figure 7 shows the argument for hazard H3 – in this case, we argue across the normal and abnormal cases, then (in Figure 8) over the possible causes of the abnormal case.

Notice how Figure 8 ends with undeveloped goals (indicated by the diamonds underneath them). Before this safety case could even potentially be accepted, the developer would need to decompose it further until they reached solutions (shown as circles in e.g. Figure 6).

In a real safety case, parallel arguments would be needed for the completeness of hazard and cause identification, and about the combined risk from all hazards. The latter is straightforward but the former can be very hard – see Section 5.5. There would probably, also, need to be more context and strategy nodes.

The 'trick' exploited by this safety case structure is that it deals with as many hazards as possible by simple means. Neither of the hazard arguments shown involves appeal to complex emergent properties – instead, we have dealt with two hazards in relatively simple terms. This allows us to focus our effort on where it is most needed (on the complex, emergent accident hazards like H4). Complex hazards will require the development of new techniques and patterns. Some progress towards this has been made in unpublished work by Polack [23].

Fig. 5. Argument over Membranes

In the argument fragments above, the argument's legs are combined informally, and would rely on expert judgement to assess the overall contribution of evidence to the top goal. The general intuition is that when there is a goal solved by N child goals, then adding an additional diverse child goal will increase the confidence in the parent. There is no accepted, explicit method of assessing and combining argument confidence, although [5] and [17] comment on what engineers seem to do in practice. (In particular, Littlewood and Wright show in [17] how the "extra leg" intuition may be misleading in some cases).

To adequately assess the strength of complex arguments, a systematic method for propagating confidence through the argument is desirable.

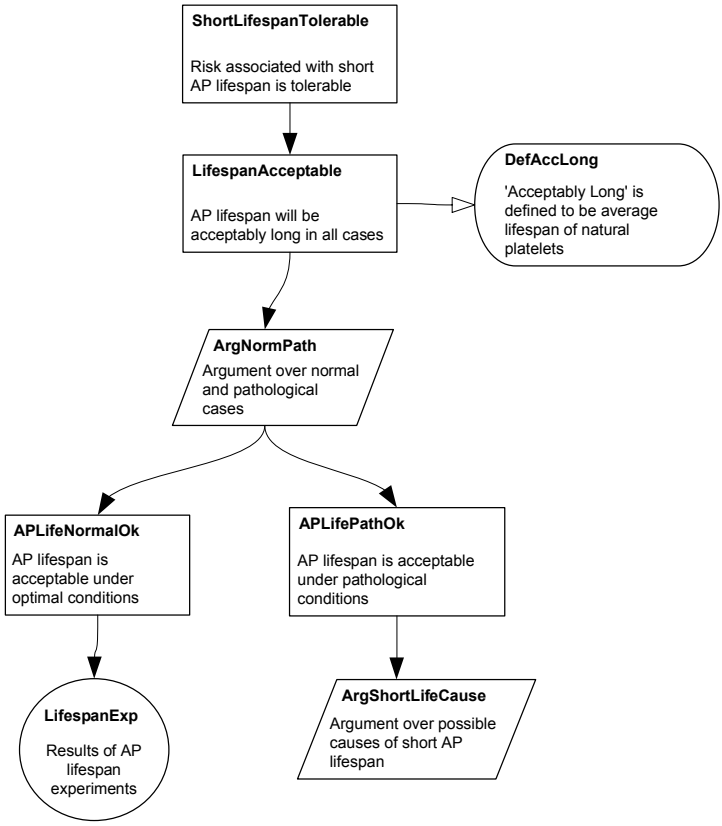

Fig. 6. Argument for Short Lifespan Hazard

Weaver, in [33], presents a system of Safety Assurance Levels (SALs) for performing this propagation. Each low-level goal is assigned a SAL (from 1 to 4) and these propagate up through the argument until the top goal is reached. Independent child goals may give their parent a SAL that is higher than either of them; interdependent child goals will give their parent a SAL that is the lowest of all of them. The weakness of SALs is that it is not clear what a given SAL means in tangible terms (if a goal has SAL 2, what exactly does that mean about the system being argued about?).

A mechanism for assigning truly quantitative confidence in useful real-world terms would be very valuable. Littlewood, in [16], discusses how such a mechanism might be found in Bayesian Belief Nets, which can express structural relationships between beliefs in terms of probability,

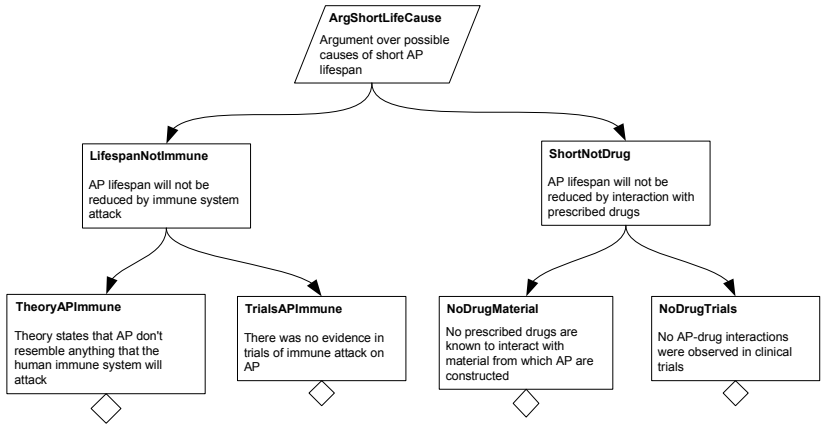

Fig. 7. Argument About Causes of Abnormally Short Lifespan

although he also notes that there are a number of unsolved problems with the approach.

5 Simulation in Safety Engineering – An Example

Two of the authors previously developed a simulation-based approach to hazard analysis for complex systems-of-systems (SoS). The work, carried out as part of the HIRTS DARP project, was in response to increasing integration of large-scale SoS such as network-enabled military units and Air Traffic Control (ATC). An overview of the work follows; for more detail, see Alexander [1].

The motivating concern for the work was that large-scale SoS are not very tractable to conventional hazard identification and analysis techniques. The SoS of concern are complex (in the sense used by CoSMoS), immensely complicated, distributed, and very heterogeneous. Our scope was limited to hazard identification and analysis – we didn't attempt to provide confirmatory safety analysis (this has certain implications, which are discussed in Section 5.5, below).

Merely identifying and explaining hazards is difficult in SoS. If we have a failure in one entity, what is the consequence at the SoS level. For example, if a reconnaissance aircraft in a military unit spots a civilian car and believes it is an enemy tank, what happens to the car? The consequence depends on the capabilities of the other entities in the SoS, the operational procedures in force, and many properties of the SoS's

dynamic state (right down to the psychological disposition of the humans involved: are they relaxed or on edge? Are they expecting the enemy or do they think there's a ceasefire? Do they *want* to see the enemy?) Timing may be a critical factor (when does the misidentification occur, with respect to the patrol times of other reconnaissance entities?).

It may be, in a complex SoS, that we don't even need a "failure" in order for an accident to occur. It may be that under certain circumstances, under certain system states, normal behaviour by all the entities involved is enough to cause an accident. This could be seen as an example of "negative emergence". The enormous state space of an SoS makes these dangerous states, and the paths to them, difficult to discover.

This leads to the concept of an *SoS Hazard* – a state of an SoS. Taking the Def Stan 00-56 definition of 'hazard' (from Section 3.1) we can define this as *"A condition of a SoS configuration, physical or otherwise, that can lead to an accident."* We can explicitly exclude from this those hazards that are confined to a single entity – states where an entity has suffered a failure and may go on to directly cause an accident (for example, an aircraft suffers an engine failure and crashes into the ground). These single entity hazards are relatively tractable using current techniques. It is the multiple-entity hazards that are challenging. (A similar, but more general, concept of *complex system hazard* could be defined). The aim of the work discussed in this section is to find and explain SoS hazards.

5.1 Method Overview

The method requires, first, that the SoS safety team develop a multi-agent model of the SoS. They do this by taking an appropriate source model (such as a description of the system in the Ministry of Defence's MODAF notation [18]) and identifying specific safety concerns that they need to model (such as collisions between aircraft). They must also identify (a) the vignettes that the SoS will be expected to participate in and (b) a set of reasonable deviations that may occur in practice, such as a system suffering a particular kind of failure. The resulting multi-agent model must be implemented in a multi-agent simulation framework, thereby making the model executable.

Once an executable model is available, the 'space' represented by the deviations of that model must be explored. This is performed by running the model with different combinations of deviations and observing the results.

As each run executes, the actions and states of the system components are logged so that they can be studied later. This invariably produces a huge volume of output. To aid comprehension of this data, ma-

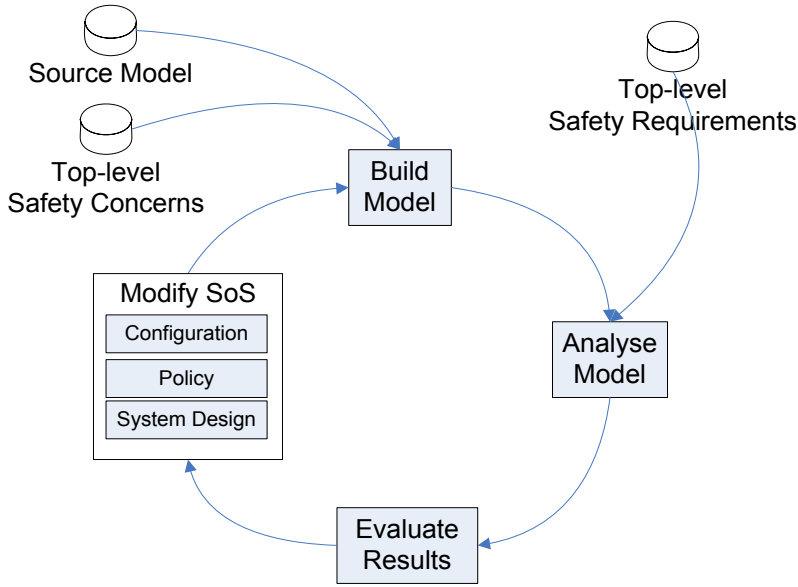

Fig. 8. Overview of the SoS Hazard Analysis Process

chine learning techniques can then be used to extract high-level descriptions of hazards. These descriptions are relationships between certain combinations of deviations and certain accident events. Once interesting accident runs are identified, causal explanations can be derived using an agent tracing tool. Engineers can then use the explanations to study the plausibility of the simulated accidents, and modify the SoS configuration or operating arrangements in light of the more credible relationships.

An outline of this process is shown in Figure 8, and key aspects are expanded on in the following sections.

5.2 Modelling Approach

A common concern in simulation modelling is that, in going from a paper model to an implemented simulation, distortions and errors can be introduced. To help alleviate this, we adopted the concept of explicit *concerns*. These concerns were initially expressed in terms of aspects of the source model, and then updated and checked at each modelling stage or iteration of the model. These concerns could be either deviations or accidents, and provided a starting point in terms of causes and their

eventual consequences. The rest of the process then works from there to derive the intermediate paths and states that linked them.

In addition to deviations coming from the concerns, we provided a method for deriving agent deviations by applying a set of five guide words to a six-part breakdown of a generic agent model. This is similar to the existing methods FFA and HAZOPS mentioned in Section 3.1.

The process we used for developing the agent model was an adaptation of the Prometheus process (see [22] for a description). Although this was not a true refinement method, requiring considerable engineer input, it structured the process and provided a range of cross-checks between the several stages of model development. For example, Prometheus leads developers to describe inter-agent communication in terms of high-level protocols before the agents themselves are developed in detail. Once the agents *are* developed, the Prometheus Design Tool (described in [21]) will automatically check that the agents have the message types defined that they need in order implement the protocol. (It cannot, however, verify that they will restrict their message sequences to those allowed by the protocol.)

The CoSMoS project aims to provide methods for developing high-quality declarative models (rather than the partial MODAF models that we worked with) and offers the prospect of more systematic refinement from model to simulation. However, even if we had been able to perform a true refinement from source model to implementation, we still would not be able to assume that the simulation represents the real world. For our explicit purposes (hazard analysis only) this is not necessarily a problem, but we cannot escape it entirely. See Section 5.5 for further discussion of this issue.

5.3 Analysis Techniques

Once the model has been built, it must be analysed. The analysis technique must select simulation runs to be performed so as to achieve adequate coverage of the parameter space of the simulation while spending the minimum of computation time. We would like to exhaustively explore the parameter space demarcated by all agent deviations, because this would reveal all behaviour paths that were implemented by the simulation model. In practice, this will be impossible unless we use a toy example.

Our solution was to specify a probability for the occurrence of each deviation (in any given simulation run), and then perform runs only for those combinations where the combined probability is above a certain threshold. The simulation engine decides whether or not to run each combination by comparing its combined probability to a threshold for

'incredibility of failure'. This concept originally stems from the nuclear industry – dangerous situations that appear to be more improbable than this threshold are not studied further in hazard analysis. A value for this is given in [2] as 10^{-7} per year of operation (equivalent to 10^{-11} per hour), and this value is adopted here.

Once those runs have been performed, the accidents that occurred need to be identified and their causes found. The former task is relatively easy, since the set of possible accidents is small. The latter, however, is harder, and machine learning techniques have been adopted to make it tractable.

For our purposes, the task of machine learning can be viewed as one of function approximation from a set of *training instances* expressed as input-output pairs; given a function specification (a set of named input parameters (the 'features' used for learning) and a particular form of output value), the algorithm learns the relationship between combinations of parameter values and the output of the target function for those values.

In our approach, the features represent parameters of the simulation and the output values are the consequences within the simulation. All the features used in the current work are deviations that are applied to the model, and the target function is the set of accidents that occurs during the simulation run. We used a decision-tree learning algorithm because it could learn from Boolean-valued parameters (deviations) and discrete-valued outputs (accidents), and present the resulting rules in human-readable form.

The output of the learning algorithm is a set of rules that describes the relationship between deviations and accidents. For example, a rule might be *"Aircraft 1 lost_radio_comms causes aircraft 1 to collide with aircraft 2"*. Such rules, however, only explain how accidents occur in very broad terms. In order to choose appropriate definitions of our hazards, or to take action to prevent or mitigate them, more detailed information about causation is required.

Lam and Barber, in [14] present a tool-supported approach to the comprehension of agent systems. Given a log of the events that occurred in a single simulation run and an event of interest within that run, the tool tries to explain *why* that event happened in terms of its immediate causes. Those causes can each then be explained in the same way, and the process repeated until the final explanation is in terms of the initial state of the simulation or 'external' events that occurred. This explanation, complete or partial, can be expressed as a causal graph leading to the event that we asked the tool to explain.

A simple example of such an explanation would be of the form *"UAV 1 received a percept indicating the location of an enemy unit. This caused it to form a goal of destroying that enemy unit, which it selected the 'air strike' plan to resolve, and as a consequence of that plan the UAV conducted the 'attack' action using a laser-guided bomb".*

The tool achieves this by storing what Lam and Barber call 'background knowledge'. This is a set of possible causal relationships between events of different types and different properties. As the tool tries to explain each event, it reviews these rules to find which earlier events could have caused it.

Once the analysis is complete, the analyst must evaluate the significance of the results, in consultation with the rest of the project safety team and other stakeholders. It is likely, particularly early on in development, that the analysis will reveal problems with the SoS that need to be resolved. As indicated in Figure 8, this may involve changes to the configuration of the SoS, to the design of the individual elements, or to the operational safety policy under which the SoS operates.

5.4 Strengths

By using simulation, our approach is able to search a huge amount of the state space. By using probability as a heuristic for the selection of runs, we can prioritise our explanations towards those parts of the space that are most likely to occur in practice. The approach is particularly strong in this regard when compared to the manual techniques that have traditionally been used in this role.

Using simulation paired with machine learning and tracing (rather than an explicit exploration technique such as model-checking) allows engineers to use the approach with almost any kind of multi-agent model. For example, agent state may be modelled in terms of discrete states or continuous-value parameters, and agent behaviour may be defined by pre-written scripts or by a neural network.

The tracing tool provides a head start to analysts who want to explain how a particular accident happened. Simple learning of relationships between deviations and accidents may not be sufficient for human comprehension – in particular, it is difficult for humans to read extensive logs of simulation events. Animation can help here, but it is still difficult to see causes that are well-separated in terms of time.

The key value of the approach is that if it finds one hazard in an SoS that was not found by other means, then it is useful. This is true even if it also identifies a number of hazards that are initially plausible but turn out to be unrealistic. As it is not intended to provide confirmatory safety analysis, errors and omissions are acceptable as long as some hazards

are realistic. (There could, of course, come a point where the cost of investigating false hazards made the approach impractical.)

In , we applied our approach to two plausible case studies (one of them purely hypothetical, the other provided by industry) and derived some interesting hazards. It is important that this be replicated in a real industrial or operational development, but these results are promising.

All of the above is likely to be true and relevant for the use of simulation in CoSMoS. It is important to note, though, that if simulation is to be used in direct support of a safety argument then it must aim higher than the work described in this section, particularly in terms of gaining confidence in the results (see 'Challenges', below).

5.5 Challenges

The approach described above raises a number of challenges. If the CoSMoS project uses simulation in a similar role, it will have to face these challenges.

The biggest concern with this work is the question of completeness – whether or not an adequately complete set of hazards, and causes of those hazards, has been identified. There are two aspects of this – completeness with respect to model (does the analysis reveal all the hazards in the simulated model), and completeness of the model with respect to reality (does the model implement, and reveal in simulation, all the hazards that exist in the real system). Some confidence that there are *no* unidentified serious hazards is vital for building a safety case. Ultimately, such a confidence cannot be empirically observed; it must come down to *theory*, theory about when it is reasonable to believe that your hazard analysis is complete.

A second concern is related – does the approach provide a false sense of security? Any safety technique or process will have an effect on the perceptions of the engineers involved. A highly detailed simulation model (especially when combined with an attractive animation) may convince engineers that lack of hazards is evidence of their absence. Put another way – any hazard identification or hazard analysis has an implicit, intuitive role in safety analysis. The key is that it should give confidence only in proportion to its adequacy.

A final concern is that the explanations produced by the tracing tool may be *too* compelling. If the tracer leads naturally to a plausible explanation of a mechanism by which a hazard can occur, they may cause engineers to ignore other alternative explanations.

6 The Challenges of Small-Scale Simulation

One can draw a distinction between large-scale simulation (such as the simulations of military units and air traffic discussed in Section 5) and small-scale simulation (such as the simulation of cells and nanites with which CoSMoS is primarily concerned). It could be suggested that in large-scale simulation the properties of the agents (including their local behaviour) are well understood, and that the role of simulation is to combine these to determine their emergent behaviour when they work together. In the small-scale case, the behaviour of the agents may be less well known, and the role of the simulation is to explain (and allow analysts to comprehend) the observed high-level behaviour in terms of low-level rules.

Alternatively, the distinction between large and small could be replaced by one between and engineering and scientific simulation. Engineering simulation is concerned with modelling well-understood parts in situations where their combined behaviour is too complex to understand; scientific simulation is concerned with understanding how parts give rise to well-understood high-level behaviour.

This second conceptualisation leaves potential for engineering simulation of small-scale entities. Can we, however, really know the fine detail of tiny things? For example, platelets may be relatively simple when compared to (e.g.) white blood cells, but they are still not completely understood. Some aspects of their high-level behaviour are also unexplained, such as the causes of the DIC phenomena mentioned in Table 1. The behaviour of entities such as platelets cannot be understood without reference to the environment that they operate in; for platelets this is, at the very least, the human vascular system. This creates a huge demand for knowledge, particularly for knowledge of combinatorial interactions between the diverse entities and processes in the system and the environment.

One can go further, however. Can we really know the precise local behaviour of large-scale entities such as vehicles? We certainly know gross detail (weight, dimension, maximum engine output), but as we move to progressively finer detail (mechanical tolerances, network protocols, presence of software faults) it becomes harder to fully describe, or indeed to know at all. Behaviour over the long term, as the system degrades through operation and is maintained, is particularly difficult to predict. Engineers have often been surprised by the behaviour of vehicles; for example, consider the 1974 air crash caused by the hard-to-shut cargo door on the DC-10 aircraft [3]. Safety engineering, when dealing with the levels of safety that we now expect in our society, deals with *tiny* probabilities.

It can be observed that, for real systems, the small-scale is always present *inside* the large scale. An aircraft may have a human pilot – the pilot contains blood which contains platelets. If we drill down further, into the individual platelets and (primarily) the mitochondria which they contain, then we have the respiratory cycle whereby they generate the power to operate. This cycle is invisible at the high level, but is intricate at the sub-cellular level; e.g. a diagram of the Krebs Cycle in [20] has 25 edges and 26 nodes (of which 17 are unique substance names).

The respiratory cycle is simple for the SoS engineer – at worst, they'll need to think of air, water and calories (e.g. blood sugar levels affect alertness). For the AP engineer, however, the respiratory cycle is close enough to be a concern – it is possible that AP will interact with the substances involved at a molecular level (e.g. it might be that AP attract and accumulate a respiratory by-product). This illustrates how *small-scale engineering dredges up fine detail*. As we move down into the micro-scale and nano-scale, phenomena that were manageable (indeed, barely of concern) in the large become critically important.

Ultimately, however, it is the high level that matters. For safety engineering, this means that we need adequately compelling evidence that the identified safety requirements are met (at the level of the whole system). For those properties of the system that are not relevant to any safety requirement, we do not need any evidence. Of course, there may need to be an argument made that these properties are truly irrelevant. Put another way - we do not need accurate models of the whole system. We *do* need evidence that we can predict certain key properties, and that the values of those properties will be suitable. And we can constrain the situations under which we need to predict these properties by explicitly restricting the situations in which the system is allowed to be used.

There is a further aspect that may make the CoSMoS approach more practical – as noted in Section 3.5, safety management and safety performance measurement are critical. Many aspects of system behaviour will only ever be discovered in real-world operation. If we can provide the initial confidence that a complex system is safe to operate, and have effective safety management in place, then we may be able to detect unexpected hazards before they occur and deal with them before they are able to cause an accident.

7 Conclusions

Safety engineering matters for many domains. In industries that are explicitly regulated (such as aviation or nuclear power) then conformance to explicit engineering standards is mandatory. Outside of those fields,

it is still often the case that employers or product manufacturers have a duty of care towards their employees or users, and therefore need to ensure that the engineered systems they use are adequately safe. Increasingly, this requires system developers to present a structured safety case, incorporating argument and evidence, that a mandated level of safety has been achieved.

When producing a safety case, we need to present an argument that the system is acceptably safe, given the risk posed by the hazards present in the system. We do not need irrefutable *proof*, and we do not need evidence (however weak) of *total* safety. In particular, we don't need to make any claims at all about those aspects of the system that are not safety-critical.

Architecture is a powerful tool – we don't need ensure that every component of a system behaves safely, provided we can argue that the overall architecture system architecture will keep it in line. For example, it doesn't matter if a controller component proposes a hazardous act if we have a monitor component that can reliably veto that act. Complex systems, however, (along with micro-scale devices) prevent the use of many common architectural approaches and so present a major safety challenge. In particular, it is not at all clear how emergent properties could be bounded in this way.

In safety engineering, conservatism is a fact. It is present at every level, from working engineers through to governments (and then on to the general public). Complex systems are likely to face opposition simply because they are novel and little-understood, quite apart from the (genuine) technical difficulties they present. Resolving the technical challenges will help gain acceptable, but showing engineered complex systems working reliably in non-safety domains will be critical.

The real engineering of complex, safety-critical, micro-scale systems such as the AP system may be some way off, but we can start to work out what claims we will need to make, and how we can generate the evidence we need to make those claims. This way, when the technology becomes ready we may actually be able to use it.

Acknowledgements

The authors would like to thank Fiona Polack and Susan Stepney for illuminating discussions about the CoSMoS project, and for providing access to reference [23].

References

[1] Robert Alexander. *Using Simulation for Systems of Systems Hazard Analysis*. PhD thesis, University of York, 2007.

[2] F Ammirato, M Bieth, O J V Chapman, L M Davies, G Engl, C Faidy, T Seldis, D Szabo, P Trampus, Ki-Sig Kang, and J Zdarek. Improvement of in-service inspection in nuclear power plants. Technical report, International Atomic Energy Agency, 2004.

[3] Aviation Safety Network. Accident description 03 MAR 1974, 8 June 2007 2007. `http://aviation-safety.net/database/record.php?id=19740303-1`, accessed 17 June 2008.

[4] John Barnes. *High Integrity Software: The SPARK Approach to Safety and Security*. Addison Wesley, 2003.

[5] Robin E Bloomfield, Bev Littlewood, and David Wright. Confidence: its role in dependability cases for risk assessment. In *Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN '07)*, 2007.

[6] CoSMoS project team. Complex systems modelling and simulation infrastructure, 2008. `http://www.cosmos-research.org/caseforsupport.html`, accessed 17 June 2008.

[7] Patrick Davey. *Medicine at a Glance*. Blackwell Science, 2002.

[8] G Despotou and T Kelly. The need for flexible requirements in dependable systems. In *Proceedings of the 4th International Workshop on Requirements for High Assurance Systems (RHAS)*, 2005.

[9] Health and Safety Executive. Health and safety at work etc act, 1974.

[10] Martin Jones. Development of an operational safety case within an existing safety management system. Master's thesis, University of York, 2007.

[11] T P Kelly. *Arguing Safety - A Systematic Approach to Managing Safety Cases*. Phd thesis, University of York, 1998.

[12] T P Kelly and R A Weaver. The goal structuring notation — a safety argument notation. In *The Dependable Systems and Networks 2004 Workshop on Assurance Cases*, 2004.

[13] Trevor Kletz. *HAZOP and HAZAN: Identifying and Assessing Process Industry Hazards*. Institution of Chemical Engineers, 3rd edition, 1992.

[14] D N Lam and K S Barber. Comprehending agent software. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2005)*, 2005.

[15] R J Liesner and S J Machin. Clinical review ABC of clinical haematology: Platelet disorders. *BMJ*, 314(7083), 1997.

[16] Bev Littlewood. Limits to dependability assurance — a controversy revisited (or: A question of 'confidence'), 2007.

[17] Bev Littlewood and David Wright. The use of multi-legged arguments to increase confidence in safety claims for software-based systems: a study based on a BBN analysis of an idealised example. *IEEE Transactions on Software Engineering*, 2007.

[18] MODAF Partners. MOD architectural framework executive summary, 2005-08 2005.
[19] Motor Industry Software Reliability Association. MISRA-C:2004 — guidelines for the use of the C language in critical systems, 2004.
[20] Michael Muller. Krebs cycle pictures and summary, 2008. `http://www.uic.edu/classes/bios/bios100/summer2003/krebsfull.htm`, accessed 21 July 2008.
[21] Lin Padgham, John Thangarajah, and Michael Winikoff. Tool support for agent development using the prometheus methodology. In *Proceedings of the first international workshop on Integration of Software Engineering and Agent Technology (ISEAT 2005)*, Melbourne, Australia, 2005.
[22] Lin Padgham and Michael Winnikoff. *Developing Intelligent Agent Systems: a Practical Guide*. John Wiley & Sons, 2004.
[23] Fiona Polack. Argumentation and the design of emergent systems. `http://www-users.cs.york.ac.uk/~fiona/PUBS/Arguments.pdf`, 2008.
[24] D J Pumfrey. *The Principled Design of Computer System Safety Analyses*. Dphil, University of York, 1999.
[25] Punit Ramrakha and Kevin Moore. *Oxford Handbook of Acute Medicine*. Oxford University Press, 2rev edition, 2004.
[26] Carl Ritson. A process orientated biological simulation using occam-. Technical report, University of Kent, 2006.
[27] RTCA and EUROCAE. DO-178B: Software considerations in airborne systems and equipment certification, March 1999 1999.
[28] Steve Schneider, Ana Cavalcanti, Helen Treharne, and Jim Woodcock. A layered behavioural model of platelets. In *ICECCS 2006*. IEEE, 2006.
[29] Stephen Toulmin. *The Uses of Argument*. Cambridge University Press, 1958.
[30] TUNA project team. TUNA: Final report, 2008. `http://www.cs.york.ac.uk/nature/tuna/outputs/finalreport.pdf`, accessed 17 June 2008.
[31] UK Ministry of Defence. Defence standard 00-56 issue 2 — safety management requirements for defence systems, December 1996 1996.
[32] UK Ministry of Defence. MOD interim defence standard 00-56 issue 4 — safety management requirements for defence systems, June 2007 2007.
[33] R. A. Weaver. *The Safety of Software - Constructing and Assuring Arguments*. PhD thesis, University of York, 2003.

Towards an Executable Model of Auxin Transport Canalisation

Philip Garnett[1], Susan Stepney[2], and Ottoline Leyser[1]

[1] Area 11, Department of Biology, University of York, YO10 5YW, UK
prg500@york.ac.uk
[2] Department of Computer Science, University of York, YO10 5DD, UK

Abstract. We describe our use of a modelling and development process to specify and implement biological simulations that involves the development of several different UML models to capture different perspectives on the system being modelled, in particular the investigation of various emergent properties. We use this process in the case of an auxin canalisation simulation, investigating the processes of auxin transport as guided by PIN proteins, in a developing plant. We discuss our preliminary results of investigating one hypothesis of PIN protein placement that fails to demonstrate canalisation in simulation.

1 Introduction

The simulation of biological systems presents a significant challenge requiring knowledge from all branches of science to capture all the relevant aspects of the biological, chemical and physical processes occurring. The challenge is made more difficult by the complex nature of biology: it is hard to make good assumptions about how a particular process is regulated. The quality of information available is important: the best solution based on the information at hand may not be the real solution, but more a reflection of how insufficient the current data are. Or the data may be excellent but missing a part of the picture altogether. Also the connectivity of processes in biology is often very high, therefore the question of the level of abstraction and simulation complexity is important. If the abstraction level is too high we risk ruling out simulations producing emergent behaviour; too low, and the simulations produced could be difficult to work with. So the decisions made when producing a simulation are important, as a balance must be sought between complexity and the all important emergent behaviours.

Here we present how we are using the CoSMoS lifecycle [3] to develop abstract models and executable simulations of a biological system, in order to test different hypotheses of how certain biological processes may work.

We are using this approach to investigate auxin canalisation in the plant *Arabidopsis*. Auxin is a plant growth hormone. The process of auxin canalisation occurs between auxin production sites and auxin sinks; in the plant stem, it results in the development of vascular tissue between new sites of auxin production and existing vascular tissue. We are interested in understanding how this complex self-organising process is regulated in the cells of the plant. Due to the complexity of canalisation, and some gaps in the biological knowledge of how and what is causing the canals to form, we are developing executable computer simulations in order to test multiple hypotheses. We are using UML (Unified Modelling Language) [25] to model the biology as we understand it, the simulated biology, and the details of the implementation. The hope is that the hypotheses tested by the simulations might indicate experiments that can then be tried in the lab to further our understanding, and to drive new simulations.

We find that the combination of UML and object-oriented programming maps naturally to the kind of biological processes that we are modelling. Biological entities, such as proteins and cells, map directly to objects in the UML models, which are then implemented as objects in the program code. The interactions between these biological entities similarly map directly to associations between objects in the UML models, which are then implemented as communications between objects in the program code. This allows us to build models containing the biological entities that we believe to be involved in canalisation, and then produce simulations that we can use test various hypotheses about the biological processes of interest. If an hypothesis is correct we should see emergent behaviour that is consistent with the real biological behaviour when the simulation is run; if not we can then return to the UML models and implement our next hypothesis. This provides a process to assist us in determining if our simulated biology is consistent with the real biology. Additionally, the UML diagrams are relatively accessible to biologists, allowing them to provide input to the model of the simulation without the need to understand the code.

This is a report of a work in progress on the modelling and simulation of a biological system. In section 2 we discuss the use of UML as a suitable modelling language. In section 3 we overview the process we are using for modelling, designing, and implementing biological system simulations. In sections 4, 5, and 6 we present our initial Domain,

Software, and Refined Software Models of auxin canalisation, and in section 7 we discuss some of the issues of building the resulting simulator. In section 8 we present some preliminary results on the auxin canalisation hypotheses, and conclude with a discussion of our experiences in section 9.

2 Modelling biology and simulations with UML

UML [25] is a modelling language comprising a suite of diagramming notations. It was originally developed to provide a standard concrete syntax for software modelling.

There are a large number of ways of developing software with UML. For example, the process can start by looking at high level interactions in the system being described [25]. This includes the ways that various external actors interact with, or use, the system (where these external actors include users and other systems). This is normally modelled in a Use Case Diagram supported by textual usage scenarios. The high level model is gradually refined by adding detail. Further diagrams are used to drill down details of how the system is built, what the objects are, what information they exchange and when those exchanges of information need to take place. This eventually results in a code skeleton being created for the objects with the attributes in place and the methods waiting to be implemented.

The process of developing biological models with UML is similar to the processes used for development using the Systems Biology Markup Language (SBML) [15, 10, 14]. However, as we are implementing our programs in object-oriented (OO) programming language Java, the ability to produce code skeletons from UML easily and flexibly with tools such as Rational Rose[16] is an advantage. UML is considered to be platform independent, as the diagrams can be transformed into a wide variety of different outputs.

As well as classic object-oriented technologies, UML is well suited to agent-based modelling [24] (where an agent can be thought of as an object with its own thread of control, allowing highly parallel systems of multiple agents). Biological "agents", such as cells and proteins, can be modeled as UML agents. There are many processes in cells acting in parallel. Some of these processes are individually sequential, such as expression of proteins in response signals detected in the cell. The signals might cause a number of events such as protein expression, which then in turn causes more events to occur. This sequential behaviour is often called a cell pathway. The parallel behaviour comes from this type of process occurring in a number of different pathways in one cell of the

plant at the same time, and in many cells at the same time throughout the plant.

We are not the first to apply UML and related modelling notations to the modelling of biology. There are a number of published cases where these have been successfully used to produce biological models [8, 17, 35].

We have taken a simple iterative approach, where UML models are developed, turned into the simulation code and tested. Then any necessary alterations are fed back into the beginning of the process, to ensure the models and code are consistent. We have not found it necessary to use all of the multifarious diagrams available in UML; we describe only the ones that we have found to be of greatest use.

3 Overview of the modelling lifecycle

In [8] we identify a conceptual framework for developing bio-inspired algorithms that takes a principled approach of building models of the biological system of interest, developing abstract computational models from this, then instantiating these computational models to produce bio-inspired algorithms. This framework can be adapted to producing *simulations* of complex (biological) systems, by implementing the abstract computational models to produce simulators.

Fowler [12, p.5] identifies two perspectives that can be used when building models: the *conceptual* perspective, representing "a description of the concepts of a domain of study", and the *software* perspective, where "the elements of the UML map pretty directly to the elements in a software system".

The CoSMoS (Complex Systems Modelling and Simulation) project[3] is developing a complex systems simulation development infrastructure (preliminary work is reported in [2, 28]). The CoSMoS lifecycle [3] has grown from the conceptual model and Fowler's perspectives, and adds the concept of an Analysis Model. It identifies the following components (summarised in figure 1):

Domain Model: a "top down" conceptual model of the real world system to be simulated, derived from the domain experts, from the literature, and (possibly) from further observations and experiments needed to provide sufficient data for modelling. Some modelling decisions about what to put in and what to leave out are made here. The model may explicitly include various emergent properties, since from a top down perspective it may not be obvious that these are emergent; or, if we are

[3] http://www.cosmos-research.org/, EPSRC grants EP/E053505/1, EP/E049419/1

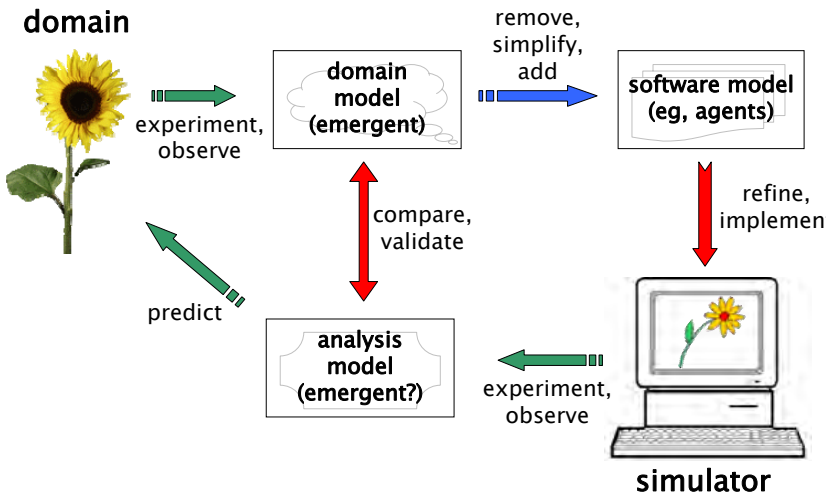

Fig. 1. The components of the CoSMoS basic lifecycle (after [3]).

aware of the emergent properties, it may not be obvious what low level processes produce them.

Software Model: a "bottom up" model of how the real world system is cast as a simulation. This includes: a definition of the system boundary (what parts of the Domain Model are being simulated); simplifying assumptions and abstractions; removal of emergent properties and replacement with the local interactions that are hypothesised to result in them; extra simulation-only concepts, such as "physics" engines to implement real world processes in possibly unnatural ways, user interfaces to view and control the simulator, and "probes" to produce output data for analysis.

Simulator: the executable implementation. The development of the Simulator from the Software Model is a standard software engineering process.

Analysis Model: a "top down" conceptual model of the *simulated* world, derived from observations and experiments on the simulation. The model may explicitly include various observed emergent properties. This model is compared to the Domain Model in order to test various hypotheses, such as the validity of the simplifications used to derive the Software Model.

In a large simulation development this basic lifecycle can be augmented with extra steps [3], such as the development of a **Refined Software Model**, describing the detailed simulator design and platform-

specific implementation details, which is a a refinement of the Software Model used as the basis for producing code.

The CoSMoS lifecycle is neutral in its choice of modelling language(s). For example, it could use a mix of text, biological "cartoons", Soft Systems' Rich Pictures [7], and mathematical equations to describe the Domain Model, and any standard software engineering technique to define the Software Model. Here we use mostly UML supported with text.

This process allows us to separate implementation details from the biology being simulated. This offers a number of advantages. It makes the individual models and accompanying diagrams simpler, as they are focussed on specific perspectives. As we are partly using UML as a communication tool it is advantageous for the diagrams to be as simple as possible. Different groups are more interested in certain perspectives: for example, biologists are probably more interested in a clear representation of the biology, rather than how the data I/O and GUI work.

4 Domain Model: auxin transport

4.1 Modelling

The process of auxin transport canalisation is complex, and is not fully understood. This makes it a natural target for modelling, but also a challenge. Our eventual goal is to produce models and simulations of shoot branching, but in order to do this we first have to understand the mechanism of auxin canalisation.

We have therefore started producing an executable model of the canalisation process. We are using the CoSMoS approach, in UML, starting from our background biology derived from the literature, and from wet lab experiments by Leyser and her group (summarised below). We use that to develop a UML Domain Model that includes all the necessary biology for the model to function, but keeps the model as simple as possible. We then develop a UML Software Model, and refine it to develop the simulator program itself, via the production of code skeletons. This helps us to ensure that the reasons for how and what is in the various models is carefully considered, and we are able to make comparisons with how the model simulates biology compared with how we think the biology works. Finally, we analyse the outputs of the simulator.

We produce abstract UML models of cells assisted by UML-based software development tool Rational Rose [16], and implement the simulator in Java.

4.2 Overview of biological domain

We summarise the background biology here to provide context; more detailed information than given here informs out full model.

There are many mathematical models of nearly all aspects of plant development [29], and many concerned with auxin transport [19]. We are interested in producing executable models as we believe that this type of modelling lends itself well to biological systems and might offer an alternative perspective that yields results [11].

We are developing models of the formation of auxin transport canals, particularly the transport canals that form in the stem of plants and often go on to differentiate into vascular tissue. This process is known as auxin transport canalisation and its regulation is not clearly understood.

We are looking at the regulation of shoot branching, where lateral axillary buds on the main stem of a plant activate and start to grow. Our eventual aim is to model the canals that form from a newly activated bud and progress into the existing main vascular tissue of the stem. Experiments indicate a correlation between the activation and growth of a bud and its ability to transport auxin out into the stem vascular tissue [5]. Therefore, the regulation of canalisation in the bud is linked to the regulation of shoot branching.

Auxin transport canals form between sites of auxin production and auxin removal: sources and sinks. In order for a canal to form between the bud (the source) and the stem vasculature (the sink), the stem vasculature must behave as a relatively strong sink compared to the surrounding tissue. If the stem vasculature is already transporting large amounts of auxin, it has no further capacity for auxin transport, so its sink strength is relatively reduced, and the canal does not form. Therefore the bud does not activate, and does not grow into a branch. However, if there is spare capacity for auxin transport in the vasculature, then it is a relatively stronger sink. This allows auxin canalisation to occur, so auxin can be transported out of the bud, allowing the bud to develop into a new branch [5, 26].

The formation of the canals between the sources and sinks is due to aspects of the biology of auxin transport (figure 2). Auxin is a weak acid and is able to enter cells passively from the more acidic apoplast (intercellular space) by crossing the cell membrane. However once in the pH-neutral cytoplasm of a cell it becomes deprotonated and therefore is not able to recross the cell membrane passively. The auxin is able to leave the cell only with the assistance of transport proteins. One particular family of auxin transport proteins, PIN proteins, are polarly localised in cells, often to only one face of the cell [23].

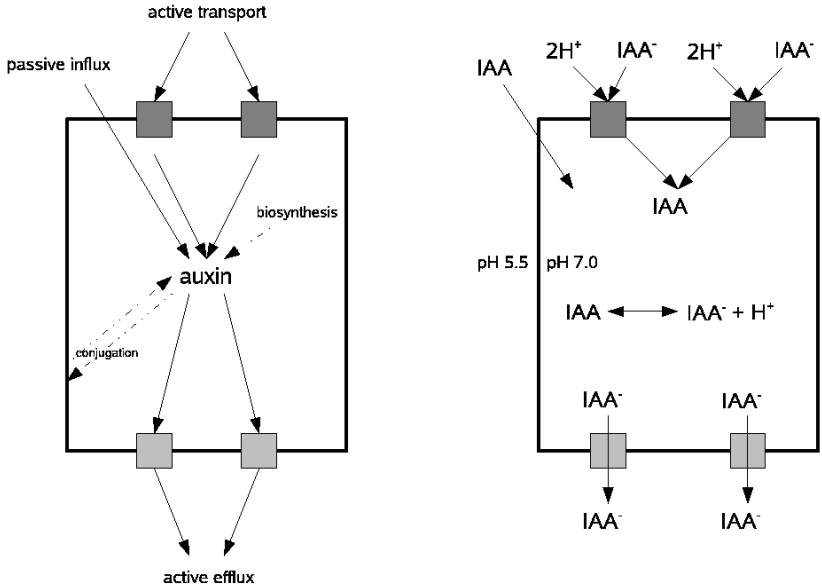

Fig. 2. A "cartoon" (informal picture) of the biology of auxin transport into cells. Auxin (IAA) can enter cells passively, or it can be actively transported by AUX/LAX proteins. Once in the cell its only method of escape is active transport by proteins like PIN [23].

In their capacity as auxin exporters, PIN proteins are important to the process of canalisation. The exact details of how PIN localisation is regulated are unknown, but increasing amounts of data and some prior modelling work are providing possible mechanisms. One hypothesis, by Sachs, suggests that auxin facilitates its own flow: both the ability of a cell to transport auxin and the polarity of the auxin flow increase with the amount of auxin being transported [31]. Therefore as the transport capacity increases the cells in the canal become better sinks and draw in more auxin. This process has been modelled by Mitchson and he showed it to work [21, 22], but the model has a few problems. Firstly, it predicts canals of high flow and low concentration, whereas experimental evidence suggests that there is both high flux and *high* concentration [34, 4]. Secondly, both Sachs' and Mitchson's models require the cells to be able to measure the flux of auxin; as yet a mechanism to do this has not been identified in the plant, although that does not mean that one does not exist.

More recent data on some aspects of the processes provides more details of what should be in the models. Auxin is up-regulating its own transport by increasing the amount of PIN protein available to transport auxin [27]. If the negative regulators of PIN accumulation, such as the MAX pathway, are removed, transport increases and the vasculature is able to transport more auxin [5]. Thus the more auxin in a cell, the more it can transport. In *Arabidopsis* the mutation of genes in the MAX pathway results in a phenotype of increased branching [33], which supports the theory that bud activation, and therefore branch number, is linked to the stem auxin sink strength.

PIN proteins are important to the canalisation process as they export auxin out of the cells, and their polar localisation patterns are responsible for complex transport patterns in a number of plant tissues [30, 9, 13]. Improved knowledge of PIN has also enabled development of simulations of auxin transport systems with high concentration and high flux [13, 18]. However, what directs the PIN in the cells into the polar patterns that are seen remains an important problem: if PIN is positioned by detection of auxin flux, as Sachs suggests [31], then what is the mechanism in cells that is detecting auxin flux?

4.3 Domain Model Use Cases

We start building our UML Domain Model from the background biological material. One approach to building UML models is to start with use cases, that capture the user requirements or how they want to use the system. That is not appropriate here, as we are not modelling what is wanted, but what is. So instead, in the domain model, we take the view that use cases capture a high level view of what the system "does", such as regulate proteins and transport auxin (figure 3).

This approach maps well to the ways biologists describe their domains. We have not found a need to develop this model further, by developing the more detailed usage scenarios necessary in a traditional system requirements elicitation process. Such scenarios might be found necessary for capturing more detailed behaviours ("so, *how* does it regulate proteins?"), and would presumably need to be modified from their traditional format.

4.4 Domain Model class diagram

We model the biological entities of interest as objects and classes. This approach works well here, because much of cellular biology can be thought of as interactions of discrete objects that result in complex behaviours.

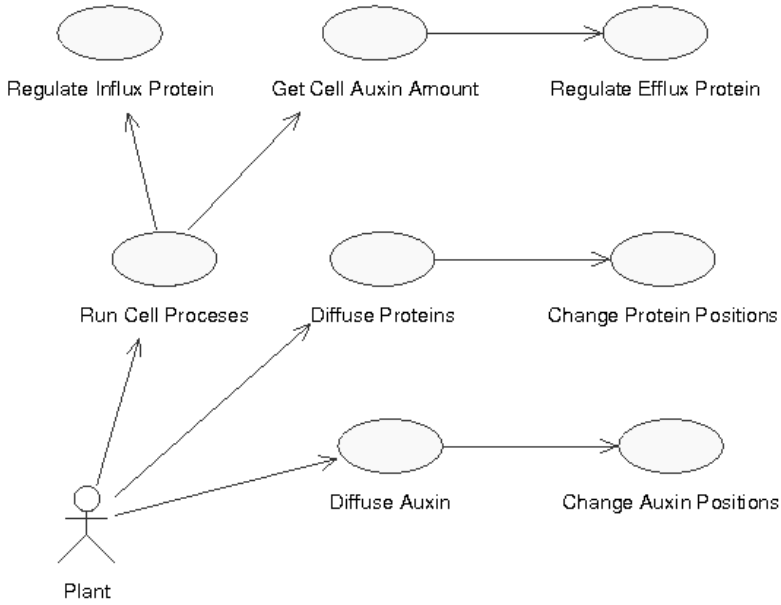

Fig. 3. The Domain Model Use Case diagram. This captures what the Plant does.

However, it should be noted that the class diagram, although maybe the most natural starting point for a object-oriented software developer, is not the most natural way for biologists think about and describe their domain, particularly in identifying associations between classes.

We consider the different parts of the cells, such as the cell membrane and vacuoles (cellular compartments), and the proteins like PIN and hormones like auxin, all as objects. One of the objects of interest at this level of model is the auxin canal. This is an emergent property of the lower level interactions. We model it explicitly here to capture its biological properties so that later simulation outputs can be related to it.

Figure 4 shows our Domain Model class diagram of the biologically relevant parts of the system (deciding what *is* biologically relevant is also part of the modelling process). In detail, it shows the following classes (type of objects) and relationships between the objects[4]:

[4] This detailed description is provided here to help biologists to interpret the diagram; there is nothing in the explanatory text not included in the diagram.

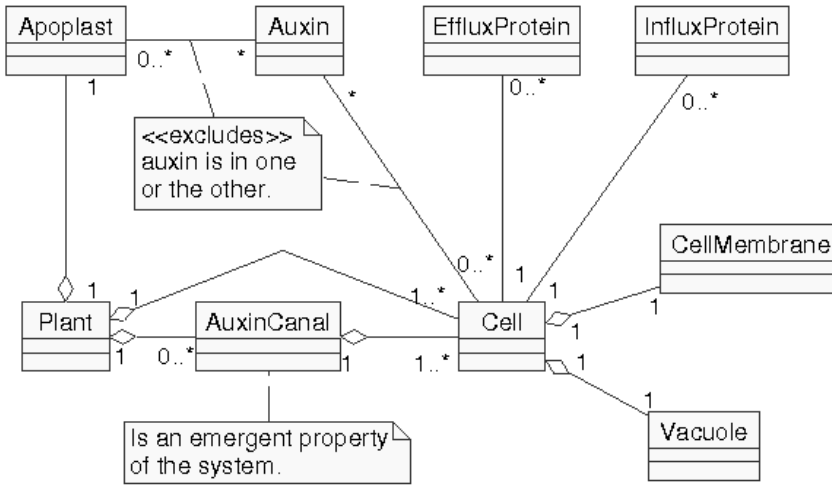

Fig. 4. The Domain Model class diagram.

- A Plant has one Apoplast (the space between cells), one or more Cells, and an optional Auxin Canal. (It also has other components, but these are not being modelled, even at the system level.)
- An Apoplast is part of one Plant, and has zero or more Auxin molecules.
- An Auxin molecule is in the Apoplast or in a Cell. (The relationship lines say that it may be in an Apoplast and it may be in a Cell; the excludes condition says that it is one or the other).
- An Auxin Canal is part of one Plant, and has one or more Cells.
- A Cell may be part of an Auxin Canal; it is part of a *Plant*. It has zero or more Auxin molecules, zero or more Efflux Proteins, and zero or more Influx Proteins. It has one Membrane and one Vacuole.
- An Efflux Protein is in one Cell; an Influx Protein is in one Cell.
- A Membrane is part of a Cell; a Vacuole is part of a Cell.

We impose an extra condition on the loop of relationships[5] containing the Plant, Auxin Canal, and the Cells: Consider a Cell that is part of an Auxin Canal that is part of a Plant, that Cell is also directly part of *the same* Plant. (There is no loop of relationships containing the Plant, Auxin Canal, Cells, Auxin, and Apoplast: the apparent loop is broken by the excludes condition.

[5] By "loop" we are referring to the two possible paths from Cell to Plant, one via the Auxin Canal, and the other directly.

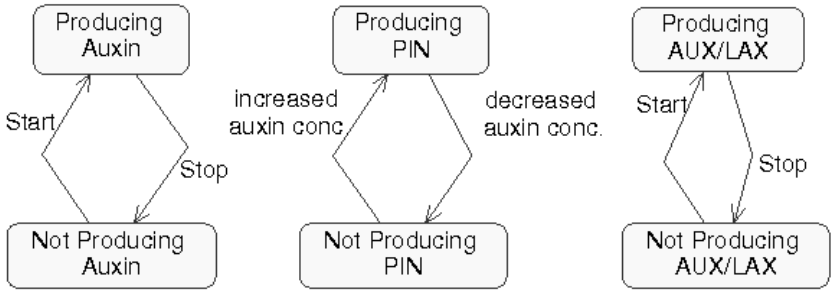

Fig. 5. State diagram for the Cell. (Since we are not explicitly modelling cell birth or death, no start or end states are needed.)

Note that there is no use of inheritance in figure 4, despite the fact that Efflux Protein and Influx Protein are both kinds of protein, for example. We find that we make relatively little use of inheritance in our Domain Models, because it is not necessary in order to understand and capture the biological domain, which is more interested in detailed differences than abstracted similarities. The Software model, on the other hand, exploits inheritance to provide abstraction and code reuse.

4.5 Domain Model state diagrams

The use case diagram and class diagram help with the identification and organisation of the different objects of the model, but provide little information about how those objects behave and how they interact.

Interactions are often captured in UML using sequence diagrams, and these show the passage of information between objects over time. In biology the order and direction of interactions is less clearly defined: the next step in the interaction sequence might not occur; the process might back up to the previous step in the sequence. For example, there is no guarantee of progression down a biochemical cell pathway. This makes capturing timing of events difficult with sequence diagrams.

UML has another way to capture how objects change over time: state diagrams. These diagrams show the different states an object can be in, and how the object moves from one state to another in response to an event. Such state diagrams also appeal to biologists, as they appear to map closely to the way that biological processes are understood.

State diagrams for the state changes associated with a Cell are shown in figure 5, for Auxin hormone in figure 6, and for PIN proteins (a kind of EffluxProtein) in figure 7. The states of these objects are linked, and a change in the state of one object is linked to that of the others. The state

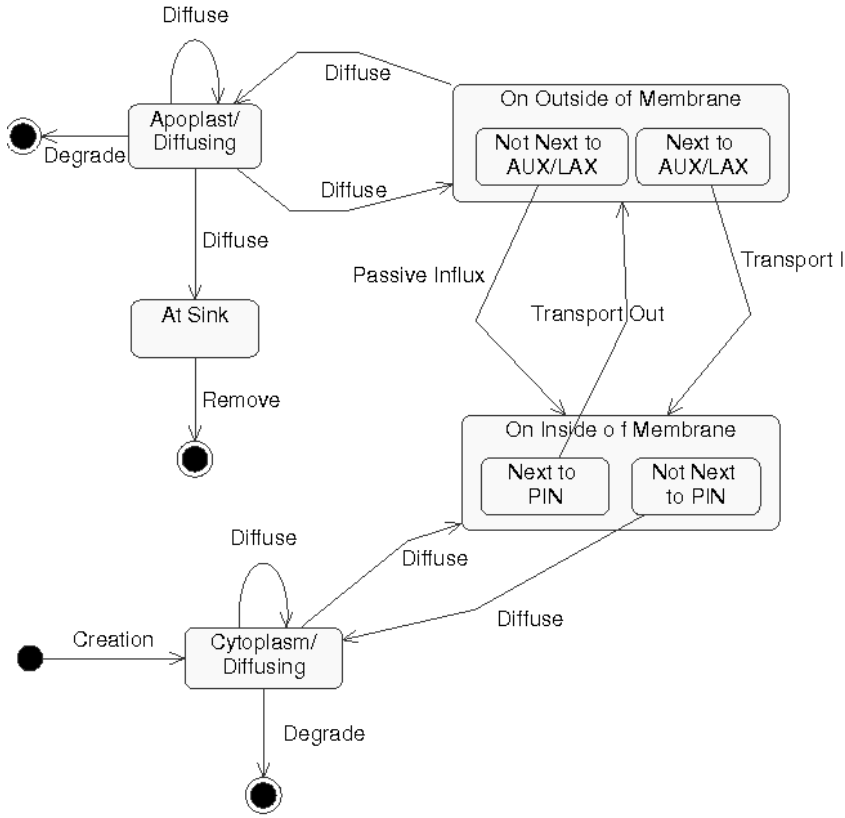

Fig. 6. State diagram for Auxin. Defining and expressing this type of complicated behaviour is where state diagrams can prove useful. The figure shows the different states auxin can progres through, and the events that can trigger those state changess.

of a cell is defined by the what is happing in that cell, which is determined by what the proteins and hormones are doing. At the moment, we are performing this linking by textual annotations.

Figure 5 shows that a Cell can be producing Auxin, PIN protein (an EffluxProtein), and AUX/LAX protein (an InfluxProtein). Not all cells are capable of making auxin, but they are all capable of producing PIN, which they do in response to the amount of auxin they have, so PIN production might be on even if there is no auxin production.

Figure 6 shows the state diagram for Auxin. It can be in four main states:

- in the cytoplasm (the inside part of the cell that is not vacuole). It is created here, and may degrade (be destroyed) here. It is in its deprotonated form. It is diffusing around, which either leaves it in the cytoplasm, or moves it to be:
- on the inside of the cell membrane, where it is in one of two substates, next to PIN, or not next to PIN. If it is not next to PIN, it diffuses back in the cytoplasm. If it is next to PIN, is transported out of the cell to be:
- on the outside of the cell membrane, where it is in one of two substates, next to AUX/LAX, or not next to AUX/LAX. If it is next to AUX/LAX, is transported into the cell to be on the inside of the cell membrane. If it is not next to AUX/LAX, is can passively influx into the cell to be on the inside of the cell membrane, or it can start diffusing to be:
- in the apoplast. It is in its protonated form. It is diffusing around, which either leaves it in the apoplast, or moves it to be on the outside of the cell membrane, or moves it to be at auxin sink, where it is removed from the system. Or it may degrade (be destroyed) here.

In reality, an auxin in the cytoplasm may be no different from an auxin adjacent to a cell membrane: it has no "sense" of where it is. Therefore the auxin may not have a different *biological* state when it is in these different situations. But we can *model* the biology in terms of such states.

State diagrams can be used to model alternative hypothesised behaviours. Simulations based on these different hypotheses can be compared. For example, figure 7 shows a state diagram for one hypothesis of PIN protein (an EffluxProtein) behaviour, and figure 8 shows the state diagram for a slightly different hypothesis for its behaviour. In the latter case, the PIN protein is allowed to move around on the cell membrane if it is not transporting auxin. Therefore the moving state is different from the transporting state when on the membrane of the cell. (Proteins are able to move around on cell membranes [32] and it is theoretically possible that a conformational change in response to actively transporting auxin might stop it from moving.)

When we are considering proteins, the different states in the Domain Model of the biology correspond more closely to real biological states than in the case of auxin. Proteins are active molecules, and can undergo conformational and other changes in response to events. Auxin however is a very simple molecule, and more of its behaviour is a passive response to its environment.

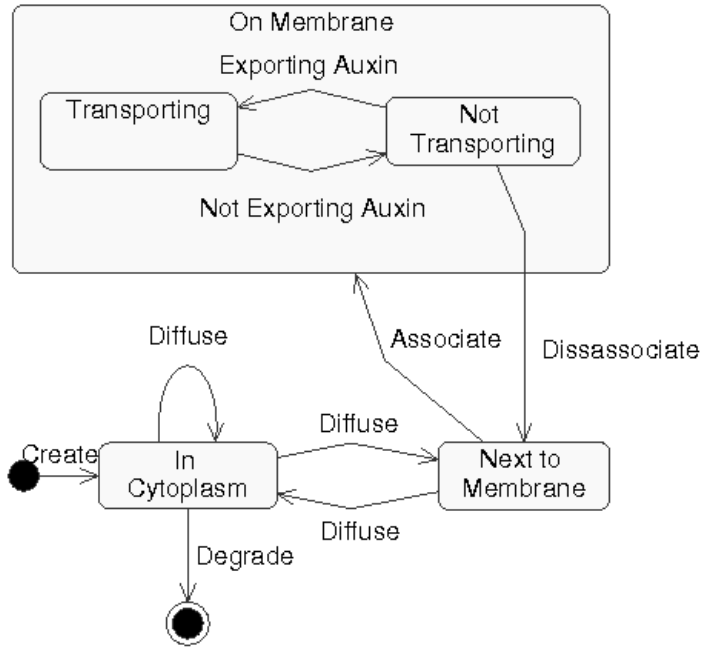

Fig. 7. State diagram for the PIN Efflux Protein. The PIN is associated with the membrane. It can either be actively transporting auxin or sitting idle. If it is not transporting auxin there is the possibility that it might disassociate and return to the cytoplasm. If it is transporting then it remains attached to the membrane.

5 Software Model

5.1 Overview of additional functionality

The Software Model includes the things that our simulation must do to be able to run. These include the setup procedures required to get the simulation to a starting point, including things like instantiating cells, and detecting the internal environment in the cell in order to produce proteins and hormones. This functionality can be considered from three points of view: that of the biological plant (captured in the Domain Model); the simulation of the biology that is required to be there but is not simulated in a particularly biological way; and the things that need to be there to produce a successful simulation but are not part of the biological Domain Model.

As our UML and simulations have developed, the simulated biology has come to represent the real biology, as currently understood, more

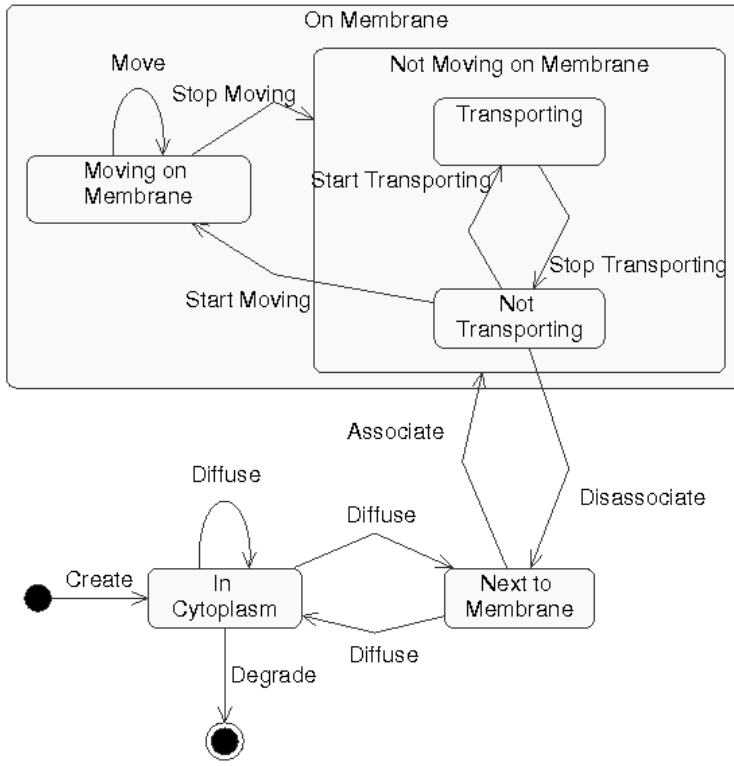

Fig. 8. State diagram of an alternative hypothesis for the PIN Efflux Protein, with different behaviour on the cell membrane: movement on the membrane is allowed.

closely. However, there are still a few areas where this has not been possible, or desirable, to achieve. For example, when a cell is created in the simulation it is necessary to create the cell and then create its membranes, a vacuole and a starting amount of proteins. In reality membranes partly define a cell: a cell cannot exist without a membrane. Therefore our simulation is not doing cell creation, or growth, in a particularly biologically realistic way. It would be more realistic for a cell be the outcome of a particular arrangement of cell membrane, vacuole and other cell elements. The increased flexibility of such an approach could in the future allow for simulation of growth, the lack of which is currently a limitation.

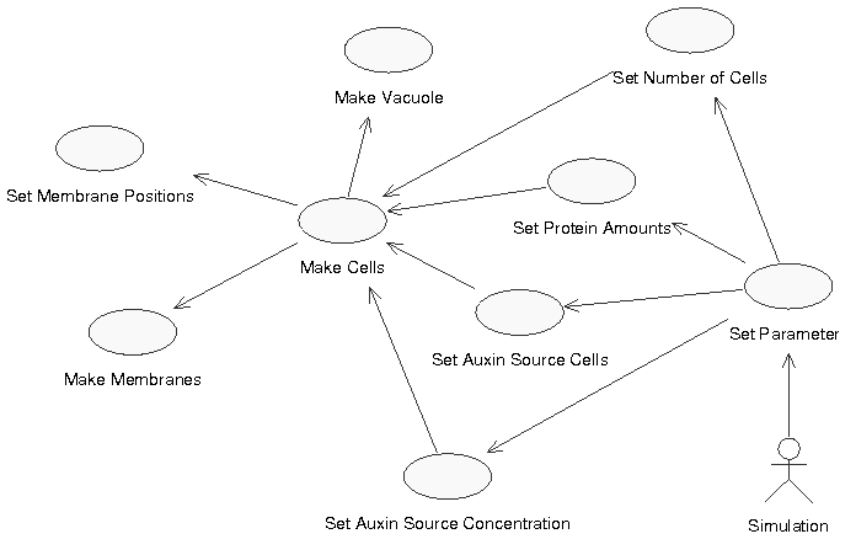

Fig. 9. The Software Model Use Case diagram, with the Simulation user actor.

The Software Model also includes things like the graphical user interface (GUI), and what I/O the simulation needs to do to provide the user or external systems with results.

5.2 Software Model Use Cases

A use case diagram is used to capture the user requirements for using the simulator, in the traditional way (figure 9).

5.3 Software Model Class diagram

The Software Model class diagram is produced from the Domain Model class diagram (figure 4) and the Software Model use cases. Figure 10 shows only the biologically relevant parts of the Software Model class diagram (to improve readability, it omits things like the data and visual output objects).

Certain classes are removed: we decide at this stage not to model the apoplast explicitly in the simulation. It appears as the gap between the cells in the visualisation (later). We also remove the explicit Auxin Canal: this is the emergent property that we desire the simulation to exhibit.

Certain classes are added: we include some inheritance. Proteins and Hormones share some common features, and we model them as subclasses

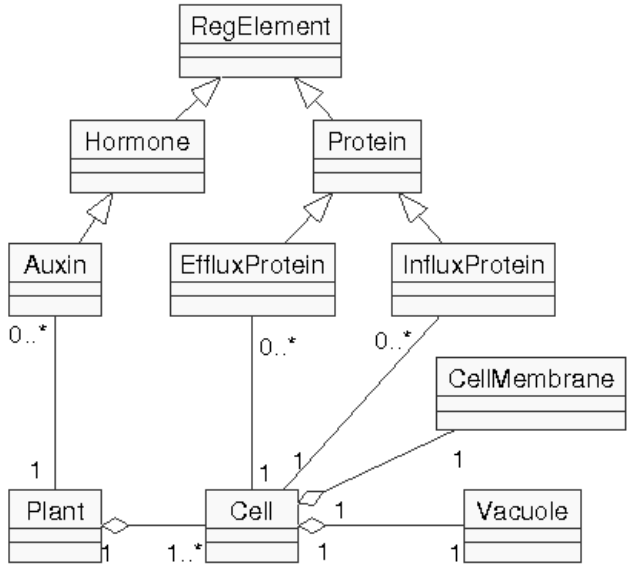

Fig. 10. The Software Model class diagram, showing only the biological part of the model.

of the RegElement (regulatory element) class. Classes that support user interaction are also added (not shown here).

Certain relationships are removed: Auxin is no longer related to Cells. This highlights a difference between the Software Model and the biology. In the Software Model, the Auxin is related only to the Plant, as are the Cells. For ease of implementation with regard to the diffusion of auxin inside and outside cells, the model records the position of the auxin in the simulation space (part of the Plant, and the Cells can query the Plant to discover how much of that auxin is internal to them. This is a suitable implementation strategy, even though it is not a good model of reality; it shows how the "same" objects in the Software Model can be quite different in structure from those in the Domain Model.

A future revision of the simulation will do this differently, by explicitly modelling the apoplast (the space between cells that the auxin is in when not in a cell).

5.4 Software Model State diagrams

The Software Model state diagrams follow the Domain Model, except that the production of AUX/LAX is left out, and expressed at a fixed amount and not (currently) regulated.

6 Refined Software Model

6.1 Refined Software Model class diagram

The Refined Software Model class diagram (figure 11) adds implementation detail to the Software Model class diagram (figure 10). It has all the methods and attributes of the objects (not shown here) and it is use to generate the code skeleton.

The Refined Software Model class diagram includes further details of how some of the biological processes are implemented. For example, the positions of components are held by Position objects and the singleton Hashmap object. It may be advantageous to split things in even more detail if the diagrams become overly complicated, as certain parts of the implementation are more important than others. Things like the implementation of diffusion and how positions of cells and hormones are stored are of greater interest than how the visual output is achieved.

This separation will be even more worthwhile when things like growth are implemented, as they are likely to be complicated and require detailed diagrams. Also, the implementation of such things is much more difficult than conceptualising them, and therefore should be open to more detailed scrutiny to ensure that it is done in a valid way.

The full class model has all the methods and attributes of the classes added, and it is this that is used to generate the code skeleton. Once the biology has been produced in both the class diagram and the state diagrams this is often enough to produce a code skeleton from the UML for the model. It might be necessary to define more clearly the interactions between the objects using sequence and collaboration diagrams if the model is large and complicated.

6.2 Refined Software Model state diagrams

The Refined Software Model links the Software Model state diagrams of different objects, particularly the overlapping parts of the auxin, PIN and cell state diagrams. This shows how a cell producing auxin influences its own state as it responds to the change in auxin concentration by making more PIN protein, and how a cell that does not make auxin, but which detects that there is auxin in its cytoplasm, responds by starting to increase production of PIN.

State diagrams are linked by shared events. The Auxin event of entering a cell, either passively or via a protein, is linked to the Cell event of detecting a change in auxin concentration, which causes the cell to enter into a PIN production state. At the moment, we are performing this linking by textual annotations.

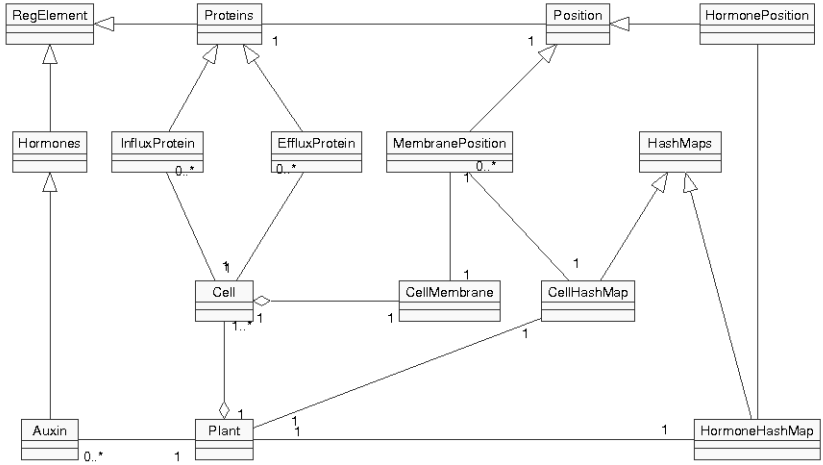

Fig. 11. A simplified implementation class diagram showing more detail of the underlying implementation of some of the biology in the model. It gives more detail on how the positions of different objects are controlled in the model. The full implementation class diagram shows the classes controlling diffusion and how the threading of the program is controlled. Some classes have been omitted for clarity.

This linking of states allows the interactions of the biological objects to be modelled at a higher level than sequence or collaboration diagrams, which are more useful for giving details of how the simulations are going to run. Linking could also be useful to include a notion of space in UML diagrams like state diagrams; for example, states of different objects may influence each other in different ways inside or outside a cell.

7 Simulator: Auxin transporter

We have taken the code skeletons produced from the Refined Software Model UML, and added the remaining code necessary to implement the executing simulator. Some implementation details are discussed here.

7.1 Molecule diffusion

We have implemented two versions of diffusion for the auxin (the work to compare the two has not yet been done).

The first to be implemented is the more simple, and will probably prove to be slightly faster in execution. The method follows an agent

based paradigm, modelling (collections of) auxin molecules as moving agents.

We need to estimate how much auxin is in a cell; this is a difficult estimate to make. The biological data for the amount of auxin in a cell is determined by crushing a section of a plant and measuring how much auxin there is in total, then dividing that value evenly amongst the individual cells. This assumes an equal distribution among the cells, which we believe not to be true, neither are the cells the same size. This method of estimation results in values for the auxin concentration in one cell that can vary by about two orders of magnitude.

In our Simulator, this auxin concentration is divided up into *auxin unit*s, the amount that would occupy 1 unit of space in the model (which is 1 square micron for the 2D model; one cubic micron for the 3D model). From the biological data, this corresponds to 20–2000 auxin molecules in each auxin unit [19, 20].

These auxin units diffuse around the model space. We are testing different ways of implementing this diffusion; currently we are using a random walk. In this approach, a "clump" of auxin molecules (all those corresponding to an auxin unit) move about together as one agent.

To test if this "clumping" is a problem, and whether it might be affecting the results of the simulation, we are currently implementing a second system of diffusion. This is a more continuous style model: every unit of space in the simulation has a number of auxin molecules associated with it. This representation allows a portion of the auxin to move into a neighbouring space, and also allows different areas of the simulation space to have different rules, allowing the rate of diffusion to be altered in different parts of the simulation. This is much more flexible, but is expected to be more costly in computing power. Having both systems is useful as it allows us to see if the more flexible but costly system is necessary, or if we can get enough information out of the simpler faster system, to determine if our hypotheses for auxin canalisation are correct.

We are interested to see if some of the features of the simpler system, that have been tested more, like intra-cellular gradients of auxin, are present in the newer system; and, if they are, can we see any differences.

We are implementing only one diffusion method for the proteins. This is the same as the simple agent based auxin diffusion system. Proteins are much larger than small molecules like auxin, and therefore each protein unit contains fewer protein molecules, and thus the approximation is less problematic.

7.2 2D and 3D simulations

The simulation has been designed to use much of the same code to model either 2D or 3D space. This means that we have 2D and 3D simulations and we know that the underlying algorithms and code is essentially the same. This is important because if we see significant differences in the behaviour of the different simulations for a given hypothesis or set of parameters we can be fairly sure that difference it due to the extra dimension, as opposed to differences in code.

We want a 2D simulator for performance reasons. A 3D simulation of 500 cells in an arrangement of $5 \times 5 \times 20$ cells high (representative of a section of plant stem exhibiting canalisation) might take a few days to run to a point where it can be considered finished (our simulations do not have end points unless some sort of target state is defined, otherwise the existence of a source and a sink allow the simulation to run forever). A 2D simulation of a similar arrangement of 5×20 cells high (representative of a longitudinal cross section) might be finished in less than an hour and require two orders of magnitude less memory (10s instead of 1000s of Megabytes: not only does the 3D simulation have more cells, but each 3D cell is bigger: $50 \times 50 \times h$ voxels, rather than $50 \times h$ pixels), and is comparatively very slow. If we find that a 3D model is necessary for realistic results, we will then investigate more efficient implementation approaches.

An important factor in the 2D simulation is the sizes of intercellular structures like vacuoles. A vacuole in a plant cells takes up a large amount of the space, squashing the cytoplasm to the inside of the cell membrane. Plant cells are 3D, so in our 3D simulation the ratio of cell size to vacuole size can be similar to what we measure in plant cells. What about 2D cells? Should a 2D cell look like a 2D cross section of a 3D cell? If so, should it be a slice through the centre, including the vacuole, or a slice near the membrane, missing the vacuole? Or should the ratio of 2D areas be the same as the ratio of the 3D volumes? Also, in a 3D cell, auxin can go round the vacuole along two axis, but in 2D it is limited to one. Should, therefore, the 2D vacuole have greater permeability to auxin than a 3D cell to account for the loss of a whole route to the other side of the cell?

We are investigating these questions in terms of the various timescales and scaling factors. However, having comparable 2D and 3D simulations based on the same implementation will allow us to validate our answers to these questions.

7.3 User interface

Biologists interact with the simulator through its user interface. In the current version of our simulator, altering model parameters and setting initial conditions can be a difficult process, particularly when defining the spatial arrangement of cells that the simulator uses.

Therefore we have started to develop a Little Language [6] (domain-specific language) to provide an easier interface to setting up the models via an interpreted language. This language provides an interface to two parts of the simulation. Firstly it can be used to set starting parameters for things such as the relationship between auxin and the expression of PIN protein, and the size of cells. Secondly, it allows the user to define what the layout of the cells in the model is, by defining cellular subunits (groups of similar cells) and then defining how the different subunits are arranged in the cell space.

8 Analysis Model: preliminary results

Eventually, will will build an analysis model of the simulator output that is analogous to the domain model. For now we conduct our analysis more informally.

The visual output from the simulation is shown in figure 12. The space between the cell membrane and the vacuole is the cytoplasm where the auxin (black dots) and proteins (gray dots) are synthesised. To avoid questions of sites of synthesis for now there is simply a certain probability that any position in the lattice produces a protein or auxin molecule when required. As the size and space of cells that canalisation occurs in varies quite widely, it is therefore important that a working hypothesis for canalisation in the simulation is not too dependent on size and shape. Therefore it is simple to vary the cell size and the size of the apoplast. It is also possible to have different sizes of cells in the same model, so long as care is taken to ensure that the cells align in a sensible way, without large gaps.

We have tested our first hypothesis for the regulation of auxin canalisation (figure 7) with the model as described here. The first hypothesis is simple and unlikely to be the full story, but it is based on ideas that might form part of the final hypothesis. The hypothesis tested concerns the positioning of the PIN proteins. PIN proteins can diffuse in the cytoplasm. When they associate with the cell membrane they are fixed in one position, unable to move. If the PIN protein exported some auxin on the timestep, it cannot disassociate from the membrane. If it did not do any transport, there is a certain probability that it can disassociate from

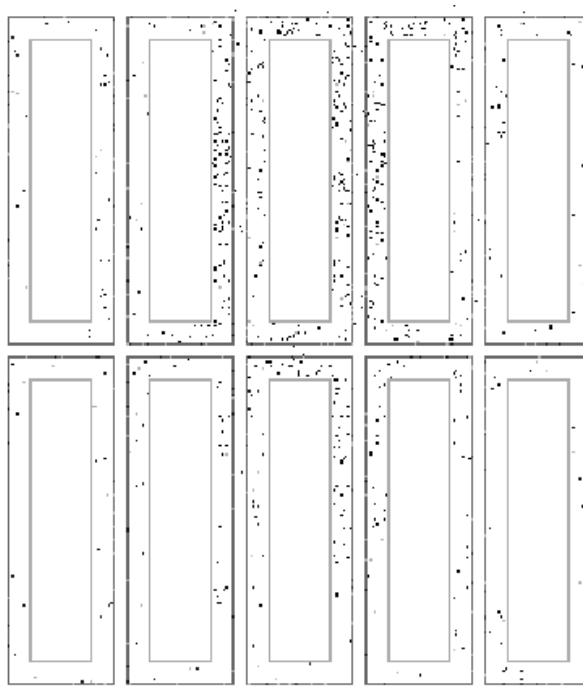

Fig. 12. Screen shot from a running simulation. The darker lines are the cell membranes, and the light lines are the vacuoles. The black dots are *auxin units* and the grey dots are either PIN or AUX/LAX proteins (darkest are the PIN). The middle cell on the top row is the only one producing auxin. The light gray line in the centre at the bottom of the figure is the auxin sink. This is where auxin leaves the model. One dot, such as an auxin unit, is 1 micron in size. To produce a clear figure, the apoplast is increased to 6 microns, from a usual 2 or 3. The cells are 147 microns high and 50 microns wide. For convenience only a few rows of cells are shown: the full simulation currently uses a 5×20 array of cells.

the membrane. The idea is that the PIN would naturally congregate on a membrane where they were able to export auxin.

Simulation of this hypothesis did *not* produce canals in the experiments carried out so far. This could be due to there being no feedback between the extra cellular auxin concentration and the PINs pumping auxin into that space. Therefore only the internal cell conditions stop the PINs from moving around randomly, as the PINs can respond only to the internal auxin concentrations of the cytoplasm, and not to the

auxin in the apoplast. Uneven distribution of auxin suggest that PINs could be responding to these internal conditions.

We intend to experiment further with this idea. We will also carry out more testing with the current hypothesis, as there maybe parameters that allow for the internal gradients of auxin to be enough to position PIN to produce canals.

Next, we will develop our model of auxin canalisation into a more complete model of shoot branching, to inform new models, and to form part of a multilayered approach to modelling shoot branching. Each layer can use an approach to modelling at the level of abstraction most suited to build a more complete picture.

9 Discussion

Our UML assisted development process has provided a number of advantages to our simulations.

The diagrammatic nature of UML as tool for producing various levels of models, including descriptions of program code, has helped produce simulations that not only work in an intuitive way, but that are built intuitively. Biology maps to UML objects in a way that can be understood by developers and biologists alike.

The resulting simulations are flexible. By concentrating on building the biological components and their interactions into the simulations we are able to test various hypotheses for the regulation of auxin transport. The results should be reflections of truly emergent behaviours, rather than due to those behaviours having being hard-coded into the simulation.

The use of different models to capture domain, software, and implementation details has helped produce conceptually cleaner models.

The Domain Model looks purely at the biology. Here class diagrams and state diagrams are of greatest use. The class diagrams allow us to look at the static structure of the model, and how the different parts are connected together. They can include the emergent properties of interest, so that we have these properties captured rigorously in a model. State diagrams provide detailed information of how the objects change in response to events. They are normally produced by thinking about the known biology of the different biological elements.

The Software Model does not explicitly include the emergent properties of the Domain Model: these should emerge from the interactions of the lower level simulated components, and can be compared against the Software Model for plausibility. At this stage inheritance is added to class diagrams, to indicate classifications and generalisations, and to

be used in implementation to reuse code and reduce duplication. The Software Model can also transform biological components to behave in non-biological ways that are more readily simulatable. For example, we may require more states to capture events than are provided in the Domain Model. The existence of the two models highlights areas where the simulation is breaking with the biology.

The biological literature gives details of how the experiments that produced the data were carried out in a lab. It should also be the case that how a simulation works and produces its data should be equally well explained, in order to allow independent validation of results and the sharing of methods and techniques among the modelling community. The increased use of modelling and the complexity of the simulations produced make this a more pressing need. UML can provide an effective way of developing and communicating simulations.

In the long term, UML could be used as an interface into models for biologists to use directly, to extend and develop models themselves. Our current work on the use of a Little Language to provide easier access to some of the deeper parts of the simulations could be extended, for example, to allow the addition of new proteins and their behaviours without the need to delve into Java code. (Of course, there is always the danger that the Little Language itself grows until it is of the complexity of Java. However, the intention is that it should be couched in biological domain specific terms, not generic programming terms.)

Eventually, biologists should be able to draw UML diagrams of these new proteins (or other objects), associate them to other biological objects, and link them to implementation objects that allow them to function. The links with the biology would confer the biological behaviour, and the links with implementation would handle diffusion, positioning, I/O etc. Or alternatively UML could be used as a simulation code navigational aid to allow direct access important parts of a simulation to allow biologists to tailor it to their own needs or add new functionality. UML could allow them to visually locate the part of the simulation that requires editing without looking though large amounts of code and needing to be able to decypher the way the simulation is constructed. The models produced would be more general in the capabilities and allow for more hypotheses to be explored.

Acknowledgements

This work is supported by a BBSRC/Microsoft Research CASE studentship. Thanks to Fiona Polack for discussions about UML models, and to the anonymous referees for their detailed comments. Thanks also to Lauren Shipley for proofreading the paper.

References

[1] *ALife XI, Winchester, UK, September 2008*. MIT Press, 2008.

[2] Paul S. Andrews, Adam T. Sampson, John Markus Bjrndalen, Susan Stepney, Jon Timmis, Douglas N. Warren, and Peter H. Welch. Investigating patterns for the process-oriented modelling and simulation of space in complex systems. In [1].

[3] Paul S. Andrews, Adam T. Sampson, Fiona Polack, Susan Stepney, and Jon Timmis. CoSMoS development lifecycle, version 0. Technical report, University of York, 2008. (in preparation).

[4] Eva Benková, Marta Michniewicz, Michael Sauer, Thomas Teichmann, Daniela Seifertová, Gerd Jürgens, and Jirí Friml. Local, efflux-dependent auxin gradients as a common module for plant organ formation. *Cell*, 115(5):591–602, Nov 2003.

[5] Tom Bennett, Tobias Sieberer, Barbara Willett, Jon Booker, Christian Luschnig, and Ottoline Leyser. The arabidopsis MAX pathway controls shoot branching by regulating auxin transport. *Curr Biol*, 16(6):553–563, Mar 2006.

[6] J. Bentley. Programming pearls: little languages. *Communications of the ACM*, 29(8):711–721, 1986.

[7] Peter Checkland and Jim Scholes. *Soft Systems Methodology in Action*. Wiley, 1990.

[8] S. Efroni, D. Harel, and I. R. Cohen. Toward rigorous comprehension of biological complexity: modeling, execution, and visualization of thymic T-cell maturation. *Genome. Res.*, 13(11):2485–97, jul 2003.

[9] François G. Feugier and Yoh Iwasa. How canalization can make loops: a new model of reticulated leaf vascular pattern formation. *J Theor Biol*, 243(2):235–244, Nov 2006.

[10] A. Finney and M. Hucka. Systems biology markup language: Level 2 and beyond. *Biochem Soc Trans*, 31(Pt 6):1472–1473, Dec 2003.

[11] Jasmin Fisher and Thomas A. Henzinger. Executable cell biology. *Nat Biotechnol*, 25(11):1239–1249, Nov 2007.

[12] Martin Fowler. *UML Distilled*. Addison-Wesley, 3rd edition, 2004.

[13] Verônica A. Grieneisen, Jian Xu, Athanasius F. M. Marée, Paulien Hogeweg, and Ben Scheres. Auxin transport is sufficient to generate a maximum and gradient guiding root growth. *Nature*, 449(7165):1008–1013, Oct 2007.

[14] M. Hucka, A. Finney, S. Hoops, S. Keating, and N. Novere. Systems biology markup language (SMBL) level 2: Structures and facilities for model definitions. *Nature Precedings*, 58(2), 2007.

[15] M. Hucka, A. Finney, H. M. Sauro, H. Bolouri, J. C. Doyle, H. Kitano, A. P. Arkin, B. J. Bornstein, D. Bray, A. Cornish-Bowden, A. A. Cuellar, S. Dronov, E. D. Gilles, M. Ginkel, V. Gor, I. I. Goryanin, W. J. Hedley, T. C. Hodgman, J-H Hofmeyr, P. J. Hunter, N. S. Juty, J. L. Kasberger, A. Kremling, U. Kummer, N. Le Novère, L. M. Loew, D. Lucio, P. Mendes, E. Minch, E. D. Mjolsness, Y. Nakayama, M. R. Nelson, P. F. Nielsen, T. Sakurada, J. C. Schaff, B. E. Shapiro, T. S. Shimizu,

H. D. Spence, J. Stelling, K. Takahashi, M. Tomita, J. Wagner, and J. Wang. The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. *Bioinformatics*, 19(4):524–531, Mar 2003.

[16] IBM. Developer of Rational Rose UML editing software. *http://www-306.ibm.com/software/awdtools/developer/rose/index.html*, 2008.

[17] N. Kam, I. R. Cohen, and D. Harel. The immune system as a reactive system. In *Proc Visual Lang. and Formal Methods*. IEEE, 2001.

[18] Eric M. Kramer. PIN and AUX/LAX proteins: their role in auxin accumulation. *Trends Plant Sci*, 9(12):578–582, Dec 2004.

[19] Eric M. Kramer. Computer models of auxin transport: a review and commentary. *J Exp Bot*, 59(1):45–53, Apr 2008.

[20] Eric M. Kramer. Mode parameters: the devil in the details. *The understanding and modelling of auxin transport in plants. Advanced Workshop, Nottingham University*, page 16, 2008.

[21] G. Mitchison. A model for vein formation in higher plants. *Pro. R. Soc. Lond.*, 207:79–109, 1980.

[22] G. Mitchison. The polar transport of auxin and vein patterns in plants. *Phil. Trans. R. Soc. Lond.*, 295:461–471, 1981.

[23] D. Morris, J. Friml, and E. Zamimalova. Auxin transport. In P. Davies, editor, *Plant Hormones: Biosynthesis, Signal Transduction, Action!*, chapter E1, pages 437–470. Kluwer Academic Publishers, 2004.

[24] J. Odell, H. Parunak, and B. Bauer. Extending UML for agents. In *AOIS Worshop at AAAI*, 2000.

[25] OMG. Maintainer of the UML standards. *http://www.omg.org*, 2008.

[26] Veronica Ongaro and Ottoline Leyser. Hormonal control of shoot branching. *J Exp Bot*, 59(1):67–74, Aug 2008.

[27] Tomasz Paciorek, Eva Zazímalová, Nadia Ruthardt, Jan Petrásek, York-Dieter Stierhof, Jürgen Kleine-Vehn, David A. Morris, Neil Emans, Gerd Jürgens, Niko Geldner, and Jirí Friml. Auxin inhibits endocytosis and promotes its own efflux from cells. *Nature*, 435(7046):1251–1256, Jun 2005.

[28] Fiona A. C. Polack, Tim Hoverd, Adam T. Sampson, Susan Stepney, and Jon Timmis. Complex systems models: Engineering simulations. In [1].

[29] Przemyslaw Prusinkiewicz and Anne-Gaëlle Rolland-Lagan. Modeling plant morphogenesis. *Curr Opin Plant Biol*, 9(1):83–88, Feb 2006.

[30] Didier Reinhardt, Eva-Rachele Pesce, Pia Stieger, Therese Mandel, Kurt Baltensperger, Malcolm Bennett, Jan Traas, Jirí Friml, and Cris Kuhlemeier. Regulation of phyllotaxis by polar auxin transport. *Nature*, 426(6964):255–260, Nov 2003.

[31] T. Sachs. The control of the patterned differentiation of vascular tissues. *Advances in Botanical Research Incorporating Advances in Plant Pathpology*, 9:151–262, Jan. 1981.

[32] S. J. Singer and G. L. Nicolson. The fluid mosaic model of the structure of cell membranes. *Science*, 175(23):720–731, Feb 1972.

[33] Petra Stirnberg, Karin van De Sande, and Ottoline Leyser. MAX1 and MAX2 control shoot lateral branching in Arabidopsis. *Development*, 129(5):1131–1141, Mar 2002.

[34] C. Uggla, T. Moritz, G. Sandberg, and B. Sundberg. Auxin as a positional signal in pattern formation in plants. *Proceedings of the National Academy of Sciences of the United States of America*, 93:9282–9286, 1996.
[35] K. Webb and T. White. UML as a cell and biochemistry modeling language. *BioSystems*, 80:283–302, 2005.

Simulating biology: towards understanding what the simulation shows

Paul S. Andrews[1], Fiona Polack[1], Adam T. Sampson[2], Jon Timmis[1,4], Lisa Scott[3], and Mark Coles[3]

[1] Department of Computer Science, University of York, UK
{psa,fiona}@cs.york.ac.uk
[2] Computing Laboratory, University of Kent, Canterbury, UK
[3] York Centre for Immunology and Infection, University of York, UK
[4] Department of Electronics, University of York, UK

Abstract. When building simulations of complex systems the task of validation is often overlooked. Validation helps provide confidence in the simulation by exploring the link between the models that we build and the real complex system. We investigate software engineering validation techniques from outside the area of complex systems to assess their applicability for the types of simulation we build. We then provide an example of how such techniques can be applied to a complex systems simulation of cells migrating from blood vessels into lymph nodes through the walls of the blood vessels. We suggest that explicitly stating the modelling and simulation assumptions we make is key to the process of validation. Concluding, we highlight a possible process for validating complex systems that explicitly incorporates environmental aspects.

Keywords: complex systems, simulation, modelling, validation

1 Introduction

Simulations are used to model complex systems such as biological phenomena, economies and human societies, for use in research investigation *in vivo*, *in vitro* and *in silico*. These systems are complex in the sense of having elaborate behaviour at a high level that is the consequence of

many simple behaviours at a lower level. The high-level behaviour cannot be deduced as a simple combination of low-level behaviours. Space, time and the environmental context are critical features.

In this paper, we present an initial engineering exploration in to the validity of a complex system simulation of part of the immune system, based on collaborative work with the York Centre for Immunology and Infection (CII), undertaken as part of the CoSMoS project[5]. We focus on a part of the conceptual model, and consider how, and to what extent, it is possible to validate this against the biological evidence of *in vivo* experiments. Our modelling assumptions, and validation problems, bring into consideration the importance of local environmental factors in modelling such complex systems. The analysis also highlights the limits of biological imaging based technologies that cannot provide dynamic insight in to the key features of the system; this in turn highlights areas where the biological research could focus.

Section 2 presents a review of simulation and its use in scientific research. The following section summarises biological research on the migration of lymphocytes to lymph nodes. We then explore the engineering validation of our simulation. Finally, in section 6, we propose an extension of Sargent's simulation process to express some of the complications of complex-systems modelling and simulation engineering.

2 Computer simulation: science and engineering

Computer simulation has been used to explore biological systems for many years. Traditional simulations generate output from equations (differential equations, Markovian models, etc.) that have been developed to mirror trends or behaviour in, for instance, biological populations. More recently, computer simulation has been used to model the possible effects (co-ordinated or emergent) of biological components or organisms acting in their environment. Essentially, simulations have two purposes: some are built in co-operation with research scientists in an effort to improve scientific understanding of natural systems; others are built as artificial systems to construct and explore alternative realities (either as science fiction or visions for future engineering). Here, we focus on a case study whose purpose is to contribute to scientific understanding in immunology permitting simulation of events that are difficult to experimentally validate *in vivo*. The aim is to produce a simulation in which the biologist has confidence and can help direct their wetlab experimental research.

[5] The CoSMoS project, EPSRC grants EP/E053505/1 and EP/E049419/1, http://www.cosmos-research.org/, is building capacity in generic modelling tools and simulation techniques for complex systems.

Computer simulation of biological phenomena is important because static models cannot capture the dynamic features that characterise the behaviour of complex systems. For example, systems biologists are increasingly adopting conventional software engineering design diagrams to express static structures, and patterns of interaction in their models; these modelling approaches cannot express time, space or the the features and consequences of large numbers of interacting instances [24]. Time and space (or at least spatial orgainisation) are essential to complex systems behaviour. A key aspect of space, which is also outside the remit of conventional modelling, is environmental interaction – both among components, and of components with their local environment: the behaviour of a complex system depends critically on the way that the (collective) components interact with their environment over time; failure to adequately model the environmental context naturally leads to non-realistic models of the complex system.

Thus, a scientifically-valid simulation has to extract suitable environmental aspects, at an appropriate level of abstraction; it also has to provide evidence that its environmental representation, as well as its scientific model, are adequate abstractions from the biological reality.

Despite the importance of realistic abstraction, the validity of computer simulations has not been a significant concern of simulators, except in the critical systems context. Typically, a simulation is judged by its ability to produce something like the expected results by a process that looks a bit like reality, and there is little concern for the quality or scientific relevance of the underlying simulation [11]. Simulations can be misleading, for instance because the output captures artefacts of the simulation software rather than patterns of behaviour, or where the attempt to approximate reality results in simulations whose complexity is as impenetrable as that of the observed system.

Scientific validity, like engineering validity, means that it must be possible to demonstrate, with evidence, how models express the scientific realities. Validity implies both adequate abstraction, and adequate development processes. Many computer simulations are poorly engineered: there is little attempt to record design of components or of the environmental context used in the simulation system. Assumptions and generalisations are not documented, and are thus not exposed to scientific scrutiny. An immediate result of this focus is a long-running intellectual debate about whether it is possible to do science through simulation (see [19, 22, 36, 5]). Bullock (in [36]) observes that, to assess the role and value of complex systems simulation, we need to address deep questions of comparability: we need a record of experience, of how good solutions are designed, of how to chose parameters and calibrate agents,

and, above all, how to validate a complex system simulation. To address these problems, we need principled approaches to the development of computer simulations.

For inspiration in engineering scientifically-valid simulations, we turn to two areas: non-complex critical systems, and agent modelling. Computer simulation of non-complex (but nevertheless *complicated*) systems has a long history, and the need to assure high-integrity and critical-systems models has led to a corpus of work on developing, verifying and validating simulation models. In intelligent agent modelling, there are methods for systems development that focus attention on the different aspects that need to be considered; some agent work has been taken forward for use in complex critical systems at the social, or macro, scale.

2.1 The process of simulation development

In high-integrity systems engineering, the foundations of a simulation process date from the late 1970s. For instance, Sargent (e.g. [28, 30]) presents a process (figure 1) that starts with a *problem entity*, or description of the phenomenon to be modelled. From the problem entity, a *conceptual model* is developed in a suitable representation – Sargent reviews diagrammatic models [29], and also notes mathematical or logical modelling paradigms [30]. Finally, a *computerised model* implements the conceptual model.

The simulation process is an iterative cycle, which includes an *experimentation* link between the problem entity and the computerised model. This allows the developers to trial-and-error simulation elements and settings, and to compare the results to the problem entity.

The second notable aspect of the simulation process is the explicit inclusion of verification (in the software engineering of the computerised model) and validation – both of models against reality, and of the data used to test the conceptual and computerised models. We return to this aspect in the next section.

The simulation process has much in common with conventional software engineering lifecycles – it presents a high-level summary of the necessary attributes of a development, rather than a comprehensive guide to achieving a high-quality engineered product. This area is addressed to some extent in agent-oriented modelling.

Sudeikat et al [33] give an insightful review of multi-agent system development methods, which, like Sargent, focuses on matching methods to the requirements of specific simulation targets. They identify as the foci of current methods: *internal architecture*, *social architecture*, *communication*, *autonomy*, *pro-activity* and *distribution*. Some of the re-

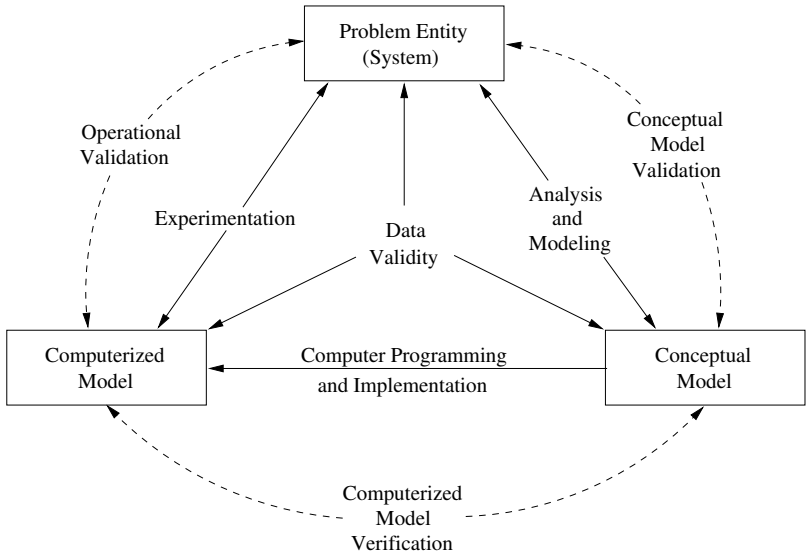

Fig. 1. Sargent's model of the simulation development process [28]

viewed methods are sophisticated software-engineering approaches, such as Prometheus [21] (figure 2).

These methods provide an integration layer for the styles of model (usually diagrams) that are needed to express the static structures and interactions of the overall conceptual model. Implementation is often well researched, with platforms, patterns, and workbenches (see, for instance, the ACE resources, www.econ.iastate.edu/tesfatsi/ace.htm). Agent modelling methods are used in robotics, social agent systems, and similar areas, but are also entering high-integrity systems engineering: for instance, Alexander [1] uses Prometheus in the simulation of command-and-control systems. Critical systems use means that some work exists on adding validation activities to the modelling activities covered in the original methods.

Two immediate issues arise with the agent modelling methods. The first is that they are oriented to social systems – including human-scale high-integrity systems. This means that the technical emphases are on capturing the autonomy or design for learning – the BDI (beliefs-desires-intentions [12]) of agents. The second issue is that these methods do not capture the time, space, and component-quantity aspects of complex systems, or the layered abstraction aspect, noted above. The representation

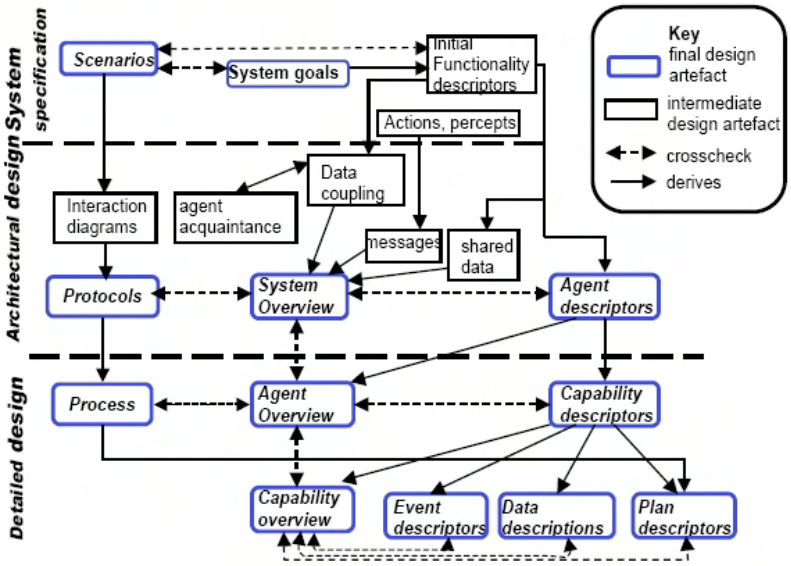

Fig. 2. The Prometheus development method [21]

of these key features are left to the inspiration of the implementer, and are thus hidden from validation scrutiny.

2.2 Verification and validation of simulations

There is a significant corpus of work from outside the area of complex systems on validating and verifying simulation models. However, although this work proffers general reminders, its direct advice is difficult to adapt to complex system simulation. Zeigler [39] presents a theory for modelling and validation of simulations; his theory is predicated on the fact of a homomorphism between conceptual models and simulations, and does not provide obvious pointers as to how the homomorphism is established – that is, verification of the development process is assumed.

Similar validation concepts come from Sargent, elaborating the validation aspects of the simulation process (figure 1, above). Sargent [30] reminds us that a *model should be developed for a specific purpose... and its validity determined with respect to that purpose.* Furthermore, Sargent notes that the level of assurance needed depends on the purpose of the simulation, and should be set independently of the development of the

Technique	**Comments on Sargent's suggestions**
Animation, operational graphics	Specifically, graphical visualisation, either of system behaviour or of operational parameters
Comparison to other models	Comparison to *valid* analytical models or other simulation models
Degenerate tests, extreme condition tests, parameter variability, sensitivity analysis	Typical domain-style testing of behaviour under normal and extreme input and operating conditions
Event validity	Compare the events in real and simulated systems
Face validity (ask a domain expert), traces	Appeal to logic or to domain experts to check the validity of model components or data
Historical data validation, predictive validation	Either drive a simulation with historical data and compare results to reality; or drive a simulation on current data and compare to independent predictions of future
Multi-stage validation, combining historical methods:	Three historical approaches can be combined to develop based on sound theory and assumptions, with empirical validity checks where possible.
Rationalism	The veracity of assumptions is rationally justifiable, and valid models arise from valid assumptions
Empiricism	All assumptions and outcomes are empirically validated
Positive economics	The model can predict the future, so causal relationships and mechanisms are of no concern
Internal validity	Used on stochastic models; comparison of consistency of results across runs
Turing tests	Can an expert tell that it is not the real system?

Table 1. Validation techniques for simulation development (based on descriptions from [30])

simulation – good software engineering practice. Sargent's development process (lifecycle) for simulations explicitly incorporates verification and validation activities, and he proposes a range of approaches to validation, summarised in table 1.

Clearly, some of Sargent's suggestions are inappropriate for complex systems work: if we knew the workings of the complex system well enough to understand event validity and traces, we would not need a computer simulation for research purposes. However, wherever such internal analyses are possible, they should be conducted. We need to be

confident that computer simulations accurately replicate contributory non-complex features. Comparison with real systems is essential, but is potentially dangerous – variants on predictive validation can lead to (accidental) construction of simulations that are self-fulfilling prophesies, whilst historical data validation tends to pick only the data that best match the simulation.

Perhaps the most useful of Sargent's suggestions relate to analysis of assumptions – though situating this in historical theories of rationalism and empiricism tends to mask their value. A common (possibly universal) failing of research simulations is the failure to document the assumptions that they make, both about the science that underpins the models, and about the means used to create the simulation. We will return to assumptions in the case study, below.

2.3 Other computer simulations for biological research

There are a number of current interdisciplinary research projects that use aspects of software engineering to produce high-quality computer simulations to support biological research.

PEPA [6, 7] is typical of several approaches that use stochastic process algebra to construct models of cell signal transduction pathways. In PEPA, complementary models are developed of a reagent view and a network view. The models are proven isomorphic with each other, and isomorphic with conventional differential equation models of transduction. Verification (that the implementation captures the conceptual model) is formal and explicit, and the ability to mimic the analytical models that the biologists create contributes to the validation of the conceptual model against the reality. This is akin to Sargent's *comparison to other models* technique, although, in common with other differential equations, the validity of the analytical models is difficult to show.

Reactive Animation (RA) [10, 9, 14] is another robustly-engineered research simulation; it uses Rhapsody statecharts (state machines) and Live Sequence Charts (connectivity diagrams), plus data from biological experimentation, to drive powerful biological visualisations (for instance, of T cell activity in the thymus). RA reverse engineers biological systems, using a well-understood software engineering analogy [27]. RA thus exemplifies a number of Sargent's comparatively-based validation techniques, as well as quality software engineering design and verification. In both PEPA and RA, high-quality computer engineering and attention to validation produces simulations that biologists can rely on to direct their research.

Whilst these initiatives are both scientifically and computationally successful, they are not easily generalisable. The specialised components

(such as process algebras) and proprietary tools (such as Rhapsody state charts) tie the initiatives closely to the groups that own them. This makes it hard to do a comprehensive evaluation of PEPA or RA as candidate for a general complex system development process – in other words, it is hard to generalise from these otherwise excellent initiatives in biologically-driven computer simulation.

In our work, we take inspiration from Sargent's process and RA modelling. Our conceptual model is a very simple form of state machine, to express the possible states of a lymphocyte. Like conventional agent modelling, much of the environmental context is captured between the conceptual model and the implementation. However, we then apply Sargent's validations, and a deviational approach to assumption generation by providing evidence for arguments (based on work by Pumfrey [25], Srivatankul [31], Allenby [2] and others applying deviational analyses to safety or security assurance work). This approaches to engineering assurance not only reveals limitations (and strengths) of our simulation, but can also be used to explore the effect of limitations of the biological knowledge of the system.

The following section provides biological background on the case study. We then review our simulations, and discuss connotations of our findings for complex systems modelling.

3 Migration of lymphocytes in the lymph node

3.1 Lymph Nodes

The mammalian immune system possesses many specialised cells that are collectively known as the leukocytes or white blood cells. The leukocytes can be divided into a number of distinct groups with different functionalities. One such group is the lymphocytes, which are vital to recognising and mounting an immune response to various harmful pathogens such as bacteria and viruses. The lymphocytes can be further classified into two distinct populations of cells: B cells and T cells. As well as specialised cells, the mammalian immune system also comprises a number of immune organs. One such organ is the lymph node, which is a small (about the size of a pea in humans) bean-shaped immune organ (figure 3) rich in lymphocytes and other leukocytes, providing a place where immune response to pathogens in the lymph may be triggered and develop. The structure of the lymph node is made up of a number of specialised areas supporting different cellular environments. There are hundreds of lymph nodes in various locations around the body.

Bodily fluid known as lymph drains into lymph node through a number of afferent lymph vessels connected to the *lymphatic system*; lymph

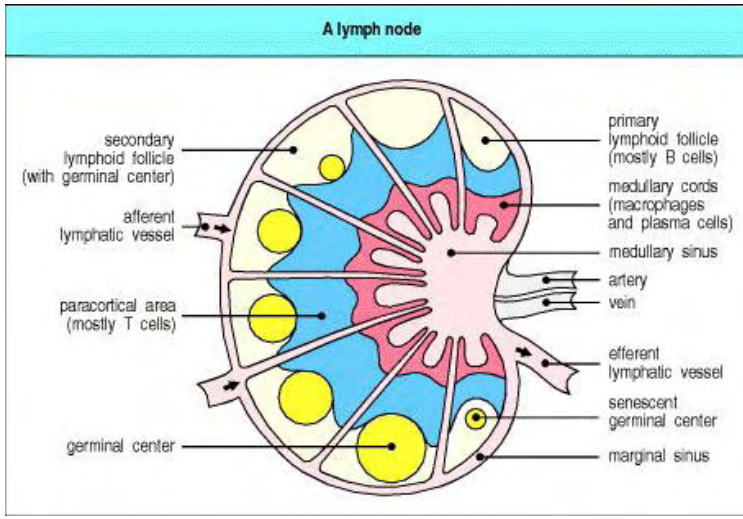

Fig. 3. Structure of a lymph node from [16]

leaves the lymph node through a single efferent lymph vessel. Lymph contains many different leukocytes, proteins and other particles (possibly pathogenic) that have drained from the peripheral parts of the body. The lymph node is also connected to the circulatory system via a lymphatic artery and vein. It is through the lymphatic artery that lymphocytes enter the lymph node. Lymphocytes can then migrate through specialised blood vessels (via a mechanism described below) into the functional tissue of the lymph node. Once there, lymphocytes can interact with other leukocytes that have encountered pathogens and entered the lymph node from the lymph, to initiate an appropriate immune response.

3.2 Endothelial Cells, Pericytes and High Endothelial Venules

Figure 4 summarises the main types of blood vessel present in the body: arteries are the large vessels that carry oxygenated blood from the heart; arterioles branch off the arteries carrying blood to the capillaries; capillaries are the smallest blood vessels in the body and allow the interchange of components between the blood and body tissues; venules carry the de-oxygenated blood from capillaries; and veins then carry de-oxygenated blood back to the heart.

The internal surface of all blood vessels is made up of endothelial cells. In the lymph node, a minority of the venules have plump (high)

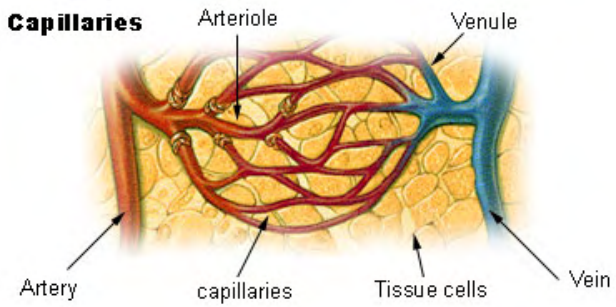

Fig. 4. The relationship between the different types of blood vessel from http://training.seer.cancer.gov/module_anatomy/images/illu_capillary.jpg

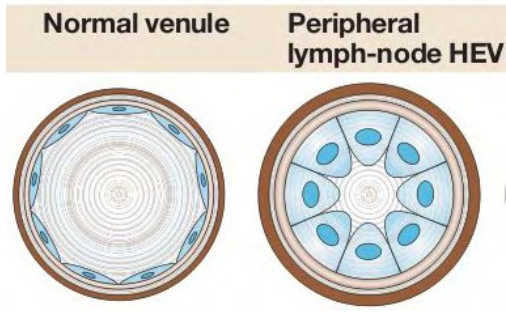

Fig. 5. A comparison of a normal venule with a lymph node HEV from [20]. In the HEV, the ring of endothelial cells are much larger.

endothelial cells that have a significantly larger diameter than normal endothelial cells. These venules are called the high endothelial venules (HEVs) [15] (see figure 5). Areas of HEVs occur in the lymph node at various points (figure 6).

It is in HEVs that lymphocytes can migrate from the blood though the endothelial cells into the functional tissue of the lymph node. Only lymphocytes can interact with and cross HEVs, other leukocytes are excluded [20]. It is estimated that a quarter of the circulating lymphocytes migrate from the blood after entering an HEV [13].

HEVs (and other small blood vessels) are surrounded by pericytes, shown in figure 7. They are a form of vascular smooth muscle cell surrounding endothelial cells that are responsible for constriction and dilation of blood vessels. This regulates blood flow and diameter of the HEV, and thus affects the ability of lymphocytes to migrate [26]. A large influx

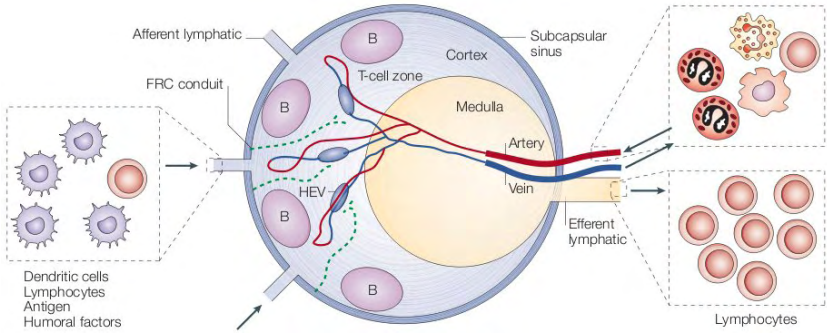

Fig. 6. The flow of cells in and out of a lymph node from [20]. Lymphocytes and dendritic cells enter the lymph node by two routes. Most dendritic cells enter though afferent lymph, settling near HEVs in paracortex (due both to lymph node structure and local production of chemokines). Most lymphocytes enter the lymph node across HEVs.

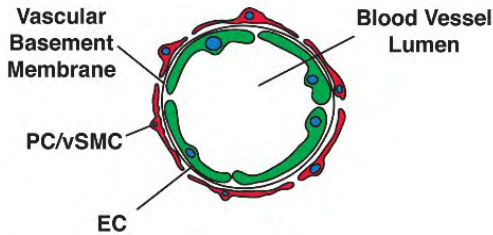

Fig. 7. Cross-section of a capillary from [4]. EC = endothelial cell, PC = pericyte, vSMC = vascular smooth muscle cell.

of lymphocytes into the lymph node during an immune response causes it to visibly swell. This is know as lymph node hypertrophy.

3.3 Lymphocyte Rolling and Migration in the HEV

All leukocytes are able to migrate through blood capillaries via the same mechanism. Only lymphocytes, however, can migrate through the specialised HEV in a lymph node. The process of migration is due to an adhesion cascade of various cell surface receptors and molecules. Originally identified as a three-step process of rolling, activation and arrest, the migration process has now been augmented, as shown in figure 8. Each step is initiated and regulated by specific signalling molecules and receptors [18].

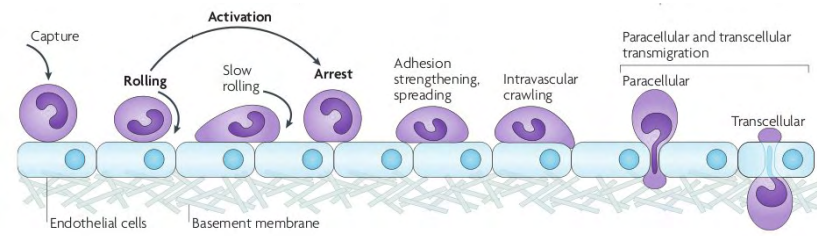

Fig. 8. The leukocyte adhesion cascade from [18]. The three historical steps are shown in bold.

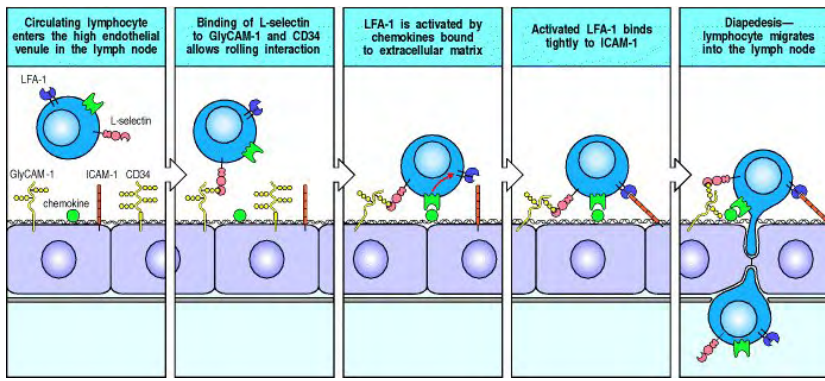

Fig. 9. Migration of T-cells across HEVs from [16]

To migrate, a leukocyte must pass through endothelial cells, the endothelial-cell basement membrane and pericytes. Migration through endothelial cells can be rapid (2-5 minutes), whilst penetrating basement membrane takes longer (5-15 minutes) [18].

Migration of lymphocytes from HEVs is controlled by a specific array of adhesion molecules that facilitates lymphocyte migration but bars other leukocytes (figure 9). Chemokines (chemical signalling molecules) that are produced by or adherent to HEVs are important in the control of lymphocyte migration, however, the precise mechanisms by which chemokines work *in vivo* are unclear [16]. Several chemokines are probably required for the movement of lymphocytes through HEV, but how many and in what order is unclear [20].

4 Developing a lymphocyte migration simulation

In section 2.2 we highlighted the importance of explicitly stating the purpose of a simulation as it will directly influence the design decisions and assumptions we make. Our aim is to model the migration of lymphocytes through the high endothelial venules (HEV) of the lymph nodes, and construct computer simulations of the lymphocytes as a complex system. Using these simulations we can investigate possible factors that lead to the observed lymph node hypertrophy during infection. We are interested in how the number of migrating lymphocytes changes under different conditions, thus the desired output of any simulation will be numerical data detailing lymphocyte migration rates. The simulations should enable us to test hypotheses such as *the increase in lymphocytes in the lymph node during infection is due to dilation of the high endothelial venules.*

In this section, we outline the processes that we use to develop the simulations; in the section 5 we turn to validation. Following Sargent's process, figure 1, the biological literature summarised above provides the starting context for our case study, the *problem entity.* Our *conceptual model* is first described followed by our *computerised model* or *simulations.* To aid the analysis of our modelling and simulation process, we have laid out our assumptions in table 2. In the descriptions that follow, we refer to this table where necessary.

4.1 A conceptual model

The first step in building the conceptual model is to identify the different parts of the system in which we are interested. In terms of a complex system we can consider the hypertrophy of the lymph node to be the non-linear (emergent) behaviour under investigation. The main active component of our system is a population of homogeneous lymphocytes, which interacts in an environment. This environment is simply the parts of the body with which the lymphocytes interact with respect to their migration through HEV in the lymph node.

In terms of the biology, the main cellular actors in the migration process are the lymphocytes, high-endothelial (HE) cells and pericytes. We consider HE cells and pericytes together in the form of a tube (a high-endothelial venule). Lymphocytes travel through the HEVs within the blood circulation. Outside the HEVs is the lymph node tissue, which the lymphocytes enter if they successfully migrate through the HEV. We have classified the different environments that the lymphocytes pass through as states, with transitions occurring when the lymphocyte moves

Label	Assumption
1	The detail described in section 3 is correct
2	Lymphocyte migration only takes place in the HEV areas of the lymph node
3	There is no interaction between lymphocyte agents. Lymphocytes do not collide.
4	There are no effects from external blood circulation e.g. blood flow is constant.
5	The volume of blood always there to accommodate size of HEV (i.e. enough blood to fill whatever size HEV expands to).
6	Once a suitable chemokine signal has been received by a lymphocyte, it will always migrate. Thus, subsequent stages in the adhesion cascade (see figure 8) are deterministic.
7	Lymphocytes are essentially equivalent. They express the same levels of receptors required for rolling and migration.
8	Lymphocyte will always re-enter blood circulation from the lymph node. Thus, lymphocytes can only exit the system (die) in the blood circulation state.
9	Lymphocytes are created and die at a constant rate.
10	The HEVs are homogeneous. The endothelial cells and pericytes that make up the HEV all behave the same making the HEV appear the same at all points.
11	Lymphocytes flow though the HEV at the same rate.
12	Lymphocytes can be captured and disassociate repeatedly whilst in the HEV.
13	There is no change in lymphocytes that have disassociated from an HEV wall, thus all free flowing lymphocyte are the equally likely to capture.
14	Whilst passing through the HEV there is no change in the state of the lymphocyte, thus there is no distinction between new and re-circulating lymphocytes.
15	Proliferation does not occur in the lymph node.
16	The multi-stage adhesion cascade show in figure 8 can be reduced to two probabilistic stages: capture leading to lymphocyte rolling, and migration after receiving a chemokine activation signal.
17	The number of lymphocyte chemokine receptors does not change on the time scale of the simulation.
18	Lymphocytes drain from the lymph node to the blood circulation at a constant rate.

Table 2. A list of many of our modelling and simulation assumptions. Each is given a label so that we can refer to it. This is not a complete list, but is illustrative of the kind of assumptions we make.

from one environment to the other. This is summarised by the *state diagram* in figure 10. The four key identified states are:

Blood Circulation: This state encompasses the parts of the body that the lymphocyte is in when it is not in the HEV or the lymph node tissue. It provides a place where lymphocytes can enter or leave the system. It is the state lymphocytes will be in for most of their existence. Assumptions 4, 5, 8 and 9 apply to this state.

HEV Lumen: This state describes the lymphocyte when it is flowing freely in the lumen of a HEV. Assumptions 10, 11, 13 and 14 apply to this state.

Rolling: This state represents the lymphocyte when it is rolling on the interior surface of an HEV (see figure 8). Assumptions 10 and 13 apply to this state.

Lymph Node: This state describes the lymphocyte when it is present in functional tissue of a lymph node. Assumptions 15 and 18 apply to this state.

In addition to these four states, **Start** and **Stop** states provide a means to introduce and remove a lymphocyte.

The state transitions in figure 10 map to the biology in the following ways:

Creation A newly created lymphocyte will automatically transition into the *blood circulation* state. Assumption 9 applies to this transition.

Enter HEV As a lymphocyte is transported around the body in the blood (the *blood circulation* state) it will at some point enter an area of HEV a lymph node. Assumptions 10 and 11 apply to this transition.

Exit HEV Just as a lymphocyte can enter the HEV lumen, it can also exit the HEV lumen via the blood, transiting from the *HEV lumen* state back into the *blood circulation*. Assumptions 10 and 11 apply to this transition.

Capture Whilst moving freely in the HEV lumen, a lymphocyte captures onto the endothelial wall transiting to the *rolling* state. Assumptions 7, 12, 13 and 16 apply to this transition.

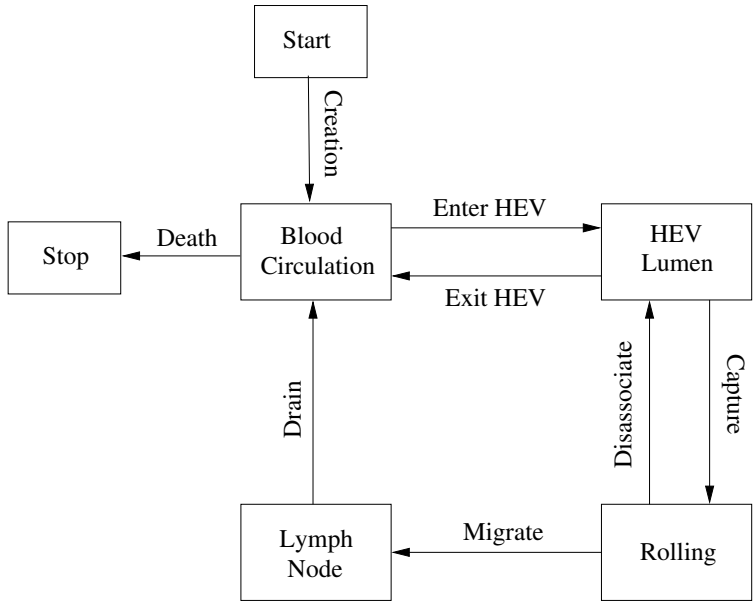

Fig. 10. A generic state transition diagram for a single lymphocyte. Boxes represent states a lymphocyte can be in and arrows represent the possible transitions between states.

Disassociate Just as the lymphocyte can transition from flowing freely in the lumen to rolling, it can also disassociate from *rolling*, moving back to the *HEV lumen*. Assumptions 7, 12, 13 and 16 apply to this transition.

Migrate During the process of rolling, a lymphocyte receives a chemokine signal from the endothelial surface of the HEV. If sufficient, this signal produces a change in the confirmation of receptors on the lymphocyte leading to a cascade that results in migration of the lymphocyte from the HEV to the functional tissues of the lymph node. Assumptions 2, 6, 7 and 16 apply to this transition.

Drain After spending time in the lymph node, a lymphocyte drains in to the lymphatic system via the efferent lymphatic vessel to rejoin the blood circulation (see figure 6). Assumption 18 applies to this transition.

Death Whilst circulating in the blood, a lymphocyte dies. Assumptions 8 and 9 apply to this transition.

Only the allowable transitions are shown, for example a lymphocyte in the *blood circulation* state can only enter the HEV via the *enter HEV* transition or cease via the *death* transition. In addition, as this is a conceptual model, the eight state transitions have a meaning, but no specific values. The values are assigned at the simulation stage, taking the form of probabilities.

The main simplification (see Assumption 16) we make with regards to lymphocyte rolling is to reduce the multi-stage adhesion cascade (represented in Fig. 8) down to two main steps. The first step, captured by the *capture* transition, models the capture of lymphocytes on to the endothelial wall. Once capture has occurred, the lymphocyte is in the *rolling* state, waiting to be activated by a chemokine signal. The second step, expressed by the *migration* transition, models the lymphocyte receiving the chemokine signal; after this it is assumed that the lymphocyte succeeds in migration. Other stages in the cascade are assumed to be either deterministic, or have such small probabilities of failing that they are insignificant.

4.2 Simulations

A simulation is best thought of as an execution of a model (such as the lymphocyte model in figure 10) over time. Typically time is implemented as atomic steps, at which each element in the simulation (each lymphocyte) updates. Within a simulation, we need to define rules to determine when a lymphocyte can transition between states. One way to achieve this is to assign each transition with a probability of occurring. These probabilities need to be extracted from the biological detail to represent what is known to happen in the biology.

We have developed two simulations of the conceptual lymphocyte migration model described above, which differ in the way spatial aspects of the environment are represented. In the first, there is no explicit coordinate system, only the four body locations in figure 10. Each of these four state spaces can contain a number of lymphocyte agents, which transit from location to location based on a set of rules. This simulation aims to capture statistically the change in lymphocyte concentrations in the lymph node as the probabilities on the capture and migration transitions change.

In the second simulation, the 3-dimensional HEV tube made up of endothelial cells, and the movement of lymphocytes through that tube, are explicitly implemented. This supports visualisation of the HEV and of the lymphocytes, with the *HEV lumen* and *rolling* states visually distinguishable via changes in colour. Again, the model is driven by the

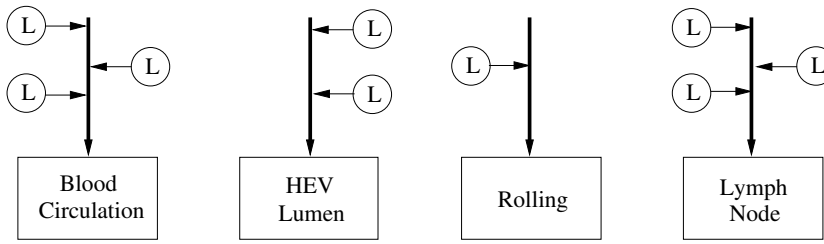

Fig. 11. Process diagram showing how lymphocyte processes (represented as circled "L"s register with state processes (boxed)

probabilities on the transitions, but because the spatial aspect is expressed explicitly, the simulation is more obviously closer to the biology.

We refer to the two different simulations as *migration-abstract* and *migration-space* respectively. The description in this section focuses on the *migration-abstract* simulation, but draws on elements of the *migration-space* simulation when necessary. Our simulations are implemented using occam-π, a process-oriented programming (POP) language capable of massive concurrency (for details of why we choose this approach see [3]).

The migration-abstract simulation implements the state diagram shown in figure 10 for a population of lymphocyte agents over a period of time. Using the POP paradigm, each lymphocyte agent is represented as a process which is connected via a *communication channel* to one of the four body place states we are interested in (also represented as processes): blood circulation, HEV lumen, rolling on the endothelium, and in the functional tissue of the lymph node. The process network structure is shown in figure 11, and directly reflects the topology of the states in figure 10. Each of the four place state processes shown has a *shared* communication channel (shown as a bold arrow), to which any number of lymphocyte processes can be connected. Each lymphocyte process is connected to one and only one place process thus ensuring it can only be in one state at a time. According to process-oriented design rules, the lymphocyte processes act as clients to the server state processes.

Depending on the transition rules, a lymphocyte process can change the state process to which it is connected, moving from one place process to the next. Channels exist between state processes to enable this movement, which directly reflect the allowable state transitions in figure 10. For example, in figure 12 the shaded lymphocyte process moves from the blood circulation state to the HEV lumen state by disconnect-

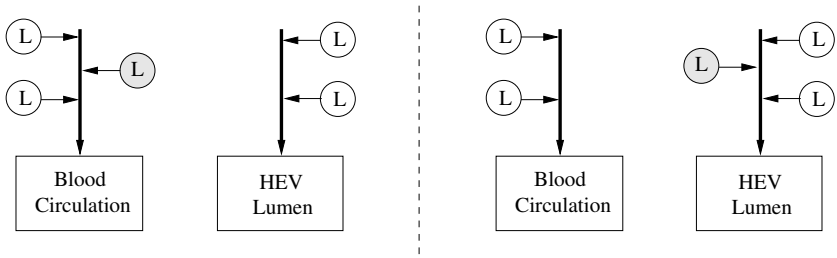

Fig. 12. Process diagram showing how lymphocytes move between state processes

ing its communication channel from the blood circulation process and reconnecting it to the HEV lumen process.

Every lymphocyte process updates once per time step (this is achieved via the occam-π barrier construct). At each step, the number of lymphocyte processes associated with each place process can be recorded, to allow numerical analysis of the number lymphocytes in each state over time. A typical simulation run necessarily contains many thousands of lymphocyte processes in order to get close to the biological scale which contains millions of lymphocytes. New lymphocyte processes can be added during a simulation run and are automatically connected to the blood circulation process.

The eight lymphocyte state transitions (figure 10) in the migration-abstract simulation are each encoded a probabilistic rule. At each time step, a lymphocyte tests for its possible for transitions. For example, a lymphocyte in the rolling state can either disassociate to the HEV lumen state, migrate to the lymph node state or stay in the rolling state. To determine which of these occurs, a random number is generated for each lymphocyte and is tested against the possible transitions. If the random number falls in the range of a transition, then the lymphocyte will transit into the relevant state. If not, the lymphocyte stays in its current state.

There is a mapping between probabilities and the biological detail. To achieve an accurate simulation, we need to choose the probabilities carefully, and try to validate to be confident that they represent what they are supposed to. It is ongoing work to find good probabilities, the job of which is not trivial for a number of reasons. Often the exact biological details are poorly understood or simply not recorded. Facts can also come from many different sources, based on different experiments utilising different technologies and subjects. Many of these facts then have to be combined into a single probability. Thus when constructing these probabilities we need to document where they have come from and

how they map to the probabilities. An example of the type of detail we would use to construct a probability is that a quarter of lymphocytes entering HEVs will migrate [13]. By combining this with the numbers of lymphocytes in the body we can start to build meaning into the capture and migration probabilities.

The following descriptions examine each transition probability and suggests the type of biological detail we would need to generate their values in a simulation:

Creation: The probability for creation is distinct from the other transition probabilities as it is a function of the population size rather than the individual lymphocytes. Studies in mice show that about 1 to 2 million T cells and B cells enter the blood circulation [16]. In a homeostatic environment, the numbers of lymphocytes created should equal the number that die (see death transition below).

Enter HEV, Exit HEV: Lymphocytes continually enter areas of HEV whilst circulating in the body. The probability of a particular lymphocyte doing so needs to reflect an average amount of time not spent in HEVs. It is consequently inextricably linked to the exit HEV probability. The probability needs to encapsulate biological details such as the relative lengths of the blood circulation system and HEVs, along with the rates of blood flow.

Capture: In the description of the conceptual model, we discussed what capture means and the assumptions involved. In the simulation, the capture probability relates to the biology of the receptors of lymphocytes and endothelial cells, and the probability that the receptors on each are close enough to interact. The interaction probability can be based on the relative sizes of lymphocytes and the diameter of the HEV lumen taken from the biological data. For example, a smaller lumen diameter would increase the chances of a lymphocyte being close enough to interact. Conversely, a larger diameter may relate to a larger surface area of endothelial cell receptors to which the lymphocytes can attach, so a larger diameter would increase the probability of capture.

Disassociate: This probability is related to the capture probability in that it takes into account the strength of binding between lymphocytes and the endothelial cells. A higher concentration of endothelial cell receptors should reduce this probability.

Migrate: The migration probability is dependent on the concentration of chemokines to induce the confirmation change in the lymphocyte and/or the likelihood that the lymphocyte picks up the chemokine signal. The last probability could take into account the numbers of chemokine receptors on the lymphocyte, and the way in which endothelial cells express and present them. Assumption 17 applies.

Drain: The drain probability reflects the amount of time it takes for a lymphocyte that has migrated into the lymph node to exit to the blood circulation. This time can be dependent on whether the lymphocyte is activated, but as this frequency is usually very small, we assume the rate is constant for all lymphocytes (Assumption 18).

Death: A lymphocyte that dies will be removed from the simulation. The probability to determine this needs to take into account the average life of lymphocytes. In a homeostatic environment, death is compensated by creation. Assumption 9 applies.

All probabilities need to be scaled to fit the time step of the simulation. Each probability is also simulation specific. Our migration-space simulation implements the same conceptual model as the migration-abstract simulation, employing the same state transitions, and has a need for the same relevant probabilities. However, the actual probability values are subtly different. For example the capture probability in the migration-abstract simulation incorporates the need for a lymphocyte to be near to the endothelium for capture to occur. In the migration-space simulation, the 3-dimensional space is explicit, thus the capture probability only encapsulates the biology of receptor binding. The process of validation needs to explicitly highlight the contributions to transitions, to make the simulations transparent and open to reasoning.

5 Verification and validation of the simulations

According to Sargent's process, we need to validate the conceptual model against the problem entity (and the purpose of the simulation). Ultimately, we also have to argue the operational validity of the computerised model (i.e. determine the behaviour has sufficient accuracy for its intended purpose), but that is largely outside the scope of this paper. As we will see, validation of the conceptual model reveals the many gaps in both the science and the computerised model, these gaps inhibit exploration of any research hypothesis.

We can think of the validation process as producing an argument of validity, in the same way that critical systems developers produce arguments of safety, dependability or security. Note that just as a safety argumentation never establishes that a system is absolutely safe (no system is safe unless it is totally closed and inert), an argument of validity merely states the case for validity, exposing it to critical consideration. This is an important observation, because, in any natural complex system, we cannot expect to provide a gold-plated guarantee of equivalence between our conceptual model and the problem entity – indeed, if the model contained all the complexity needed to exactly mimic the natural system, it would be intractably large, and too complex to provide any new research insight.

An argument is expressed as a *proposition*, and is reasoned on the basis of some *premises*, to reach a *conclusion*. A variety of textual and diagrammatic techniques allow an argument to be presented with a degree of formality – exposing the premises to analysis and scrutiny (see [17, §2.6] for a succinct review). Kelly and others [17, 35] adopt the goal structuring notation (GSN), proposed in [37], to present arguments including those relating to the safety or safe design of critical systems [17] and system dependability [8], and to emphasise the role of argumentation in design [38].

In conventional safety case argumentation, basic elements are used to express an argument: goals (decomposable), strategies (to meeting the goals), justifications, assumptions, contexts, and solutions. Here, our goal is to validate the conceptual model against the problem entity. For illustration, we focus on one aspect of the conceptual model, the transition labelled *Capture*. This reveals many of the issues in validating a complex system model against a natural problem entity. Whilst we do not present a systematic analysis here, the approach could be systematised, applying a deviational approach in the way that is common in safety work (see [25]) and has more recently been used in security analysis (for example, [31]).

5.1 The Capture transition and its connotations

The *Capture* transition takes the lymphocyte from the state *HEV Lumen*, where it is moving freely within the HEV, to the state *Rolling*, where it is in the preliminary stage of the migration process. From the *Rolling* state, a lymphocyte can revert to the *HEV Lumen* state by disassociation, or continue its migration to the functional part of the lymph node. We might equate *Capture* to the second stage shown in figure 9, above.

The *Capture* transition is an abstraction from the cell biology and biochemistry – which has been well-researched (see the top panel of figure 9, above). However, validation of the *Capture* probability would require a separate, lower-level simulation of the capture bio-chemistry (or *in vivo* research). In our validation argument, the probability of capture is an area that we must expose to external review, or to further work.

However, there are other aspects of capture that we must validate, where the biological basis for our conceptual model is less well understood. We might postulate a context for capture: the chance of a lymphocyte being captured depends on (a) the density of active receptors on the endothelium; (b) the receptors on the lymphocyte; (c) the likelihood of a lymphocyte being close enough to the endothelium for capture. Thus, our simulation needs to take into account the geometry (as well as the biochemistry) of the lymphocyte and the HEV, as well as flow characteristics in the venule. Validation has a choice here. On one hand, we could attempt to simulate the three-dimensional structural biology of a HEV – capturing typical venule cell structures and their behaviour as the HEV constricts and relaxes (an animated version of figures 5 and 7) – and analysing the flow and contact characteristics that determine lymphocyte interaction with the endothelium chemicals. On the other hand, we could state our assumption that the probabilities on the transition from flowing in the *HEV Lumen* to captured in the *Rolling* state adequately captures the geometric and flow aspects of the HEV. The first option tends to improve the biological realism (and validatability) of the simulation, at the cost of complicating the conceptual basis of the model. If we opt for simplicity, we must record the assumption – which come under a general heading of *environmental factors* – so that scientists appreciate the limitations of the simulation.

In the conceptual model, the transition probabilities are constant. Each probability approximates the effect of many environmental factors, and each factor must cause fluctuations in the rate of transition in the short term. However, we assume that behaviour tend towards the norm over the timescale of the simulation (or the real biology). Again, we could postulate further research or low-level simulations to validate this assumption: for instance, we would like to understand the effect of irregular coverage of pericytes – does this give rise to an uneven longitudinal profile in the HEV, and if so, does this promote lymphocyte capture at upstream locations?

This last question identifies a potential paradox in our conceptual model: we assume constant probabilities of transition, so all lymphocyte in the *HEV Lumen* state have an equal probability of being captured; equal probability implies a homogeneous HEV environment. However,

the hypothesis seeks to relate lymphocyte volume in the lymph node with dilation of the HEV, so we are required to vary the cross-section (at least) of the HEV, and it is not obvious that this is consistent with a homogeneous HEV environment (chemically or geometrically).

As we explore the question of the HEV environment further, we discover that one possible conformation of the contracted HEV has the lining cells packed in tight folds; the folds relax as the HEV dilates. There must, therefore, be an intermediate point at which pockets arise between folds, which we might expect to trap lymphocytes, increasing the chance of capture. Taking an opposing view, widening the HEV means that lymphocytes are more able to move away from the venule walls, potentially causing a fall in capture. We cannot currently validate any of the assumptions about the geometry and flow of the HEV environment, but we can highlight these assumptions in relation to our simulation results.

We assume that all lymphocytes are the same size – that is, there is no differential probability across lymphocytes. This can be validated biologically: the literature gives size ranges for lymphocytes, and we can determine (by asking immunologists!) whether the range represents sizes with one lymphatic system or across a species (or what): the simulation can be driven accordingly. It is noted that much of the data taken from the literature and used for our simulation was gathered for purposes other than ours. We must, therefore, assume that the data is still applicable in our domain.

Returning to the probability of the *Capture* transition, we have used the conceptual model as the basis for two simulations. In the abstract simulation (non-spatial), the transition probability must be used to reflect the effects of all factors in HEV environment and other relevant environmental factors. However, in the spatial model, the spatial relationship between lymphocyte size and location and venule diameter and conformance is incorporated directly – the transition probabilities abstract only from the biochemistry and the surface characteristics of the HEV.

We have not identified all assumptions of our conceptual model here. For example, the model also abstracts from all biochemical factors: we assume that the probability of transition expresses any underlying variability in receptor form, binding strength, mechanisms of binding and expansion, etc. We also abstract away from the cascade details of the rolling state and the capture, disassociation and migration transitions.

5.2 Breaking with assumptions

Validating the conceptual model against the problem domain (the biology) can reveal inconsistencies in the biological detail and mismatches

with our assumptions. For example, we may have based a probability on data that has been revisited and altered by subsequent scientific research; mixed data from incompatible pieces of research; used data from fixed biological material not appropriate to the simulation. By presenting our model and simulation assumptions, and the biological detail that has influenced the design of our simulation, the validation process is made easier.

If we find a problem in an assumption that breaks the model, we need to analyse the effects and change the model accordingly. For example, assumption 15 states that proliferation (the generation of cellular clones) of lymphocytes does not occur in the lymph node. Based on experimentation with the simulator, we might decide that this has an effect. Consequently we can update the conceptual model with extra states and transitions. These changes then cascade through subsequent models to the simulator. We need to check that changing one assumption or value does not affect the others, and if it does then change these accordingly. Most incorrect assumptions will not invalidate the entire model or simulation, but just require editing. Tables of assumptions and biological details allow traceability through the modelling and simulation process. For example a domain expert might inspect them and highlight areas of inconsistencies leading to the model or simulation to be updated.

6 Discussion

Our conceptual model of the lymphocyte system represents a set of design decisions: we have chosen to abstract to certain (key) states of the lymphocyte lifecycle, and we have selected probability-based transitions as a suitable basis for experimenting with lymphocyte concentrations and HEV dilation. The consequent validation requirements are clearly dependent on these design decisions. Had we chosen to model at a different level of abstraction, or to represent lymphocyte behaviour differently, we would have different validation requirements (but a similar range of problems relating our model to the problem entity).

Disparate levels is an inherent problem of complex systems research and simulation. The validation proposals suggest that some of the conceptual model features could be validated by either lower-level simulation, or exploration of the biology (and biochemistry) at the lower level. This idea is also inherent to the CoSMoS project – the modelling and simulation platform that it seeks to develop has as one goal support for multiple simulation levels. A lesson of validation might be that we need to identify component *levels*, as well as component state-and-operations, so as to facilitate validation where biological detail is uncertain.

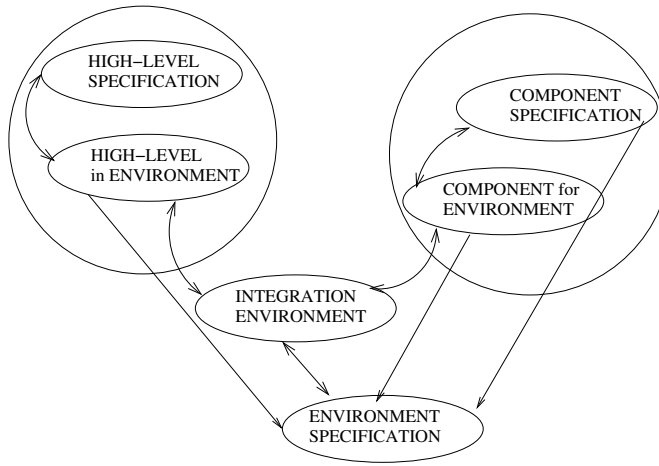

Fig. 13. An architecture for complex system engineering, after [23]

Elsewhere, we have proposed an architecture, and some principles, for the engineering of simulations exhibiting several layers of complex behaviour (see [34, 32, 23]). The architecture, figure 13, for complex systems proposes component specifications, higher-level system specifications, and a reconciliation (or linguistic integration) via strategic parts of the common environment. Putting together these architectural ideas with the need to identify environmental factors in determining the conceptual model (and the implementation detail of the subsequent computerised model), we propose an extension to Sargent's process, as shown in figure 14.

There is much work still be to done in the area of validating our complex system models and simulations. This includes establishing structured ways to layout our assumptions and biological details that have influenced our designs. We also need to establish schemes for mapping between our biological details and simulator parameters. In addition we are investigating structured argumentation techniques to talk about validity of complex systems simulations. Our aim is to establish patterns of validation that are applicable to the validation of many different complex systems.

7 Summary

We have presented a selective review of engineering approaches to engineering simulations in non-complex systems and agent systems, and used

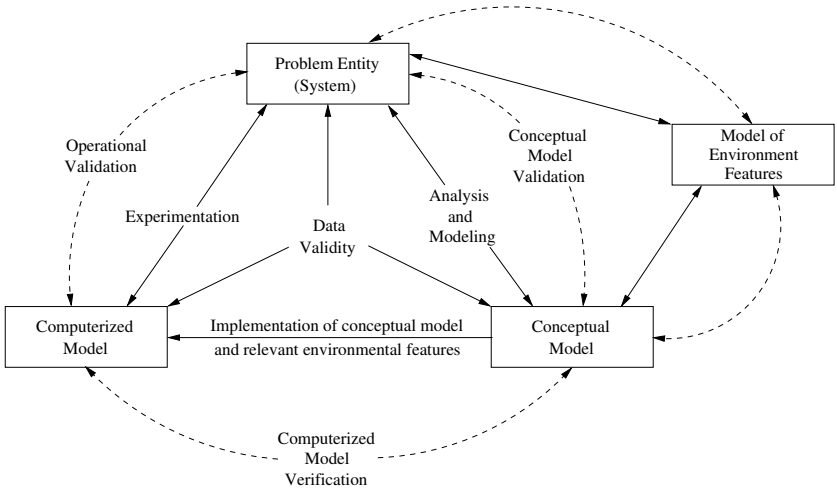

Fig. 14. Adding environmental concerns to Sargent's process (figure 1), to propose a process for complex system simulation

this to initiate a case study simulation development for part of the lymphocyte system. The simulation is based on information from biological literature and part of an ongoing research project in the CII.

Proposing the biological information as our problem entity, and exploration of proposed immune mechanisms as our goal, we have presented a simple conceptual model (from which two simulations of different abstractions have been created). Focusing on one part of the conceptual model, we discuss issues relating to validation of biological research simulations.

Drawing on work in non-complex systems simulation and agent systems modelling, we have identified possible features of a process for engineering complex systems simulations. The process is speculative, but fits our experience of simulating part of the immune system.

The paper presents a first step in an engineering approach towards scientific computer simulation; the findings are preliminary and not yet substantiated by repetition or systematic use. However, our findings are well grounded in wider work on critical systems engineering and assurance, as well as other areas of system simulation.

From the brief exploration of validation, it is clear that arguments of validity for research-oriented simulations of complex systems are going to be complicated, and often incomplete – the principle of exposing assumptions to external scrutiny is an important contribution of this paper. We expect that argumentation approaches, and deviational anal-

ysis, will contribute to the quality and visibility of validation. In short, we believe that the scepticism over use of computer simulation in complex systems research can be addressed through well-established engineering principles, just as it is being addressed in macro-scale complex systems.

8 Acknowledgements

This work is part of the CoSMoS project, funded by EPSRC grants EP/E053505/1 and EP/E049419/1.

References

[1] R. Alexander. *Using Simulation for Systems of Systems Hazard Analysis*. PhD thesis, Department of Computer Science, University of York, 2007.

[2] K. Allenby and T. P. Kelly. Deriving safety requirements using scenarios. In *5th IEEE International Symposium on Requirements Engineering (RE'01)*. IEEE Computer Society Press, 2001.

[3] Paul S. Andrews, Adam T. Sampson, John Markus Bjorndalen, Susan Stepney, Jon Timmis, Douglas N. Warren, and Peter H. Welch. Investigating patterns for the process-oriented modelling and simulation of space in complex systems. In *To appear: Artificial Life XI: Proceedings of the Eleventh International Conference on the Simulation and Synthesis of Living Systems*. MIT Press, 2008.

[4] G. Bergers and S. Song. The role of pericytes in blood-vessel formation and maintenance. *Neuro-Oncology*, 7(4):452–464, 2005.

[5] J. Bryden and J. Noble. Computational modelling, explicit mathematical treatments, and scientific explanation. In *Artificial Life X*, pages 520–526. MIT Press, 2006.

[6] M. Calder, S. Gilmore, and J. Hillston. Modelling the influence of RKIP on the ERK signalling pathway using the stochastic process algebra PEPA. *Transactions on Computational Systems Biology VII*, 4230:1–23, 2006.

[7] M. Calder, S. Gilmore, J. Hillston, and V. Vyshemirsky. Formal methods for biochemical signalling pathways. In *Formal Methods: State of the Art and New Directions*. Springer, 2008.

[8] G Despotou and T Kelly. Design and development of dependability case architecture during system development. In *25th International System Safety Conference*. System Safety Society, 2007.

[9] S. Efroni, D. Harel, and I. R. Cohen. Reactive animation: realistic modeling of complex dynamic systems. *IEEE Computer*, 38(1):38–47, 2005.

[10] S. Efroni, D. Harel, and I. R. Cohen. Emergent dynamics of thymocyte development and lineage determination. *PLoS Computational Biology*, 3(1):0127–0135, 2007.

[11] J. M. Epstein. Agent-based computational models and generative social science. *Complexity*, 4(5):41–60, 1999.

[12] M. Georgeff, B. Pell, M. Pollack, M. Tambe, and M. Wooldridge. The belief-desire-intention model of agency. In *ATAL'98*, volume 1555 of *LNCS*, pages 1–10. Springer, 2000.

[13] J. Girard and T. Springer. High endothelial venules (HEVs): specialized endothelium for lymphocyte migration. *Immunology Today*, 15:449–457, 1995.

[14] D. Harel, Y. Setty, S. Efroni, N. Swerdlin, and I. R. Cohen. Concurrency in biological modeling: Behavior, execution and visualization. *FBTC 2007: Electronic Notes in Theoretical Computer Science*, 194(3):119–131, 2008.

[15] P. G. Herman, I. Yamamoto, and H. Z. Mellins. Blood microcirculation in the lymph node during the primary immune response. *The Journal of Experimental Medicine*, 136:697–713, 1972.

[16] C. A. Janeway, P. Travers, M. Walport, and M. J. Shlomchik. *Immunobiology: The Immune System in Health and Disease (6th Edition)*. Garland Science Publishing, 2005.

[17] T. P. Kelly. *Arguing safety – a systematic approach to managing safety cases*. PhD thesis, Department of Computer Science, University of York, 1999. YCST 99/05.

[18] K. Ley, C. Laudanna, M. I. Cybulsky, and S. Nourshargh. Getting to the site of inflammation: the leukocyte adhesion cascade updated. *Nature Reviews Immunology*, 7(9):678–689, 2007.

[19] G. F. Miller. Artificial life as theoretical biology: How to do real science with computer simulation. Technical Report Cognitive Science Research Paper 378, School of Cognitive and Computing Sciences, University of Sussex, 1995.

[20] M. Miyasaka and T. Tanaka. Lymphocyte trafficking across high endothelial venules: dogmas and enigmas. *Nature Reviews Immunology*, 4(5):360–370, 2004.

[21] L Padgham and M Winikoff. Prometheus: A methodology for developing intelligent agents. In *AOSE III*, volume 2585 of *LNCS*, pages 174–185. Springer, 2003.

[22] E. Di Paolo, J. Noble, and S. Bullock. Simulation models as opaque thought experiments. In *Artificial Life VII*, pages 497–506. MIT Press, 2000.

[23] F. Polack, S. Stepney, H. Turner, P. Welch, and F. Barnes. An architecture for modelling emergence in CA-like systems. In *ECAL*, volume 3630 of *LNAI*, pages 433–442. Springer, 2005.

[24] F. A. C. Polack, T. Hoverd, A. T. Sampson, S. Stepney, and J. Timmis. Complex systems models: Engineering simulations. In *ALife XI*. MIT press, 2008. to appear.

[25] D. J. Pumfrey. *The Principled Design of Computer System Safety Analyses*. PhD thesis, Department of Computer Science, University of York, 2000.

[26] H. K. Rucker, H. J. Wynder, and W. E. Thomas. Cellular mechanisms of CNS pericytes. *Brain Research Bulletin*, 51(5):363–369, 2000.

[27] A. Sadot, J. Fisher, D. Barak, Y. Admanit, M. J. Stern, E. J. A. Hubbard, and D. Harel. Towards verified biological models. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2007.

[28] R. G. Sargent. An exposition on verification and validation of simulation models. In *17th Winter Simulation Conference*, pages 15–22. ACM, 1985.

[29] R. G. Sargent. The use of graphical models in model validation. In *18th Winter Simulation Conference*, pages 237–241. ACM, 1986.

[30] R. G. Sargent. Verification and validation of simulation models. In *37th Winter Simulation Conference*, pages 130–143. ACM, 2005.

[31] T. Srivatanakul. *Security Analysis with Deviational Techniques*. PhD thesis, Department of Computer Science, University of York, UK, 2005. http://www.cs.york.ac.uk/ftpdir/reports/YCST-2005-12.pdf.

[32] S. Stepney, F. Polack, and H. Turner. Engineering emergence. In *ICECCS'06*, pages 89–97. IEEE Computer Society, 2006.

[33] J. Sudeikat, L. Braubach, A. Pokahr, and W. Lamersdorf. Evaluation of agent-oriented software methodologies – examination of the gap between modeling and platform. In *AOSE 2004*, volume 3382 of *LNCS*, pages 126–141. Sringer, 2004.

[34] H. Turner, S. Stepney, and F. Polack. Rule migration: Exploring a design framework for emergence. *Int. J. Unconventional Computing*, 3(1):49–66, 2007.

[35] R. A. Weaver. *The Safety of Software – Constructing and Assuring Arguments*. PhD thesis, Department of Computer Science, University of York, 2003. YCST-2004-01.

[36] M. Wheeler, S. Bullock, E. Di Paolo, J. Noble, M. Bedau, P. Husbands, S. Kirby, and A. Seth. The view from elsewhere: Perspectives on alife modelling. *Artificial Life*, 8(1):87–100, 2002.

[37] S. Wilson, J. McDermid, P. Fenelon, and P. Kirkham. No more spineless safety cases: A structured method and comprehensive tool support for the production of safety cases. In *2nd International Conference on Control and Instrumentation in Nuclear Installations (INEC'95)*, 1995.

[38] W. Wu and T. Kelly. Towards evidence-based architectural design for safety-critical software applications. In *Architecting Dependable Systems*, volume 4615 of *LNCS*. Springer, 2007.

[39] B. P. Zeigler. A theory-based conceptual terminology for m&s vv&a. Technical Report 99S-SIW-064, Arizona Center for Integrative Modeling and Simulation, 1999. http://www.acims.arizona.edu/PUBLICATIONS/publications.shtml.