The engineering of concurrent simulations of complex systems

Fiona A. C. Polack Paul S. Andrews Adam T. Sampson

Abstract— **Concurrent process-oriented programming is a natural medium for simulating complex systems, particularly systems where many simple components interact in an environment (which may itself be complex). There is little guidance for engineering complex systems simulation. In the context of simulation work to support immunological research, we explore relevant approaches to modelling, and draw on concepts from dependable and high-integrity systems engineering, including the emphasis on the need to validate all aspects of the simulation.**

I. Introduction

Two software engineering issues in simulations of complex systems are design modelling and validation. We explore how these can be addressed, based on the experience of two research projects on complex systems engineering[1]. We focus on simulations designed to assist immunological understanding, developed in close collaboration with immunologists at the York Centre for Immunology and Infection (CII).

Our work addresses systems that 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. Time and the environmental context are also critical. In a complex system, many things happen in parallel, and thus a concurrent paradigm is an obvious choice for computer simulation. Here, we focus on simulations constructed as concurrent process-oriented programs (POP).

Engineering guidance for such systems is limited. In agent-based systems, support for complex systems simulation and agent modelling tends to be at the implementation level (see, for instance, the ACE resources, www.econ.iastate.edu/tesfatsi/ace.htm), and engineering, though widely discussed, tends to focus on low-level design and architectural issues. Our work draws heavily on principles from state-of-the-art dependable systems engineering. A guiding principle in dependable systems engineering is that it must be possible to demonstrate how the quality attributes of a system are met – that risks associated with the system are *as low as reasonably practicable* (as in UK Health and Safety Executive's guidance on risk, www.hse.gov.uk/risk/index.htm, and the UK MoD defence standard 00-56 on safety requirements management). In general, the safety (or security, reliability etc) of a system is evaluated via a *case* that argues that risks are identified and suitably mitigated; the argument is assessed by independent evaluators (as, for instance, in the UK Civil Aviation Authority air-safety certification processes).

Traditionally, computer simulation can be used in dependable systems engineering, but it is not considered suitable as evidence in a dependability case. However, the need to validate complex systems (such as human-scale systems of systems – command and control, evacuation simulation) has raised the profile of simulation in dependable system development – see for example, recent work on the validity of simulation evidence in safety assurance of systems of systems [1].

There is a similar scepticism about the ability of computer simulations to contribute to scientific research (see [2], [3], [4], [5]). Computer simulations are often built by computer scientists who are interested in a visual result (a game, an imitation of Life), rather than any scientific reality. Typically, a valid simulation is any program that produces the expected (visual) results. At a slightly more rigorous level, there might be an attempt to produce the expected results by a process that looks a bit like reality, but there has generally been little concern for the quality of the underlying simulation [6]. Note that this lack of rigour extends also to the well-respected mathematical models of many natural scientists – it is rare to find an in-depth justification of the choice of variables in models using differential equations, Markovian or Bayesian processes. In the context of artificial life, Bullock [4] represents an increasing concern in noting 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.

Our simulations are built in process-oriented programming (POP) languages – mainly in the occam-π language (for details of why we choose this approach see [7]). This is a fast, efficient concurrent implementation medium. Because there is typically a close analogy between complex system components and programming processes, the construction and maintenance of simulations is intuitive, and it is easy to modify, adapt and reuse simulation programs [8]. Currently, there are no software engineering methods for working with POP media. Here, we report on our attempts to extend and adapt approaches to the modelling and validation of computer simulations.

Fiona Polack and Paul Andrews: Department of Computer Science, University of York, YO10 5DD, UK: email{fiona,psa}@cs.york.ac.uk. Adam Sampson: Computing Laboratory, University of Kent, Canterbury, UK: email A.T.Sampson@kent.ac.uk

[1]The CoSMoS project, EPSRC grants EP/E053505/1 and EP/E049419/1, www.cosmos-research.org/, is building capacity in generic modelling tools and simulation techniques for complex systems; it follows on from a feasibility study, TUNA, EPSRC grant EP/C516966/1, that investigated ways to model and engineer complex systems using the occam-π language.

978-1-4244-2959-2/09/$25.00

A. Some existing work on simulation engineering

Sudeikat et al [9] have an insightful review of multi-agent system development methods, which focuses on matching methods to the requirements of specific simulation targets. Some of the reviewed methods are sophisticated software-engineering approaches. For example, the Prometheus development method for agent-oriented systems is illustrated in Figure 1.

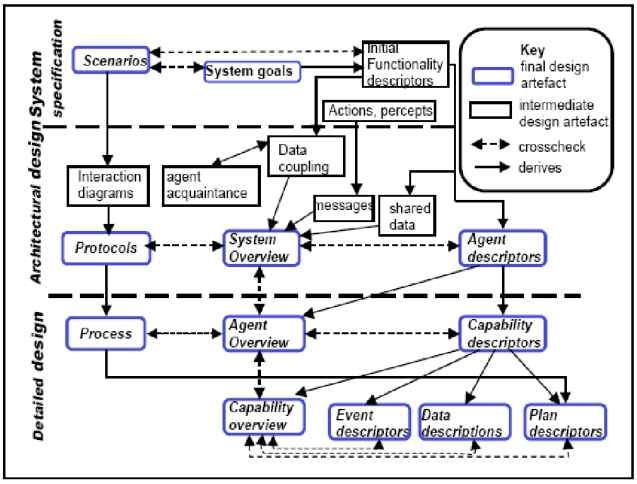

Fig. 1. The Prometheus development method [10]

Approaches such as Prometheus are based on general object-oriented (OO) or procedural design methods. The focus is on different system views (the agents, their data and interactions, actions and object message-passing), rather than on critical features of complex systems – the components, their environment, and desired or undesirable emergent behaviours. In software engineering terms, these approaches do not provide the right tools for our POP approach. We can characterise the general problem as one of *continuity*.

Continuity of paradigm is an important consideration in software engineering. For example, object or class models map cleanly on to OO programming, reducing the scope for introducing inconsistencies between design and implementation. Problems arise where inappropriate design models are used as the basis for implementation. Historically, this has arisen whenever a new paradigm appears, as witness methods from the 1990s that use procedural models such as Jackson structures to design information systems (ie. relational database applications – for example, SSADM [11]) or OO applications (as reviewed in [12, chapter 12]).

Continuity is improved if design models are compact. However, the need to express designs through multiple views can lead to continuity pitfalls. For example, an OO design language such as UML [13] support many different views of a system, but, currently, there is limited support for consistency checking across views (this is being addressed in the context of metamodelling and model-driven engineering, but is not yet widely accepted). Although a UML class diagram maps well to an OO program, the mapping from the other design diagrams to OO program concepts is less than obvious.

If design modelling can achieve continuity and consistency, this helps in verifying a program. For POP, we need modelling tools that are as simple as is compatible with expressivity, and that allow us to identify and design the components of a complex system, the environmental context of the simulation, and the interactions of components with each other and the environment. The design concepts also need to map cleanly to POP programming concepts.

B. Some existing work on simulation validation

Validation checks that the right system is built – it is about meeting requirements and quality (whereas *verification* checks that the system is built right – it is about correctness of construction). Here, we are concerned with scientific validity as well as engineering validity. It must be possible to demonstrate, with evidence, how models express the scientific realities. Validity implies both adequate abstraction, and adequate development processes.

In high-integrity systems engineering, the validation of simulation has been a focus of interest since the late 1970s. Although several groups [1], [14], [15] are now applying this research to complex system simulations, until recently the work had little impact outside its original context. The central theme is Sargent's development lifecycle [16], shown in Figure 2. The *problem entity* is the phenomenon to be modelled. From understanding the problem entity, a *conceptual model* is developed in a suitable representation – Sargent reviews diagrammatic models [17], and also considers mathematical or logical representations [16]. The *computerised model* implements the conceptual model as a simulation.

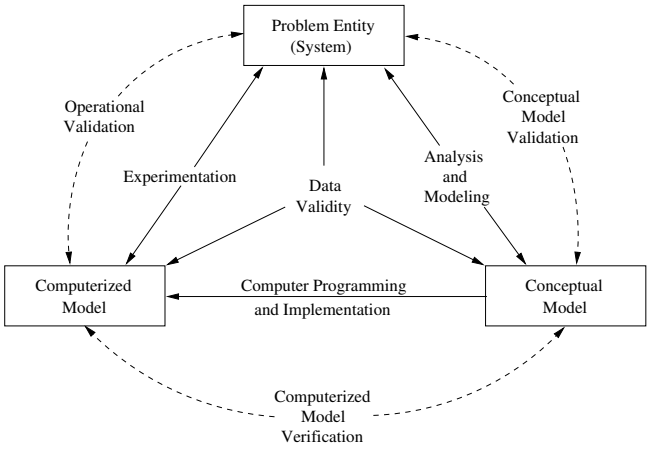

Fig. 2. Sargent's model of the simulation development process [17]

The *experimentation* link between the problem entity and the computerised model allows iterative trial-and-error simulation, with the models and results compared to the problem entity at each step. The model explicitly incorporates verification (in the software engineering of the computerised model) and validation – of all models against the problem entity, and of the data used to test or populate the conceptual

and computerised models. The lifecycle 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.

Elsewhere [18], we consider how complex systems differ from conventional systems, and use this understanding to propose an extension to Sargent's process [15], [19]. Essentially, the components of a complex system can be designed and verified by a conventional engineering process, but the complex effects of the components only become apparent when they are considered in their environmental context. A small change in the environment can change the nature, and even the occurrence, of high-level behaviours. In describing the problem entity and developing the conceptual model, the extraction of relevant environmental characteristics is as important as the modelling of the components. As we noted above, the quality of software engineering, and particularly the feasibility of verification, is enhanced by continuity and consistency. Given that we now have to model environment, components and their interactions, there is an even greater need to follow principles of continuity.

In relation to validation, Sargent [16] reminds us that a *model should be developed for a specific purpose... and its validity determined with respect to that purpose.* The level of assurance also depends on the purpose of the simulation, and should be set independently of the development of the simulation – good software engineering practice.

Work on validation of non-complex system simulations does not transfer easily to complex systems. Zeigler [20] presents a theory for modelling and validation of simulations predicated on a homomorphism between conceptual models and simulations; he does not show how the homomorphism is established. Sargent [16], focusing on the validation of the computational model against the problem entity, proposes a range of approaches to validation, summarised in Table I.

Many of Sargent's validation techniques are inappropriate for complex systems work. For instance, if we knew the workings of the complex system well enough to understand event validity and traces, we would not need a computer simulation to help us understand it. The the most useful of Sargent's suggestions is the analysis of assumptions – which he disguises under the historical theories of rationalism and empiricism. It is rare for a research simulation to document its assumptions and design decisions.

TABLE I
WAYS TO VALIDATE SIMULATIONS (BASED ON [16])

Technique	Comments on Sargent's suggestions
Animation	Graphical visualisation, of system behaviour or of operational parameters
Comparison	Comparison to *valid* analytical models or other simulation models
Degenerate and extreme tests, parameter variability, sensitivity analysis	Typical domain-style testing of behaviour under normal and extreme input and operations
Event validity	Compare the events in real and simulated systems
Face validity	Appeal to logic or domain experts to validate 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
Combination of traditional methods:	Combine approaches using sound theory, assumptions and empirical validity checks
Rationalism	Assumptions are rationally justifiable; valid models arise from valid assumptions
Empiricism	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 experts distinguish it from reality?

II. DESIGN FOR CONCURRENT SIMULATION APPLICATIONS

We wish to engineer simulations of complex systems in a process-oriented concurrent programming paradigm. We need design notations that we can use to validate our models with domain experts. The design notations need to admit continuity in development, and be verifiable. We also need to view the development process as part of the construction of an argument of validity, which requires us to systematically identify and record assumptions. In this section, we first consider issues and possible solutions to the design problem. We then review the sorts and sources of assumptions in our models.

A. Modelling for POP

Design processes targeting OO or procedural implementations are too elaborate for the POP paradigm. This can be seen by considering POP and the way it is used in practice. Here we consider design of occam-π programs, but many of the principles apply more generally.

The occam-π programming language [21] is a practical implementation of the concepts from Hoare's CSP [22] and Milner's π-calculus [23]. As in CSP, an occam-π program is a composition of parallel processes that communicate using channels (two-way events) and barriers (multiway events). Channels carry messages (structured data) between processes; allowable messages on a channel are specified using a protocol (see [24]). Each channel has a writing end and a reading end; they may be used in a point-to-point manner, or a channel end may be explicitly shared between processes to allow any-to-one communication.

From the π-calculus, occam-π takes the concept of mobility. Channels and barriers can be created dynamically; channel and barrier ends are first-class objects that can be transferred between processes at any time. Mobility allows a process's alphabet of events to vary during its execution, and the network of processes to be dynamically reconfigured as appropriate. New processes can be started dynamically, and may themselves be first-class objects – *mobile processes* that can be suspended and transferred over channels.

In terms of verification, occam-π's formal basis allows the programmer to reason about the behaviour of concurrent programs, and simple rules, expressed as design patterns, make it possible to construct programs that are guaranteed to have properties such as freedom from deadlock [25]. In addition, the occam-π compiler applies extensive static checks that detect common concurrency errors. occam-π is particularly appropriate for complex system simulation, as a specialised language for process-oriented programming with an efficient run-time scheduler, that allows real-time simulation of millions of processes.

Process-oriented programs are often described using informal process diagrams of connections in a process network at a particular instant (see [26]). Processes are drawn as labelled boxes, triangles, circles, and so on. Different shapes can be used to distinguish different uses of process in a system. Channels are drawn as arrows in the direction of communication, and shared channels as thick arrows to which multiple processes may be connected. Figure 3 shows a representation of part of our CII case study (below) with agent processes drawn as circles.

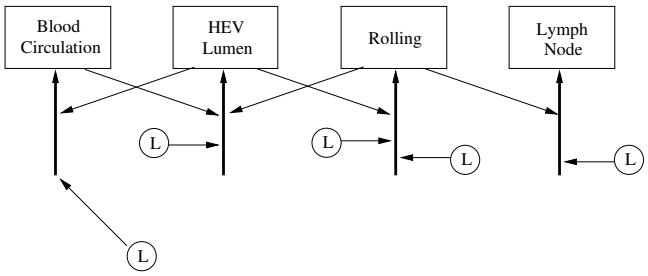

Fig. 3. The design of an occam-π program. Four processes, *Blood Circulation*, *HEV Lumen*, *Rolling* and *Lymph Node*, are connected to each other via each process's shared channel (thick arrow). A lymphocyte process (*L*) can also connect to a shared channel (based on [15])

Processes are used to represent the lymphocytes, and the different states of lymphocytes. At a higher level of abstraction, this is neatly expressed by a state chart – Figure 4 is a basic example.

In the design described in Figure 3, (at least in process-oriented languages, using the client-server design pattern which has been proved to be free from deadlock), the simple state chart is sufficient to define the interaction of this system. Each state machine transition is associated with a biologically-derived probability; a lymphocyte has this probability of engaging in a transition in any time step. In the implementation, the channel connecting a lymphocyte processes to a state process is passed to the next state processes upon a transition. This is a common occam-π pattern of channel mobility, used in many other simulations.

In a non-POP context, a method such as Prometheus has to include separate interaction models to define object message passing protocols. This is also the case in perhaps the most impressive simulation of the immune system, Reactive Animation [27], [28], [29], where the conceptual model uses object diagrams, state charts, and live sequence charts

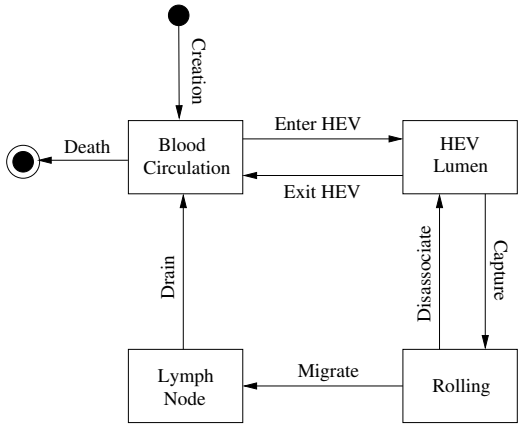

Fig. 4. A state chart abstraction of Figure 3. The four states (boxes) map to the state processes. The linking lines are the permitted transitions, and give rise to the directional channels (based on [15])

(for protocol description). The Harel state charts used by Reactive Animation have transitions labelled with guards (necessary firing context), and details of events generated by the transition, to model cascading events; these alone would be sufficient for the conceptual design of POP simulations.

In [19], we discuss other limitations of diagrammatic models for simulation design. Diagrams represent static structures – of data, of interaction. Spatial and temporal characteristics of systems, that determine how components can be laid out and evolve over time, cannot be captured in static views. Furthermore, current diagramming approaches focus on single components, and have difficulty expressing the idea that there are many components. Even in OO modelling approaches – where a class represents a set of objects, each of which behaves as an individual in the system – there are only limited ways of describing the number of components that take part in the overall system behaviour. Whilst we might use simulation to overcome the temporal and spatial limitations of static modelling, we need to be aware that we are always making assumptions about the temporal, spatial, and collective aspects of the problem entity to be simulated. However, within these limitations, the state chart approach used here supports concurrent POP implementation naturally. Our designs have led to very fast simulations (of tens of millions of processes). The design models and the implemented programs are easy to reuse and adapt, and have been used in a range of applications from robotics to biological simulation by researchers and student projects.

B. Assumptions at each stage of development

In our work with CII, the collection and analysis of assumptions has improved our understanding of the environmental context. Furthermore, our biological collaborators can see exactly the extent to which our simulations capture their knowledge, and how gaps in their knowledge (as well as ours) affect the understanding of the simulated biology.

The CII case study focuses on the behaviour of the mammalian immune system, and in particular on rates of transfer of lymphocytes (specialised white blood cells) between bodily blood circulation and lymph nodes. An account of the biological background for a non-specialist audience and a fuller account of the assumptions and issues can be found in [15]. Here, we summarise the biology very briefly, then consider the sorts of assumption made.

Lymphocytes migrate from the blood to the lymph node through high-endothelial venules (HEV – small blood vessel with a specialised endothelial cell lining). The biological description, after [30], has lymphocytes in a HEV encountering the cell wall and being captured with a certain biochemically-determined probability. A captured lymphocyte can initially disassociate back into the HEV lumen (the inside, or content, of the HEV), or it may become bound. Binding starts a rolling effect, during which biochemical activation results in slowing and finally adhesion to the endothelial cells. Finally, the lymphocyte passes through the HEV membrane into the lymph node. In the presence of infection, experimentalists observe hypertrophy (swelling) of the lymph node. The case study with CII aims to produce simulations that can be used to explore hypotheses about why and how hypertrophy occurs. For example, ultimately we should be able to test hypotheses such as *the increase in lymphocytes in the lymph node during infection is due to HEV dilation*.

In constructing suitable simulations, we follow a development cycle similar to that in the Sargent model; we use that structure here to summarise our assumptions.

1) Problem entity: Our starting point is the biological literature, interpreted for us by expert collaborators at CII. Thus, our problem entity is not the lymphocyte migration and its context, but a *particular biological understanding* of that reality (note that immunologists are unclear about some aspects of the biological process).

We focus on a level of abstraction above the biochemistry of attraction (an adhesion cascade of various cell-surface receptors and molecules), and we ignore the wider context of the lymphatic and blood circulations. This accords with a proposition that we are exploring in CoSMoS, that complex simulations can be layered – if we had a simulation of the biochemistry, and a simulation of the circulation system, then these could be used in the environment of the simulation of the migration process, in place of some of the probabilities that we use at the higher level.

Even in arriving at a mutually-acceptable understanding of the problem entity, we have made a number of significant assumptions. The main one is to reduce the multi-stage adhesion cascade, with its biochemical receptors and activators, to (a) the *capture* of a lymphocyte on the endothelial wall; (b) *rolling*, to encompasses all processes from initial binding to receipt of the chemokine signal for migration; and (c) *migration*. Other stages in the cascade are assumed to be deterministic.

Some more of the assumptions about the migration process and about the environmental context, are given in the following two lists, summarised from [15].

C1: The biological background, including CII experts' advice on quantities and probabilities, is a sufficient, correct and consistent basis for the simulation.
C2: There is no interaction between lymphocytes.
C3: A captured lymphocyte that reaches the rolling stage will always migrate.
C4: All lymphocytes eventually exit to re-enter blood circulation – they do not die during migration.
C5: All lymphocytes in a given state are equivalent: there is no difference between a lymphocyte that enters the HEV for the first time, a lymphocyte that has gone through capture-and-disassociation, and a lymphocyte that has already migrated and re-entered the HEV.
C6: There is no proliferation of lymphocytes during migration.
C7: The probabilities of each change of state do not change on the time scale of the simulation.

The environment of the system represents the parts of the body with which lymphocytes interact in the migration process. In our work so far, the environment model is significantly simplified.

E1: All necessary environmental information is encoded in biologically derived quantities and probabilities.
E2: The details of the structure of the HEV and its biochemistry are not relevant to the simulation.
E3: No detail of the lymph node (that lymphocytes enter if they migrate) is needed in the simulation.
E4: The different environmental conditions that the lymphocytes pass through can be represented as simple lymphocyte states, and the lymphocyte's moving through different environmental conditions can be represented as transitions of lymphocyte state.
E5: There are no relevant effects from blood circulation – blood flow is constant.

In our research, a crucial activity is recording how and why we arrived at each assumption, and to what extent it is accepted by the CII experts. (A similar approach is taken in Reactive Animation [27], where the data that drives the simulation is selected from explicit research reports, so different researchers' theories and findings can be compared).

2) Conceptual model: A conceptual model is a model that has enough detail to understand and analyse the problem, but does not commit the developer to a specific implementation. Our conceptual model, Figure 4, can be implemented in many different ways.

The conceptual model was derived from the problem entity with several reviews by the CII experts. The validation of the conceptual model is expressed in the process of acquiring knowledge, extracting what is relevant to the chosen level of abstraction, and checking that the understanding and assumptions were consistent with expert opinion. Note that, like any validation of a non-deterministic system, the conclusion of the validation is *not* that the conceptual model is correct, but that within the abilities and understandings of all concerned, it is a sufficient starting point. As in dependable systems

engineering, a validation conclusion can change in the light of new evidence.

The assumptions about the problem entity translate into assumptions about the conceptual model. Perhaps the key assumption is that the transitions in the model represent all the allowable transitions that are consistent with the biological information and the assumptions.

3) Computational model: design: In moving from a conceptual model to a simulation, we produce an occam-π program design in which states map to processes and the potential transitions map to appropriate channels. In the design, a lymphocyte is created in the *Blood circulation* state. At each time step in the simulation, a random variable is used to determine whether each lymphocyte remains in its current state or makes one of the possible transitions to another state. On a transition, the communication channel for the lymphocyte is transferred among state processes, using a well-known and verified pattern in occam-π. The necessary synchronisations are handled using the client-server design pattern, which is proven deadlock free for this usage. Appeal to well-defined and formally proven programming patterns contributes to the verification of the simulation construction.

The conceptual model is a static representation of the process structure, that maps cleanly on to the process architecture of the occam-π program (Figure 3, above). A simulation is an execution of a model over time. We assume that it is realistic to implement time as atomic steps, and that a lymphocyte process may change the state process to which it is attached at each time step.

The update time step can be finer grained than the time steps at which data is collected or a visualisation rendered. The granularity of the different time steps can be adjusted so that the simulation shows the fine detail evolving, or so that the rendering (data collection) is more fluid (for instance, for visual realism and to focus on observation of any emergent behaviours).

In general, a complex system simulation is also sensitive to the quantities of interacting components. We have not yet addressed issues of quantity in any detail here[2]. From other work we know that we can produce an efficient occam-π implementation with enough lymphocyte processes to approximate the biological scale (millions). Suffice to say that the population of components in a model must also be subject to validation activities – Sargent's *data validity*.

In engineering terms, we construct an occam-π simulation according to simple mapping assumptions – that states map to processes and that a lymphocyte maps to a process connected to a state process, and so on. Although continuity of concepts makes validation straightforward here, these mappings are also recorded in the catalogue of assumptions. If (when) the simulation is found to be unrealistic, the engineering assumptions must be examined as well as the component and environmental assumptions.

[2]The TUNA project case study, which simulated artificial blood platelets, introduced considerable variation in quantities of components and rates of movement, with some interesting results.

4) Computational model: simulation: From the occam-π design in Figure 3, we have so far constructed two simulations. The first simply measures and reports the number of lymphocytes in each state (the original biological objective). In this version, the transition probabilities are closely related to biological observation supplied by the biologists – from a validation point of view, it is the biologists who are making the assumptions, not the simulators.

The first simulation allows simple comparisons between biological data and the simulation-derived data, which is an important validation of the simulation against the problem entity, but gives little insight into what might be happening – just as monitoring vital signs has little to say about how a mammal works.

A second simulation has been constructed from the same occam-π design (Figure 3), in which the environment is enriched to include a spatial dimension. Lymphocytes are visualised moving through the HEV, with colour-coding of different states. The visualisation is programmed to refresh at each simulation time-step. Again, we use relatively simple, well-understood, and efficient, occam-π patterns to create this extension.

The inclusion of space in the simulation requires more design decisions, which are added to the catalogue of assumptions. Rather than detail these extensive assumptions, here we summarise one set of issues, relating to simulation of lymphocytes in the HEV. In consultation with CII experts, we simulated the HEV as a homogeneous tube, the diameter of which can be changed in a biologically-realistic manner. However, we know that the simulation is not realistic – that the inner surface of a HEV is not smooth and its texture changes with its diameter; that, in general, fluid flow through a tube is not even. Thus, our spatial assumptions are seen to be more tenuous than the earlier assumptions. Essentially, we need information from other branches of biology to complete the spatial model and to use the simulation to address hypotheses that relate hypertrophy to rates of migration. The CII experts are consulting other experts, and designing new experiments to explore the new questions that the spatial simulation has raised.

When we add a spatial dimension to the simulation, we raise new issues relating to the biologically-derived probabilities. The data from biological experiments is for points in time, and does not reflect any three-dimensional or dynamic aspects of the real HEV context. However, when we simulate in three dimensions, we program lymphocytes to be captured (with a certain probability) only when in contact with the HEV lining. This is biologically accurate, but the point-in-time data cannot provide the probabilities needed to run the new simulation. It is ongoing, non-trivial, work to find good probabilities, but because of the way assumptions and decisions are documented, we can easily determine where and how the new inconsistencies affect the validity of the simulation. Ultimately, we would hope to engineer in more of the environmental context, to reduce reliance on probabilities and increase the scope for – hopefully realistic – emergent

lymphocyte behaviours, that can be analysed in simulation to gain insight into the real process of lymphocyte capture.

Two final observations are worth recording. First, the creation of the computational model raised so many issues that, even before the model was complete, validation against the problem entity and experimental trial-and-error iteration had started – but the experimentation is as much about improving the understanding of the problem entity as it is about improving the simulation. Second, as it happens, the CII experts do find even the "invalid" spatial simulation useful, so we could claim that we have met at least part of the engineering goal!

III. Discussion

This brief summary of our simulations of lymphocyte migration using POP is the first phase of our research into engineering complex systems simulation. The development that we have done so far uses intuition, ad hoc analysis, and unstructured recording. Validation is based on recording all the assumptions and decisions that we noticed, and discussing each stage and the results with the CII experts.

We are looking to dependable and high-integrity systems engineering for guidance as to how to systematise our development process. Two areas are particularly relevant: argumentation and systematic analysis.

Validation is the process of arguing the validity of something from the evidence available. This applies equally in scientific research and in engineering systems such as simulations. Support for argumentation is well developed in safety-critical systems engineering, and has been applied to other dependability areas. A safety case (for example as part of the certification of a new aircraft) is made via a safety case argument; the argument structure is summarised diagrammatically, to assist evaluation of the case. 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 – exactly what is needed when working with domain experts to construct simulations for scientific research. 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 [31]. A common approach is to represent the case using the Goal Structuring Notation [32] – this is used commercially for safety cases, and has also been applied to aspects of system dependability [33], including two "thought experiments" on a hypothetical blood platelet system [34], [35]. Some work already exists on the role of argumentation in design [36]. In [15], we present initial work on a validation argument for the *Capture* transition in the CII case study.

Systematic analysis is also a feature of any form of critical systems engineering. In constructing a dependability argument, it is necessary to identify as many issues (hazards, threats) as possible. Systematic *deviational techniques* are used to challenge assumptions, decisions, and model components (see [37], [38]). A typical example is *Hazard and Operability* (HAZOP). Developed initially for analysing the designs for chemical plants, the HAZOP approach comprises systematic application of guidewords to components of a model. The guidewords vary with application – for example the UK Defence Standard 00-58 provides guidewords for use on systems containing programmable electronics.

A systematic deviational approach would systematically challenge each of our assumptions and model components. For example, we make assumptions, based on the biological advice, about the size of a lymphocyte relative to the HEV. A deviational analysis might ask us to consider at least the following:

- All lymphocytes are larger/smaller relative to the HEV;
- Some lymphocytes are larger/smaller than others;
- Some HEV sections are larger/smaller than others;
- Even though the relative sizes are correct, lymphocytes differ in some other way, such as their attraction to the HEV lining.

Having checked each deviation, we can then construct this part of the validation argument, using as evidence the support from (cited) literature and (named) experts that our sizes are valid.

Because we are working in a developing area of biology, we know that subsequent research may show that our assumptions and evidence are inappropriate. The validation argument, and the records of assumptions, decisions and evidence, provide a basis for revising the models, and then revising the simulation – rebuilding all the necessary validation arguments accordingly.

Similarly, if we ultimately produce a layered simulation, in which, for example, capture and rolling is an emergent effect from a lower-level simulation (suitably validated) of the biochemistry, then we would have to modify the validity argument to refer to evidence of the validity of the biochemical model and of the validity of assumptions about the lymphocyte and HEV biochemical interaction.

Note that we have already encountered situations where the interplay with the CII experts has produced revised understanding of the problem entity – either because the experts have spotted places where we have misinterpreted their advice, or because the experts have improved their knowledge of the migration environment. Tables of assumptions and biological details support traceability through the modelling and simulation process, and experts can inspect them to highlight inconsistencies.

There is much work still be to done on validation of complex system models and simulations. As well as reflecting state-of-the-art dependable and high-integrity systems

engineering analysis and argumentation techniques, we need to investigate structured ways to lay out our assumptions, design decisions and biological evidence. We also need to establish schemes for mapping between our biological details and simulator parameters. We intend ultimately to establish patterns of development and patterns of validation that are applicable to the validation of many different complex systems.

IV. Acknowledgements

This work is part of the CoSMoS project, funded by EPSRC grants EP/E053505/1 and EP/E049419/1. The original immunological model is based on collaborative work with Lisa Scott and Mark Coles from the York Centre for Immunology and Infection, University of York, UK, and extensive discussion with Jon Timmis of the Departments of Computer Science and of Electronics, University of York, UK. Work on continuity in modelling draws on discussion with Tim Hoverd, Susan Stepney, and other CoSMoS researchers.

References

[1] R. Alexander, "Using simulation for systems of systems hazard analysis," Ph.D. dissertation, Department of Computer Science, University of York, 2007.

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

[3] E. D. Paolo, J. Noble, and S. Bullock, "Simulation models as opaque thought experiments," in *Articial Life VII*. MIT Press, 2000, pp. 497–506.

[4] M. Wheeler, S. Bullock, E. D. Paolo, J. Noble, M. Bedau, P. Husbands, S. Kirby, and A. Seth, "The view from elsewhere: Perspectives on alife modelling," *Artificial Life*, vol. 8, no. 1, pp. 87–100, 2002.

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

[6] J. M. Epstein, "Agent-based computational models and generative social science," *Complexity*, vol. 4, no. 5, pp. 41–60, 1999.

[7] P. S. Andrews, A. T. Sampson, J. M. Bjorndalen, S. Stepney, J. Timmis, D. N. Warren, and P. H. Welch, "Investigating patterns for the process-oriented modelling and simulation of space in complex systems," in *Artificial Life XI*. MIT Press, 2008.

[8] C. G. Ritson and P. H. Welch, "A process-oriented architecture for complex system modelling," in *Communicating Process Architectures 2007*, vol. 65. IOS Press, 2007, pp. 249–266.

[9] 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*, ser. LNCS, vol. 3382. Sringer, 2004, pp. 126–141.

[10] L. Padgham and M. Winikoff, "Prometheus: A methodology for developing intelligent agents," in *AOSE III*, ser. LNCS, vol. 2585. Springer, 2003, pp. 174–185.

[11] CCTA, *SSADM Version 4 Reference Manual*. NCC Blackwell Ltd, 1990.

[12] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen, *Object-Oriented Modelling and Design*. Prentice Hall, 1991.

[13] P. Stevens and R. Pooley, *Using UML*, 2nd ed. Pearson, 2006.

[14] O. Paunovski, G. Eleftherakis, and T. Cowling, "Framework for empirical exploration of emergence using multi-agent simulation," in *Workshop on Complex Systems Modelling and Simulation*. Luniver Press, 2008, pp. 1–31.

[15] P. S. Andrews, F. Polack, A. T. Sampson, J. Timmis, L. Scott, and M. Coles, "Simulating biology: towards understanding what the simulation shows," in *Workshop on Complex Systems Modelling and Simulation*. Luniver Press, 2008, pp. 93–123.

[16] R. G. Sargent, "Verification and validation of simulation models," in *37th Winter Simulation Conference*. ACM, 2005, pp. 130–143.

[17] ——, "The use of graphical models in model validation," in *18th Winter Simulation Conference*. ACM, 1986, pp. 237–241.

[18] F. Polack, S. Stepney, H. Turner, P. Welch, and F. Barnes, "An architecture for modelling emergence in CA-like systems," in *ECAL*, ser. LNAI, vol. 3630. Springer, 2005, pp. 433–442.

[19] 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.

[20] B. P. Zeigler, "A theory-based conceptual terminology for M&S VV&A," Arizona Center for Integrative Modeling and Simulation, Tech. Rep. 99S-SIW-064, 1999, www.acims.arizona.edu/PUBLICATIONS/publications.shtml.

[21] P. Welch and F. Barnes, "Communicating mobile processes: introducing occam-pi," in *25 Years of CSP*, ser. LNCS, vol. 3525. Springer, 2005, pp. 175–210.

[22] C. Hoare, *Communicating Sequential Processes*. Prentice-Hall, 1985.

[23] R. Milner, *The Pi Calculus*. Cambridge University Press, 1999.

[24] A. T. Sampson, "Two-Way Protocols for occam-π," in *Communicating Process Architectures 2008*, WoTUG. IOS Press, 2008, pp. 85–97.

[25] J. Martin and P. Welch, "A Design Strategy for Deadlock-Free Concurrent Systems," *Transputer Communications*, vol. 3, no. 4, 1997.

[26] P. Welch and F. R. Barnes, "A CSP Model for Mobile Channels," in *Communicating Process Architectures 2008*, WoTUG. IOS Press, 2008, pp. 17–33.

[27] D. Harel, Y. Setty, S. Efroni, N. Swerdlin, and I. R. Cohen, "Concurrency in biological modeling: Behavior, execution and visualization," in *FBTC 2007*, ser. ENTCS, vol. 194, no. 3. Elsevier, 2007, pp. 119–131.

[28] S. Efroni, D. Harel, and I. R. Cohen, "Reactive Animation: realistic modeling of complex dynamic systems," *IEEE Computer*, vol. 38, no. 1, pp. 38–47, 2005.

[29] I. R. Cohen and D. Harel, "Explaining a complex living system: dynamics, multi-scaling and emergence," *Journal of the Royal Society Interface*, vol. 4, pp. 175–182, 2007.

[30] K. Ley, C. Laudanna, M. I. Cybulsky, and S. Nourshargh, "Getting to the site of inflammation: the leukocyte adhesion cascade updated," *Nature Reviews Immunology*, vol. 7, no. 9, pp. 678–689, 2007.

[31] T. P. Kelly, "Arguing safety – a systematic approach to managing safety cases," Ph.D. dissertation, Department of Computer Science, University of York, 1999, yCST 99/05.

[32] R. A. Weaver, "The safety of software – constructing and assuring arguments," Ph.D. dissertation, Department of Computer Science, University of York, 2003, yCST-2004-01.

[33] 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.

[34] R. Alexander, R. Alexander-Bown, and T. Kelly, "Engineering safety-critical complex systems," in *Workshop on Complex Systems Modelling and Simulation*. Luniver Press, 2008, pp. 33–63.

[35] F. Polack, "Argumentation and the design of emergent systems," working paper, available at www-users.cs.york.ac.uk/~fiona/PUBS/Arguments.pdf.

[36] W. Wu and T. Kelly, "Towards evidence-based architectural design for safety-critical software applications," in *Architecting Dependable Systems*, ser. LNCS, vol. 4615. Springer, 2007.

[37] D. J. Pumfrey, "The principled design of computer system safety analyses," Ph.D. dissertation, Department of Computer Science, University of York, 2000, yCST 2000/05.

[38] T. Srivatanakul, "Security analysis with deviational techniques," Ph.D. dissertation, Department of Computer Science, University of York, UK, 2005, yCST-2005-12.