Equivalence Arguments for Complex Systems Simulations – A Case-Study

Teodor Ghetiu[1], Robert D. Alexander[1], Paul S. Andrews[1], Fiona A. C. Polack[1], and James Bown[2]

[1] Dept of Computer Science, University of York, York, YO10 5DD, UK
{teodorg,rda,psa,fiona}@cs.york.ac.uk
[2] University of Abertay, Dundee, UK j.bown@abertay.ac.uk

Abstract. Complex systems are often simulated to provide a basis for research or analysis. However, complex systems simulation often fails to properly demonstrate that the constructed simulation is an adequate tool to support investigation of the system under study. To address this issue we adopt and adapt argumentation techniques traditionally used for safety critical systems (SCS). Here we present part of an on-going case-study in which these techniques are used to demonstrate that two different implementations of a complex system simulation are adequately equivalent. This is a first step in producing further simulations of the system under study, which will be shown to be valid models on which to explore particular ecological phenomena.

1 Introduction

This paper presents part of a case study that is using a principled approach to computer simulation of a complex system. The work is part of the CoSMoS project[1], which is developing a general framework for the simulation of complex systems using agent-based approaches. One of our long-term goals is to argue the validity of complex systems simulations against domain models that capture an explicit expression of scientific understanding. More generally, we want to present properly-evidenced arguments that one model is an adequate representation of another model, or a particular perspective on reality. Such arguments form the basis for discussion between the simulators and the domain experts, and capture the rationale for the simulation, in terms of both the domain understanding of the science, and the engineering of the simulation – see [1, 2] for further discussion. We believe the ability to argue properties of a complex system simulation (such as equivalence or validity) is an important element of CoSMoS and any other similar approach.

In this case study, a first step is to re-engineer a simulation of intra-specific plant variation [3]. The existing object-oriented simulation, in C++, needs to be scaled-up to support the scientific research. Elsewhere, we discuss the use of occam-π for efficient, process-oriented parallel agent-based simulation [4]. A

[1] http://www.cosmos-research.org

number of recent complex systems simulations [5–8] have successfully used this programming language. Here, we will ultimately exploit parallelisation to distribute the occam-π simulations over a cluster of machines [9]. This will allow us to simulate larger numbers and variations of plant, and a greater range of environmental influences than is possible in a purely sequential implementation.

The re-engineering is supported by construction of an argument that the occam-π simulation is *adequately equivalent* to the original C++ simulation. We call this an equivalence argument. The description of this argument forms the the main subject of this paper. The original C++ simulation was used in ecological research that has some credibility within its research community, and through the equivalence argument we can support a claim that our re-implementation should share that credibility. Thus, our definition of adequate equivalence must make a case that the simulations capture equivalent aspects of the scientific domain, rather than simply presenting evidence of the technical equivalence of the two programs. Our argument is acceptable if the scientists – here represented by the person who has overseen the C++ simulation effort, James Bown (referred to subsequently as *the scientist*) – accept the argument that we have captured equivalent aspects of the scientific domain.

The paper continues with a brief introduction to argumentation techniques and their relation to simulation validity, section 2. Section 3 then describes the plant ecology case-study. Section 4 considers what adequately equivalent means and shows how we can build an explicit, structured argument of adequate equivalence for the two simulations. Section 5 gives examples of the required evidence from the simulations, to support the argument presented in section 4. The paper concludes with a discussion, section 6, conclusions and proposals for future work.

2 Argumentation and Complex Systems Validity

Elsewhere, we summarise current scepticism about the ability of computer simulations to adequately support scientific research (see [10], and cited work, [11–15]). In [1, 2], we report on an immunological case study undertaken in conjunction with immunologists, in which we found that a systematic collection and exposure of assumptions, made by the immunologists in relation to the scientific domain and by us as modellers and simulation-engineers, helped the immunologists to understand the value and limitations of our simulations. This understanding meant that the immunologists could use even basic agent-based models to test their understanding and guide their laboratory experiments; the documented assumptions gave rise to new avenues of scientific research. Here, we follow a suggestion in [2], and turn to conventional techniques from critical systems engineering, to start the process of systematising the use of *arguments* to capture and analyse evidence and assumptions.

In critical systems engineering, arguments are used to demonstrate a *case* to regulators that a system has certain properties, most commonly properties related to safety. In critical systems, it is impossible to absolutely demonstrate properties such as safety; instead evidence is collected based on criteria such as

use of accepted development practices, software, system and sub-system testing, mechanical analysis, past experience or cumulative usage outcomes, and field trials. The evidence is used to support an argument that the risk associated with the system is *As Low As Reasonably Practicable* (ALARP), within the operational environment for which the system is designed. A general approach to constructing and documenting safety cases can be found in Kelly [16], whose other published research includes a range of studies and applications of critical systems argumentation. For an example of safety case creation for a – hypothetical – complex system, see [17].

2.1 Argumentation in Safety Critical Systems

Early safety-critical systems were unregulated, and were potentially grossly unsafe [18]. Consequent deaths and damage costs from, for instance, industrial and vehicle accidents, led in time to regulation, part of which is usually *certification*. Potentially-dangerous systems are allowed if there is sufficient evidence that they would be safe to operate. For a long time, *evidence* was based on process – "I have followed good engineering practice, so my system is safe". This approach is unsatisfactory in many ways, not least of which is its limiting of engineers to use only approved processes, thus inhibiting innovation.

A significant improvement in safety management came with *product-based certification*. Independent regulators are appointed, who set the safety criteria that a system must meet, in terms of specific evidence requirements. Developers collect evidence, and tie it together by means of a structured argument known as a *safety case*. It is still possible to cite an approved process as evidence, but this evidence is relegated to an appropriately-subordinate role. A safety case is accepted or rejected based on independent review of its arguments and evidence. Acceptability is not an absolute, and can change over time, in the light of experience or new evidence. This presents an important parallel to scientific investigation, particularly in biological domains, where the understanding of complex natural systems is a developing area, with much debate and many competing theories.

2.2 Summarising the Structure of an Argument: GSN

Safety cases were conventionally presented as free text, which is easy to create and immediately readable, but hard to systematically review. As Kelly [16] notes, not all safety engineers are gifted writers, and free text safety cases are often ambiguous. Construction and review of cases is improved if the structure of the argument and evidence can be summarised, for example using the Goal Structuring Notation (GSN) [19, 16]. Existing examples and patterns for GSN are predominantly concerned with safety cases.

GSN is a graphical way to express argument structures. A GSN diagram shows a hierarchy from the top-level claim – a typical safety case might seek to establish that *The system is safe* – down through sub-claims that support that claim (e.g. *The hazard 'loss of temperature control' will not occur*) and

eventually to the evidence supporting those claims (e.g. *Software test results for component X show no faults*). Anybody using GSN is guided by the rules of the notation, which helps to avoid gross errors of logic.

It is important to understand that GSN as a notation is of limited value – it is the argumentation culture and the safety-case literature that gives it its power in the safety field. Similarly, it would be a culture of argued validation that would be most important in addressing the criticisms (noted above) of complex systems simulation for scientific research.

2.3 Adapting Argumentation for Scientific Simulation Validity

When a computer simulation is used in a scientific study, the user of a simulation needs to demonstrate the extent to which the computer simulation matches reality (and other models). Traditionally, these arguments have been, at best, informal discussions in papers and reports. This causes many problems. Evidence or detail is omitted, making it difficult to assess the validity of simulation results. In an attempt at clarity, many arguments are reduced to vacuous or partial claims. There is a need to improve the quality and presentation of validation arguments; GSN is an obvious candidate for constructing argument structures and recording the evidence that supports (or could support) the argument.

Whilst there is a range of work on the validity in simulation, for example [20, 21], we are not aware of any existing work on structured arguments of computer simulation validity.

In safety analysis, the safety properties and the case for safety are normally created and rehearsed by the developers before the argument is constructed and represented in GSN. There are few specific argument construction methods, and experience shows that, whilst a top-down approach is impractical because it requires oversight of the body of evidence before the top-down structure can be identified, a bottom-up approach risks losing sight of the point of the argument.

The argumentation that we require for simulations is somewhat different to safety case argumentation, in that we are constructing arguments in parallel to simulation development, and can use the top-down construction of the argument to guide development. Similarly, we do not have a regulator dictating what is and is not acceptable evidence, but instead we have a scientific collaborator who must be able to understand and review our argument. In this paper, the goal is to demonstrate that two simulations are adequately equivalent. Our argument proceeds by analysing and recording what we will accept as evidence of adequately equivalent. We then establish this evidence by systematic analysis, recording the result as a GSN argument structure. First, we briefly introduce the intra-specific plant variation domain and the existing C++ simulation.

3 The Example: Intra-specific Plant Variation Simulations

The work presented in this paper is the first phase of a case study to provide computer simulations to support extensions to the ecological research of Bown

et al [3], based on their novel model of plant physiology and interactions, based on physiological traits.

In [3], computer simulation is used to demonstrate that defining plants in terms of a suitable set of traits yields results that are acceptable to the ecological community, for example, the model produces species-area and species-abundance distributions that have typical characteristic statistical signatures (curves) [22]. However, the existing simulations are limited in the number and complexity of components that can be modelled, even if the implementation and platform were fully optimised, because of the difficulty of distributing a C++ program.

3.1 Ecological Modelling and Plant Trait Models

Begon et al define ecology as the "scientific study of the distribution and abundance of organisms and the interactions that determine distribution and abundance" [23]. The "holy grail" of ecology [24] is to find general rules that relate environmental conditions, species characteristics and community composition.

To complement field experiments, ecologists attempt to capture observational patterns and behaviours in models. At one extreme, equation-based models (EBMs) focuses on characteristics of the plant population as a whole, while at the other extreme individual-based models (IBMs) that allow for some of the individual variations within and between species. IBM is the more appropriate technique for study of intra-specific variation, and has the advantage that IBM individuals can map directly to and from real plants, so biological understanding can be mechanistically reflected in computer models. However, a computer model cannot hope to express all the characteristics of a real plant. A popular ecological technique is to summarise the characteristics of a plant in terms of numerical traits, with much ecological research to establish the most appropriate traits and value-ranges. Traits typically characterise visible, phenotypical properties such as shoot height, as well as ongoing biological processes such as water uptake capacity. A good model has rich informational content built using traits whose validity is supported by the direct mapping to biological data.

Ecological research has shown that *trait trade-off* is important in explaining the distribution and abundance of ecological communities [25]. Computer-based IBMs that model plants in terms of traits play a key role in this research. However, the models do not always map well to research goals, and it has been shown that the identification and representation of traits has a significant influence on the simulation results [26, 27].

3.2 The Computer Simulation of Bown et al

The intra-specific plant variation models of Bown et al [3] uses an IBM based on a resource-centric physiological scheme [28]. The model allows the study of the relationship between trait trade-off and the distribution and abundance of species.

Firstly, Bown et al [3] establish twelve traits (table 1) that adequately describe plant physiology. The plant species is described by a set of twelve distributions, one for each of these traits. The distributions determine the probability of each trait value across the set of plants, with individual trait values assigned to achieve the species distribution. This approach gives appropriate intra-specific variation.

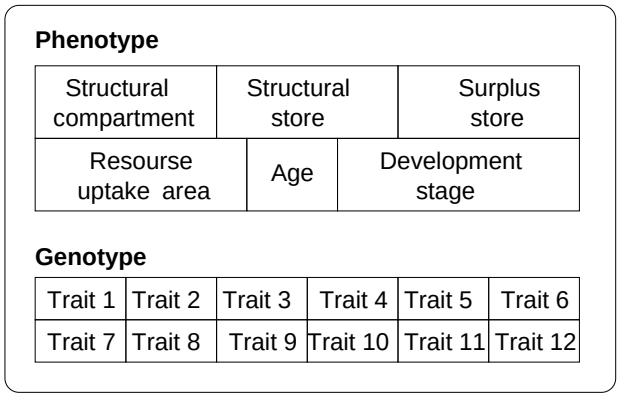

Fig. 1. Bown et al's model of an individual plant [3]

In the model of Bown et al [3], a plant individual is modelled as a *phenotype* and a *genotype*, figure 1. The phenotype consists of appropriate representations of the resource storage and usage of a plant: the structural compartment represents resources corresponding to the plant's fixed structure; the structural store holds resources that are used for reproduction; and the surplus store represents any excess of resource-uptake over the level essential to maintain the plant. In addition, the phenotype records age and development stage. In the genotype, a value is assigned to each of the twelve species traits, using a random sampling of the trait distribution to introduce intra-specific variation. Trait value distributions were obtained from field observations of the *Rumex acetosa* plant species [29].

Four biological processes drive the generic life-cycle of a plant: resource uptake, resource allocation, reproduction and development, as shown in figure 2. In the model, each plant takes up resources from the environment and allocate it to the three resource components of the phenotype. As resource is accumulated, the plant develops, which is denoted by incrementing the Development stage in the phenotype. Four of the trait values are related to a plant's development stage: spatial distribution of uptake, development dependent reproduction, and the two uptake traits.

There is an initial population of plants. When a plant reproduces the distribution of seeds is controlled by the seed dispersal pattern trait. A seed is only

Table 1. Bown et al's twelve plant traits [3]

Trait	Description
Essential uptake	Amount of resources that a plant needs for normal development without reproduction
Requested uptake	Amount of resources that a plant will request to support development and reproduction.
Spatial distribution of uptake	Uptake capacity of a plant with respect to the distance
Compartment partition	Resource allocation ratio of structural compartment to structural store
Structural store release proportion	Proportion of structural store that can be released
Surplus store release proportion	Proportion of surplus store resource that can be released
Time dependent reproduction	Time needed before initiating reproduction.
Development dependent reproduction	Resource level needed to initiate reproduction
Storage/fecundity relation	Ratio of the resource available for reproduction to the resources necessary for creating a seed
Seed dispersal pattern	Radius of the area of local seed dispersal
Survival threshold	Minimal resource level for plant survival
Survival assessment period	Number of consecutive timesteps over which the resources level can be below the survival threshold before the plant dies

viable if it lands at a valid location that does not contain a plant. In [3], reproduction is clonal, so a seed has the same trait values as its (single) parent plant. The Reproduction process may be triggered according to the trait value for time dependent reproduction or for development dependent reproduction.

The environment is represented by a single type of resource, which is distributed evenly across its surface. The resource level has an upper limit defined by a saturation level. The flow or resource to plants is constrained by release and replenishment rates, which specify the maximum quantity of resource that can be released or added to the environment at any time. In the computer simulation, the environment is modelled on a two-dimensional grid. Bown et al [3] note that a cell represents an area of approximately 100cm^2, which, in the model, can be occupied by at most one plant. The number of plants that take resource from a cell is determined by the location of each plant and its root area, as represented by the spatial distribution of uptake trait. Grid cells contain a resource substrate, which is parametrised by the saturation level and the release and replenishment rates.

A timestep in the simulation represents one day in the real-world. Accordingly, the values that are used for parametrisation of the model reflect the resource flow through a plant during one day [3].

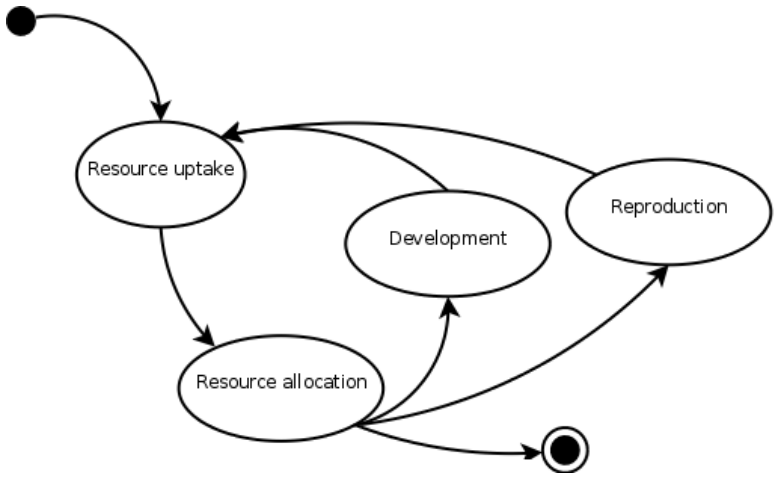

Fig. 2. State machine model of the biological processes of Bown et al [3]: ellipses represent the states of the plant associated with each biological process, and arrows represent possible transitions between theses states; the plant is created in the Resource uptake state and must be in the Resource allocation state when its death is determined

In order to compare the trait-model intra-specific results to inter-species distribution results, Bown et al [3] introduce 75 individual plants, which are treated as representing 75 different species. Because the model uses clonal reproduction, these 75 species either persist and increase in numbers, or die out. A simulation run lasted for 50 000 timesteps, which corresponded to around 1250 generations of plants. The simulation was run over different environment sizes (grids of 10×10 up to 50×50 cells) to collect statistics on the relationship of the size of the environment to the number of species co-existing (the species-area curve), and to the abundance of each species (the species-abundance curve).

3.3 The C++ and occam-π Simulations

Bown et al [3] use a mechanistic model of plants through which community level processes can be studied. We have re-implemented the simulation in occam-π, but in order to use our simulation to scale up the original experiments, we need to show that the new simulation is still based on the same underlying biological model.

The C++ simulation code is sequential, running on a single thread of execution. The model uses two passes per timestep to reduce sequential bias. For example, for resource uptake, all plant demands are made in the first pass, then, in a second pass, each plant receives a normalised percentage of the quantity it requested – where the total demand on a grid cell is more then the cell can release then each demand is reduced accordingly. The limitation of running on a single thread constrains the size of the environment and population that can

be used in this simulator, which cannot handle the real-world scale of several hectares containing millions of plants.

A traditional re-engineering approach would create an abstract model of the data and processing implemented in the C++ simulation, and then re-develop this model in occam-π. This would, in theory at least, allow formal refinement relations to be established between each implementation and the abstract model, and a formal proof of equivalence. In practice, whilst model-driven engineering provides semi-formal transformation approaches to move between object-oriented models at different levels of abstraction, the potential for formal refinement between abstract models and object-oriented code is limited. Furthermore, having extracted an abstract model from the object-oriented code, there is no established way to refine this model into the process-oriented occam-π language – occam-π is formally underpinned, but by CSP [30], an event-driven formal language.

If a formal approach were to be found, it could establish a measure of equivalence between the implementation codes of the two simulations, but would not allow the re-engineered version to take full advantage of the strengths of occam-π. Most significantly, here, the re-engineered occam-π simulation can represent plants and locations as individual occam-π *processes*, each having its own thread of control. The occam-π processes communicate through *channels* through which data can be passed. This gives a closer mapping between the implementation and the biological reality than was evident in the C++ simulation.

4 A Structured Argument for Adequate Equivalence

This section works through the argument of adequate equivalence constructed for the C++ and occam-π implementations. For simplicity, we will refer to the C++ implementation as C, and the occam-π implementation as O. The argument is presented in GSN, using the standard notations, given in figure 3. The meaning of these symbols in our work is elucidated in the description of the argument that follows.

Note that the equivalence argument does not attempt to address the rationale or engineering of the C++ simulation – this is an established system that we cannot change. We do not compare the performance of the two implementations, as the motive for the re-engineering is not any immediate performance gain, but the distribution potential of the occam-π simulation across computer grids [6], with the efficient management of processes and events [31].

4.1 The Top Goal

A GSN argument starts with a *top goal*. In figure 5, this is shown as the rectangle labelled OCEquiv – O simulation is adequately equivalent to C simulation. This is the claim that we want to make, and the whole argument below is devoted to making that claim. In the diagram, lines with solid arrowheads connect each goal to lower-level components that together meet the goal.

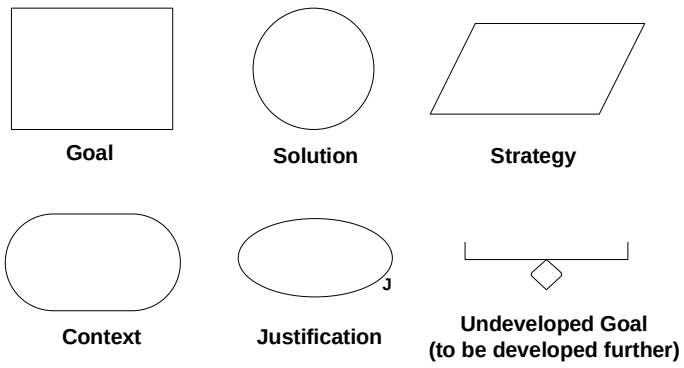

Fig. 3. GSN notations used in the equivalence argument: explanations are given in the text description of the arguments that follow

A goal exists in a context. In figure 5, DefAdEq labels a context node, here promising that a definition of *adequately equivalent* is given elsewhere – in fact, the definition is given and explained in this section of the paper.

It is hard to definitively define equivalence. Structures in different languages may be syntactically different but semantically equivalent, or *vice versa*; we may have behavioural bi-similarity from different structures, or, since we are modelling complex systems, we may observe different results from similar initial conditions even within the same implementation. Despite this we need a definition of what we mean by *adequate equivalence* in order to argue convincingly about it.

We therefore propose that:

> *the two simulations are adequately equivalent if they produce the same results over the whole range of concern.*

In common with most analyses of complex systems, *same results* can be defined by statistical analysis – we run each simulation many times, and collect the results. This gives a distribution for each result. We then use an accepted statistical test (usually a non-parametric test that medians and inter-quartile ranges represent the same distribution at some confidence level) to determine whether the results can be considered equivalent.

The *range of concern* is defined by ranges for parameters over which the equivalence should hold. In the plant simulations, this relates to the range of environment sizes and initial plant numbers. Note that, because we cannot execute the C++ simulation on very large populations, we can only consider equivalence within the range of this simulation. Instead, we present direct comparisons of results within the range of the C++, and theoretical arguments for the rest of the range. The comparison of the results gives us high confidence within part

of the range, while the theoretical arguments give us some confidence, but at a lower level, beyond that. This is represented figuratively in figure 4.

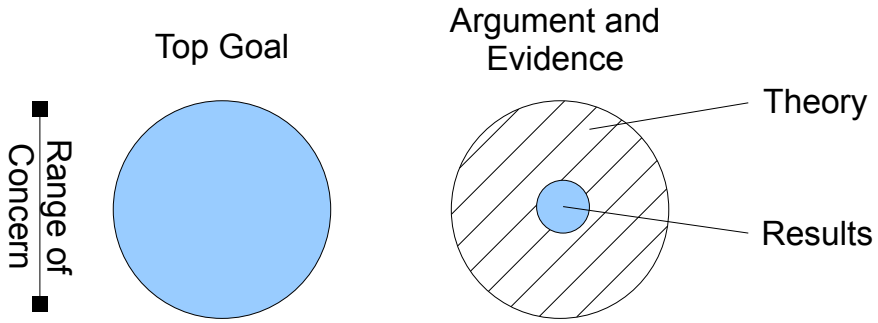

Fig. 4. Range of concern for arguing *adequate equivalence*: we wish to be convinced over the whole range of both simulations (the Top Goal), but we can only produce results evidence for part of the range; in the rest of the range we rely on other forms of evidence

Note that the crucial factor in determining whether the definition of *adequately equivalent* is sufficient is a discussion with the scientists. Thus, in our case study, we consult the scientist directly; since he considers that our definition is sufficient, we can proceed. It is, of course, possible that this initial acceptance may be reversed when the evidence is complete and the whole argument presented – perhaps the scientist can demonstrate that our non-parametric tests of statistical equivalence are inappropriate, or our theoretical arguments are flawed, or perhaps we find that there are bugs in one of the simulations that affect the comparability of the results in other ways. The dialogue to establish the definition and associated argument is essential in the establishment of trust and understanding between simulators and scientists [1, 2].

4.2 Decomposing the Top Goal

Having agreed a top goal and the definition of the key terms that it uses, we need to provide an argument that the goal is satisfied. In figure 5, the top goal OCEquiv is met by following the ArgSciImplRes *strategy*. A strategy in an argument explains the connection between a goal and its sub-goals. Here, ArgSciImplRes states that we argue over three distinct areas – the underlying science, the details of how the simulations are implemented, and the actual results that they produce. The relationship here is complementary – each child goal gives us some confidence that the parent goal holds, and together, they give us *adequate* confidence that the goal is met.

Note that the three-goal sub-argument in figure 5 is not an alternative definition of what it means for two simulations to be adequately equivalent. Rather, it is an approach to substantiating such a claim. We are using the three-legged argument to support a claim that the results will be the same across the whole range of concern.

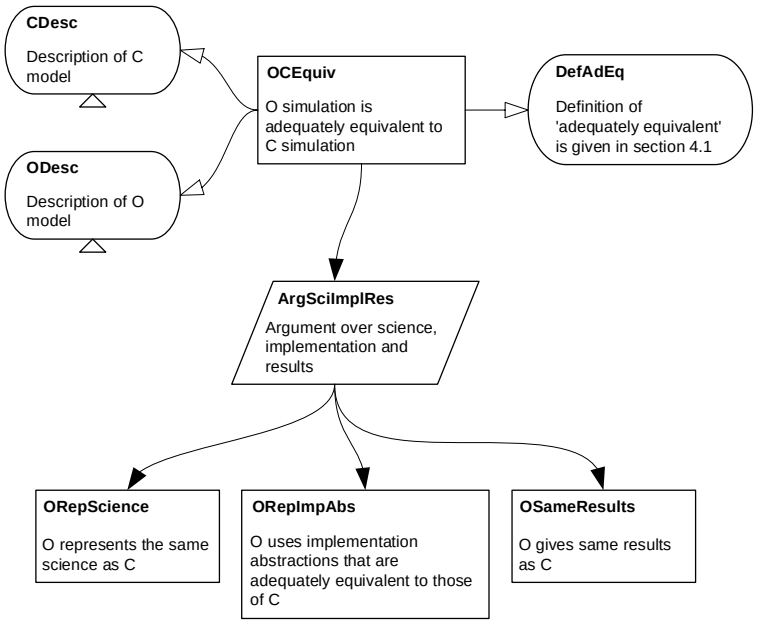

Fig. 5. Top level of the argument that the C++ (C) and occam-π (O) simulations are adequately equivalent

The text in the GSN goal boxes is necessarily terse, and refers to concepts that need to be defined, as in the above discussion of *adequately equivalent*. It is hard to provide compelling contexts and definitive definitions. This is seen as a benefit, not a cost, of making structured arguments – you get to see where your definitions are vague or unsatisfying. (It is also much easier to see when another person's arguments are weak.)

In GSN, an upward triangle beneath a context box means that it has yet to be instantiated – it is a placeholder for concrete content that is not yet available. In figure 5, the CDesc and ODesc context boxes could be instantiated by a reference to the code of the simulations, to common abstractions such as figure 2, above, or to summary text such as the descriptions in section 3. The argument is not complete until this instantiation is performed.

Again, the point of GSN arguments is not to demonstrate with absolute certainty that the top goal holds, but to demonstrate why the author of the argument believes that it is holds. The reviewer can disagree with the assumptions, strategy, and eventual evidence, and can challenge the author to find a better argument. Here, for instance, our scientist may dispute the strategy of arguing over three distinct areas, or may dispute the totality of these complementary areas, and challenge the author to make better justifications for its argument.

The three lowest-level goals shown in figure 5 are expanded in figures 6 to 8. Each of these argument fragments terminates in a circular *solution* node. Solutions refer to the *evidence* that supports a claim. In very simple arguments, evidence might directly support the top goal, but in practice, such intermediate sub-goals and strategies are needed to create a compelling argument. The following sections consider each of the three sub-goals in turn.

4.3 The Science Goal

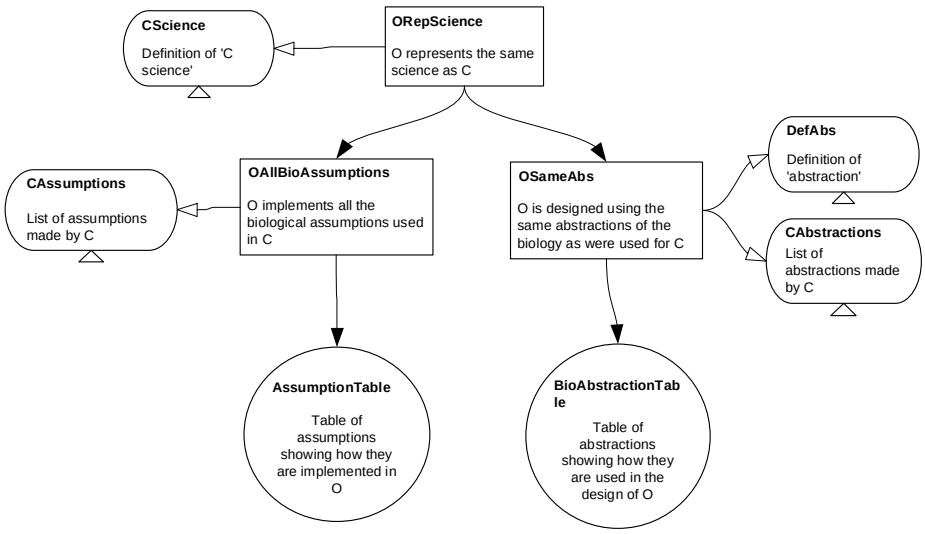

Fig. 6. Elaboration of the sub-goal to show that the simulations represent the same science, from figure 5

In figure 6, the goal, ORepScience, is shown as being solved by two further goals. OAllBioAssumptions presents an argument that the occam-π version is based on the same assumptions about the actual biology as the C++ version. OSameAbs argues that the occam-π simulation abstracts from the details of the biology in the same way as the C++ version. Again, we expand the goals by

providing context. From these goals, we directly reach the evidence required, with solutions pointing to tabular comparisons.

We could expand the argument, and the GSN, further to argue over each compared assumption or abstraction, providing a specific argument that each pair is adequately equivalent. This might be necessary if scientist found the comparison tables unacceptable without further evidence.

4.4 The Implementation Goal

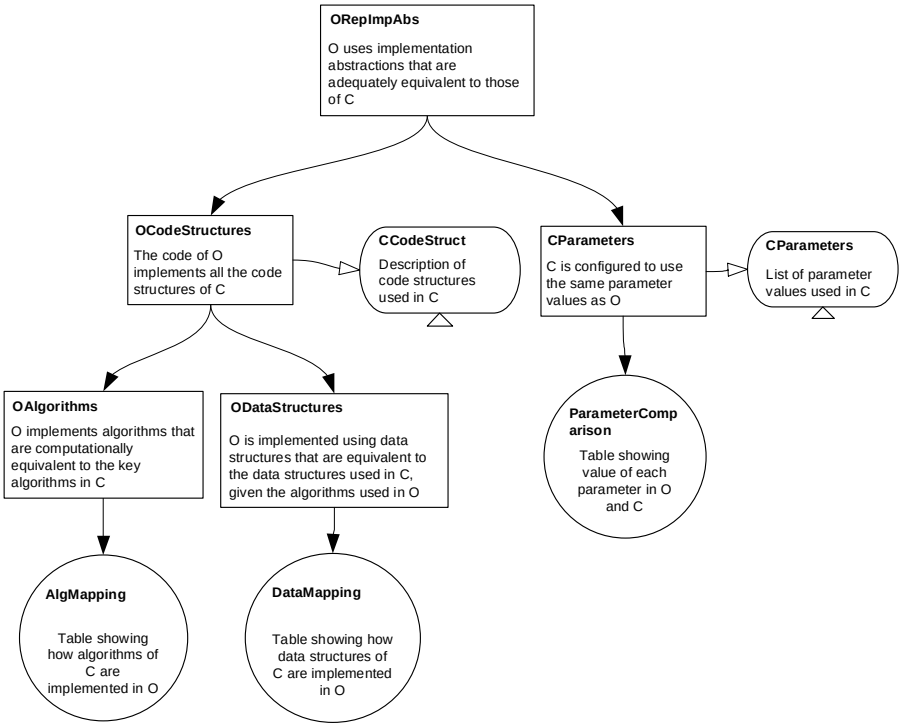

Fig. 7. Elaboration of the goal to show that the simulations represent the same implementation abstractions, from figure 5

The second child goal in figure 5, ORepImpAbs, is expanded in figure 7, with new sub-goal relating to the adequate equivalence of the code structure (OCodeStructures) and parameters (OParameters) in the two simulations. The reasoning here is that the simulation implementations are adequately equivalent if they run equivalent algorithms on equivalent data structures and use the same parameter settings.

OParameters is solved directly by a table comparing parameters in the two simulations, whilst OCodeStructures is further decomposed into a claim about algorithms and a claim about data structures. Each of these is, again, solved by a table that compares key elements of the two implementations.

Again, despite the appearance of precision provided by the GSN notation, much of the argument here is still implicit and left to the reader to infer. For example, it is implicit that equivalence of code structures and equivalence of parameters is sufficient to argue equivalence of implementation. Similarly, the argument that two algorithms in the table are equivalent is not made explicit. A software expert could verify or refute our claims, by whatever means they chose, but the non-expert must take our assertions on trust or ask for a further level of argument.

Also note that although the top-level goal talks about equivalence in terms of a black box that produces results, the argument here is white-box – we talk about how the simulation works internally. We are using white-box methods to support a claim expressed in black-box terms. This is similar to software testing, where it is common to combine white-box and black-box methods.

4.5 The Results Goal

The third child goal in figure 5, OSameResults, is decomposed, in figure 8, into claims relating to the testing and experimentation on the two simulations.

OCBoundaryCases claims that the two simulations provide the same results for boundary and extreme cases within the valid range. This is based on a common testing strategy, to establish that unusual situations are properly managed. We have not developed this goal yet, as is shown by the diamond beneath the goal box. To develop it, we need to consider what cases to test, in terms of the parameter and value settings that characterise each case – for instance, we may test both simulations on the case where all plants are the same, in order to check that clonal reproduction is implemented similarly; we might then check the behaviours that result with very small and very large initial numbers of plants (starting with the same plant populations), then look at the effects of extreme environments. Unless equivalence were obvious – in a very poor environment, we might be able to see that all plants died as soon as the minimum time (trait survival assessment period) had elapsed – in all cases, we would be using statistical analysis to determine acceptable equivalence of the results, as described above.

OCExperiments states that, when the simulations are set up to replicate the same experiments (e.g. same environment and plant population, same trait and resource distributions), the results are the same – again using statistical analysis to determine equivalence.

As OCExperiments is critical to our argument, we expand the goal further to argue under the strategy of result similarity from n experiments (ArgCExp) – we could add a context here, that n represents the specific experiments conducted on the C++ simulation, as reported in the literature. Below ArgCExp, experiments are enumerated – here using the GSN version of ellipsis for brevity. We are showing that each entry in some list has been considered, and evidence produced.

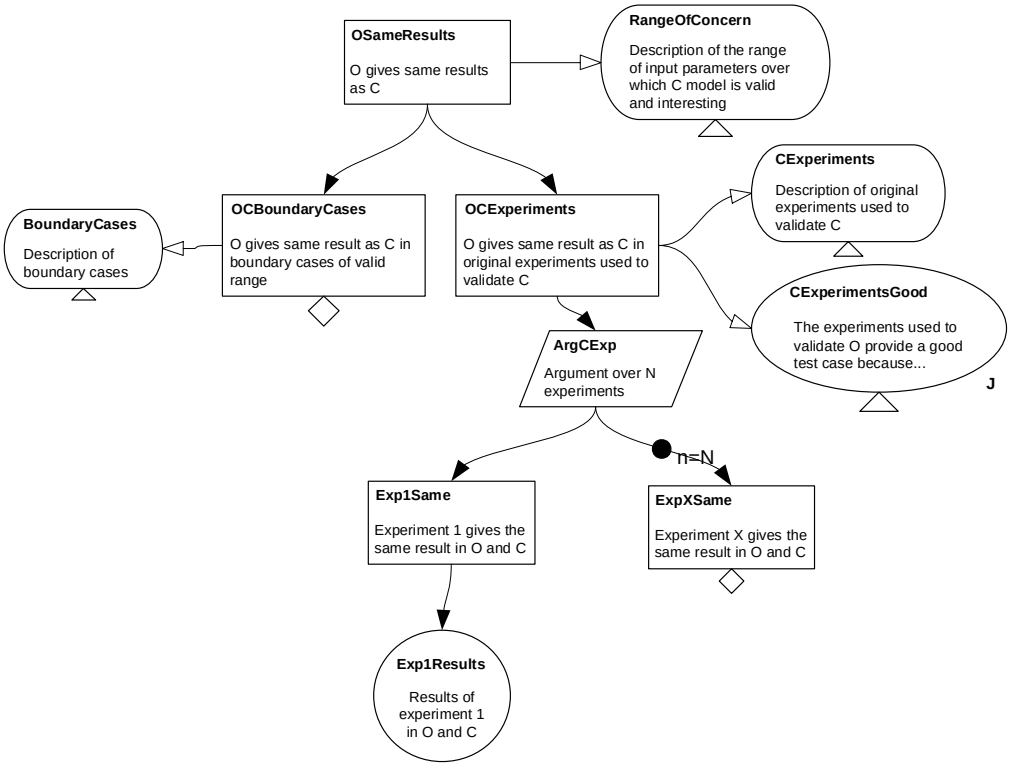

Fig. 8. Elaboration of the goal to show that the simulations produce equivalent results, from figure 5

The whole argument fragment in figure 8 is in the context of RangeOfConcern. This returns to the point made in defining *adequately equivalent* for the top goal, that there is a range over which we can produce equivalent results, and that, in this case, we can only claim that the two simulations are equivalent when performing the type and scale of experiments for which the C++ simulation was originally designed.

In figure 8, CExperimentsGood is a *justification* node – shown by a **J** next to the node. When expanded, it identifies a justification of why we can assume that OCExperiments supports OSameResults.

5 Solution Data

The previous section summarises the argument of adequate equivalence which we are making, and which we present to the scientist for review and external

Table 2. Expanding CAssumptions – some of the assumptions made in the C++ model [3], and mirrored in the occam-π simulation

Environment Assumptions	
1	The soil properties do not change radically in time.
2	The environment can be seen as a plain. Various three-dimensional landscapes will not affect the outcome.
Plant Assumptions	
3	The uptake area of a plant can be considered conic.
4	The tap root is generally more developed than the fine roots.
5	The ratio between resource allocation towards growth and towards reproduction varies slowly in time.
6	Germination takes place in no longer than one day.
7	Plants develop unhindered, if having necessary resources.
8	Plants release their resources back into their environment, when they die.
9	Plants may die of starvation or due to unpredictable events.
10	Seed dispersal happens over a short period (a matter of days).
11	Each seed requires a similar amount of resource.
12	Seeds that fall in populated areas, most often do not germinate.

scrutiny. We now consider some of the evidence, or solutions, that support the argument.

Most of our argument of adequate equivalence points to tabular comparisons. We briefly cover two of the biological aspects, but then focus on structural comparison from the implementation argument structure, which raises most of the interesting issues of equivalence. The generation of evidence for the argument of *adequately equivalent science* is, in general, more interesting, and the establishment of this argument will be essential when we extend the simulation to support further experiments on the intra-specific plant variation. However, for the argument of simulation equivalence, the science has already been captured by J. Bown in constructing the original C++ simulation (and reviewed by the scientists with whom he was working). We have essentially one source, Bown et al [3], and, throughout, we refer to an interpretation of it that is directly expressed in the C++ implementation.

5.1 Biological Assumptions

Biological assumptions were not explicitly identified in the body of work represented by Bown et al [3]. However, we have had to identify some assumptions in order to complete the re-engineering, and can use these to strengthen the argument of equivalence. Table 2 lists some of the assumptions that form the context CAssumptions in figure 6. These have been confirmed by the scientist, giving us confidence that the occam-π simulation captures the assumptions on which the C++ simulation was based.

Table 3. Expanding CAbstractions – some of the abstractions made in the C++ model [3], and mirrored in the occam-π simulation

Environment Abstractions	
1	Resource release and replenishment rates are constant.
2	The environment is 2D and each grid cell can hold only one plant.
3	The maximal level of resource is homogeneous across the environment.
Plant Abstractions	
4	Requested uptake is homogeneous with respect to the distance from the plant.
5	The uptake area has a regular shape and is not affected by neighbouring competitors roots.
6	The ratio between resource allocation towards growth and towards reproduction, does not vary in time.
7	Germination is instantaneous (takes only one time step).
8	When they die, plants release all of their resources into the environment.
9	Plants die of random events and starvation.
10	Reproduction is instantaneous (takes only one time step).
11	Each seed requires an identical amount of resource.
11	Seeds die if cells are occupied, otherwise they become plants.
12	Reproduction is clonal.

5.2 Biological Abstractions

Between biological facts and assumptions and the construction of computer simulations, we make various abstractions to map the real world into the computational one. The abstractions are influenced by the platform on which the computer simulation is built, as well as subjective factors. To expand the CAbstractions context in figure 6, we collect the abstractions made by Bown et al [3], some of which are listed in table 3). We then checked that the occam-π simulation respects each of these abstractions.

5.3 Algorithm Mapping

To compare the algorithms of the two simulations, a sub-goal of OCodeStructures in figure 7, we present pseudo-code summaries and check subjectively for similarity. Figure 9 gives a pseudo-code overview of the two simulations, whilst figure 10 focuses on the algorithm for resource uptake. Note that the pseudo-code for the occam-π implementation is written to facilitate comparison with the C++, rather than in a way that native occam-π programmers would use.

The sequential C++ implementation has a centralised architecture. This requires loop-iteration over, for example, all instances of `location` and all `plant` individuals. Because occam-π is a parallel language, all the occam-π processes (plants, locations) could execute in parallel, shown in figure 9 as `each individual` and `each location`.

In the C++ model, a double-pass approach is used to reduce positional biases – resource uptake and usage are separated into two phases, otherwise subsequent

C++ simulation

```
instantiate locations
instantiate individuals

 for each timestep
  /* resource uptake */
  for each location
    assess resource demand
    release resources
    replenish substrate

  /* resource usage */
  for each individual
    allocate uptake
    assess death
    if not dead
      assess development
      assess reproduction
```

occam-π simulation

```
instantiate locations and servers
instantiate individuals

 while simulation_running
  /* resource uptake */
  each individual
    place resource demand
    SYNCHRONISE
  each location
    process resource demands
    replenish substrate

  /* resource usage */
  each individual
    allocate uptake
    assess death
    if not dead
      assess development
      assess reproduction
```

Fig. 9. Comparing the C++ and occam-π simulations

behaviours such as seed dispersal would take place in the order in which plants are iterated. In the occam-π simulation, *synchronisation* means that the plant processes will be blocked until all have finished sending their resource requests, when all processes will be released to proceed to resource uptake.

In reviewing the complete comparison of the high-level algorithm, we found that, in terms of semantics and results, the two implementations can be considered equivalent. The resource flow is identical; only the architecture through which it is carried out differs.

The second pseudo-code comparison, figure 10, refers to the process of resource uptake. In the C++ implementation, resource uptake is location-centric – the neighbourhood of each location is scanned for plants and the demand of each plant is calculated and stored. A normalisation process is necessary to divide the resource fairly among the plants. In the occam-π implementation, however, the process relates more closely to the biology, as each plant interacts directly with its location. The computational abstraction is, in this case, that of plants and locations interacting through a *client-server protocol* [32].

In this case, the algorithm comparison shows that, although the input and output of the algorithms is equivalent, the detail is different. To make a strong equivalence argument, we would need also to look at evidence of resource uptake behaviours through experimentation and testing.

Resource uptake (C++)
Sequential algorithm

```
Main control loop:
for each location
 select uptake_area of location
 create empty demand_list
 for each loc in uptake_area
   if loc occupied
     select occupying plant
     calculate plant's demand on
                        location
     add demand to demand_list
 normalise demand_list
 for each demand in demand_list
   select demanding plant
   add resources to plant's uptake
 replenish location's substrate
```

Resource uptake (occam-π)
Parallel algorithm

```
Plants:
for each location in uptake_area
  send resource request
SYNCHRONISE
for each location in uptake_area
  if resources released
    uptake resource

Locations:
create empty demand_list
while running
  if resource request received
    store request
  if all requests received
    normalise demand vector
    for each demand in demand_list
      select demanding plant
      send resources to plant
    reset demand_list
    replenish substrate
```

Fig. 10. Comparing resource uptake algorithms for the C++ and occam-π simulations

5.4 Data Mapping

The second sub-goal of OCodeStructures in figure 7 concerns the argument of adequate equivalence of data representations. We can explore this similarity starting from a class diagram of the C++ implementation, figure 11, and a similar diagram of the occam-π processes and channels, figure 12.

UML provides an object-oriented modelling notation which is well-adapted to expressing the class structure of C++, but the notation of figure 12 is just an *ad hoc* representation of occam-π processes and channels. However, informally, we can compare data types between the two diagrams. As in earlier argument fragments, we present the evidence at this level for review; we only need to elaborate the comparison if the scientist is not prepared to accept it.

The C++ implementation of Bown et al [3] uses the class Location to represent represents cells of the environment. Each location contains a resource substrate, of class Substrate, and a plant individual, of class Plant. Because plants do not move, a Location instance is represented as being composed of one Plant instance and one Substrate instance. The Location attribute, `occupied` takes the value `1` if there is a plant growing at a location, and `0` when a location is empty – in the C++ an unoccupied location is associated to an "empty" plant instance,

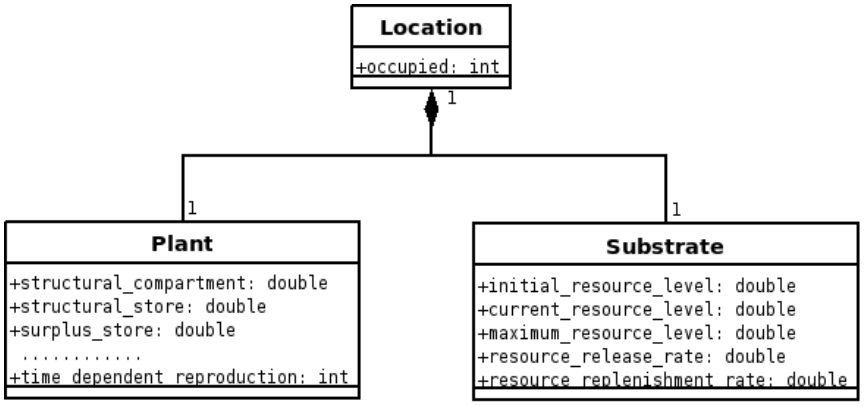

Fig. 11. Class diagram (UML notation) of classes in the C++ simulation

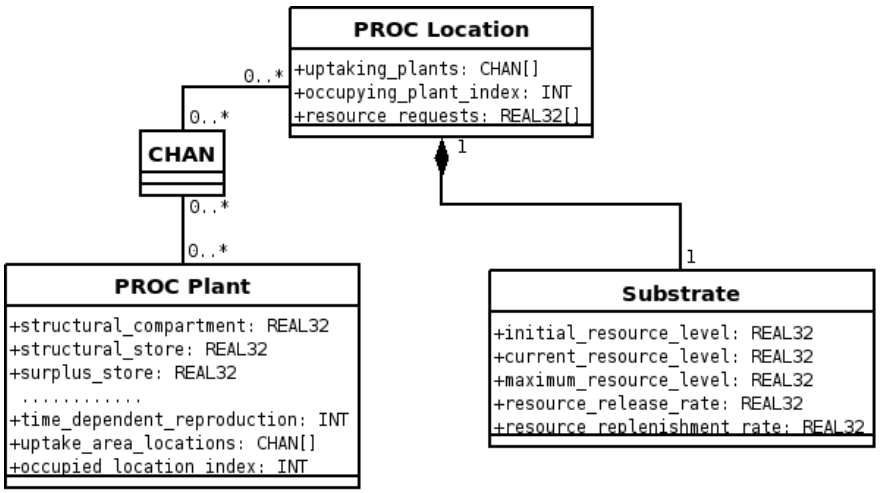

Fig. 12. Processes and channels in the occam-π implementation (notation undefined)

rather than to no plant instance. The diagram does not show the relationship between a location and the plants that are taking up resource from it, as the C++ implementation calculates this from the plant traits and location at run-time.

In the occam-π implementation, a similar form is used for the location substrate, but occam-π supports more flexible data structuring for locations and plants. These are dynamic processes (the occam-π `PROC` structure), which interact through channels (the occam-π `CHAN` structure). The relation between plants and locations is implemented through explicit channel communication.

The channel ends held by each plant process can be connected to the corresponding channel ends in any location process.

Comparing the two data structure implementations, we can observe differences in terms of attributes and their data types, the nature of plants, locations and their relationship. By reference to the biological model that these represent, we could declare ourselves adequately confident that these implementations represent implementations of the same abstract model. However, there are some subtleties that may present problems, such as the subtle quantitative effects of internal data formats: the C++ implementation uses the type double while the occam-π one uses REAL32. The two differ in terms of precision, double being represented on 64 bits, while REAL32 on 32 bits. Again, we need to check the effect of this difference through appropriate experimentation and testing – at this stage, we do not believe that the difference in precision qualitatively affects the simulations results, but we may need more evidence to convince the scientist.

6 Discussion

In this paper, we present a summary of an argument of adequate equivalence between an existing C++ simulation and a re-engineered version in occam-π. If we can assume that the original simulation is valid, then establishing the equivalence of the occam-π version would imply its validity in the same context and for the same purposes as the original simulation (see [21]). We used GSN to visualise the argument structure: those faced with evaluating our argument can immediately see the basis of the belief that the two models are adequately equivalent, and can challenge areas that they do not consider to be sufficiently supported by evidence.

This work is part of the CoSMoS project[2], which is developing a general framework for the simulation of complex systems. Part of this framework concerns the routine collection of assumptions – about the domain, the design, and the implementation. In the CoSMoS context, just the exposure of assumptions has led to scientific acceptance of some of our experimental simulations [1]. The work presented in this paper is a first step towards producing guidance and techniques for systematising argumentation relating to simulation development. However, turning assumptions into evidence for arguments that a simulation is a *valid imitation of the real world*, for a given scientific purpose, is a non-trivial activity, which is the subject of ongoing research.

Computer scientists who have spent a career in the deterministic world of the digital computer are often sceptical about the value of arguments of validity, safety etc. However, in simulating complex systems for scientific study, we are not seeking to model or implement traditional deterministic computer systems. A simulation that reduces the interacting complex systems of the real world to a deterministic system is unlikely to be adequate for the areas of scientific research that we seek to support.

[2] http://www.cosmos-research.org

Our work, here and in the CoSMoS project, also signals a departure from the common form of computer applications, in that our simulations are designed to support specific domain models – a particular expert's view of a particular scientific context. The aim of the simulation is to support those areas of scientific experimentation that rely on that specific scientific context. If the scientist wishes to extend or adjust the context, then the simulation models must be extended or adjusted, and the adequate equivalence re-established. Later revisions can be facilitated through the careful recording of the argument of equivalence or validity for each simulation; if a preceding simulation was already acceptable, scientifically, and a new simulation corresponds to that simulation for part of its range, then we need concentrate only on what has changed.

This brings us to the use of occam-π in the re-engineered simulation presented here. To support the need to extend or adjust simulations in line with scientists' requirements, we need flexible implementations. In CoSMoS, we have used a range of implementation languages, and, although occam-π does not have the mature support of languages like C++ and Java, we have found that applications written in occam-π are easy to adapt and re-use. The CoSMoS project is assembling concrete evidence of this assertion, as well as seeking to improve the maturity of the occam-π programming environment.

7 Future Work

In relation to the specific example presented here, we need to complete the argument of adequate equivalence, and expose it all to the critical review of our scientist and his colleagues. Our next step is then to use the occam-π implementation to scale up the original experiments, which will improve the quality of the scientific evidence we can provide. We will then produce a series of modified simulations to support other experiments on intra-species and inter-species plant ecology, in collaboration with Bown's group.

In relation to the CoSMoS project, the work is contributing to the body of evidence on use and suitability of occam-π for developing flexible, validated simulations to support scientific work. The argumentation processes will form part of the CoSMoS framework for complex systems modelling and simulation – we continue to review SCS work for guidance in analysis, evidence collection and management, argument construction and validation. We plan to provide specific guidance on producing GSN type arguments in the context of complex systems simulation. In addition, we are applying the activities outlined here in a range of other case studies including various scientific studies of the immune system and work on swarm robotics.

8 Acknowledgements

The work of Teodor Ghetiu, Paul Andrews and Fiona Polack is supported by the CoSMoS project, EPSRC grants EP/E053505/1 and EP/E049419/1.

References

1. Andrews, P., Polack, F., Sampson, A., J, T., Scott, L., Coles, M.: Simulating biology: towards understanding what the simulation shows. In: Proceedings of the 2008 Workshop on Complex Systems Modelling and Simulation, York, UK, September 2008, Luniver Press (2008) 93–123
2. Polack, F.A.C., Andrews, P.S., Sampson, A.T.: The engineering of concurrent simulations of complex systems. In: CEC 2009. (2009) 217224
3. Bown, J.L., Pachepsky, E., Eberst, A., Bausenwein, U., Millard, P., Squire, G.R., Crawford, J.W.: Consequences of intraspecific variation for the structure and function of ecological communities: Part 1. model development and predicted patterns of diversity. Ecological Modelling **207**(2-4) (2007) 264–276
4. Polack, F., Stepney, S., Turner, H., Welch, P., Barnes, F.: An architecture for modelling emergence in CA-like systems. In: ECAL. Volume 3630 of LNAI., Springer (2005) 433–442
5. Bonnici, E., Welch, P.H.: Mobile processes, mobile channels and dynamic systems. In: 2009 IEEE Congress on Evolutionary Computation (CEC 2009), IEEE Press (2009) 232–239
6. Sampson, A.T., Bjorndalen, J.M., Andrews, P.S.: Birds on the wall: Distributing a process-oriented simulation. In: 2009 IEEE Congress on Evolutionary Computation (CEC 2009), IEEE Press (2009) 225–231
7. Ritson, C.G., Welch, P.H.: A process-oriented architecture for complex system modelling. In Mcewan, A.A., Schneider, S., Ifill, W., Welch, P., eds.: Communicating Process Architectures 2007. Volume 65 of Concurrent Systems Engineering Series., Amsterdam, The Netherlands, IOS Press (2007) 249–266
8. Turner, H., Stepney, S., Polack, F.: Rule migration: Exploring a design framework for emergence. International Journal of Unconventional Computing **3**(1) (2007) 49–66
9. Ritson, C.G., Sampson, A.T., Barnes, F.R.M.: Multicore Scheduling for Lightweight Communicating Processes. In Field, J., Vasconcelos, V.T., eds.: Coordination Models and Languages, 11th International Conference, COORDINATION 2009, Lisboa, Portugal, June 9-12, 2009. Proceedings. Volume 5521 of Lecture Notes in Computer Science., Springer (2009) 163–183
10. Polack, F.A.C., Hoverd, T., Sampson, A.T., Stepney, S., Timmis, J.: Complex systems models: Engineering simulations. In: ALife XI, MIT press (2008) 482–489
11. Miller, G.F.: 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)
12. Epstein, J.M.: Agent-based computational models and generative social science. Complexity **4**(5) (1999) 41–60
13. Paolo, E.D., Noble, J., Bullock, S.: Simulation models as opaque thought experiments. In: Artificial Life VII, MIT Press (2000) 497–506
14. Wheeler, M., Bullock, S., Paolo, E.D., Noble, J., Bedau, M., Husbands, P., Kirby, S., Seth, A.: The view from elsewhere: Perspectives on alife modelling. Artificial Life **8**(1) (2002) 87–100
15. Bryden, J., Noble, J.: Computational modelling, explicit mathematical treatments, and scientific explanation. In: Artificial Life X, MIT Press (2006) 520–526
16. Kelly, T.P.: Arguing safety – a systematic approach to managing safety cases. PhD thesis, Department of Computer Science, University of York (1999) YCST 99/05.

17. Alexander, R., Alexander-Bown, R., Kelly, T.: Engineering safety-critical complex systems. In: Proceedings of the 2008 Workshop on Complex Systems Modelling and Simulation, York, UK, September 2008, Luniver Press (2008) 33–62
18. Leveson, N.: High-pressure steam engines and computer software. IEEE Computer (1994)
19. Wilson, S.P., McDermid, J.A., Pygott, C.H., Tombs, D.J.: Assessing complex computer based systems using the goal structuring notation. In: 2nd Int. Conf. Engineering of Complex Computer Systems. (1996) 498–505
20. Sargent, R.G.: The use of graphical models in model validation. In: 18th Winter Simulation Conference, ACM (1986) 237–241
21. Sargent, R.G.: Verification and validation of simulation models. In: 37th Winter Simulation Conference, ACM (2005) 130–143
22. Preston, F.W.: The canonical distribution of commonness and rarity. Ecology **43**(3) (1962) 185–215, 410–432
23. Begon, M., Townsend, C.R., Harper, J.L.: Ecology: From individuals to ecosystems. Fourth edn. Blackwell Publishing (2006)
24. Lavorel, S., Garnier, E.: Predicting changes in community composition and ecosystem functioning from plant traits: revisiting the holy grail. Functional Ecology **16**(5) (2002) 545–556
25. Tilman, D.: Causes, consequences and ethics of biodiversity. Nature **405**(6783) (2000) 208–211
26. Reineking, B., Veste, M., Wissel, C., Huth, A.: Environmental variability and allocation trade-offs maintain species diversity in a process-based model of succulent plant communities. Ecological Modelling **199**(4) (2006) 486–504
27. Marks, C., Lechowicz, M.: A holistic tree seedling model for the investigation of functional trait diversity. Ecological Modelling **193**(3-4) (2006) 141–181
28. Squire, G.R.: The Physiology of Tropical Crop Production. Oxon (UK). C.A.B. International (1990)
29. Bausenwein, U., Millard, P., Thornton, B., Raven, J.A.: Seasonal nitrogen storage and remobilization in the forb rumex acetosa. Functional Ecology **15**(3) (2001) 370–377
30. Hoare, C.: Communicating Sequential Processes. Prentice-Hall (1985)
31. Wood, D.C., Welch, P.H.: The kent retargetable occam compiler. In: WoTUG '96: Proceedings of the 19th world occam and transputer user group technical meeting on Parallel processing developments, Amsterdam, The Netherlands, The Netherlands, IOS Press (1996) 143–166
32. Martin, J.M.R., Welch, P.H.: A Design Strategy for Deadlock-Free Concurrent Systems. Transputer Communications **3**(4) (1997) 215–232