Proceedings of the 2012 Workshop on
Complex Systems Modelling and Simulation

CoSMoS 2012

Susan Stepney, Paul S. Andrews, Mark N. Read
Editors

CoSMoS 2012

Luniver Press
2012

Published by Luniver Press
Frome BA11 6TT United Kingdom

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

CoSMoS 2012

ISBN-10: 1-905986-37-8
ISBN-13: 978-1-905986-37-8

Preface

The CoSMoS workshops series has been organised to disseminate best practice in complex systems modelling and simulation, with its genesis in the similarly-named CoSMoS research project[1], a four year EPSRC funded research project at the Universities of York and Kent in the UK. Funding for the CoSMoS project is now complete, but we have decided to continue running the workshop series as a forum for research examining all aspects of the modelling and simulation of complex systems. To allow authors the space to describe their systems in depth we put no stringent page limit on the submissions.

We are pleased to be running the fifth CoSMoS workshop as a satellite event at the 11th International Conference on Unconventional Computation and Natural Computation (UCUN 2012) at the University of Orléans, France. UCUN explores all aspects of unconventional and natural computation, an area rich in the inherent complexity within systems, providing a natural complement to the issues addressed by the CoSMoS workshop.

The main session of the workshop is based on five full paper submissions:

Andrews et al. explore the analogy between traditional scientific instruments and simulations used in scientific investigations. They highlight the need to understand the model underpinning a simulation and engineer simulations in a rigorous way to provide open and reproducible science.

Evora et al. present an approach to improve performance when simulating power grids. Their approach uses asynchronous timing to reduce unnecessary calculations and to allow such calculations to execute at a natural pace.

Garnett describes how the CoSMoS process has facilitated the transformation of an existing agent-based simulation developed to investigate plant development to one that examines the phenomena of standing ovations in audiences.

Greaves et al. apply the CoSMoS process to a social-ecological case-study. This domain raises difficult-to-model "soft element" concepts, which have been tackled at the domain modelling stage to help clarify and understand their interactions.

Stepney provides an overview of a pattern language that has been developed within the CoSMoS project to support the development of

[1] The CoSMoS project, EPSRC grants EP/E053505/1 and EP/E049419/1, http://www.cosmos-research.org

scientifically useful and credible simulations. Several pattern and antipattern examples are given.

We also invited authors to submit abstracts for discussion at the workshop. One abstract is presented in the proceedings:

Goltsov et al. explore linking scales in both modelling and visualisation by combining two existing blood clotting simulations to provide immediate visualisation of tissue-scale results to cell-scale changes.

Our thanks go to all the contributors for their hard work in getting these papers, abstracts and posters prepared and revised. All submissions received multiple reviews, and we thank the programme committee for their prompt, extensive and in-depth reviews. We would also like to extend a special thanks to the organising committee of UCUN 2012 for enabling our workshop to be co-located with this conference. We hope that readers will enjoy this set of papers, and come away with insight on the state of the art, and some understanding of current progress in complex systems modelling and simulation.

Programme Committee

Paul Andrews, University of York, UK

George Eleftherakis, CITY College, International Faculty of the University of Sheffield, Greece

Philip Garnett, Durham University, UK

Richard Greaves, University of York, UK

Colin Johnson, University of Kent, UK

Fiona Polack, University of York, UK

Mark Read, University of York, UK

Carl Ritson, University of Kent, UK

Adam Sampson, University of Abertay, Dundee, UK

Susan Stepney, University of York, UK

Table of Contents

CoSMoS 2012

x

Simulation as a Scientific Instrument

Paul S. Andrews[1], Susan Stepney[1], and Jon Timmis[1,2]

[1] Department of Computer Science, University of York, UK
[2] Department of Electronics, University of York, UK

Abstract. Computer simulation approaches are starting to be used more extensively throughout scientific investigations. Some scientists, however, are skeptical about the benefits of simulation. We present computer simulation as a scientific instrument in order to explore issues of their construction and use, which we believe might increase their acceptance within science. We highlight the need to understand the model which the simulation implements, and examine the importance of calibrating simulations and presenting them in an open way to provide scientific reproducibility.

1 Introduction

The use of predictive simulation based approaches to facilitate research in a wide range of scientific disciplines is becoming ever more prevalent in the literature. The acceptance of this trend towards the use of simulation methods, however, is by no means universal with some scientists skeptical regarding the benefits of computer simulation to scientific understanding [6]. We believe it is the responsibility of the people engaged in constructing simulations to provide evidence demonstrating why simulation results can be used to provide real insight into scientific investigations.

In this paper we examine the analogy between computer simulators and a view of scientific instruments expressed by [4] to show how we can make simulation a tool more accessible to science. Computer simulations should be subject to the same rigour that goes into constructing other kinds of scientific instrument. They need to be calibrated to understand how the outputs relate to the system under study, and they should be presented in such a way that their findings can be reproduced.

2 Computer Simulation and Science

In general, Frigg and Reiss [2] state that the term simulation "refers to the entire process of constructing, using, and justifying a model that

involves analytically intractable mathematics". Models do not, however, always involve analytically intractable mathematics. For example, simulation is often used where other approaches are intractable owing to ethical reasons (social experiments), cost, time, danger, or impossibility (galaxy formation or climate models). Often, computer simulation is the only way to greater insight into a system [4].

Simulators are built based on an underlying model that is used to represent the system or domain under investigation. Simulation runs (executing the simulator) then allow us to animate that model, exploring its temporal behaviour, hence a "simulation imitates a (usually real) process by another process" [1]. Models are used throughout science as surrogates to learn about the world, revealling features of the system the model represents. Learning takes place during both construction and manipulation of the model that underlies the simulation. In the simplest scenario, model construction results in computer code (the simulator), and model manipulation takes the form of in silico experimentation. During the former we learn about the system and gain an idea of the questions we wish to ask of it; during the latter we explore these questions and enhance our understanding of the model upon which the simulator is based.

Broadly speaking computational methods serve two purposes for scientists: brute-force/informatics approaches deal with large amounts of data or numerical calculations, for example genome sequencing; and predictive simulation aims to to explain observed natural phenomena by capturing the underlying behavioural processes. This paper is concerned with the latter, whereby simulation aims to explain real-world phenomena rather than describe it, and the model underlying the simulation provides a theory for how the components of the system interact to produce a particular outcome [8]. The results of computer simulation can be used for many complementary purposes, for example: to inform real-world experimentation on the system being investigated; to validate such experimentation; or simply to explore both concrete and abstract hypotheses.

One of the main advantages of computer simulation approaches is the complete control of the elements and parameters that make up the simulation. This allows us to explore various aspects that relate to a real-world system under study, which are otherwise difficult or even impossible to achieve. The flexibility of computational approaches, however, can have negative consequences. For instance, the increased access to computational power and reliance on computer simulations may lead to reduced levels of more expensive (but more informative) laboratory or field experiments [4], or overly complex and heavily parameterised

models containing poorly understood assumptions [1]. Humphreys [4] also warns of the problems inherent in exploratory agent-based models that aim to show how simple rules can account for complex behaviour. It is possible to use simple models to produce patterns that have little connection to the actual underlying mechanism. In such cases, care must be taken not to blindly accept the explanation given by the computer simulation.

The outputs of simulations will depend on the way in which they have been constructed. This makes understanding the construction process key to interpreting and presenting what the simulation shows. We need to understand the details of the model that the simulation implements, and show how that model then relates to the simulation outputs in order that the outputs can be properly interpreted. This issue is true of most computational approaches in science; for example, Nyce [7] describes how to some radiographers, traditional x-ray images are not representations of the underlying biological structures, but are "very much the same kind of 'thing' " that do not need to be interpreted. This contrasts with their relationship with digital techniques, where a level of mistrust exists because the resultant images are produced via unknown machine operations. This produces a perceived distance between the digital images and the thing they represent.

To increase trust and confidence in computer simulations further, their outputs need to be reproducible. Reproducibility is a key axiom of science, and Timmer [11] reports that the increased reliance on computational methods in most areas of science has lead to an inadvertent loss of scientific reproducibility. The examples given by Timmer [11] for the apparent loss of reproducibility focus on the more data intensive aspects of using computers to analyse large quantities of data. However, many of the issues raised are equally applicable to the simulation techniques used in the Alife community. These issues include: a complex mix of data from both public and internally generated; data that is often quick to become out of date; a complex pipeline of software programs used for analysis, etc; multiple sets of parameters for each piece of software; different software versions; and software bugs.

3 Simulation as a Scientific Instrument

We can consider computer simulation in the same terms as other tools or instruments used by scientists in their scientific endeavours. Given this, such simulations should be subject to the same rigour that goes into constructing other kinds of scientific instrument. They need to be calibrated to understand how the outputs relate to the system under

study, and they should be presented in such a way that their findings can be reproduced. We present here a discussion that aims to show that computer simulations fall within a spectrum of instruments used every day by the scientific community.

3.1 Scientific Instruments

The description presented throughout this section has been drawn from the analysis of Humphreys [4], who examines the role of computational models (including simulation) in science.

The role of scientific instruments is to enhance the range of our natural human abilities, such as our perceptual and mathematical capacities. In the case of mathematics, computational devices can be used to move beyond what is accessible naturally to the human brain, such as the number of calculations performed, which is many orders of magnitude greater on a computer.

Instruments of all types have been used for hundreds of years throughout science. These range from everyday instruments such as bench microscopes and optical telescopes to specialised medical imaging equipment. Many modern day instruments incorporate explicit computational approaches in addition to physical detection to provide enhancements. For example, magnetic resonance imaging (MRI) and computerised axial tomography (CAT) scanners contain various physical devices to measure nuclear spin and radiation respectively and then use computer algorithms to transform these readings into two- and three-dimensional images.

One thing in common with all instruments is that they are calibrated to produce outputs that are directly accessible to the human observer. The process of calibration relies on correctly observing and reproducing the structure of known features measured by the instrument. This affords confidence in using the instrument, establishing the scope of its usage along with its accuracy, precision and resolution.

We can often take for granted familiar instruments such as microscopes and telescopes, which are the product of many years of testing, refinement and adjustment. Because of this, the user of such instruments does not need to know the precise details of the theory behind how it works as it has been deliberately designed to be used without need for that knowledge. However, when dealing with contemporary research-level instruments, the user needs to understand the instrument in much greater detail owing to their complexity and the lack of many years of refinement. This type of instrument may routinely malfunction and produce spurious data or need close attention to work in the desired way. Knowing how the instrument works should help reduce the occurrence of

unwanted artefacts increasing the instrument's stability and highlighting situations in which malfunctions take place. Another related issue is knowing how to interpret the outputs of the instrument. It is these issues that we tackle when calibrating an instrument. The main benefit of knowing how instruments work is when they provide unexpected outputs, whether this be because something has gone wrong or not. Instrumental knowledge should tell us when we have gone outside the domain of application of the instrument, and how this can be corrected.

3.2 A Spectrum of Instruments

Based on the previous description of scientific instruments, we consider computer simulation as a technique that allows us to develop bespoke scientific instruments. Once engineered, scientific instruments are applied to some object/system of study, which we call the domain. Here we explore how computer simulations relate to their domain of study in the context of other types of scientific instrument. The purpose is to show how the inputs and outputs of a simulation instrument might be applied to understand its intended domain. Conceptually, an instrument takes some form of observation as input from a domain and transforms it based on a model of understanding that has been encoded into the instrument during its construction. The output of the instrument is then presented to a human observer, with the aim of extending the understanding of the domain. Consider three different examples of scientific instruments within the setting just described:

Optical telescope: light is passively received as a direct physical input from the domain (the object that the telescope is directed towards). Lenses are used to refract the light, which is emitted as the output so that the domain appears magnified to the human observer.

MRI scanner: a physical input from the domain is achieved through manipulation. A magnetic field is used to line up proton spins of hydrogen atoms of the intended domain and a radio signal used to disrupt this and measurements are made of the distortion. Computer algorithms are then used to transform the measurements to create an image of the domain that is displayed to the human observer.

Predictive computer simulation: no physical input is received directly from the domain, but a set of derived starting conditions for computational agents is represented within the simulation. These starting conditions then drive the dynamics of the computational model encoded within the simulation, and may be subject to further inputs. A representation of the model is output to a human observer over a period of time.

Whilst each of the three instruments just described attempts to help us understand the domain of study, the relationship with that domain differs. Even though all three instruments are based on models of domain understanding, the way in which these models are encoded differs. For example, it might be within the physical components of the instrument and/or within a computational model. Computer simulations are at the far end of this spectrum, based purely on a computational model with no direct domain understanding specifically encoded within their hardware.

The way in which inputs are received from the domain also differs with scientific instruments. As we move from instruments such as the optical telescope to an MRI scanner, the domain input changes from a passive observation to requiring a direct perturbation of the domain in order to measure its effects. When we move to computer simulation, this domain input becomes far more indirect with regard to space and time in the sense that no direct physical input is present. In this case, we rely more on logical connections to the domain rather than direct physical inputs. This results in an added layer of interpretation required to understand how the inputs of computer simulations map to the entities of the domain under investigation.

In summary, even though different instruments are fulfilling the same role of investigating a domain of study, the way in which instruments interact with that domain can differ immensely. Understanding the relationship between an instrument and the domain it measures is vital to interpreting its output. This is especially true of computer simulation instruments that fall at the far end of a spectrum of instruments, with a reliance on computational models and an indirect relationship to their domain of study.

4 Calibration

We have discussed in the previous section that computer simulations can be viewed as scientific instruments. It follows that simulations should be subject to the same rigorous process of construction as other scientific instruments, generating an understand of how the model upon which the simulation instrument is based relates its inputs to its outputs. In order to help achieve this, the simulation needs to be calibrated.

Construction of a computer simulation, and any other kind of scientific instrument, relies on both processes of science and engineering. Science is employed in the development of the model upon which the instrument will be based. Engineering is then used to implement this model resulting in the construction of an instrument suitable for the purpose of scientific investigation. Before the instrument can be used,

however, it should be subjected to the process of calibration. As previously mentioned, calibration involves establishing the relationship between the output of an instrument (across a range of operating conditions) and the system under observation; it lets us interpret what the simulation is showing us.

For instruments such as microscopes and telescopes, Humphreys [4] tells us that the calibration process relies first on correctly observing and reproducing the structure of features that are already known. For computational devices, calibration standards often include reproducing analytically derived reference points. One problem is that due to the complexity of calculations, access to results independent of the simulation is often impossible, thus comparison with existing (instrumental) techniques is required [4].

It is often the case with predictive computer simulation that we do not have access to the types of data typically used to calibrate (for example a set of reference standards). In this case calibration can only be achieved by comparing simulation observations with predictions from a pre-existing and explicitly stated model that formed the basis for simulation construction. One example would be in the case of emergent properties such as flocking. Our model might predict that flocking is the result of a combination of certain agent behaviours. These behaviours would be encoded within the simulation, and then calibration would ascertain whether or not the flocking behaviours are perceived in the actual simulation. This emphasises that need to understand and identify what the underlying model of a simulation is actually a model of.

A further problem with calibration and computer simulation is that all simulators are essentially different specialised instruments that have been constructed to answer a specific question or set of questions. The purpose of these instruments is more often than not different, therefore calibration is going to be different for each individual simulator. When changes are made to the simulator (no matter how small), it may have to be re-calibrated depending on how the change affects the encoded behaviours.

5 Openness

Previously we highlighted the need for the results generated by computer simulations to be reproducible. Whilst proper calibration would be a first step towards this, we need to be open about further aspects of the simulation. Timmer [11] describes the beginning of a movement towards researchers adopting approaches to ensure that computational tools are in line with existing scientific methods. However, whilst there

may be a recognition that nearly everyone doing science uses some form of computation, there are few who know what is needed to make sure that documentation of approaches is sufficient for reproducibility.

It is probably intractable to expect complete reproducibility of a piece of science performed using computer simulation without access to the exact piece of computer code and all the initialising variables (parameter settings, initial states of data). This gives support to the argument for complete openness of code, an emotive issue amongst many who develop computational tools. This can only be tackled by a sea-change within research communities to require this level of openness.

There is, however, a more subtle and no less important source of knowledge that should also be open. We have previously mentioned that all simulations encode a model. This model is often only explicitly expressed as the computer code and is the process of much work. It contains many different assumptions which are vital to understanding what the model represents. This is an issue of validation (see [10]): how do you know that you have built the right system to answer the questions you are exploring? This is hard to express as a yes/no answer and it typically expressed as a level of confidence. In some circumstances, for example where outputs of a simulation instrument have a high level of criticality, that a structured argument is required to express confidence in a computer simulation (see [9] for a more in depth discussion).

6 Conclusion

There are many reasons why scientific instruments based on computer simulation might not be accepted for use in scientific investigations. In this paper we have suggested that if we can show how simulations relate to more traditional scientific instruments, and highlight some important issues regarding how they might be constructed and presented, that we may stand a greater chance of simulation approaches becoming a useful instrument for science. It is also important to emphasise that simulation does not replace direct experimentation; simulation is a tool to assist more traditional approaches. Simulation should be part of the scientists toolkit and used where it is appropriate.

Humphreys [4] argues that when we use new instruments we need to understand how they are built. Over time they can become more generally accepted: in the case of simulation the 'acceptance' is going to be the acceptance of a class of predictive software artefacts and development strategies rather than of an instance of one. This general acceptance is only going to come from an increased number of instance acceptances.

In summary, we can consider general purpose computers as physical instruments that can be used to construct a wide variety of logical instruments in the form of simulations. However, as previously discussed, the benefit of computational approaches conceals its drawback, with many unknowns liable to populate the simulation. In the aftermath of 'Climategate' [3], there should be greater scrutiny on the way in which scientists use computational devices as part of their scientific process [5]. There have been calls for open-source code to enable repeatability. In our view, this is not enough: we need to perform calibration and present the results of this calibration to provide us with the knowledge to decode the output of simulation and interpret in the context of the real domain being modelled and simulated. We need to show how the simulation has been engineered and why it is a good instrument to enhance our domain knowledge.

7 Acknowledgements

This work is part of the CoSMoS project (www.cosmos-research.org), funded by EPSRC grants EP/E053505/1 and EP/E049419/1.

References

[1] R. Frigg and S. Hartmann. Models in science. In E. Zalta, editor, *Stanford Encyclopedia of Philosophy*. 2006.
[2] Roman Frigg and Julian Reiss. The philosophy of simulation: hot new issues or same old stew? *Synthese*, 169:593–613, 2009.
[3] Roger Harrabin. Climate science must be more open, say MPs. 2010. http://news.bbc.co.uk/1/hi/sci/tech/8595483.stm.
[4] Paul Humphreys. *Extending Ourselves: Computational Science, Empiricism, and Scientific Method*. Oxford University Press, New York, 2004.
[5] Darrel C. Ince, Leslie Hatton, and John Graham-Cumming. The case for open computer programs. *Nature*, (482):485–488, 2012.
[6] Lucas Laursen. Biological logic. *Nature*, 462(26):408–410, 2009.
[7] James M. Nyce. Artifice, interpretation and nature: Key categories in radiology work. In *Unconventional Computation, 8th International Conference, UC 2009*, volume 5715 of *Lecture Notes in Computer Science*, pages 11–15, 2009.
[8] Steven L. Peck. Simulation as experiment: a philosophical reassessment for biological modeling. *Trends in Ecology and Evolution*, 19(10):530–534, 2004.
[9] Fiona A.C. Polack, Tim Hoverd, Adam T. Sampson, Susan Stepney, and Jon Timmis. Complex systems models: engineering simulations. In S. Bullock, J. Noble, R. Watson, and M. A. Bedau, editors, *Eleventh International Conference on the Simulation and Synthesis of Living Systems*, pages 482–489. MIT Press, Cambridge, MA, 2008.

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

[11] John Timmer. Keeping computers from ending science's reproducibility. 2010. http://arstechnica.com/science/news/2010/01/keeping-computers-from-ending-sciences-reproducibility.ars.

Asynchronous Smart Grid Simulations

Jose Evora[1], Jose Juan Hernandez[1], Mario Hernandez[1], and Enrique Kremers[2]

[1] SIANI, University of Las Palmas de Gran Canaria, Las Palmas, Spain,
jose.evora@siani.es, josejuanhernandez@siani.es,
mhernandez@siani.es
[2] EIFER, European Institute for Energy Research (KIT & EDF), Karlsruhe, Germany,
enrique.kremers@eifer.org

Abstract. A shift from traditional power grids to future smart grids requires a different approach to the analysis of power grid systems. In the smart grid conception, the system is analysed in a dis-aggregated manner through simulations. Many objects and relationships must be considered for a complex system to be eventually modelled and simulated. Usually, the simulation is performed by synchronising calculations associated with objects. The main problem of this approach is that every calculation has to be executed at the same speed in spite of objects not requiring an update with the same frequency. So, lots of unnecessary calculations are done, making performance worse. With the objective of improving the simulation performance, in this paper a new approach for simulating power grids based on asynchronous timing is presented. This approach is orientated towards allowing calculations to be executed at their own pace.

1 Introduction

The climate change and liberalisation of markets are pushing the energy sector towards a new paradigm known as the smart grid. This paradigm is characterised by the introduction in the power grids of renewable energy sources (RES), new technologies such as storage mechanisms, massive integration of sensors and decision makers distributed along the grid. There is also a trend towards the introduction of a communication layer for the management and control of these technologies. The smart grid paradigm is also based on the use of the Demand Side Management (DSM) whose objectives include the minimisation of the peak demand and the system operation and planning improvement [10]. The system

complexity is therefore increased and new tools are needed for the analysis and design of smart grids.

Traditionally, simulators have been an essential tool for analysing and designing power grid systems. Many simulation tools have been developed for this purpose: UWPFLOW [8], TEFTS [6], MatPower [18], VST [15], PSAT [13], InterPSS [17], AMES [1], DCOPFJ [2], Pylon [4], and OpenDSS [3]. However, these tools are limited to simulating smart grids specific issues, like a communication system integrated in a large-scale simulation. GridSim [7] was developed to deal with these problems. GridSim is a modified version of TSAT [5] (an industry-proven transient stability simulator) which addresses the electro-mechanic working mode of the power grid system. GridSim is a real-time simulator adapted to integrate sensing with a high data rate. The modelling approach of these tools manage the production and demand in an aggregated manner.

However, smart grid simulations require the representation of both demand and production in a dis-aggregated manner. Tafat is a tool able to simulate smart grids that enables a bottom-up representation which includes, not only a technical system description, but also a sociological description of people interacting with the system [9]. With this representation, it is possible to design, implement and test smart grid simulations. All these tools execute the simulation with a synchronised approach. Synchronous simulations have the advantage of simple time management as all objects of the modelled system are running in the same time instant. It forces objects to always perform calculations, in every time step. Sometimes these calculations are unnecessary due to the fact they cannot provide new results. For example, a washing machine is usually waiting for an agent to be turned on, considering this as an event. Later on, it develops some washing cycles where the power may vary along the time. Whenever the washing machine state does not change, calculations could be avoided.

In this paper, it is proposed that an asynchronous simulation approach is included in Tafat which would allow objects to develop their own time as desired. They could behave both event and time-based according to their nature. Furthermore, they could use variable steps from one calculation to another. For example, in an asynchronous simulation where the power consumption of a washing machine is analysed, calculations would be done only when the washing machine state changes. The advantage with respect to a synchronous simulation is clear since in the synchronised case, calculations are done every time step.

In the context of discrete event simulation the asynchronous concept has dual connotation. One of them consists in variable time-increment procedures as opposed to a ˝synchronous˝ or fixed time-increment pro-

cedures for simulation control. This connotation is related to the known concept Distributed Discrete Event Simulations (DDES) [12, 14]. For instance, Simula [16], a simulation-oriented programming language, is based on this asynchrony concept where the time-management is mainly event-based. This kind of asynchrony was already considered in Tafat through using different time steps for each mode of behaviour [9]. On the other hand, the asynchrony can be understood as a non-sequential processing where simulation parts may not be executed in the proper temporal order. That is to say, later parts of the simulation may be executed before previous ones [11]. The last connotation is the one to which we subscribe in this paper. The objective is to apply the time-management to each model element allowing them to be in different time instants.

2 Tafat asynchronous simulation

In initial Tafat framework releases, the simulation of power grids was done following a synchronous timing approach. This paper examines a new approach to achieve asynchronous simulations with Tafat. This section introduces the concepts and constructions that Tafat architecture includes to model power grids. Theses constructions are focused on dependencies between objects that are massive and very relevant in a complex system simulation. In order to properly handle an asynchronous simulation, it is important to understand the dynamics of coupled objects. For the sake of clarity, a traced execution of objects interaction during an asynchronous simulation is demonstrated.

2.1 Tafat system modelling

Modelling in Tafat is done by developing two views: an object oriented description of the scenario, and a behavioural specification of these objects. The first view is the static representation of the real world objects, where each single object is described with features (static attributes) and variables (dynamic attributes). This representation also includes the specification of object relations. The second view focuses on objects' dynamic : that is, how objects should behave, emulating the way they act in the real world (behaviour). A single object can be associated with several behaviours. These associated behaviours are responsible for modifying the model object variables along the time. Object variables are encapsulated and can be only accessed and modified by their associated behaviours.

The solution of separating objects from their behaviours, makes it straightforward to change the method for calculating variables. In this way, it is possible to simulate different behavioural aspects with the same representation.

For example, a washing machine representation contains:

1. Static View
   - The description of their features such as capacity, installed power and energy labelling, and their variables such as mode (on, off), active programme (temperature, cycle, timeout...), and active power.
   - The topological relation to the electrical installation in a household
2. Dynamic View
   - The specification of the washing machine-operating mode. The behaviour is then associated with this model object.

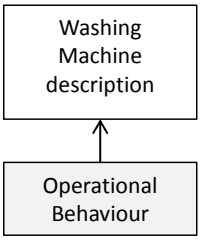


Fig. 1. The operational behaviour of a Washing Machine is associated with the Washing Machine object description

Normally, a behaviour is coupled with other objects, both for querying their states or sending messages in order to change their states. In the Tafat model representation, defining behaviour which interacts with other objects is allowed.

This representation approach consists of interfaces that should be defined in the object which could be externally accessed. In Tafat, there are two types of interfaces:

1. event interfaces that handle messages and are responsible for modifying the object internal variables as requested, and
2. data interfaces that handle queries and provide the value of requested attributes

An example of these types of interfaces is shown in the figure 2. On the one hand, the thermal behaviour within a household has a data dependence with the temperature of the surrounding Outdoor. In this case, the Outdoor temperature data is requested by the associated object through the outdoor data interface. On the other hand, an agent sociological behaviour wants to turn on the washing machine. Then, this sociological agent must use the washing machine event interface to achieve this task. The washing machine event interface would change the washing machine mode to "ON". The washing machine operational behaviour would calculate the proper power consumption based on this mode. Later on, when the cycles end, the operational behaviour turns off the washing machine.

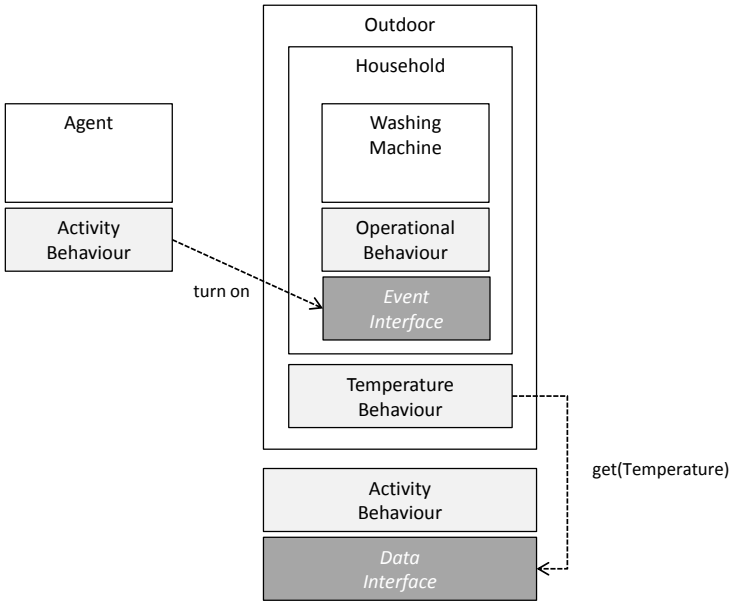


Fig. 2. Dependencies examples between objects

2.2 A power grid simulation case

In order to consider the main issues that involve asynchronous simulation a simulation case is proposed to show how objects interact when working in different times (Figure: 3).

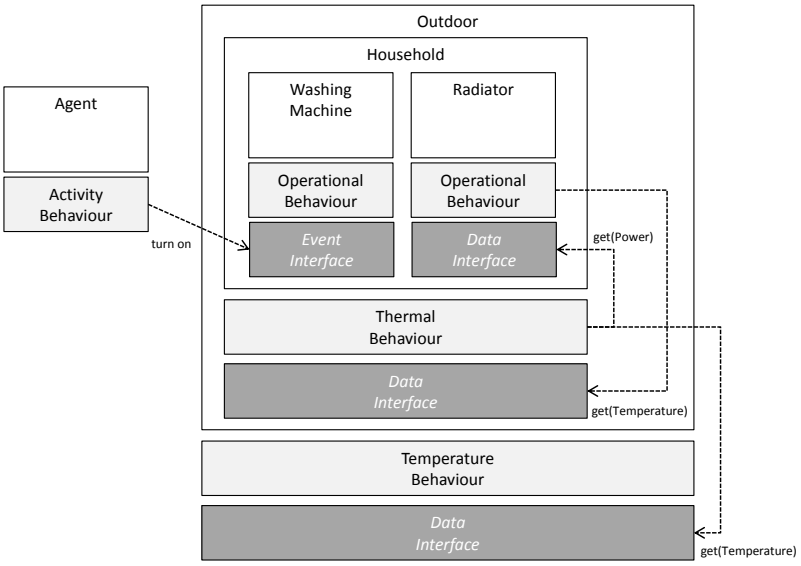


Fig. 3. Model composition

The objects within this simulation case are an Outdoor, a Household, a Washing Machine and a Radiator.

- The Outdoor is the object that represents environmental conditions, in this case, the temperature. The Outdoor temperature behaviour is responsible for setting the temperature which can be loaded from an external database.
- The Household works as a container of the appliances of a household, a Washing Machine and Radiator in this case. The Household Behaviour is concerned with the thermal dynamics inside the household.
- The Electrical devices inside the Household are a Radiator and a Washing Machine. These devices are handled by an Agent.
- Finally, the Agent represents the people living in the Household and the associated behaviour defines the actions that these people are performing. For example: a person turning on the Washing Machine.

The coupling in this model is represented by the dotted lines in the figure 3. This coupling is always defined from behaviours to interfaces. The Agent depends on the Washing Machine to change the operation

mode of this device. The Radiator depends on the Household temperature, since the heat radiation is calculated based on the gap between the Radiator reference temperature and the Household temperature. The Household has two dependencies: with the Outdoor temperature and with the Radiator power, since the Household temperature is calculated by a numerical solution of a differential equation which includes these two variables. Note that, in this case, there is a cyclic dependence between the Household and the Radiator.

2.3 Asynchronous simulation dynamics

A system simulation requires time-management to ensure that temporal aspects are correctly represented and emulated. This temporal representation only exists during the simulation process and is referred to as "Simulation Time". Simulation Time is represented as a timestamp, a long integer where a unit corresponds to a millisecond of real time.

The time-management in a synchronous simulation is centralised while the time-management in an asynchronous simulation is distributed. That is, an asynchronous simulation involves that every object manages its time, so they could have different timestamps (Figure: 4).

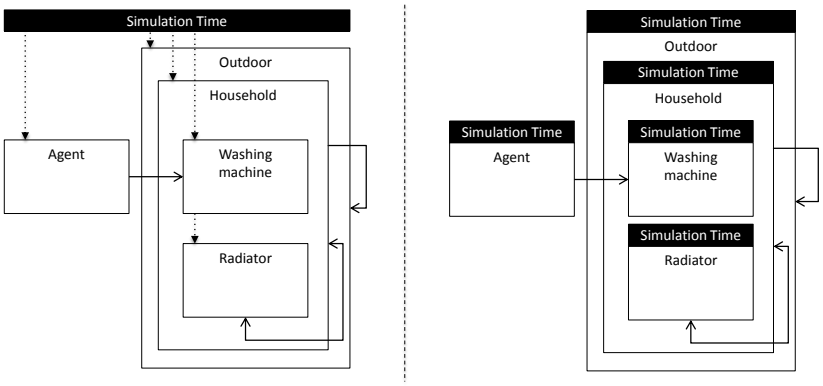


Fig. 4. Synchronous vs Asynchronous simulation

In this simulation paradigm, when an object is not coupled with other objects, its Simulation Time develops without considering other object Simulation Times. In this simulation case, the Outdoor is completely independent of other objects.

However, when objects are coupled, the challenge consists of correctly reproducing temporal relationships. The identified temporal relationships are as follows:

1. Coupling with a data interface
2. Cyclic coupling with data interfaces
3. Coupling with an event interface

In the following sections these relationships are discussed.

Coupling with a data interface Since an object could access a variable of an external object which may be in a different time instant, every object must keep the different states that have been calculated during the simulation execution. So, when a variable is modified, a state snapshot is created in order to keep the object state in this time instant.

If an object is querying for a variable value in a time instant t_i, there are two cases: the object Simulation Time is delayed or ahead with respect to the external object Simulation Time. In the first case, the external object is able to provide the value by retrieving the last snapshot previous to this time instant (t_i). I n the second case, the dependent object must wait until the external object reaches this time instant (t_i).

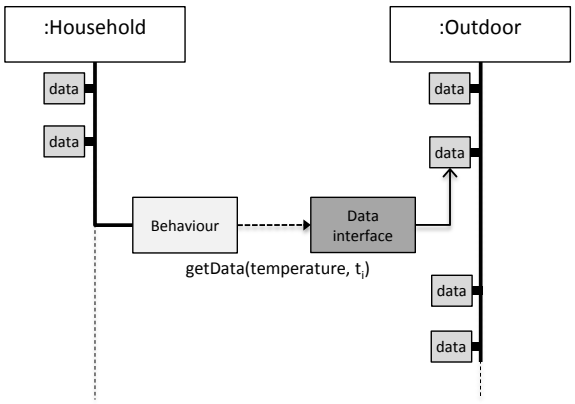


Fig. 5. Household requires the external variable temperature from the Outdoor. Note that the time is vertically represented

In the figure 5, the first case is shown. The Household Simulation Time is t_i and the Outdoor Simulation Time is t_j. Whenever t_i is lesser

or equal than t_j, the requested data can be delivered since the data has already been calculated and stored.

However, when the Household Simulation Time (t_i) is greater than the Outdoor Simulation Time (t_j), the Household behaviour is blocked (Figure: 6) until t_j is greater or equal than t_i (Figure: 7) delivering the last Outdoor Temperature value stored in the last calculated snapshot.

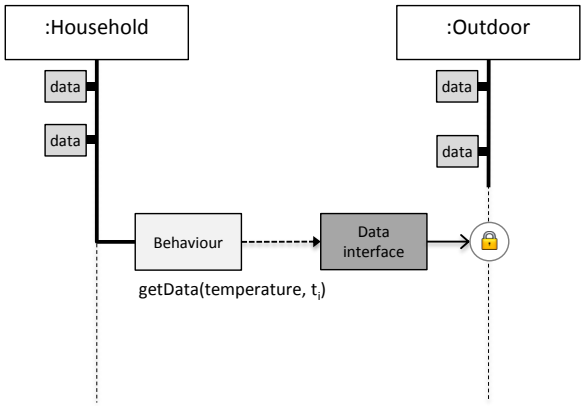


Fig. 6. The Household behaviour request is blocked since the Outdoor Simulation Time is delayed with respect to the Household one

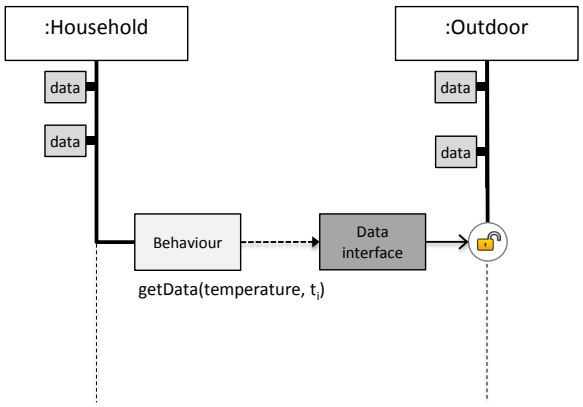


Fig. 7. When the Outdoor Simulation Time reaches the Household one the data is delivered.

Cyclic coupling with data interfaces The cyclic dependence is a concrete case of the data dependence. Two objects depending on each other whose Simulation Times are different, is handled with the following rules: the most delayed one will always retrieve the required data while the most advanced will be blocked until the delayed reaches its Simulation Time (Figure: 8). The mutual blocking is not possible since objects retrieve the value for the current Simulation Time to calculate the next Simulation Time value.

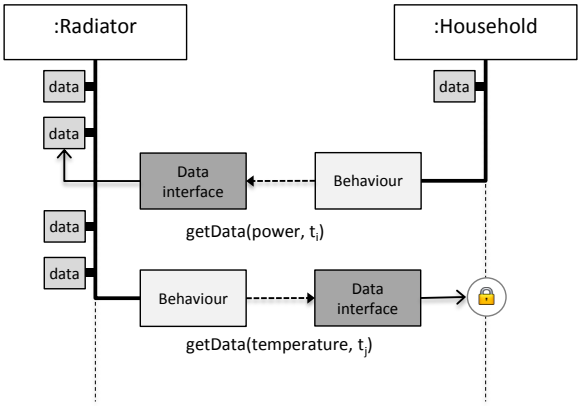


Fig. 8. Radiator and Household cyclic dependence resolution

In the example shown in the figure 8, the Household requires the power consumption of the Radiator in order to calculate the new temperature value. On the other hand, the Radiator behaviour needs the Household temperature value to modify the Radiator state, since the reference temperature at the Radiator thermostat serves as a control mechanism.

Coupling with an event interface The event coupling means that an object receives external messages that contain orders for changing its internal variables. This is the case of objects which are managed by people that are represented as Agents in the model. The Agent interacts with these objects by sending a message using the object event interface. When the message is received by the object interface, the object Simulation Time is developed and then, a new snapshot state is created.

It could happen that the agent develops its simulation time without the intention of sending an order to any object. In this case, the agent

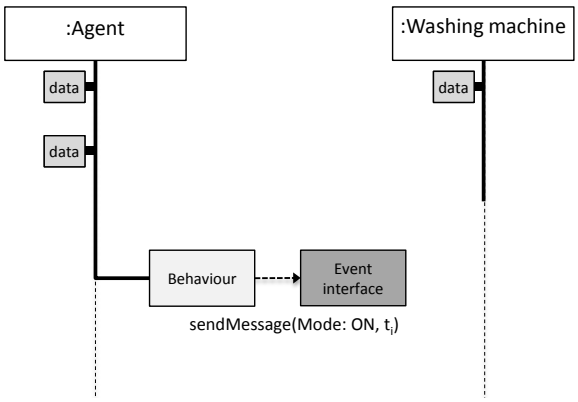


Fig. 9. The Agent sends a message to turn on the Washing Machine

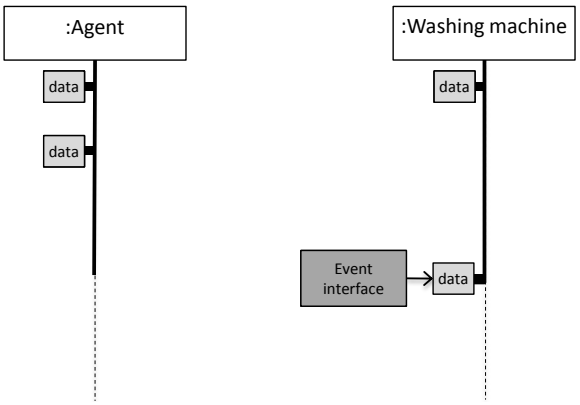


Fig. 10. The Washing Machine event interface changes the object mode to on

behaviour must send a "Notification Time Message"to the object. In fact, when the agent simulation time develops, the agent behaviour must send a Notification Time Message to all objects the agent is controlling. This notification determines how long an object can develop its Simulation Time. This type of relationship means that object's Simulation Time that is controlled by an agent, will never exceed the agent Simulation Time.

Figures 9-12 show an event relationship between a social Agent that turns on the Washing Machine. In this example, the Washing Machine Simulation Time is always behind the Agent Simulation Time. In other

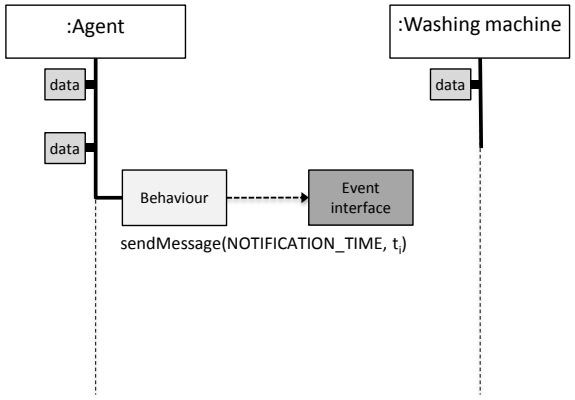


Fig. 11. The Agent indicates the Simulation Time in which it is to its controlled objects

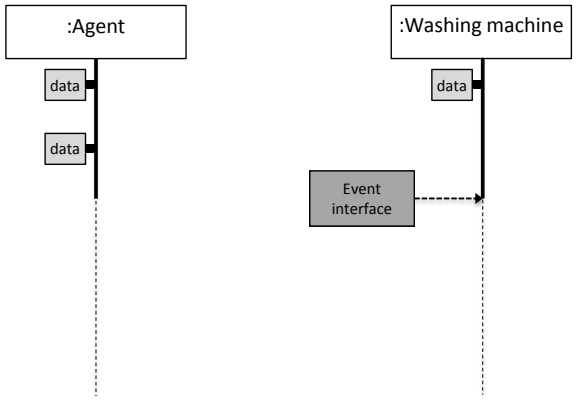


Fig. 12. The washing machine receives the message. Now the washing machine can develop its time until the temporal point indicated in the message

words, the Agent Simulation Time sets a restriction for the Washing Machine Simulation Time.

In the case of the Washing Machine, its power consumption would be 0 at the beginning of the simulation as it's off. Therefore, a new snapshot is created when the Agent turns on the Washing Machine. From that moment, the Washing Machine behaviour will calculate the new power consumption with the restriction that the calculations development should not exceed the Agent Simulation Time, in case the agent turns off the Washing Machine.

Scales The dependencies explanation has been focused on the low scale level. This is due to the fact more complex interactions take place at this level in the demand simulation of the power grids. Scaling up from the presented case to power grid levels demonstrates how the time would be developed following a bottom-up approach. In the figure 13, information flows are shown which indicate how the demand power is aggregated from the lowest levels to the highest ones at a concrete time slice. This aggregation is required to calculate the demand at every scale. Assuming that every element of a level makes the same calculations, it could be observed that each level may be delayed with respect to the lower one. This is the typical case since the upper elements are waiting for the information coming from the lower elements. However, it is possible for all of them are in the same time instant. It is not possible for upper levels to be ahead of the lower ones.

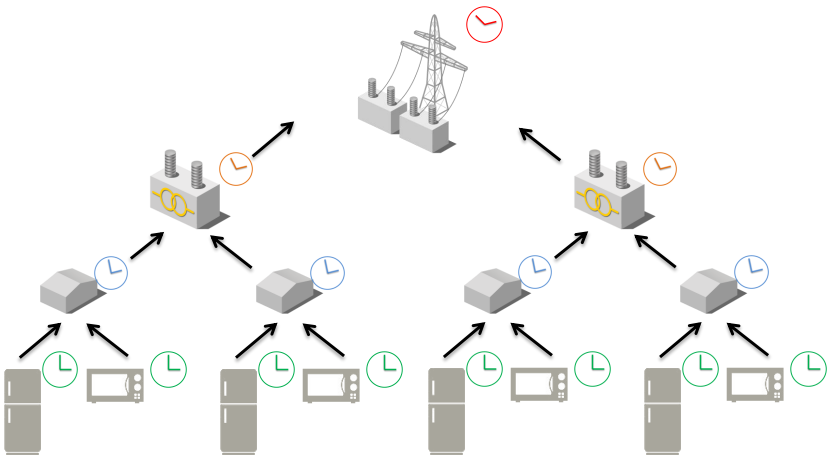


Fig. 13. Demand simulation in a higher scale

2.4 Object time management

In the previous cases, the discussion was focused on objects with a single type of behaviour. However the time-management of an object with several behaviours or/and several event interfaces must be dealt with. Every time a type of behaviour is executed, it registers the Next Time Execution, that represents when it should be executed. The object time-manager selects the behaviour with the nearest Next Time execution to

the current Simulation time. The event interfaces are dealt with in the same way, so that the Interface Next Time Execution corresponds with the time defined in the last message received. Whenever a received message concerns a variable value modification, the object behaviours will be executed afterwards allowing a change in their Next Time Execution, according to the new state. Therefore, objects can dynamically develop their Simulation Times : that is, their Pace could vary from one Simulation Time to the next one. To illustrate the internal time-management, the photovoltaic cell behaviour is studied. This behaviour calculates the generated power, based on the environmental solar radiation. Therefore, the generation power variable will vary along the day until the sunset when the production will become 0. Then, this variable will not change until sunrise. According to this behaviour, three solutions can be proposed to avoid systematic calculus along the night:

1. When the sunset is reached the behaviour registers the Next Time Execution in the sunrise time, whenever this data is available.
2. When the sunset is reached the behaviour registers the Next Time Execution of the previous known sunrise time. This temporal jump may avoid the first solar radiation when the sunrise time is before the already known one. Therefore, the Next Time Execution could be the previous known sunrise time minus ten minutes.
3. The photovoltaic cell outdoors could send messages to the photovoltaic cell event interface whenever solar radiation changes. Following this, the photovoltaic behaviour could register its Next Time Execution to infinite (sleep mode). Therefore, solar radiation changes are received by the photovoltaic cell event interface-allowing mode of behaviour to access this information.

2.5 Implementation

In this section, architectural methods to implement this approach are presented. This architectural proposal takes into account the previously described requirements for simulating a power grid, using an asynchronous approach.

A Tafat Thread represents the execution of a single Model Object and from this point of view describes the execution state, awake or sleeping, and the simulation time in which it is (Figure: 14). During the execution of the whole simulation, Tafat Core request awake Tafat Threads to be executed. After this execution, a Model Object will have changed its simulation time and/or its state. In order to improve the performance, Tafat Core keeps a list of the awake threads and it is listening for state changes in threads to update this list.

A single Model Object has many controllers that can modify the Simulation Time. A controller factor could be either Behaviour or an Event Interface. These controllers, that implement the Develop Time interface, participate in the Model Object simulation, each of them proposing different Next Simulation times. When the Next Simulation Time of any of these controllers is undefined, the Tafat Thread that represents the Model Object turns into a sleeping mode. Once, the Next Simulation Time of all the Model Object controllers are defined, the thread will wake up. Next Simulation Time of Develop Time Controllers could be set to undefined or a value that should be greater than the current Model Object Simulation Time. A feasible value for a Next Simulation Time could be infinite, meaning that Behaviour is suspended, pending an external event.

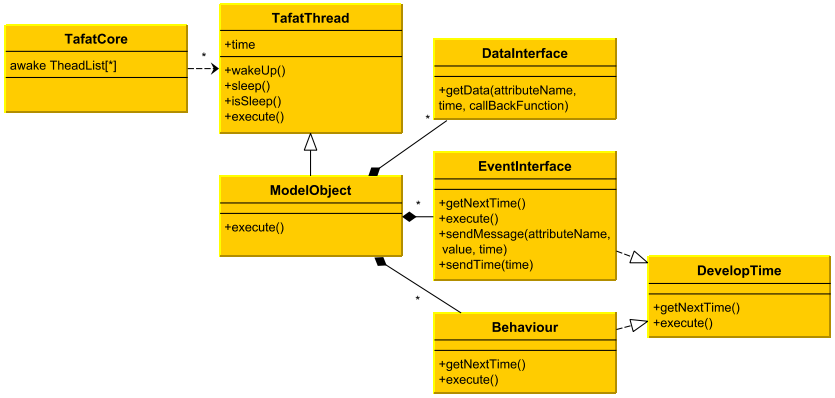


Fig. 14. Tafat asynchronous simulation architecture

For example, the Next Simulation Time of a Washing Machine Behaviour can be infinite, so that the washing machine is off and therefore, it is waiting to be turned on (Figure: 15). On the other hand, the Next Simulation Time of this Washing Machine Event Interface is undefined until other Model Object Behaviours that use it, set the Next Time Simulation. Since the Washing Machine depends on the Social Agent to be modified, the Social Agent must inform this device of this. This Current Time is transmitted through a message which arrives at the Washing Machine Event Interface. When the Social Agent Current Time arrives, the Washing Machine Event Interface will modify its Next Simulation Time from an undefined value to the one which has arrived in the message. Whenever an event for modifying the state of the Washing Machine

arrives, the Washing Machine behaviour will be executed once, allowing to it to calculate its Next Simulation Time based on this new state.

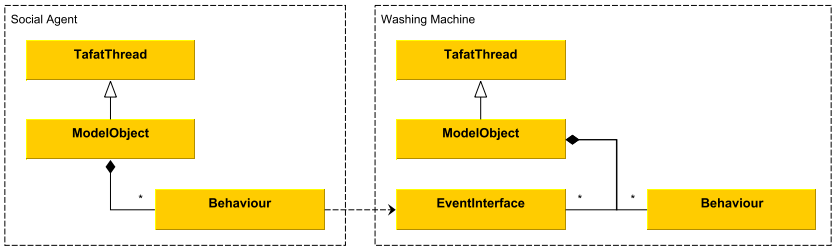


Fig. 15. The Washing Machine Event Interface Next Simulation Time turns from undefined to a defined value when the behaviour of the Social Agent sends its Current Time. On the other hand, the Washing Machine behaviour Nex t Simulation Time turns from infinite to a reachable time when its state is changed to ON by the Agent

Another improvement from the performance point of view is based on the Snapshots removing. A concrete Model Object may have dependences for requesting data or set values in external Model Objects. Similarly, other Model Objects could require this one to be accessed. For this reason, the Model Object must keep the snapshots for all the Model Objects which request data. As this Model Object knows the data requesters, it is able to find out the time in which the requesters are and, therefore, it could delete the Snapshots which are previous to the Current Time of the most underdeveloped Model Object requester.

3 Conclusions and outlook

Going towards asynchronous complex system simulations involves a re-conceptualisation. This re-conceptualisation affords objects interaction issues which could come from both data and event dependencies. In a synchronised execution environment, every object of the system is in the same time slice and the time-management is usually handled using a single clock. The main advantage of this approach is, among others, the simplicity when accessing or modifying an object since all of them are in the same time slice. However, the main disadvantage of the execution of object calculations, is that some of them are unnecessary because the execution is not going to produce any different output.

The use of an asynchronous approach for simulating complex systems provides flexibility in the object evolution. Objects can freely develop as

far as their dependencies are satisfied. Furthermore, object behaviours can be both event and time-based which provides the possibility of having sleeping behaviours. This sleeping behaviour could change their status to active by receiving external events. The behaviour step may vary from one execution to the next at a dynamic speed. Both sleep mode and dynamic speed are important features to avoid the systematic calculations at fixed steps which produce the same values. Finally, we think this approach may facilitate the parallel complex simulation execution.

4 Acknowledgment

This work has been partially supported by Agencia Canaria de Investigación, Innovación y Sociedad de la Información of Canary Islands Autonomic Government through the PhD grant funding to José Évora with reference TESIS20100095 and also through the project "Framework para la Simulación de la Gestión de Mercado y Técnica de Redes Eléctricas Insulares basado en Agentes Inteligentes. Caso de la Red Eléctrica de Gran Canaria", with reference SolSub200801000137.

References

[1] AMES Market Package. http://www.econ.iastate.edu/tesfatsi/DCOPFJHome.htm.
[2] DCOPFJ Package. http://www.econ.iastate.edu/tesfatsi/DCOPFJHome.htm.
[3] OpenDSS. http://sourceforge.net/projects/electricdss/.
[4] Pylon, Power system and energy market analysis with Python. http://pylon.eee.strath.ac.uk/pylon.
[5] TSAT - Transient Security Assessment Tool. http://www.powertechlabs.com/software-modeling/dynamic-security-assessment-software/transient-security-assessment-tool/.
[6] TEFTS Program, University of Waterloo, 2000. http://www.power.uwaterloo.ca.
[7] David Anderson, Chuanlin Zhao, Carl H. Hauser, Vaithianathan Venkatasubramanian, David E. Bakken, and Anjan Bose. A virtual Smart Grid. *IEEE Power and Energy Magazine*, 2012.
[8] C.A. Cañizares and F.L. Alvarado. UWPFLOW Program, University of Waterloo, 2000. http://www.power.uwaterloo.ca.
[9] Jose Evora, Enrique Kremers, Susana Morales, Mario Hernandez, Jose Juan Hernandez, and Pablo Viejo. Agent-Based Modelling of Electrical Load at Household Level. In *ECAL 2011: CoSMoS - Proceedings of the 2011 Workshop on Complex Systems Modelling and Simulation*, page 12, 2011.

[10] A. Gabaldon, A. Molina, C. Roldan, J.A. Fuentes, E. Gomez, IJ Ramirez-Rosado, P. Lara, JA Dominguez, E. Garcia-Garrido, and E. Tarancon. Assessment and simulation of demand-side management potential in urban power distribution networks. In *Power Tech Conference Proceedings, 2003 IEEE Bologna*, volume 4. IEEE, 2003.

[11] Jay B Ghosh. Asynchronous simulation of some discrete time models. In *Proceedings of the 16th conference on Winter simulation*, WSC '84, pages 466–469, Piscataway, NJ, USA, 1984. IEEE Press.

[12] Fred J Kaudel. A literature survey on distributed discrete event simulation. *SIGSIM Simul. Dig.*, 18(2):11–21, 1987.

[13] F. Milano. An Open Source Power System Analysis Toolbox. *IEEE Transaction on Power System, vol. 20, no.3*, 2005.

[14] Jayadev Misra. Distributed discrete-event simulation. *ACM Comput. Surv.*, 18(1):39–65, 1986.

[15] C. Nwankpa. Voltage Stability Toolbox, version 2, Center for Electric Power Engineering, Drexel University, 2002. http://power.ece.drexel.edu/research/VST/vst.htm.

[16] R J Pooley. *An introduction to programming in SIMULA*. Blackwell Scientific Publications, Ltd., Oxford, UK, UK, 1987.

[17] M. Zhou. InterPSS. http://www.interpss.org.

[18] R.D. Zimmerman and D. Gan. Matpower, Documentation for Version 2, Power System Engineering Research Center, Cornell University, 1997. http://www.pserc.cornell.edu/matpower/matpower.html.

Going Around Again – Modelling Standing Ovations with a Flexible Agent-based Simulation Framework

Philip Garnett

Department of Anthropology, Durham University, Dawson Building, South Road, Durham, DH1 3LE. UK philip.garnett@durham.ac.uk

Abstract. We describe how we have used the CoSMoS process to transform a computer simulation originally developed for the simulation of plant development for use in modelling aspects of audience behaviour. An existing agent-based simulator is refactored to simulate a completely different type of agent in 2D space. This is possible and desirable because the original simulator was designed with the intention that it could easily be use to model a variety of different agents interacting in 2D and 3D space. The resulting simulation will be used to simulate the phenomena of standing ovations in audiences as a model system of tipping point behaviour. Continued development of this simulator, assisted by the CoSMoS process, has resulted in a general purpose lightweight simulation framework.

1 Introduction

The dynamics of standing ovations incorporates many interesting aspects of human behaviour and even with a simple computer simulation there are many things to explore. The collective desire of an audience (or at least some parts of an audience) to display their appreciation of something is interfered with by that very human desire to not embarrass oneself. On the face of it this sounds like a difficult system to understand. There are complex individual decisions about how much you want to stand up and clap, or stay firmly rooted to your seat. There is also the behaviour of your immediate neighbours to consider. If the people sitting next to you start standing up, what do you do? Does that overcome your reluctance to get up? Or if you stand up and they don't, what then? Sit down and shrink back into your seat? Then there is the pressure to conform with the wider audience. If everyone on the other side is standing do you spring up to get your side going, or wait to see if

the enthusiasm diffuses round to your section? From the perspective of the individual agents (or people) there is a lot of complex decision making going on. However, at the population level could there be a simple set of rules that are governing the global behaviour of the system? In this paper we describe the process of re-factoring an existing simulator that was built to be a flexible agent-based modelling platform, but with a focus on a particular use, to allow us to model standing ovations as an emergent behaviour. During the re-factoring process the simulation framework has become increasingly modular and generalizable.

A simulation must be developed using a rigorous process of design, implementation, and validation if it is to be scientifically respectable, understandable and reproducible. As we aim to maintain flexibility, the simulation tool will need to be upgraded and enhanced in a principled manner as its requirements change, and we use it to address new research questions. This helps ensure that we fully understand the foundations of the platform before we build something new on top. The CoSMoS (Complex Systems Modelling and Simulation) process [1] provides a flexible approach designed to support the modelling and analysis of complex systems, including the design and validation of appropriate computer simulations.

We have previously used the CoSMoS process to guide the initial development of a simulation of an abstract tissue level model of plant cells [8]. We then used the CoSMoS process to enhance that existing model to produce a more capable and efficient version of the simulation [7]. Here we present work using the same process [1] to guide modification of the basic simulation framework to produce a new simulation of standing ovations in simulated audiences. This work is further evidence that the CoSMoS process can be used in an incremental manner to assist in the reuse of existing simulation code.

In §2.1 we overview the CoSMoS process as used for modelling, designing, and implementing simulations of human behaviour. In §2.2 we discuss the use of UML as a suitable modelling language to support this process. We then use the CoSMoS process components to structure the remaining sections. In §3 we introduce the Research Context. In §4 we summarise the standing ovation Domain Model. In §5 we discuss how the Platform Model was developed using the CoSMoS process. In §6 we conclude with a discussion of our experiences and some preliminary results.

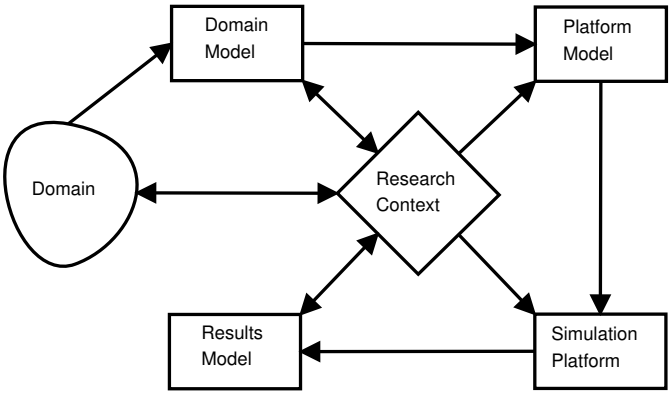


Fig. 1. The components of the CoSMoS process [1, fig.2.1]. Arrows indicate the main information flows during the development of the different components. There is no prescribed route through the process, in so far as going back a step at any point in the process is allowed and often useful.

2 Background

2.1 CoSMoS Process: The modelling lifecycle

For this work we use the CoSMoS process as described in detail by Andrews et al. [1], and used in our earlier work [7, 8]. The CoSMoS process provides a systematic approach to building models and simulations of complex systems, and here we apply it to modelling human behaviour. The CoSMoS process is an inherently incremental process without a defined end point. It therefore lends itself to producing a series of simulations aimed at answering a particular question. This flexibility also allows for taking an existing simulation down a new path to answer different questions. We [7, 8] and others [4, 13, 14] have successfully used this process to assist in the production of simulations of complex biological systems. Summarised in figure 1, the version of the process used here contains the following components (summarised from [1], and the description of the process is taken from [7]):

Research Context : the overall scientific Research Context. This includes the motivation for doing the research, the questions to be addressed, and the requirements for success.

Domain Model : conceptual "top-down" model of the real world system to be simulated. The Domain Model is developed in conjunction with the domain experts, with its scope determined by the Research

Context. The model may explicitly include various emergent properties of the system.

Platform Model : a "bottom up" model of how the real world system is to be cast into a simulation. This includes: the system boundary, what parts of the the Domain Model are being simulated; simplifying assumptions or abstractions; assumptions made due to lack of information from the domain experts; removal of emergent properties (properties that should be consequences of the simulation, rather than explicitly implemented in it).

Simulation Platform : the executable implementation. The development of the simulator from the Platform Model is a standard software engineering process.

Results Model : a "top down" conceptual model of the simulated world. This model is compared with the Domain Model in order to test various hypotheses. This part of the process is on-going research.

2.2 Modelling human behaviour and simulations with UML

UML (Unified Modelling Language) [12] is a suite of diagramming notations designed to aid in the development of large object-oriented software engineering projects by groups of developers working in teams. Ordinarily it is used in conjunction with an object-oriented programming language; it has been shown to be well suited to agent-based modelling [11], and the production of agent based models [6–8, 14]. Here "agents", representing humans, map naturally to agents that can be described using UML. This allows for much of the structure of the behavioural simulation to be represented in UML. We have also found that UML is suitable for capturing simiplified human mental states that are significant to the model (see §4.1).

3 The Research Context

Although we are modelling standing ovations we are interested in the wider phenomena of tipping points and emergence in human behaviour. We use the standing ovation as a simplified case of tipping points in social systems. Human behaviour on both the individual and population level is very complex. People can frequently behave in unpredictable ways, often doing things that may seem counterintuitive or unexpected. People also have the capacity to consciously go against, or follow the crowd, making predicting an individual's behaviour very difficult. At the population level however human behaviour becomes more predictable. A herd mentality (the desire to belong) may mean that in the short term at least

it is possible to predict a future state of a group (or herd) of people [15]. However, it is often the unexpected shifts in population behaviour that we would like to be able to predict or detect. These tipping points are the focus of our wider research and the model presented here was designed as a simplified model of tipping point behaviour.

In the context of wider society social tipping points are very interesting [2, 3, 9]. Even when a human system appears to be relatively stable it could go through a tipping point and change state completely. Here we define a tipping point in a system as when it moves rapidly from one stable state to another state which may or may not be stable. We therefore allow our definition to include reversible changes, so even if the system goes back to its initial state we consider it to have gone through a tipping point. Of particular interest to us are systems that appear to be stable but have the potential to go through a tipping point. In order to understand these systems we must identify the triggers and thresholds that indicate the point at which the system tips.

Modelling an individual human would be extremely difficult but at the population level it might be possible to break the behaviours thought to be of importance in a system down into a set of simple rules. The model can then capture the abstract simple behaviours between agents that when simulated produce the overall systems behaviour as an emergent property. In the model the synchronised human behaviour of standing ovation is an emergent property of the combination of the basic rules of the system, and the interaction between the agents in their environment.

Standing ovations have been used as a model system for synchronised human behaviour in a number of different contexts [5, 10, 16]. We focus on standing ovations as a model system for a social tipping point, paying particular attention to how the system is triggered; what constitutes a trigger; at what point can a system be considered to have tipped; and does that allow us to predict the outcome of a system?

4 The Domain Model: the standing ovation

Standing ovations capture many interesting aspects of synchronised human behaviour. Individual members of an audience are influenced by how good they thought a performance was, and by the behaviours of the people around. Depending on the relative influence of these factors individuals might jump up and start clapping, with little regard to what others might think, or wait to see if others are going to stand up first. In fact there are many possibilities for both showing appreciation, or dissatisfaction. In §4.1 we outline in more detail the aspects of stand-

ing ovations that we are going to capture in the simulation. In §4.2 we summarise how we capture these behaviours in UML.

4.1 The Mental Domain

A standing ovation is an emergent property of the relative influences on individuals in an audience. We break the influences down into three simple parts:

- The individuals own personal enjoyment of the performance. If their enjoyment is very high and they are likely to stand without paying much attention to the rest of the audience. Similarly, if it is very low they are likely to remain seated. The interesting social effects in the system will operate mainly on the people who are somewhere in the middle.
- The behaviour of an individual's immediate neighbours will modify this background likelihood of standing to either suppress it so they remain seated, or activate it causing the individual to stand.
- The final influence on an individual is the room size. We assume that an individual can make an assessment of the larger space and that this acts on their background likelihood of standing in a similar way to the neighbour interactions.

We will also look at what affect an individual being able to stand up again has on the behaviour of the system.

4.2 Domain Model UML

In order to re-factor the simulator for its new use we start the process at the Domain Model UML to get a sense of which parts of the simulator can remain and which need to be wholely removed or significantly altered.

Domain Model use cases. During the development of the biological models use case diagrams were used to capture the higher level interactions that were to be included in the simulation [8]. For re-factoring the simulation for a new use we didn't find it advantageous to revisit the use case diagrams as we have come to the conclusion that all of the relevant interactions can be captured in the description of the domain class and state domain diagrams. Therefore unless it is the case the application of the process would benefit greatly from use case diagrams, they will not be used.

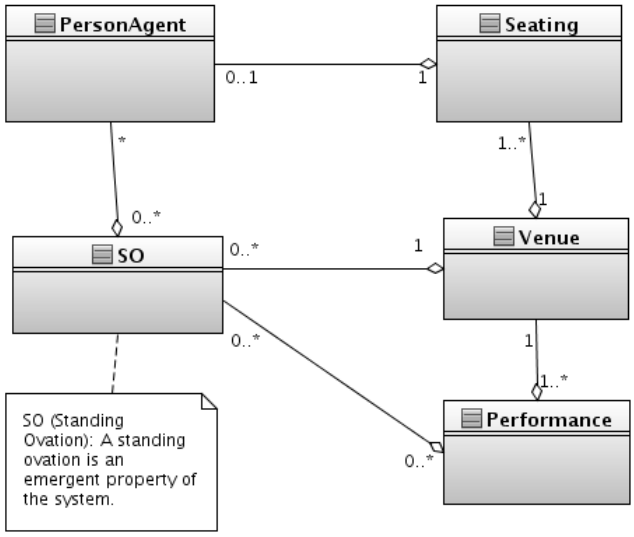


Fig. 2. Domain Model class diagram.

Domain Model class diagram. This captures the different aspects of the system that are required as classes. At this point in the process is it desirable to take forward only the parts of the Domain that we believe to be absolutely necessary. The classes map directly to either the required spatial elements of the system that we need to include in order to simulate standing ovations, or the agents that exist in this space. This includes the Venue, and the Seating within the Venue, which together define the space the system exists in. There is also the class, PersonAgent, which is a simplified representation of the people in the simulation. We also capture the required aspects of the performance in the Performance class. These classes provide enough structure to allow us to model a standing ovation. We also capture the emergent property of a standing ovation SO. At this point it is a desired outcome of the interaction between the Venue, Performance and PersonAgent classes (See figure 2).

Domain Model state diagrams. When describing a biological process state diagrams prove very useful as they can easily show the different states that biological objects can exist in and how they move between states [7]. In figure 3 we can see that in this model the number of physical states the PersonAgents can be in is very limited, Sitting or Standing. We therefore propose that in the case of modelling human behaviour (or

any behaviour that is not manifested in a physical change) it is advantageous to also model more abstract mental states that are of importance to the model. These can then be included in the diagrams which assists communication of important parts of the simulation. In the mental state diagram, mental states provide information about what needs to be captured by the model in order for the PersonAgents to transition from the Sitting to the Standing state. Figure 4 shows the mental state diagram for a PersonAgent. When a PersonAgent changes state they can go through a loop of mental state transitions before they commit to either remaining in the same state or transitioning to a different state. This loop incorporates what the individual is considering doing, the state that their neighbours are in, and the different states that the wider audience is in. In figure 3 it is possible to clearly label the state transitions as Stands and Sits. However, when the mental processes are included the transitions are more difficult to label as the physical state might not change even though the PersonAgent has gone through a process of assessing whether it is going to change state.

In some sense what we are attempting to capture by including mental states in the Domain Model state digrams is an agents abstract individual tipping point. The point at which an agent is required to make the decision to change state they (for a short time) are no longer simply passively Sitting or Standing, but instead are in a separate state of making a mental decision. The outcome of this rather abstract mental state is either the Sitting or Standing, but during the process a transition has occurred.

Upon entering the venue people are standing. A mental process then occurs during which individuals assess what they want to do, taking into account the state of the wider audience and their neighbours, in order to find their seat. Throughout the performance this process will continue as there are a number of possible reasons why a person may have to transition from a Sitting to a Standing state. For the purposes of this simulation we are only interested in people's behaviour at the end of the performance.

5 The Platform Model

The Platform Model includes all the extra components that allow the simulation to run. This includes all the processes required to get the simulation to a point where it is able to start, and the components identified in the Domain Model that are important to allow the behaviours of interest to occur. Also included are any additional behaviours that are required for the model to function but that might not be of explicit

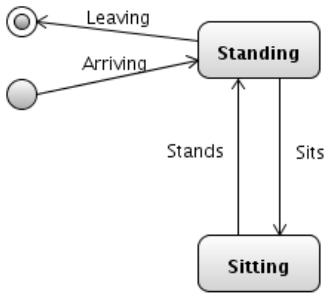


Fig. 3. Domain Model PersonAgent state diagram.

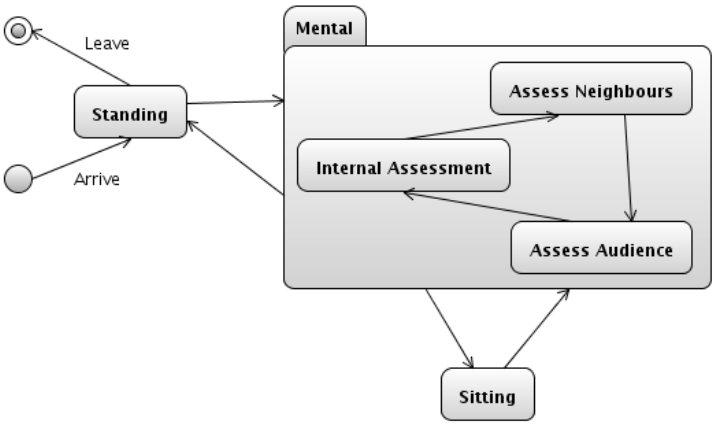


Fig. 4. Domain Model mental PersonAgent state diagram.

interest to the Research Context. These can be implemented with a view to producing an efficient simulation rather than system fidelity. Finally, we need to include aspects of a simulation that are not part of the Domain but are required in order that simulation results may be observed and documented.

5.1 Platform Model UML

The platform model UML bridges the gap between the Domain Model UML and the final implementation of the model in code.

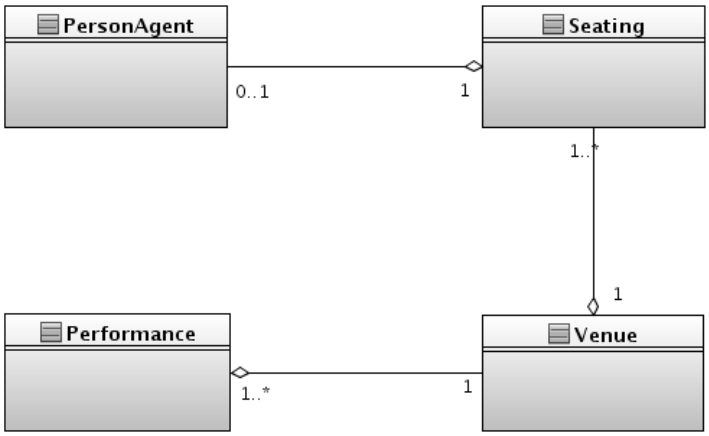


Fig. 5. Platform Model class diagram: included are the four main components of the system being models, the Venue, PersonAgent, Seating and Performance.

The Platform Model class diagram. The is produced from the Domain Class diagram, with all emergent properties (such as a standing ovation, SO) removed. Shown in figure 5 this is a high level diagram and includes the four main components of the system, the Venue, PersonAgent, Seating, and Performance. There is only one Venue, we assume that it has at least one Seating. That Seating can either be unoccupied or have one PersonAgent in it. A Venue is assumed to have at least one Performance.

The Platform Model class diagram, implementation level. This diagram represents the structure of the underlying code of the simulation by including implementation level data structures and any generalisable classes. Figure 6 shows the implementation of the classes carried forward from the Domain model. Seating is a child of the Area class. In this simulation each Area can only hold 0 or 1 PersonAgents, which is implemented as a specific class. The Areas are stored in the Space, and to improve the performance of the simulation can either be (in conjunction with a suitable programming language) separate threads of execution or grouped together in the separate executable buckets of Areas (executable buckets are used in the Java implementation but left out of figure 6 to improve readability). Structuring the Space in this way has advantages for investigating the affect of the interaction between agents and the

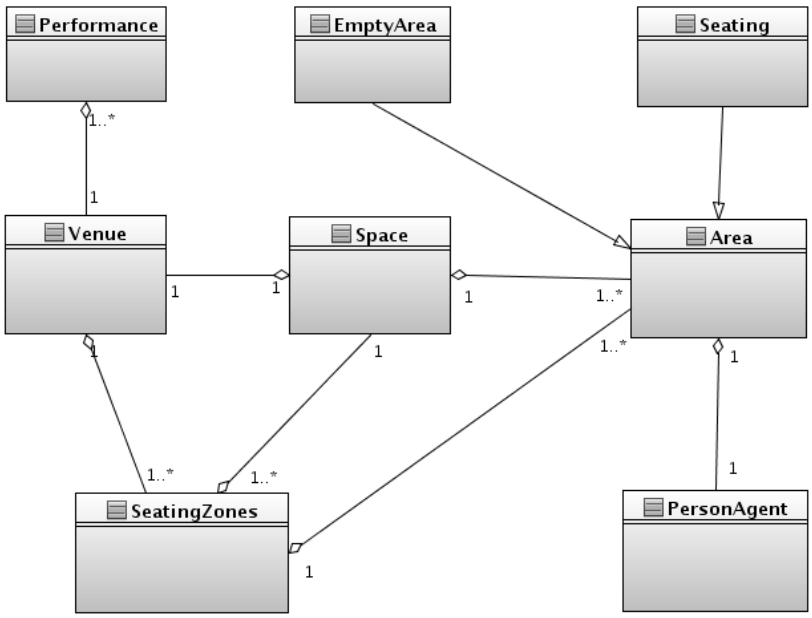


Fig. 6. Implementation level Platform Model class diagram: The Venue has one Space, at least one Performance, and at least one SeatingZones. The Space and is composed of many Areas which can be one of the two child classes EmptyArea or Seating. Seating belongs to one SeatingZones.

spacial environment, and allows easy reproduction of interesting spaces (see §5.3 and §6).

Space has at least one SeatingZones, and a SeatingZones has at least one Area. There are also EmptyAreas which are used for Space that are not processed by the running simulation. The Space, Performance and SeatingZones are stored in the Venue class. The purpose of the PersonAgent class is to contain the current state of the PersonAgent. The PersonAgent's interaction with the wider system is mediated via the Seating and SeatingZones classes. In its current implementation the Performance is almost unnecessary but in the future when we extend the simulation to include more interaction between the performance and the audience the complexity and importance of this class will increase. The Venue class acts as a container for the simulated system. It could also fulfill any more abstract requirements that the venue might have.

5.2 Comparing two models

The inclusion, during redevelopment, of a flexible Space makes it easier adapt the simulation to its new purpose. The original modelling framework had at its centre a flexible and extendible class that is the basis for all the different types of space in the system [7]. This flexibility allows for the many different types of space to be described. In this simulation the Area class that is held in the Space class is extended to describe seating (child class Seating). Each area of Seating contains within it one agent representing a single person, PersonAgent. The Seating class connects that agent with this environment. Via the Seating class the agent can contact its neighbours, it can also sample from all of the Seating areas of the simulation to gather information about any of the agents in the system. Figure 7 shows the Platform Model class diagram for the auxin simulation model described in [7]. By comparing this diagram with figure 6 is it possible to see how at the implementation level very little of the underlying structure of the model needed to be altered for its new purpose.

The behaviour of the model is controlled by running a method within the extended Area classes called 'process'. This processes any agents held in this part of the space. What the different agents do is controlled by their implementation. This structure means that no assumptions about how the agents are behaving in space are transferred from one model implementation to another. What does remain the same is the channels through which the agents get information about their environment. However because this is ultimately determined by the extended Area classes the exact information that flows is also specific to each model. To move from an auxin transport model to a standing ovation model the Area class was extended to describe the Seating in the Venue. The auxin model's Plant class, which is a general holding class, providing functionality similar to a database, was simply renamed as Venue. The Cell class in the auxin model implemented an abstract collection of different Areas (three different types, Cytoplasm, Membrane, and Vacuole, see figure 7). For the standing ovation model this class is completely reimplemented as SeatingZones. SeatingZones is equivalent to Cell as it is a collection of areas of Seating that can be considered as 'together' in the space, and information may flow differently between different zones or within a zone. Currently these spatial zones are not generalised from a parent class and therefore for this model the class was rewritten from the ground up to ensure it provided the correct functionality.

During the process of reimplementation it became clear that the Cell and SeatingZones classes share a number of common features. We are now working on two generalizable classes, SpatialNetwork and AgentNet-

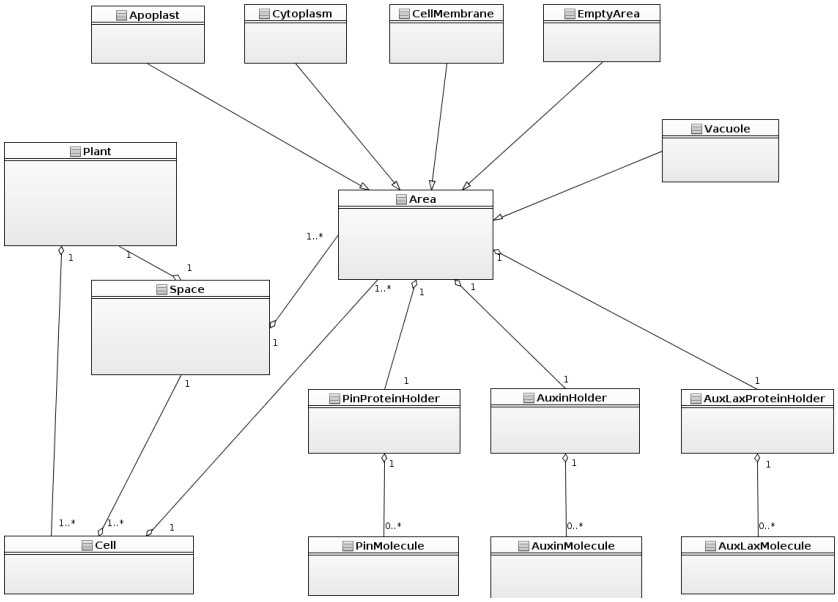


Fig. 7. Implementation level Platform Model class diagram for the original auxin simulation:

work. SpatialNetwork provides a basis for the implementation of abstract collections of spatial Areas, which Cells and SeatingZones are examples off. AgentNetwork provides a basis for abstract collections of agents. In the case of our standing ovation model this could be men and women PersonAgents for example. These should not be confused with the executable buckets of Areas mentioned in §5.1. It would however be useful to implement the multi-threaded execution of the simulation with these abstract collections if that seems feasible and desirable. The generalizable form of the future framework is shown in figure 8. The introduction of these higher level collections brings this framework closer to the framework described in [4, 13] (which the author was involved in developing). The modular design is deliberate as it helps to make clear what the foundations of the simulator are, these foundations can be extended helping avoid the reusing code implemented for old models that are not suitable for the new purpose. The modeller is encouraged to start from the generalizable classes not a previous implementation.

Platform Model state diagrams. These remain the same as the Domain Model State Diagrams.

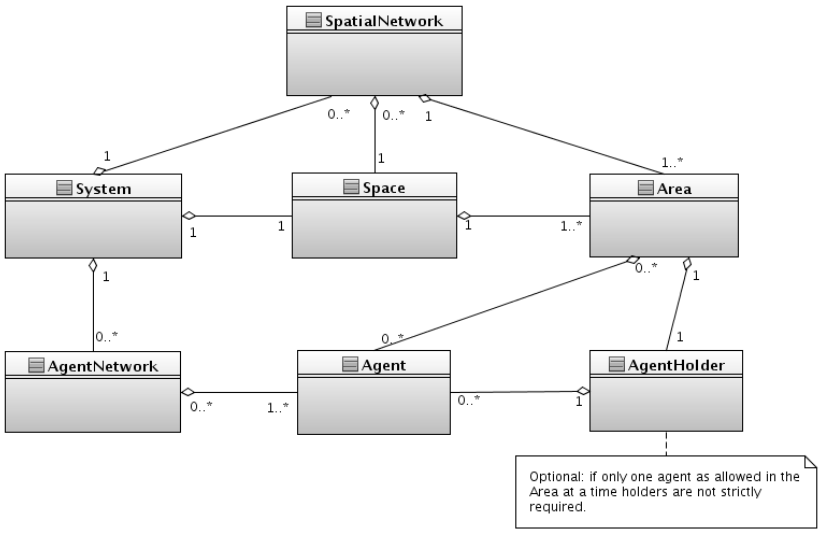
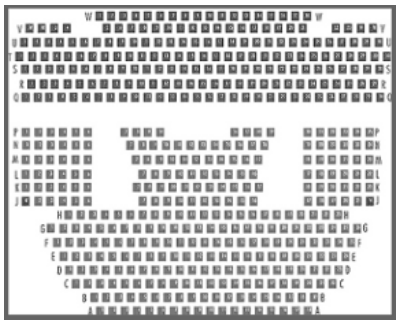


Fig. 8. The class diagram of the generalisable simulation framework.

5.3 Space from Templates

The original auxin model was designed to allow for cell tissues to be loaded into the simulation from a section through a real tissue [7]. This feature allows the standing ovation model to load in approximations of real seating plans. These seating layouts are read from template images into the simulation when it starts. The template provides information on the position of all the seats in the system, and how they are arranged in 2D space. Including any empty areas or gaps between different sections of the audience.

This allows us in future to explore the effect of boundary conditions in the system. Not only those at the edge of the seating, but also boundaries within the seating. It is possible that the probability of a standing ovation fully taking hold might be affected by the seating in many venues being in sections. The small breaks in the seating may alter the dynamics of system, and the flow of information within the system, changing how the audience responds as a whole. It might be that the sectioning of the audience encourages a greater degree of autonomy (I'm part of this group, the people overthere are a different group), reducing the ability of a standing ovation to spread over the entire system, or perhaps giving small sections of the audience the confidence to act alone.



(a) A real seating plan of a lecture theatre.

(b) A template approximated from the real plan. For this simulation we chose not to use seating zones but they could have been used to produce a more faithful plan.

(c) A still from the running simulation. The light grey area of the seating indicates that the PersonAgent is standing.

Fig. 9. By allowing the use of real seating plans we can compare real data with simulations allowing the investigation of effect of changing the layout of the venue on the system.

6 Discussion

We have been able to show that the CoSMoS process can be used to assist with the implementation of an existing model in a new area of scientific study. This is in some ways an inherently dangerous method of producing a simulation. There is the potential to use code that when originally written had at its foundation assumptions that are not suitable for the new purpose. Where the CoSMoS process becomes valuable is that it insists that the model implementation starts from a suitable place: the Domain. Starting at the domain and working forward provides useful information about which parts of the original model can be safely reused. Once the unsuitable parts of the system are removed the process can then be followed as normal to produce the new simulation.

This systematic process has two added benefits. Simply casting a simulation into UML along with using the CoSMoS process can often highlight possible improvements that can be made to the model and resulting simulator. Good examples include the identification of code that has been put in the wrong place, and the discovery that parts of the system that have been excluded from the Domain Model or not explicitly identified when they need to be. One example is that there is a temptation to include aspects of agent behaviour in both the Area and Agent classes which could be a source of confusion further on in the development process. Avoiding this potential confusion also increases the modularity of the simulation by encouraging consistency in the placement of methods. Increasing the modularity of the simulator has helped in the development of an efficient generalizable multi-threaded agent-based simulation framework. In fact the generalizable framework emerged from the process. The adherence to the CoSMoS approach has also ensured that its development has been systematic and well understood. At each stage of the process effort has been made to understand and acknowledge the decisions that have been made and why, and many of the decisions are recorded.

It is true that the simulation framework produced is not completely generalizable and could not be used to produce *any* agent-based simulation. It is unlikely that any such tool could be produced. We are also of the opinion that the attempt should not be made. A completely generalizable framework would either be susceptible to bloat in both size and complexity, resulting in a tool that was difficult to maintain, fully understand, or apply appropriately. Or it would become such light-weight collection of extendible classes that it would be of greater advantage to develop a system of patterns instead. There then remains the question of how do you determine when a general tool should be used over a one-off efficient simulator? This is not an easy question as this framework

started as an efficient one-off simulator and has developed into a more generalizable tool. The CoSMoS process has a lot of offer as a way of assisting this decision making process. Even though we intuitively believed that the foundations of simulation of auxin transport were suitable for modelling another 2D agent-based system, following the CoSMoS process assisted in determining which parts could remain and which needed to be rewritten or simply removed. The CoSMoS process should therefore allow the developers and domain experts to determine if there is a suitable existing tool (as long as they actually understand what the tool has to offer), or if a new simulator is required.

Acknowledgements

We gratefully acknowledge the financial support from the Leverhulme Trust who funds the Tipping Point project based in the Institute of Hazard, Risk and Resilience at Durham University. We would also like to thank the developers of the CoSMoS process. Finally we would like to thank the reviewers for their detailed and very helpful comments, and Lauren Shipley for her assistance with proof reading.

References

[1] Paul S Andrews, Fiona A C Polack, Adam T Sampson, Susan Stepney, and Jon Timmis. The CoSMoS Process version 0.1: A process for the modelling and simulation of complex systems. Technical report, University of York, 2010.

[2] M Batty. Discontinuities, tipping points, and singularities: the quest for a new social dynamics. *Environment and Planning B: Planning and Design*, 35(3):379–380, 2008.

[3] William A Brock. *Tipping Points , Abrupt Opinion Changes , and Punctuated Policy Change by.* PhD thesis, University of Wisconsin, 2004.

[4] Alastair Droop, Philip Garnett, Fiona A C Polack, and Susan Stepney. Multiple model simulation: modelling cell division and differentiation in the prostate. In Susan Stepney, Peter Welch, Paul S Andrews, and Carl G Ritson, editors, *Proceedings of the 2011 Workshop on Complex Systems Modelling and Simulation, Paris, France, August 2011*, pages 79–111. Luniver Press, 2011.

[5] Fernando Eesponda, Matías Vera-Cruz, Jorge Tarrasó, and Marco Morales. The complexity of partition tasks. *Complexity*, 16(1):56–64, 2010.

[6] S. Efroni, D. Harel, and I. R. Cohen. Towards rigorour comprehension of biological complexity: modeling, execution, and visualization of thymic T-cell maturation. *Genome Res*, 13(11):2485–2497, 2003.

[7] Philip Garnett, Susan Stepney, Francesca Day, and Ottoline Leyser. Using the CoSMoS Process to Enhance an Executable Model of Auxin Transport Canalisation. In S Stepney, P Welch, P. S. Andrews, and A. T Sampson, editors, *CoSMoS 2010*, pages 9–32, 2010.

[8] Philip Garnett, Susan Stepney, and Ottoline Leyser. Towards an Executable Model of Auxin Transport Canalisation. In W P Stepney Susan, Polack Fiona, editor, *CoSMoS 2008*, pages 63–91. Luniver Press, 2008.

[9] Suzanne B. Goldberg. Constitutional tipping points: Civil rights, social change, and fact-based adjudication. *COLUMBIA LAW REVIEW*, 106(8):1955–2022, 2006.

[10] John H Miller and Scott E Page. The standing ovation problem. *Complexity*, 9(5):8–16, 2004.

[11] J Odell, H Parunak, and B Bauer. Extending UML for agents. In *AOIS Workshop at AAAI*, pages 3–17, Austin, 2000.

[12] OMG. Maintainer of the UML Standards., 2012.

[13] Fiona A C Polack, Alastair Droop, Philip Garnett, Teodor Ghetiu, and Susan Stepney. Simulation validation: exploring the suitability of a simulation of cell division and differentiation in the prostate. In Susan Stepney, Peter Welch, Paul S Andrews, and Carl G Ritson, editors, *Proceedings of the 2011 Workshop on Complex Systems Modelling and Simulation, Paris, France, August 2011*, pages 113–133. Luniver Press, 2011.

[14] Mark Read, Jon Timmis, Paul S. Andrews, and Kumar Vipin. Using UML to Model EAE and its Regulatory Network. In Paul S. Andrews, Jon Timmis, Nick D. L. Owens, Uwe Aickelin, Emma Hart, Andrew Hone, and Andy M. Tyrrell, editors, *Proceedings of 8th International Conference on AIS*, volume 5666 of *Lecture Notes in Computer Science*, pages 4–6–6, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[15] E.M. Rogers. *Diffusion of Innovations*. Free Press, New York, 1962.

[16] Miklos N Szilagyi and Matthew D Jallo. Standing ovation: an attempt to simulate human personalities. *Systems Research & Behavioral Science*, 23(6):825–838, 2006.

CoSMoS in the Context of Social-Ecological Systems Research

Richard B. Greaves[1,2], Fiona A. C. Polack[1,2], and John Forrester[1,3]

[1] YCCSA, University of York, UK, YO10 5DD
[2] Department of Computer Science, University of York, UK
[3] Environment Department, University of York, York, UK.
Fiona.Polack@york.ac.uk

Abstract. Social-ecological systems are an innately complex interaction among social systems and ecosystems. Computer modellers have been working with the WD-NACE social-ecological systems research project to develop useful models, following a principled approach to complex systems modelling and simulation (the CoSMoS process). Key considerations include comprehensibility by domain scientists; software engineering rigour; and consideration of soft elements in the context of physical structures and behaviours. In social sciences, there is an unresolved problem in the modelling of *soft elements* and the structures and behaviours that underlie soft elements. There is a need to model, and simulate, systems that include soft elements, not least to inform policy and decision options. Focusing on the domain model provides a context for understanding the interaction of soft and hard elements, and helps to clarify what sort of soft-element representation would be appropriate in a computer simulation.

1 Introduction

In social-ecological systems research, modelling and simulation are often proposed as support tools for decision making. However, a tradition of *ad hoc* modelling and simulation focused on exploration of theoretical hypotheses (such as the Schelling model [27] and its variants e.g. [4]) does not offer sufficient real-world understanding of case study scenarios. In developing models and simulations for use as tools in scientific research (e.g. immunology, bio-medicine[4]), we have shown that fit-for-purpose modelling needs an understanding of how to map from domain concepts

[4] see www.cosmos-research.org/publications

to model concepts. To do this, we must pay attention to the semantics of domain concepts and to the modelling-notation (model) semantics.

In this paper, we bring ideas from the modelling and simulating of complex scientific domains to the social-ecological systems context (Section 1.1). First, we outline some ways in which social scientists have addressed modelling, identifying as a key issue the problem of integrating soft elements and physical structures and behaviours (Section 1.2). We then outline the principled CoSMoS approach to modelling and simulation (Section 2). Taking inspiration from the CoSMoS process and philosophy, we determine a purpose for the modelling. We identify that the requirement for a single, overview model of the domain (the Kenyan southern coastal social-ecological system) can be addressed through CoSMoS domain modelling activities. In Section 3, we outline the process of developing a domain model, and present some of the diagrammatic elements of this model. We also show how systematic software-engineering checking of diagram consistency can expose modelling flaws and missing detail. The modelling exercise provides some clarification of the soft element representations needed in the system, but it is apparent that further conceptual work and understanding is needed to fully address soft aspects of the modelled domain. Section 4 provides some commentary on features of the CoSMoS process and philosophy in the context of such social ecosystems modelling. The paper does not reach any definite conclusions, but we believe that it helps to clarifies the "problem" of integrating soft elements in models.

1.1 The WD-NACE Project

This paper reports modelling work done in association with researchers on the WD-NACE project. WD-NACE (Whole Decision-Network Analysis for Coastal Ecosystems) is part of a UK-government initiative to support development of local systems of governance in developing countries, as a route to alleviation of poverty through ecosystems management. WD-NACE brings together social scientists and ecologists, in the UK and in the field. The project focuses on two coastal production systems: this paper concerns wild fisheries and the reef ecosystems of southern Kenya. WD-NACE also considers shrimp aquaculture in the Sundarbans mangrove delta of Bangladesh. In the Kenyan and Bangladeshi study areas, fragile ecosystems support millions of people, and over-exploitation of coastal resources threatens many livelihoods. The underlying philosophy of WD-NACE is that management of sustainability needs to be underpinned by knowledge of why people choose to use resources in a particular way.

The WD-NACE project is attempting to provide support, or the understanding of how to provide support, for people to make choices that may reduce poverty and protect ecosystems. WD-NACE is collecting information from stakeholders at many levels, such as national policy makers, local government, regional authorities, and local decision takers such as individual farmers and fishers. The project aims to try to understand social drivers of ecosystem change, with a long-term goal to develop simulators that support testing and analysis of policy and decision options.

WD-NACE has three internal modelling strands. The Stockholm Environment Institute Oxford (SEI Oxford) has created an individual-based model of the social systems, which has not yet been validated by the domain researchers. Separate ecosystem models are being developed by Kenyan and Bangladeshi local researchers: the ecosystems models are described using STELLA [5], an equational modelling package that has a graphical visualisation[5]. In the future, the WD-NACE researchers would like to integrate models into a simulation platform to support policy and decision options.

The work presented in this paper arises from a collaboration between WD-NACE researchers and independent software engineering modellers. The goal of the collaboration is to to create a meaningful overview model that can express the complexity of viewpoints of the different actors involved at different levels. A key requirement is that the model be comprehensible to significant stakeholders and the full research team, so that it can be checked and agreed as a suitable representation of the domain.

1.2 Modelling in a Social Science Context

WD-NACE is studying social-ecological systems. A social-ecological system is a multi-dimensional complex system. Information collected in a social-ecological research project such as WD-NACE includes concrete, factual material on the structures (fish, people etc.) and behaviours that are observable in the social and ecosystems contexts. In addition, social-ecological information includes *soft elements*, such as opinions, decisions, social interactions etc., and measures such as "wealth". The soft elements may be intangible, and may not be easily represented in a quantifiable or computational form.

There is a significant literature on the incorporation of soft elements in social science modelling. A common approach in social science is to consider only the soft dimension, avoiding mathematical or computational approaches. For example, modelling by Barron et al. [2] is used

[5] www.iseesystems.com/softwares/education/stellasoftware.aspx

to attribute economic benefit to soft elements such as institutional and human social capital. de Vries and Peterson [8] propose a systematic approach to soft element modelling, which starts by analysing people's interpretation of sustainability problems in terms of value orientations and beliefs; the resulting worldviews are translated into narrative scenarios which provide a basis for (qualitative) investigation of risks, opportunities and robustness of policy options [8].

Coulthard et al. [7] argue that the subjective focus on social conceptualisation ignores many other dimensions of what needs to be understood. Soft systems modelling approaches (after Checkland's soft systems method [3]) allow informal visualisation of many dimensions and of both soft and physical elements of a system, accommodating actors' perceptions and how actors formulate ideas about the system. For instance, Powell and Osbeck [26] use rich pictures to express a human activity system (actors, rules, power structures, norms) governing resource use and interaction. Étienne [12] extends such approaches in his *companion modelling*, which attempts to enable simulation of complex social systems. Companion modelling uses visual representations of the views of multiple stakeholders, creating cartoon models (that is, models in a notation without a rigorous semantics). The cartoon models are used as a heuristic aid to communication, to promote social learning among stakeholders, but they can also be used to inform scenarios that in turn inform agent-based modelling.

It is far from obvious how soft systems modelling of beliefs and actor perspectives can be aligned with ecosystems modelling concepts in a simulator. Kemp-Benedict et al. [18] attempt to align social and ecosystems views by using actors' perceptions of the linkage of social and ecosystems concepts: they systematically deduce knowledge-representation rules through interactive questioning. This approach, which rewrites ecosystem models in social terms, makes it difficult to interface with traditional ecosystem models or to interpret results to an ecosystems context. Thus, traceability between reality and social model concepts is enhanced, but traceability to ecosystem concepts is jeopardized. Furthermore, soft systems approaches to social ecosystems do not provide an easy starting point for software engineering. It is hard to know how to quantify and accommodate soft elements, which are usually highly interdependent. In approaches such as companion modelling, it is hard to trace from agent-based models back to the domain concepts, in order to interpret observed model behaviours.

1.3 Modelling in a Software Engineering Context

Faced with a significant amount of information about societies and ecosystem impacts, collected from all levels of the decision-making hierarchy in the field, the WD-NACE researchers approached software engineers at York Centre for Complex Systems Analysis (YCCSA) who have experience of modelling complex systems with emergent properties, and of principled development and use of complex systems simulation in scientific domains. The WD-NACE context differs from the scientific domains in that it includes soft elements, such as power and influence, wealth and well-being.

The software engineering challenge in modelling the WD-NACE systems is to identify what is relevant to the purpose of the model, and what abstractions are appropriate. This means that participants in the modelling exercise need to understand the purpose of the model, the level of abstraction, and the rationale and implications of the abstraction. Modelling needs to be sufficiently realistic for realistic emergent behaviours and properties to emerge. However models should not be so complicated that traceability and understanding are lost.

WD-NACE researchers are interested in creating agent-based simulations. Agent- or individual-based simulation is now widely used in the social science and ecology (e.g. [10, 14]). The motivation for simulation is usually to try to provide support for hypotheses about how a system functions, to generate predictions about the effects of some perturbation to the system, or to assist in policy formulation. There are many software engineering approaches that seek to guide development of agent-based simulations (e.g. [22]), or support the implementation of agent-based simulation (e.g. [19]). However, there is little guidance on principled development of fit for purpose simulations (see [23]).

Ecological researchers have developed standards that support documentation of developed simulators (e.g. ODD and TRACE [16, 17]). However, the standards document the actual simulation, its parameterisation and results (analogous to the way that a scientist documents the set-up and results of a laboratory experiment), rather than recording the design decisions and rationale for the simulator construction or evidence of software quality. Design decisions and quality are important in understanding the credibility of results, and thus important in policy formulation and decision making. Since complex domains are impossible to fully understand and represent, and change continuously, it is never possible to give an absolute guarantee of correctness for a model or a simulator.

The engineering of simulations of complex domains is addressed in the CoSMoS process[6]. The CoSMoS process is often assumed to represent a particular style of modelling, and, indeed, this was what attracted WD-NACE researchers to the approach. However, CoSMoS provides a philosophy of complex systems modelling and simulation, not a simple technique, and in applying CoSMoS-style modelling to the Kenyan coastal ecosystem, other benefits of the CoSMoS approach became evident. In the next section, we summarise the CoSMoS process. We then (section 3) present the domain modelling for the Kenyan coastal ecosystem. After this (section 4), we reflect on the CoSMoS approach and its advantages or disadvantages for this sort of social science project.

2 The CoSMoS Process

In this section, we present an overview of the CoSMoS process [1, 24], then consider the aspects that were important to making progress in WD-NACE.

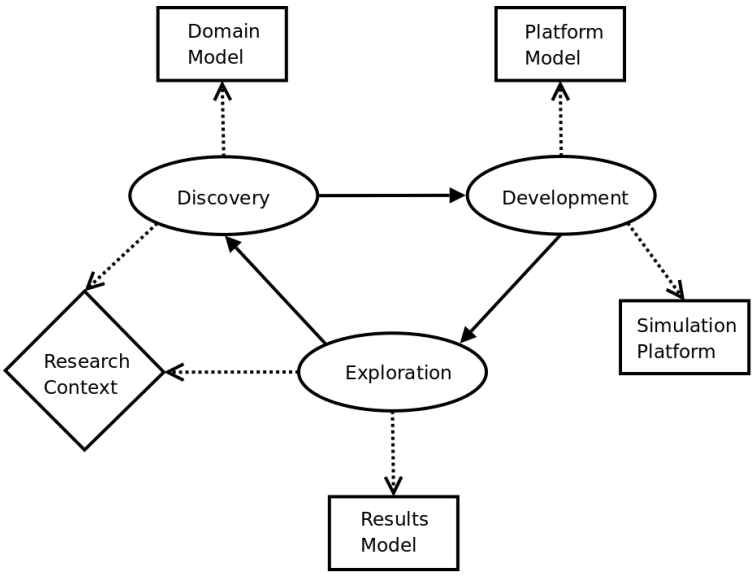


Fig. 1. Overview of the phases of the CoSMoS process, from [1]

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

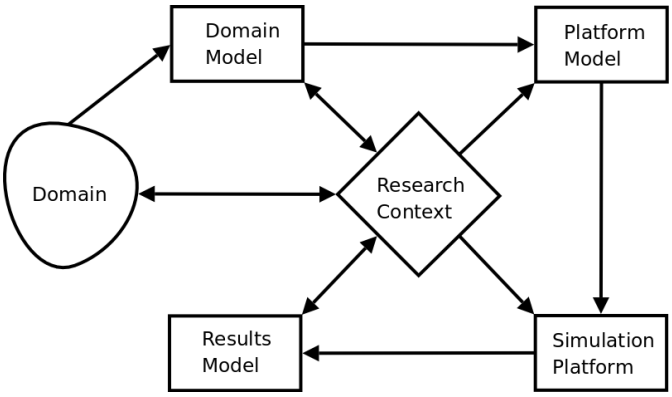


Fig. 2. Overview of models in the CoSMoS process, from [1]. Arrows represent flow of information. Note that, as in most software-engineering lifecycles, any and all steps can be iterated. The arrows can also be associated with validation activities.

CoSMoS provides a principled approach for creation of a fit-for-purpose simulator, for use in engineering or scientific research. CoSMoS assumes a collaborative project involving domain experts and software engineers [24]. The CoSMoS process comprises three phases: *discovery*, *development* and *exploration* (figure 1). Each phase is associated with particular models (figure 2), and has a specific relationship to the project goals and roles [1]. Here we focus on simulation as a tool supporting research in a (social) scientific context.

During the *discovery* phase (figure 1), the collaborators in the simulation project are engaged in developing mutual understanding. The purpose, scope, scale and potential impact of the simulator and simulation results are carefully explored. Once it is apparent that the collaboration is working and that there is a reasonable expectation of feasibility, the *development* phase focuses on interpreting the *Domain* (figure 2) from its research description – papers, cartoons, research models, domain expert commentary, and so on – to an abstract software engineering description, referred to as the *Domain Model* (figure 2).

The domain is the preserve of the *domain expert*, and represents the research area in which the simulator is expected to be used. An important aspect of the CoSMoS approach is that the software engineers (modellers and developers) are guided by the domain expert on all questions relating to interpretation and representation of the domain. The software engineers do not become involved in, for instance, choice of

domain theory or academic decisions that relate to the concepts of the domain.

The domain model is the responsibility of the software engineers. In developing a domain model, the software engineers are taking information from the domain expert (such as information from published research, diagrams and other models of the domain) and interpreting it into a consistent format that is still amenable to checking and review by the domain expert. The ideal is an internally-consistent abstract model that uses the language and concepts of the domain, but which is capable of supporting subsequent systematic software engineering development. One way to achieve the aims of the domain model is to use existing well-defined software modelling notations.

From the domain model, the software engineers (modellers) develop a *Platform Model* (figure 2), which is a software-engineering design for the simulator. From the platform model, the software engineers (implementers) implement a *Simulation Platform* (figure 2), which should be verified and validated appropriately as a software engineering artifact. The simulation platform should be calibrated against existing domain data.

In the *exploration* phase (figure 1), simulated experiments are run on the calibrated simulation platform. The experiments are designed in a similar way to the simulator: the domain experts determine what experiments would assist their research; these are mapped to simulator experiments through collaboration with the software engineers, and simulation experiments are run on the simulation platform. The simulated experiments produce *Results*. These are simulation results, not real experimental results. It is important that these are carefully interpreted, and not assumed to represent the real system directly. Interpretation uses the *Research Context*, a repository of shared knowledge from all phases of the project, about the domain and the development (abstractions, design decisions, motivations, assumptions).

Purpose and Fitness for Purpose In a CoSMoS development, it is explicit that a simulation platform is developed for a specific purpose, and that any validity of the simulation platform relates only to the specific purpose. Thus, a simulator is considered fit for purpose if: the team (domain experts *and* software engineers) is satisfied that the model of the domain is adequate for the intended purpose; the software engineering quality is such that the simulator can be "trusted"; and the results produced by the simulator are consistent with expectation. To keep track of the collaborators' understanding of the suitability and quality of the simulator, CoSMoS proposes arguments of fitness for purpose (discussed

in, e.g. [15, 24, 25]). A typical approach is to instantiate a generic high-level argument, as shown in figure 3, for the appropriate domain and engineering approach. Each claim is then elaborated using appropriate strategies, until either the participants agree that the argument is sufficient to record their confidence in the simulator, or each lowest-level claim is substantiated by evidence.

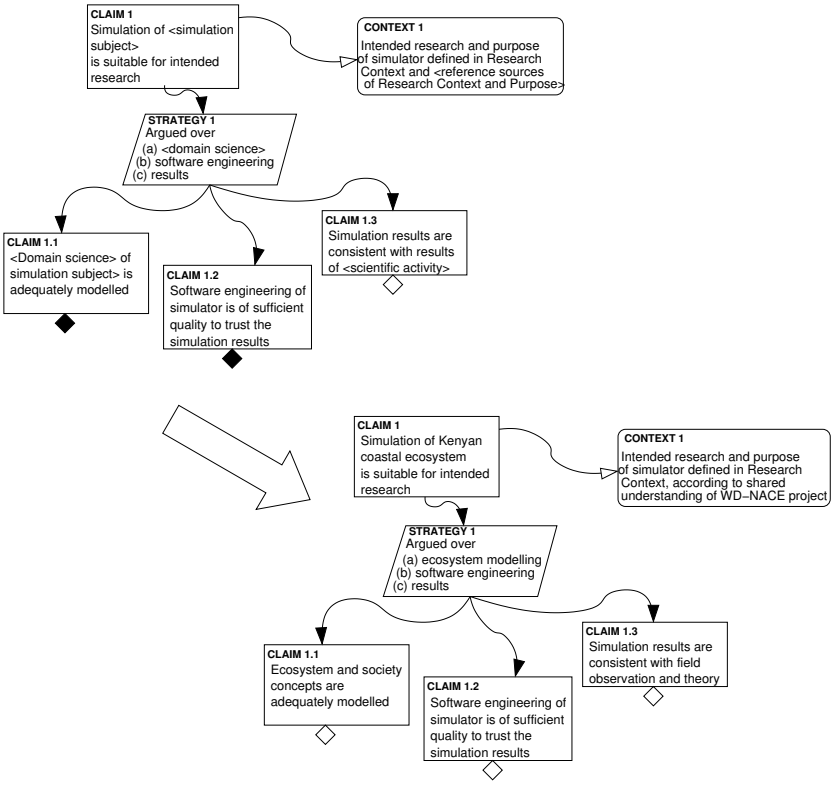


Fig. 3. Instantiating a generic fitness-for-purpose argument for the WD-NACE model of the Kenyan coastal ecosystem. Rectangles represent claims. Parallelograms are strategies. Black-headed arrows indicate the successive elaborations of a claim. White-headed arrows here indicate context, but can also link in assumptions or justifications. Diamonds indicate whether the claim has been developed further (black) or not (white).

3 Modelling the Kenyan Southern Coastal Ecosystem

In this section, we present the process and results of arriving at a domain model that gives an overview of the Kenyan southern coastal ecosystem, as viewed by the WD-NACE project researchers. We show some modelling iterations, but do not attempt a chronological account of the modelling process.

WD-NACE researchers seek to understand how autonomous people interact to generate, share and select knowledge and act on it, and how multi-scale social and ecological interactions produce feedbacks that affect the sustainability of ecosystems services and the livelihoods of those that depend on them. The initial plan of the modelling collaboration was to create an integrated model by translating social and ecosystems models prepared by the WD-NACE researchers into a common software engineering notation and identifying concept mappings across the three models. However, only the Kenyan ecosystem model was easily available, and the modelling goal was changed to creation of a single overview model of the Kenyan southern coastal ecosystem. Since the existing descriptions of the ecosystem represent interaction between the local population and the ecosystem, the overview model inherently integrates social and ecosystem concepts.

3.1 The domain

WD-NACE has collected a wealth of information on the Kenyan southern coastal society and ecosystem over a number of years. The complex interactions of the domain are summarised by Forrester's sketch, figure 4, which shows the principal feedback loops and interactions between people and ecosystem services. People's behaviour impacts the environment and potentially degrades it; changes in the ecosystem affect people's behaviour. At the same time, changes to the ecosystem may lead directly to further changes in the ecosystem; and people's changing behaviour may bring about changes in the behaviour of others.

At a more detailed level, the information collected by WD-NACE researchers is expressed informally in figure 5. This sketch is the starting point for development of the Kenyan ecosystem STELLA model. STELLA [6] provides a graphical interface to sets of differential and linear equations that describe stocks and flows. Because the model is based on equations, it can be simulated to determine consequences of specified flows on stocks. The role of equational models in guiding policy, and particularly in analysing decision options across multiple scales, is limited,

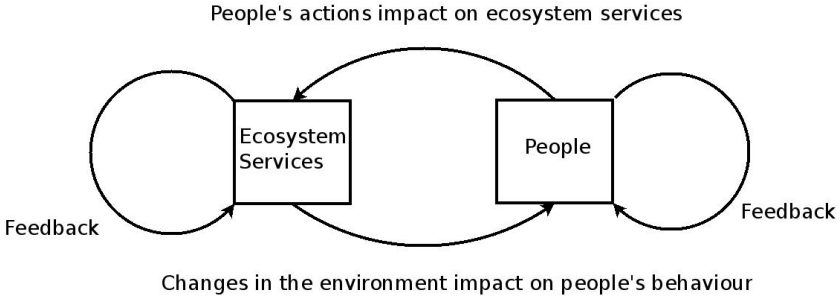


Fig. 4. Forrester's schematic representation of the WD-NACE project domain

since only general trends can be analysed and it is impossible to map the effect of individual inputs or actions.

The domain researchers need a model and simulator that allows more flexibility and better insight into the interactions of the systems than is possible in the STELLA approach. WD-NACE researchers have proposed to use agent-based modelling, with its emphasis on individual behaviour and interaction, as a basis for understanding the influence of policies and decisions on the complex behaviours of the integrated social and ecological systems.

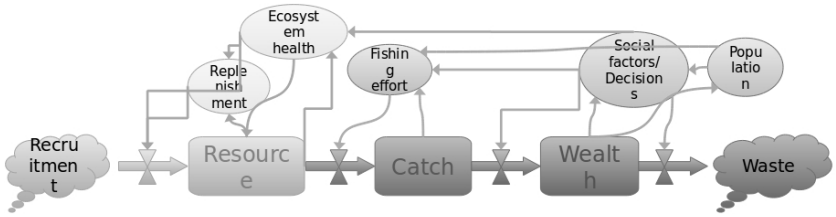


Fig. 5. Obura's sketch model of stocks and flows in the Kenyan ecosystem model. The notation is informal, but clouds represent stocks, thick arrows represent flows. Shaded boxes and ovals are things that are recognised as influences the stocks and flows, with thin arrows showing where influence is known. The shading has no significant semantics.

3.2 The Domain Model

In CoSMoS, the domain model is an abstract (conceptual) representation in a form that can be understood by domain experts but also used to

engineer a concrete (implementational) software system. To support the eventual development of an agent-based model, the domain model needs to interpret the existing information, such as the sketches in figures 4 and 5, into a software engineering model in a well-defined design language. To help the process of interpretation, we need an idea of the purpose of the domain model. Here, the purpose is to capture the structures and processes that affect (a) the decision of individuals to fish or not to fish; (b) the effect of fishing on a reef ecosystem; (c) the effect of the health of the reef ecosystem on fishing.

Apart from the need to support subsequent software engineering, CoSMoS provides no guidance on the choice of modelling language for the domain model. Here, the UML language (from [13]) was chosen based on the experience of the lead modeller (Greaves). The development of a domain model considered adequate by the domain experts and modellers requires many iterations. The first model was derived from Figure 5 (with domain-expert input from Forrester), and comprises a class diagram, the state diagram for objects of one class, and two simple activity diagrams. WD-NACE researchers reviewed these diagrams and helped to elaborate and extend the model; each subsequent iteration was similarly reviewed and extended.

Initial domain model The first iteration undertaken by software engineering modellers is based on careful analysis of the existing STELLA model, which incorporates social as well as ecosystem stocks and flows, and offers hints to an underlying structure for an agent-based model. Components identified at this stage are:

1. the `reef`, the `ecosystem health` of which is central to the project;
2. the `fish stock`, representing the fish `population` living in association with the reef;
3. the `community`, which represents the body of people living in the coastal area described by the ecosystem model;
4. the `fisher`s, whose interaction with the `community` results in fishing practices that may be benign or may have negative consequences such as destruction of the reef and depletion of `fish stock`s.

These concepts are represented as classes in a UML class diagram (figure 6). From Figure 5, `reef`, `fish stock` and `community` would represent singleton classes, but the class model admits the possibility of adding further reefs, communities or fish stocks in future. In the UML notation, associations enable communication (message passing) between objects. The associations represent the reef ecosystem helping or hindering fish stock replenishment; the ecosystem enhancing or diminishing

the personal wealth of the fishers; and the fishers negatively impacting fish stocks and reef ecosystem health.

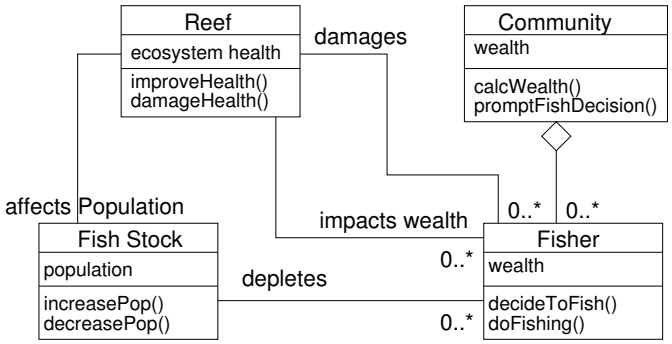


Fig. 6. Preliminary UML class diagram. Association multiplicities are omitted on ends where a singleton class is implied in Figure 5.

From Figure 5 the relevant states of a `fisher` are deduced as `fishing` or `not fishing`, but there is little information about the determinants and drivers of fishing activity.

Extending the class model The initial class model was reviewed by domain experts in Kenya and UK. It was agreed that social parts of the model need to be more detailed. Domain experts also proposed enrichment drawn from a PhD thesis [20] and a study of the effects of fishing gear on the ecosystem [21]. The revised diagram is shown in Figure 7. The model subclasses `fish stock` into `food` and `non-food` stocks (that is, fish sought and caught to be eaten, and other fish, which may be caught accidentally), with an association between the subclasses modelling the possibly-asymmetric interplay of these populations.

The social structure representations added to the class diagram are considered important in influencing the number of fishers and extent of fishing at any time. However, many are soft elements (above), and the domain model does not attempt to give a concrete realisation for these concepts. For instance, there is no detail of the modelling of `wealth`, its adjustment or calculation, and there is no indicator of how kin, household, village or community `influences` the decision to fish.

State diagrams A UML state diagram is a software engineering design model that allows expression of the relevant and distinguishable (compu-

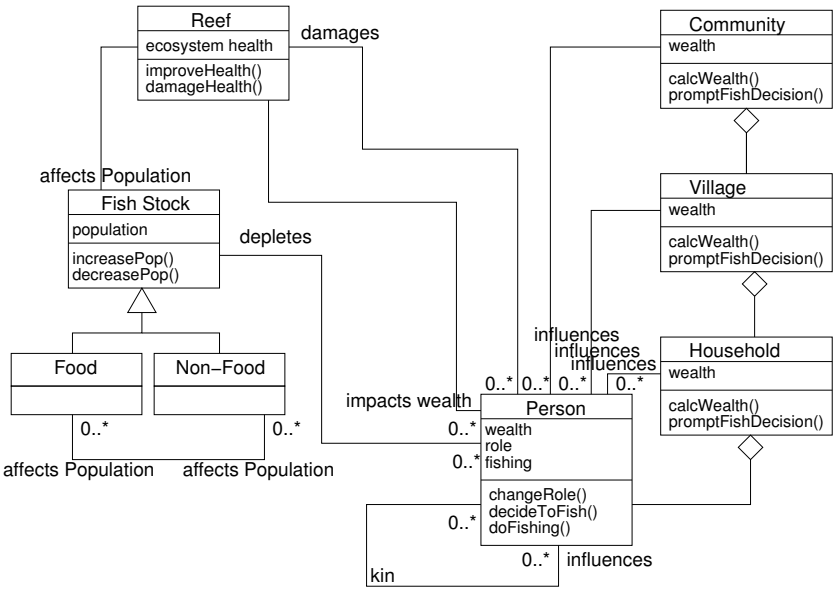


Fig. 7. Extending the UML class diagram of the Kenyan coastal ecosystem

tationally) states of the objects of a class (from a UML class diagram). It is a software engineering design decision as to what states and transitions are relevant. A state diagram for `Person` is shown in Figure 8. The model uses nested and parallel states to capture the selected relevant transitions that can arise (in the model) for all "alive" *person* objects, as follows.

- The *Child* state is for infants and children who are not yet available for work. The domain model makes no assumptions about age, simply about activity.
- The *Student* state is for those studying for qualifications and not available to work. People may enter and re-enter study at any age. The model makes no distinction between forms of education, but to enter the *student* state in the model, the household or individual must be able to afford the education offered (this may be a monetary fee, or may be a soft measure of cost, such as not being "needed" domestically).
- The *Unpaid* state represents people who could be in paid work but are not so currently. The sub-states are:
  - *Domestic*, for those who work in the home without pay.

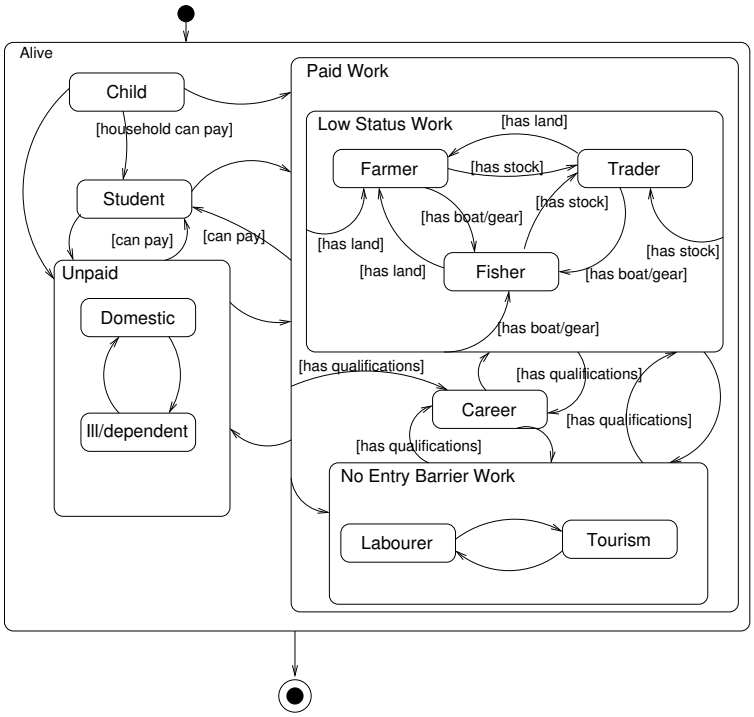


Fig. 8. `Person` state diagram: layered states are used both to map domain concepts, and to simplify capture of the transitions among states – an object that transitions to a super-state can enter any sub-state, subject to meeting any entry condition.

- *Ill/Dependent*, for those who could work but for illness, old age, disability or other incapacity, which forces them out of paid employment. A person in this state may be temporarily incapacitated by a short illness, or may be permanently outside paid employment.

– The *Paid Work* state captures relevant employment; it comprises sub-states as follows:
  - *Career* represents the state of any individual who is in employment and whose job requires educational qualifications. This would cover, for instance, tourism managers, government officials etc. Given the purpose of the model, it is not useful to sub-divide this category.
  - *No Entry Barrier Work* is used for people who are currently doing a paid job with no specific entry requirement. The choice

of sub-states (*Labourer* and *Tourism*) is determined by the need to distinguish different influences on the decision to fish: e.g. labourers might fish when agricultural demand is low, whilst low-status tourism employees might fish outside the tourist season.
- *Low Status Work* is distinguished from other states by physical entry barriers: a *Farmer* must have control over land; a *Fisher* must have access to a boat and fishing gear; a *Trader* must have access to stock.

State diagrams for `Fish Stock` and the `Reef` objects were also created in collaboration with the field expert in Kenya. The diagrams are not shown here: the states of `Fish Stock` and `Reef` are arbitrary, and the determining characteristics and transition triggers need to be worked out with domain experts.

- `Fish Stock` distinguishes states, *overstocked*, *unexploited*, *sustainably exploited* and *over-exploited*. Transitions between the states in this order are known to arise as a result of fishers using net designs that adversely affect stocks; pollution; predation (assumed to include removal of fish by fishers); or poor reef health. In the reverse order, improvement in fish stocks is ascribed to stock management. These factors cannot be expressed directly in a UML state diagram.
- The state diagram of `Reef` distinguishes `ecosystem health` states: *unspoilt*, *slightly damaged*, *heavily damaged*, *deteriorating* and *dying*. Degrading transitions are known to arise as a result of physical damage by fishing gear, whilst improvement in reef ecosystem health arises if the reef is allowed to recover from such damage. Again, these factors cannot be captured directly in a state diagram.

Adding activity diagrams The interrelated processes of degradation of the reef ecosystem and depletion of fish stocks cannot be captured directly on state diagrams that model possible states and transitions of objects of individual classes. The connected behaviours can be expressed in UML activity diagrams, which include notations for activities and guarded choice, as well convergence of inputs to (or divergence of outputs from) an activity. Figure 9 shows activities associated with reef damage and repair, whilst Figure 10 shows activities associated with fish stock depletion and recovery. The activity diagrams draw on the description of stocks and flows, and, extensively, on the advice of the domain experts in the field.

Figure 9 concerns degradation to the reef through damage by fishing gear, as well as the effects on the reef of other coastal activities (tourism, farming). The reef health may start to recover if fishing ceases or the reef

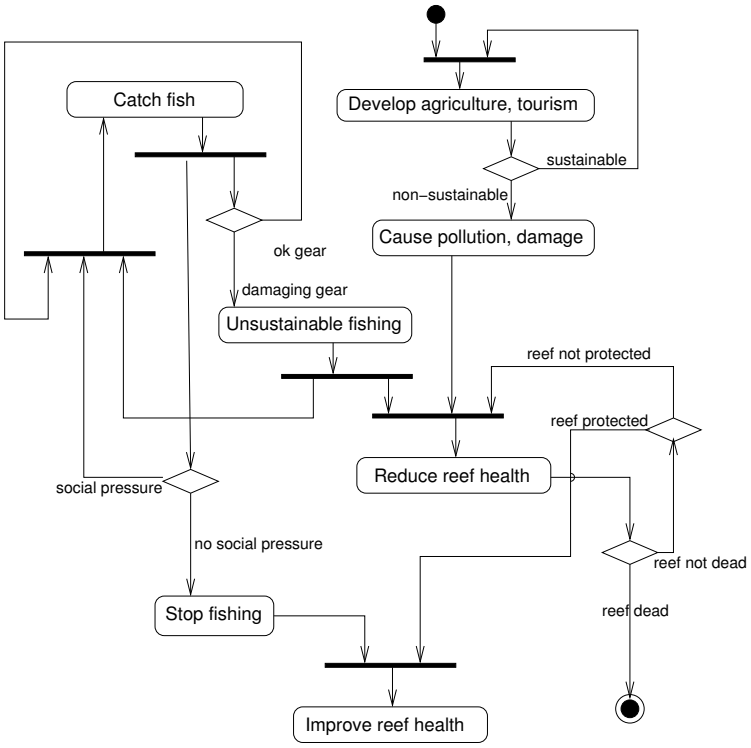


Fig. 9. Activity diagram for reef ecosystem destruction and recovery

is protected by legislating and policing a marine reserve. This activity diagram also models social pressure driving fishing activity.

Figure 10 focuses on depletion and recovery of fish stocks. The factors are similar but not identical. For instance, depletion of fish can be caused by using techniques that deplete not only adult stocks but also juveniles (seining, gill nets, big traps and hand-lining). Fish stocks may recover if fishing ceases or through active stock management programmes, with or without legislation.

The two activity diagrams are, in effect, two simultaneous views on the fishing activity, and can be combined on the shared activities (`catch fish`, `stop fishing` and `unsustainable fishing`). Differences in naming between the two diagrams are deliberate: for instance, in Figure 9, `develop agriculture, tourism` relates to activities that cause the physical destruction of the reef or its environment, whereas in Figure 10,

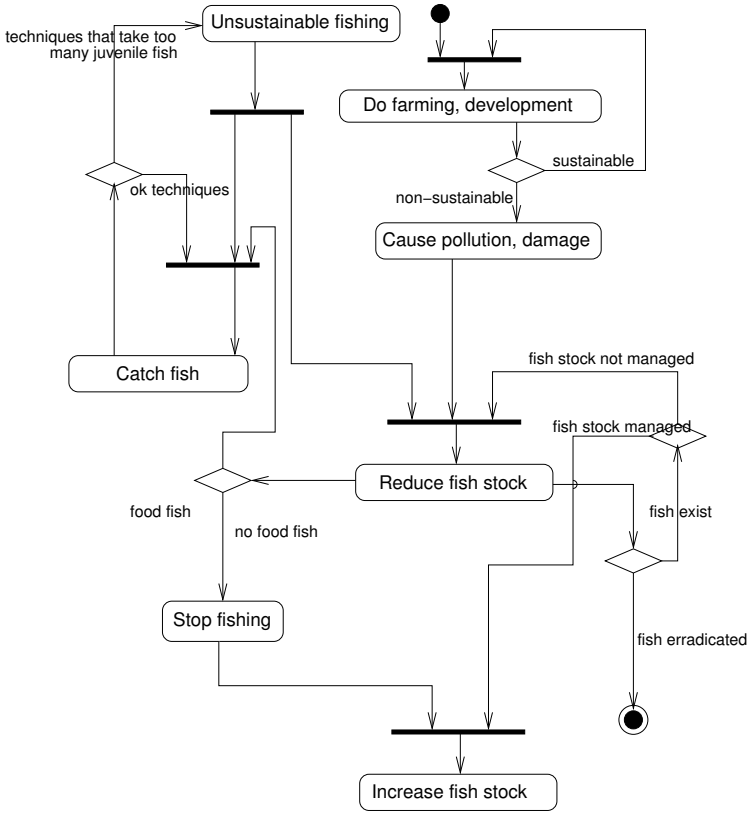


Fig. 10. Activity diagram for fish stock depletion and recovery

`do farming, development` relates to activities that cause the forms of pollution and damage that impact directly on fish stocks.

Apart from capturing the concurrent activities and effects on fishing, fish stocks and reef health, these diagrams begin to illuminate how soft elements would be used in a computer simulation. For instance, we need variables that relate to the quality of the reef and the fish stocks – these could be represented ultimately as quantities or enumerations, depending on what is considered most appropriate by the domain experts.

The activity diagrams can also help to separate the social ecosystem from the social system that it supports and sustains. Thus, we can envisage a separate model that provides the "social pressure" conditions to drive fishing activity, and another separate model that provides inputs on the development, agriculture and tourism aspect. Understanding the

interplay of separable models would be important in determining design decisions in taking the domain model forward into a simulation development.

Checking the model Checking the domain model can contribute to confidence in fitness for purpose of both the modelling and the software engineering basis of the simulator (see Figure 3). Consistency checking also helps to remove from the software engineering model features of the domain that are not relevant or at the wrong abstraction level relative to the purpose of the model. Here, we also check that the domain model uses UML concepts to represent domain concepts in a way that can be used to develop software in a principled manner: we check for consistency between diagrams, and review the semantics expressed in the diagrams.

Consistency checking for the diagrams presented above requires us to check that the state diagrams are consistent with the class diagram. For example, all actions implied in Figure 8 must be either explicitly represented as operations in the Class diagram, Figure 7, or clearly associated to external events. Tables 2 to 4, in the Appendix, summarise the consistency analysis of `Person`.

In relation to soft elements, consistency checking of the domain model cannot be exhaustive. Since the domain model does not define mechanisms and representations for concepts such as `wealth` and `influence`, `reef health` and `fish stock`, consistency checking only points to factors that will be required in concrete representations of these elements in the platform model and simulation engineering.

4 Commentary on CoSMoS Use

The WD-NACE project was well-advanced before the need for "CoSMoS modelling" was identified. The CoSMoS process has been used to guide development of the domain model. The development leads to identification of further modelling needs, such as ways to model the processes needed to express wealth and influence. This section is a retrospective commentary on aspects of CoSMoS that were useful in developing the model.

Software engineering roles: CoSMoS proposes use of *roles*. The key function of a software engineering role is to determine responsibilities. Each role can be taken by many researchers, and each researcher can take many roles. The *domain expert* role here, as in other projects, expresses the involvement of many individual domain researchers. In WD-NACE, a

lot of data had already been collected by the various domain researchers. It is not generally constructive, or relevant, for most of the individual domain researchers to be involved in software engineering modelling, but all researchers must be able to contribute to the domain understanding, and, potentially, must be able to interpret simulation results. As in other projects [9, 25], it proved useful to designate a spokesman for the domain experts (John Forrester) who was responsible for liaison with the other WD-NACE researchers and interpretation of domain concepts.

The role of *modeller* was led by Richard Greaves, with input from Fiona Polack. Again, providing a lead modeller facilitates interaction and decision making, whilst allowing software engineering discussion to take place.

Discussion is currently under way about the role of *implementer*. In simulation development, the implementer is often the same person as the modeller, but this is not necessary, and is often not the case in software engineering. Separating modelling and implementation roles necessitates careful adherence to modelling and documentation standards, plus ongoing validation. A commitment to seamless development, or clearly documented mappings across abstraction levels, is crucial. It is essential that the implementer understand the purpose of simulation and the design decisions that underlie the domain and platform models.

Separation of Concerns (Roles): Roles assist separation of concerns, but are not the only element required. The domain experts seek to understand the complex interactions of the real social-ecological system, whilst the software engineering roles (modeller, implementer) seek only to capture the domain experts' understanding and intuition about the domain concepts in implementable ways.

Although different phases are led by different roles, the whole project must be seen as collaborative. The implementer and modeller must be prepared to seek guidance from, and discuss design implications with, the domain expert. The domain expert must be prepared to challenge details and quality of the models and implementation.

At the point where "CoSMoS modelling" was introduced, there was confusion in the WD-NACE project over the purpose of existing models: models of interacting agents, and of interacting feedback loops, existed, but attempts to find a simulation for these models were stalled. In CoSMoS terms, we identified the existing models as domain "sketches" – pictorial models used in the domain to express understanding, rather than diagrams with explicit semantics that could be used to create a software engineering artifact. Separation of concerns helps to keep the process of describing the domain separate from the process of domain

modelling. The software engineering diagrams are used for software engineering, not for domain exploration. Sometimes domain experts want to use UML-like notations (or even variants of the domain model diagrams) for their own modelling purposes. By treating domain-expert use of the diagrams a part of the domain, not part of the domain model, we remove the need to enforce syntactic and semantic consistency on models that will not be used in software engineering.

The difference between the use of notations by a domain expert and by modellers and implementers (software engineers) is summarised in table 1. Note that a key element in the fitness-for-purpose of a simulation is to be able to express the mappings between domain concepts and their representations in the domain model (and on into the platform model and simulation platform). This is essential information used in the validation and calibration of the simulation platform and interpretation of results.

Table 1. Observed differences between domain expert use of diagrams and software engineering use of diagrams. KISS and KIDS refer to "simplistic" and "anti-simplistic" modelling, see [11].

Domain: domain expert use	**Domain Model: software engineering use**
Well-defined notations not essential	Well-defined syntax and software engineering semantics essential
Any notational semantics may be ignored	Syntactic and semantic consistency must be checked
Diagrams express domain structures and interactions	Diagrams define potential computational structures and interactions
Exhaustive description and modelling (KIDS) advocated	Abstraction to what is needed for modelling purpose (KISS)

Justifications and rationale: CoSMoS encourages practicality over academic rationale. At each phase of the simulation project, practical solutions are needed to expedite good-quality research and a good-quality software engineering.

For example, in WD-NACE, the domain sketches focus on process, but there was little clarity over the components of the system. As usual in a CoSMoS (or software engineering) project, this led to some essentially arbitrary decisions about the domain modelling notation to be used. The domain expert and modeller were aware that some people use UML notations to design simulators; the modeller was familiar with the UML

class diagram notation; the domain expert felt that they could understand class diagrams. Thus, they decided to use UML notations for the WD-NACE models. It is important to note that here, as in many other projects, there is no academic justification for using UML – indeed, there are plenty of academic reasons not to use UML, such as the fact that most existing sketches were of process, not structure. The justification is purely pragmatic: the relevant individuals felt confident in using UML to express the domain model.

The decision about a modelling notation always has consequences. A software engineering notation has explicit semantics. The decision to use UML as-is (rather than a UML profile or domain specific variant of UML) means that the diagrams have a specific object-oriented semantics. For instance:

- a box on a class diagram represents a class; a class implies a set of objects that have inherent identity, conform to the type of the class, support the methods of the class, etc.
- a line on a class diagram represents an association; an association is a link that allows objects of one class access to public data and methods of linked objects (of another class).

In asking the domain expert to "validate" the class diagram, it is important that the domain expert understands that the notation here has an inherent semantics. For instance, if an association links two classes, this says no more than that the proposal for the simulator structure allows objects of the linked classes to find out about objects of the other class. The association does not express what happens in reality, and it does not express how the interaction would happen in the simulator.

Validation in context: CoSMoS calls on software engineers to create demonstrably fit-for-purpose software. This requires validation checks between levels of abstraction, and verification checks of models, such as the consistency checks described in Section 3.2. A full consistency analysis of this sort reveals syntactic inconsistencies, such as missing methods on classes. It also identifies underspecified elements, such as conditions or invariants that require methods or data not represented in the diagrams. The process of consistency analysis can prompt new insights or questions to domain experts, identifying misunderstandings or possible omissions and oversights in the domain model.

5 Discussion and Conclusions

Domain modelling in approaches such as CoSMoS is an iterative process. We have striven at every step to be clear about the assumptions made

in the creation of the model. The ideas incorporated into the model developed out of a simple initial model, enriched by guided use of relevant literature. Each draft is discussed with the relevant domain experts in Kenya, though the model remains to be reviewed by stakeholders.

The process seeks to make a model that is fit-for-purpose, and has well-understood mappings from (simplifications of) the observed reality. It is unclear whether this is what WD-NACE needs: social science does not always distinguish clearly when it is seeking a theoretical model, and when it wants to model reality. It is also unclear whether a model of reality can be used to guide policy and decision options: a model that is considered faithful to reality should faithfully express the effect of changes in policy and decisions that have been made, but, on the other hand, it is difficult to generalise from a model that is finely tuned to one society and ecosystem. These questions need to be addressed before we can draw any conclusions about the ultimate value of principled modelling and simulation in social-ecological systems research of this sort.

The domain model summarised here represents a unified social and ecosystems model that combines soft and physical elements. In many ways, this is just the start of the modelling of the WD-NACE domain. The extent to which soft elements are modelled is limited: we represent wealth and influence in the model, but do not give any detail of what these mean. We do not model any mechanism of decision making or influence. We could extend the model, so that the class diagram (Figure 7) includes classes that represent regional and local government responsible for policy, and research organisations that monitor the ecosystems and advise government organisations. However, this does not address the need to understand what the soft elements mean or how they interact with the physical elements of the model. Furthermore, the purpose of this domain modelling exercise was to represent the influences on fishing and reef ecosystem health; the extra organisations all operate through the local communities or villages, or on individuals already represented in the model. Alternatively, the WD-NACE researchers can use the existing domain model as a starting point for understanding how to incorporate existing social science models or measures of wealth and influence, and we can then adapt the domain model as necessary to support this. It is possible that parts of the STELLA model may prove to be more appropriate for the soft-element modelling of wealth and influence than the UML notations; this would require mapping between domain concepts represented in UML and STELLA. Alternatively, "influences" might be represented systematically as rule-bases, with values taken from databases of representative data [7].

[7] Suggested by WD-NACE researcher, Richard Taylor (SEI, Oxford)

The domain model here is expressed in UML. CoSMoS does not dictate UML. Here, the choice of notation reflected the specific experience of individuals involved in WD-NACE, and was purely practical. The CoSMoS philosophy supports pragmatic engineering, but, in this case, it is arguable that UML, and particularly the "standard" UML approach of starting by deriving a class diagram, is a poor choice. The existing STELLA model uses a metaphor of stocks and flows, and is essentially a behavioural model. The social-ecological system is a set of interacting behaviours. There is thus a significant level of interpretation needed to identify UML classes, which could inhibit checking of the new domain model by those trained in STELLA modelling. It would probably be useful for the modellers to work with the relevant domain experts to cross-check the STELLA model and the UML activity diagrams, and then to conduct a systematic consistency check between the UML activity diagrams and the other components of the UML domain model.

Overall, the CoSMoS philosophy, and guidance from software engineers who have used CoSMoS ideas in other simulation and modelling projects, has brought clarity to the WD-NACE domain modelling. However, it is still unclear whether, in the longer term, the software-engineering emphasis of the UML domain model described here is appropriate for the WD-NACE project: this is a good approach if a fit-for-purpose agent-based simulator is eventually required, but for domain understanding and expression, a less rigorously-enforced modelling approach (which could still use UML, but worry less about consistency checking and engineering correctness) might be sufficient.

Acknowledgments

The WD-NACE project supports the work of John Forrester and Richard Greaves. Fiona Polack's contribution builds on the work of the CoSMoS project, funded by UK EPSRC. WD-NACE is a Programme Framework Grant funded jointly by the UK Department for International Development, NERC and ESRC, under the Ecosystems Services for Poverty Alleviation (ESPA) programme. WD-NACE also supports the work of David Obura (CORDIO, Kenya), whose permission to use figure 5 is acknowledged with thanks.

References

[1] P. S. Andrews, F. A. C. Polack, A. T. Sampson, S. Stepney, and J. Timmis. The CoSMoS Process, version 0.1. Technical Report YCS-2010-450, Dept of Computer Science, Univ. of York, 2010. www.cs.york.ac.uk/ftpdir/reports/2010/YCS/453/YCS-2010-453.pdf.

[2] J. Barron, S. Noel, M. Malesu, A. Oduor, G. Shone, and J. Rockström. *Agricultural water management in smallholder farming systems: the value of soft components in mesoscale interventions.* Stockholm Environment Institute, 2008.

[3] P. Checkland and J. Poulter. *Learning for action: a short definitive account of soft systems methodology and its use for practitioner, teachers, and students.* Wiley, 2006.

[4] P. Collard and S. Mesmoudi. How to prevent intolerant agents from high segregation? In *ECAL*, pages 168–175. MIT Press, 2011.

[5] R. Costanza, D. Duplisea, and U. Kautsky. Ecological modelling on modelling ecological and economic systems with stella. *Ecological Modelling*, 110:1–4, 1998.

[6] R. Costanza and T. Maxwell. Spatial ecosystem modelling using parallel processors. *Ecological Modelling*, 58:159–183, 1991.

[7] S. Coulthard, D. Johnson, and A. McGregor. Poverty, sustainability and human wellbeing: A social wellbeing approach to the global fisheries crisis. *Global Environmental Change*, 21(2):453–463, 2011.

[8] B. de Vries and A. Petersen. onceptualizing sustainable development: an assessment methodology connecting values, knowledge, worldviews and scenarios. *Ecological Economics*, 68:1006–1019, 2009.

[9] A. Droop, P. Garnett, F. A. C. Polack, and S. Stepney. Multiple model simulation: modelling cell division and differentiation in the prostate. In *Workshop on Complex Systems Modelling and Simulation*, pages 79 – 112. Luniver Press, 2011.

[10] B. Edmonds. Bootstrapping knowledge about social phenomena using simulation models. *Journal of Artificial Societies and Social Simulation*, 13(1):8, 2010.

[11] B. Edmonds and S. Moss. From KISS to KIDS an 'anti-simplistic modelling approach. In *Multi Agent Based Simulation*, volume 3415 of *LNCS*, pages 130–144. Springer, 2005.

[12] M. Étienne. Companion modelling: a tool for dialogue and concertation. In *Biodiversity and stakeholders: concertation itineraries (Technical Notes 1)*, pages 44–52. Biosphere Reserves UNESCO-MAB, 2006.

[13] M. Fowler and K. Scott. *UML distilled: a brief guide to the standard object modeling language.* Addison-Wesley Longman, 2 edition, 2000.

[14] F. Franzén, G. Kinell, J. Walve, R. Elmgren, and T. Söderqvist. Participatory social-ecological modeling in eutrophication management: the case of Himmerfjärden, Sweden. *Ecology and Society*, 16(4):27, 2011.

[15] T. Ghetiu, F. A.C. Polack, and J. Bown. Argument-driven validation of computer simulations – a necessity rather than an option. In *VALID*, pages 1–4. IEEE, 2010.

[16] V. Grimm, U. Berger, F. Bastiansen, S. Eliassen, V. Ginot, J. Giske, J. Goss-Custard, T. Grand, S. Heinz, G. Huse, A. Huth, J. U. Jepsen, C. Jørgensen, W. M. Mooij, B. Müller, G. Peer, C. Piou, S. F. Railsback, A. M. Robbins, M. M. Robbins, E. Rossmanith, N. Rüger, E. Strand, S. Souissi, R. A. Stillman, R. Vabø, U. Visser, and D. L. DeAngelis. A

standard protocol for describing individual-based and agent-based models. *Ecological Modelling*, 198:115–126, 2006.

[17] V. Grimm, U. Berger, D. L. DeAngelis, G. Polhill, J. Giske, and S. F. Railsback. The ODD protocol: A review and first update. *Ecological Modelling*, 221(23):2760 – 2768, 2010.

[18] E. Kemp-Benedict, S. Bharwani, and M. Fischer. Using matching methods to link social and physical analyses for sustainability planning. *Ecology and Society*, 15(3), 2010.

[19] M. Kiran, P. Richmond, M. Holcombe, L. Shawn Chin, D. Worth, and C. Greenough. FLAME: simulating large populations of agents on parallel hardware architectures. In *AAMAS*, pages 1633–1636, 2010.

[20] D. Malleret-King. *A Food Security Approach to Marine Protected Area Impacts on Surrounding Fishing Communities: the Case of Kisite Marine National Park in Kenya*. PhD thesis, Biological Sciences, University of Warwick, 2000.

[21] S. C. Mangi and C. M. Roberts. Quantifying the environmental impacts of artisanal fishing gear on Kenya's coral reef ecosystems. *Marine Pollution Bulletin*, 52:1646–1660, 2006.

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

[23] F. A. C. Polack. Arguing validation of simulations in science. In *Workshop on Complex Systems Modelling and Simulation*, pages 51–74. Luniver Press, 2010.

[24] F. A. C. Polack, P. S. Andrews, T. Ghetiu, M. Read, S. Stepney, J. Timmis, and A. T. Sampson. Reflections on the simulation of complex systems for science. In *ICECCS*, pages 276–285. IEEE Press, 2010.

[25] F. A. C. Polack, A. Droop, P. Garnett, T. Ghetiu, and S. Stepney. Simulation validation: exploring the suitability of a simulation of cell division and differentiation in the prostate. In *Workshop on Complex Systems Modelling and Simulation*, pages 113 – 133. Luniver Press, 2011.

[26] N. Powell and M. Osbeck. Approaches for understanding and embedding stakeholder realities in mangrove rehabilitation processes in Southeast Asia: lessons learnt from Mahakam Delta, East Kalimantan. *Sustainable Development*, 18(5):260–270, 2010.

[27] T. C. Schelling. Dynamic models of segregation. *Journal of Mathematical Sociology*, 1(2):143–186, 1971.

Appendix: Validation of `Person` Modelling

This appendix presents a summary of consistency checks described in section 3.2, focusing on the `Person` class and associated UML state diagram. The consistency checks systematically consider the representation of common abstract-syntax concepts (based on the UML 2.x metamodel

concepts[8]) between and within the models. In each table, entries preceded by ** explain how the consistency requirement is, or could be, met.

Table 2. Results of consistency checks (Section 3.2) on the `Person` state diagram (Figure 8) and class diagram (Figure 7)

State diagram element	**Consistency summary**
Alive state	not needed: any `Person` object currently represented in the system is alive
Other states and substates	`Person.role` records the current status of the object. The domain of `Person.role` is an enumeration, $\{child, student, ill/dependent, domestic, farmer, fisher, trader, career, labourer, tourism\}$ ** `Person` has an operation, `changeRole()` to change the value of this attribute; this needs to be called as an action on the transitions between these states
Condition, *household can pay*	** Condition needs a threshold calculation using `Household.wealth` and the roles/incomes of the people who make up the household ** `Household` needs an operation to calculate its ability to pay for education
Condition, *Can pay*	** Condition needs a threshold calculation using `Person.wealth` ** `Person` needs an operation to calculate its ability to pay for education
Condition, *Has farm*	assumed to be a precondition of the operation to change status: the precondition can be fulfilled by prompting for confirmation that the Person object of which the role is being changed to $farmer$ has access to land ** define precondition on `Person.changeRole()` where the target role is `Farmer`

[8] www.omg.org/spec/UML/

Condition, *Has stock*	assumed to be a precondition of the operation to change status: the precondition can be fulfilled by prompting for confirmation that a Person object of which the role is being changed to *trader* has access to stock ** define precondition on `Person.changeRole()` where the target role is `Trader`
Condition, *Has boat / gear*	assumed to be a precondition of the operation to change status: the precondition can be fulfilled by prompting for confirmation that a Person object of which the role is being changed to $fisher$ has access to a boat or fishing gear ** define precondition on `Person.changeRole()` where the target role is `Fisher`
Condition, *Has qualifications*	assumed to be a precondition of the operation to change status: the precondition can be fulfilled by prompting for confirmation that a Person object of which the role is being changed to *career* has the appropriate qualifications ** define precondition on `Person.changeRole()` where the target role is `Career`

Table 3. Results of consistency checks (Section 3.2) on the class diagram (Figure 7)

Class diagram element	**Consistency summary**
`Person.wealth`	attribute recording a measure of wealth ** need to define wealth, and determine a domain for this attribute ** need operations on `Person` that set or calculate and update wealth
`Person.role`	see above
`Person.fishing`	Boolean attribute recording whether the person is currently engaged in fishing ** need an invariant that `Person.fishing` is $false$ unless `Person.role` is $fisher$
`Person.changeRole()`	see above

`Person.decideToFish()`	operation assessing overall influences and determining whether `Person.fishing` should be *false* or *true*: the operation executes periodically, and if it establishes a change to *true*, initiates a change of role (which checks preconditions and sets `Person.role` as appropriate) ** need to establish how `influences` (association ends linking `Person` object to their `Household`, `Village` and `Community`, and any other `Person` objects in `kin` associations with the object) are used to determine fishing activity, and to express this in operation details ** need relevant operations on `Household`, `Village` and `Community` to provide required information to `Person` ** need attribute and operations on `Person` to represent how that `Person influences kin Person` objects
`Person.doFishing()`	operation that models the fishing activity of a `Person` whose `role` is *fisher* and for whom `fishing` is *true* ** need to determine how the operation affects `Fish Stock.population` and `Reef.ecosystem health` ** need relevant operations on `Fish Stock.population` and `Reef.ecosystem health` to provide any required information to `Person`
association end, `damages`	`Person` calls `Reef.damageHealth()` with appropriate parameter values. This operation call is directly related to the effect of `Person.doFishing()` (above)
association end, `depletes`	`Person` calls `FishStock.depletePop()` with appropriate parameter values. This operation call is directly related to the effect of `Person.doFishing()` (above)
association end, `impact wealth`	`Reef` calls an operation on `Person` that modifies `Person.wealth` according to relevant parameters based on `Reef.ecosystem health` with appropriate parameter values. The operation only applies if `fishing` is *true* ** the association end would call the wealth-modification operation noted above

Table 4. More results of class diagram (Figure 7) consistency checks (Section 3.2)

Class diagram unnamed association ends	**Consistency summary**
with `Person`	association end identifies `kin` objects whose influence should be heeded
with `Household` (aggregation)	association end identifies `Household` to which a `Person` object belongs, whose influence should be heeded
with `Household`	`Person` calls `Household.promptFishDecision()` to access information needed to determine whether to fish or not: uses aggregation link to identify relevant household, and passes result as a parameter of `Person.decideToFish()`
with `Village` (aggregation)	association end identifies `Village` to which a `Person` object belongs, whose influence should be heeded
with `Village`	`Person` calls `Village.promptFishDecision()` to access information needed to determine whether to fish or not: uses aggregation link to identify relevant household, and passes result as a parameter of `Person.decideToFish()`
with `Community` (aggregation)	association end identifies `Community` to which a `Person` object belongs, whose influence should be heeded
with `Community`	`Person` calls `Community.promptFishDecision()` to access information needed to determine whether to fish or not: uses aggregation link to identify relevant household, and passes result as a parameter of `Person.decideToFish()`

A Pattern Language for Scientific Simulations

Susan Stepney

Department of Computer Science, University of York, UK

Abstract. For computer-based simulations to be scientifically useful and scientifically credible, they need to be developed to high standards, and argued fit-for-purpose. The CoSMoS project has developed an approach to support such development, and codified its approach in a pattern language. Here we overview this pattern language, and discuss several example simulation development patterns and antipatterns.

1 Introduction

Computer-based simulation is a key tool in many fields of scientific research. In silico experiments can be used to explore and understand complex processes, to guide and complement in vitro and in vivo experiments, to suggest new hypotheses to investigate, and to predict results where experiments are infeasible. Simulation is an attractive, accessible tool: producing new simulations of simple systems is relatively easy. But it is also a dangerous tool: simulations are often complex, buggy, and difficult to relate to the real-world system.

A simulation needs to be both scientifically useful to the researcher, and scientifically credible to third parties; it needs to have the properties of a well-designed *scientific instrument*. The CoSMoS project has been developing an approach to simulation of complex systems that supports such development of simulations as a scientific instruments. The CoSMoS approach emphasises two key aspects: the use of models to capture the scientific domain and the simulation platform; and the close co-working of scientific domain experts and simulation software engineers. This requires the development of a suite of models, of the scientific domain, of the simulation platform, and of the simulation results, in addition to the simulation platform implementation. It also provides an approach for developing a rigorous argument of "fitness for purpose" of the simulation for its intended task.

The CoSMoS approach is generic: it does not mandate a particular modelling technique, or particular implementation language. What it

does mandate is the careful and structured use of models and arguments, to ensure that the simulation both is well-engineered, and seen to be well-engineered. In order to help developers through this careful and structured approach, we have developed a pattern language to help guide development, promote good simulation engineering practice, and warn of potential pitfalls. This paper overviews the CoSMoS pattern language.

The structure of the rest of the paper is as follows. Section 2 overviews the CoSMoS approach and its main features and components. Section 3 overviews the pattern language approach, and defines the pattern templates used in this paper. Section 4 presents several specific example patterns and antipatterns. Section 5 concludes.

2 Overview of the CoSMoS approach

The CoSMoS approach enables the construction and exploration of simulations for the purpose of scientific research. It has been designed to be adaptable both to a variety of simulation problems and to changing circumstances during simulation construction and use. Application of the approach should be tailored to suit the criticality and intended impact of the research outcomes.

The construction and use of simulations is a necessarily interdisciplinary endeavour between scientists who study a particular domain (the *domain experts*), and software engineers who construct simulations to facilitate the study of that domain (the *developers*). Together, the domain experts and developers are involved in open-ended scientific research: the simulations are used as a tool to support theory exploration, hypothesis generation, and design of real-world experimentation.

To run computer simulations we need to engineer a *simulation platform*. A properly calibrated simulation platform is the scientific instrument, the basis for running multiple *simulation experiments*. To engineer such a platform requires us to explicitly represent some knowledge of the system being studied in a form that can be implemented on a computer. This representation, the source code, is either designed manually by the developers or automatically generated from a higher-level description.

In many existing approaches the source code is the only explicit description of the aspects of the target domain that are being simulated. Source code contains numerous implicit assumptions (including abstractions, simplifications, axioms, idealisations, approximations) concerning both the scientific aspects of the work, and the engineering design of the simulation platform. Source code also contains many implementation details, which are needed to make the simulation run on a computer, but

are not part of the underlying scientific model. Hence source code is not a satisfactory basis for modelling.

To mitigate inappropriate assumptions in the design of simulation platforms, and to have greater confidence that simulation results can actually tell us something that relates to the real system being studied, we use a series of related models to drive and describe the development of the simulation platform and simulation results generated from its use. Systematic development assists interaction between domain experts and developers, and improves our confidence in, and interpretation of, the results of simulations.

2.1 Phases

We identify three main phases in a simulation project.

Discovery, or "deciding what scientific instrument to build". This establishes the scientific basis of the project; identifies the domain of interest, models the domain, and sheds light on scientific questions.

Development, or "building the instrument". This produces a simulation platform to perform repeated simulation, based on the output of discovery.

Exploration, or "using the instrument in experiments". This uses the simulation platform resulting from development to explore the scientific questions established during discovery.

These phases are not intended to be performed purely sequentially. A project naturally begins with a discovery phase followed by development and then exploration. But many iterations of discovery, development and exploration may be required to build a robust, fit for purpose instrument. The separation into phases helps provide a focus on what particular pieces of information are needed at each phase for each model.

Indeed, some projects might not perform all phases. A prior project may have performed the necessary discovery, and only development and exploration is needed (although it will be necessary to check that the assumptions of the prior discovery phase are valid for this project). Similarly, a suitable existing simulation platform might exist, and only the exploration phase is followed in this project (again, it will be necessary to check that the assumptions underlying the existing simulation platform are valid for this project). On the other hand, it may be that only the discovery phase occurs, and discovers that a simulation is not appropriate, or not needed.

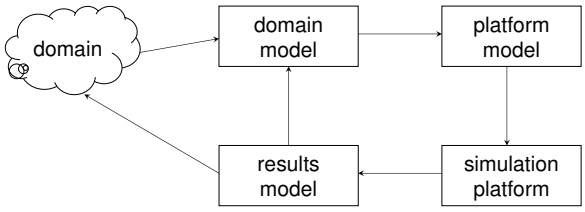


Fig. 1. Relationship between simulation components; arrows represent flows of information. These are all framed by the research context.

2.2 Models

Our simulation approach uses the following model concepts (figure 1): domain, domain model, platform model, simulation platform, and results model.

Each of these components has a different role to play in the building, verifying, and use of the simulation:

Domain represents the real-world system of study.

Domain Model encapsulates understanding of appropriate aspects of the domain. It focuses on the scientific understanding; no simulation implementation details are considered.

Platform Model comprises design and implementation details for the simulation platform, based on the domain model concepts.

Simulation Platform encodes the platform model into a software and hardware platform with which simulation experiments can be performed.

Results Model encapsulates the understanding of outputs and results from simulation experiments, in domain terms, enabling comparison with results from domain experiments.

2.3 Experiments

The models described above are used to build the simulation platform. The platform can be thought of as a computational implementation of the model of the real world system under study.

The simulation platform can be used to run simulation experiments that are analogies of the real world experiments run in the domain. The results of a simulation experiment (after suitable translation into domain terms, and data analysis, via the results model) can be compared to the real world experimental results (see figure 2; the later Data Dictionary and Calibration pattern descriptions have further details).

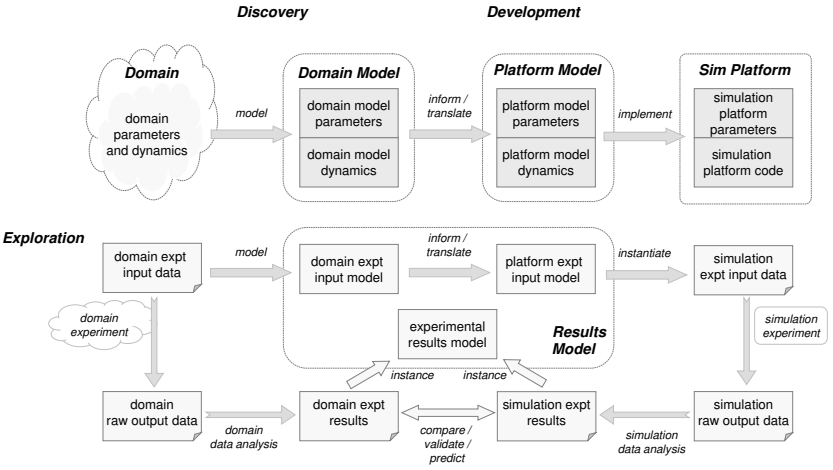


Fig. 2. The relationship between the various models and phases, and how the simulation platform is used to perform simulation experiments. See text for details.

Initial runs are used to calibrate the simulation platform. This is needed to determine how to *translate* domain parameters and variables into their corresponding platform values (for example how to translate between real-world time, and simulated time), and how to take simulation experiment raw output data and *analyse* it to enable *comparison* with domain results.

Subsequent runs can be used to *validate* the simulation. If these disagree with domain experiments, it may be because:

- the variables and parameters are not being translated appropriately (calibration may have *overfit* their values)
- there are faults in the platform model or in the simulation platform implementation (the simulation platform has not been adequately engineered)
- there are faults in the domain model (the science is imperfectly understood)

Once the simulation has been validated, experiment runs can be used to make *predictions* about the results of domain experiments. Even in such a case, predictions should be checked against real world data, particularly if the simulation experiment is being run outside the calibration range of the instrument.

2.4 Arguments

To build confidence in a particular simulation-based study, the team needs to argue the appropriateness of the entire simulation project (including modelling and simulator development, input data, and analysis of results). This requires an argument, based on evidence, that the simulation platform is fit for purpose, and is being used appropriately to perform the simulation experiments. This argument can be used to drive the shape of the simulation development process: it is easier to argue a system is fit for purpose if the development has been guided with such a need in mind, and the system is more likely to be valid if it has been structured in such a way.

We use the terms "fit for purpose" (with the meaning "good enough to do the job it was designed to do" [26]) and "appropriate" for our argument structure. These terms emphasise that they are *relative*, to the simulation purpose, and hence that there is a need to revisit arguments should that purpose change. We choose not use more common terminology such as "valid" or "correct". These terms have implications of being absolute terms: "this instrument is correct", as opposed to "this instrument is appropriate *for a given purpose*". Hence these terms do not capture the need to revisit arguments if circumstances change. Additionally, they have implications of being either true or false: something is either "valid" or "invalid", whereas we want to capture a continuum of possibilities, allowing a simulation platform to have degrees of fitness for purpose.

An appropriateness argument is usually incomplete: its purpose is to capture the understanding about fitness for purpose of its audience, so that it can be referenced in future, challenged and revisited. A thorough and fully documented argumentation exercise is unnecessary in most situations, particularly in cases where the simulation criticality is low.

As well as documenting what you do, and arguing that it is the right thing to do, it is important to document what you don't do, and argue why it would be wrong to do it. This saves much grief later in the project, when a previously dismissed approach is retried, and the reason for its dismissal rediscovered.

There are two approaches to arguing the fitness for purpose: retrospectively, after the simulation platform has been developed, or incrementally, as the development of the simulation platform proceeds.

3 Patterns

As can be seen, the CoSMoS approach has many components: phases, models, implementations, arguments. In order to help structure a sim-

ulation project, the approach is captured in the CoSMoS pattern language [33]. This pattern language provides the structure, detail, and rationale for developing all the necessary components, to aid the developer in producing a high quality, scientifically viable simulation instrument.

In 1977, Christopher Alexander and his co-authors published *A Pattern Language* [1], one in a series of books "intended to provide a complete working alternative to our present ideas about architecture, building, and planning". It is a handbook of 253 patterns, where "Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice." [1, p.x]. The patterns describe how quality buildings should be designed, and together provide a language covering a wide range of spatial scales, from whole towns, through small clusters of buildings, and individual buildings, to tiny detailing.

And that, as far as the computing community goes, would have been that, were it not that the concept of Patterns inspired a group of software engineers. Buildings are not the only things described by "architecture": software engineering uses the same word to describe its own structuring concepts. In 1995, the so-called "Gang of Four" published *Design Patterns* [13], which took Alexander's concept and applied it, to produce a catalogue of patterns found in good software architectures. Things have not looked back: there are now analysis patterns [11], coding patterns [5], patterns conferences and catalogues [7, 21, 27, 38], antipatterns [6, 25], metapatterns (patterns that describe patterns), and more (including arguments that the whole software patterns community have completely missed Alexander's point [12]).

The initial flurry of publications may have slowed somewhat since those early days, but Patterns are now part of the everyday culture of software engineering. One impact of Alexander's ideas, as adapted by the Gang of Four, on software development has been to make it clear that there is much more to object-oriented architecture than just the single concept of an object. The patterns provide a simple vocabulary, letting us all talk of the Visitor Pattern, or the Factory Pattern [13], without having to explain what we mean.

3.1 Pattern template

It is important that a pattern is a practical, tried-and-tested solution to a problem, not merely something the pattern writer hopes or theorises might be a good solution.

We use the following template to document a pattern:

Pattern Template

Intent

What the pattern is for; what its use will achieve.

Context

The place or circumstance where the pattern is applicable.

Discussion

An explanation of what the pattern provides, and how to use it. This may include references out to other patterns (formatted as the name of the pattern, followed by the page number where it is defined).

Summary

A pithy summary of how to achieve the pattern's intent.

Related patterns

A list of related patterns (not otherwise mentioned in the body) and antipatterns (common mistakes that may be made when applying this pattern).

We refer to a pattern in the text by its name, in sans serif font: Pattern Template. In the pattern catalogue, we also give the page number where the referenced pattern is documented. However, in this paper, only a few patterns are documented, so instead we provide appendix A documenting its intent.

A pattern language provides a *vocabulary* for talking about a problem situation. This is analogous to the manner in which the names of the modelling concepts in Domain Driven Design [9] provide a vocabulary for talking about a software system. A full pattern language is more than just a vocabulary, however. As in Alexander's original work [1], a full language is *morphogenetic*, in that it provides a way composing patterns to building a full solution to a problem. Patterns refer to other patterns, and the consequences of using one pattern impact what other patterns are relevant and applicable. Such a full pattern language is much harder to develop; the CoSMoS pattern language provides some such structure, but is not a fully morphogenetic language yet.

3.2 Anti-pattern template

Patterns provide guidance on what to do. It is just as important to give guidance on what *not* to do, particularly when this superficially appears to be a good idea, a clever shortcut, a sensible compromise, or even just normal practice. Antipatterns [6, 25] provide a means to give such guidance. An antipattern documents a pattern of bad behaviour or an often repeated mistake, together with a solution of what to do instead, or how to recover from the mistake. The solution is often a pointer to which pattern(s) to use instead.

We use the following template to document an antipattern:

Antipattern Template

Problem

What the problem is.

Context

The place or circumstance where the mistake is often made.

Discussion

Further discussion of the problem.

Solution

A pithy summary of what to do instead, or how to recover from the mistake.

We refer to an antipattern in the text by its name, in italic sans serif font: *Antipattern Template*. In the pattern catalogue, we also give the page number where the referenced antipattern is documented. However, in this paper, only a few antipatterns are documented, so instead we provide appendix B documenting its problem statement.

In addition to antipatterns of the form "doing the wrong thing", antipatterns can often appear in pairs (for example, *Analysis Paralysis* and *Premature Implementation*) where one of the antipatterns is "doing too much" and its pair is "doing too little".

4 Example patterns and antipatterns

This section describes and discusses a few selected examples of specific patterns and antipatterns, to help illuminate both the CoSMoS approach, and the pattern language approach. The high level Research Context pattern is important for setting the scope of a simulation project. The more detailed Data Dictionary and Calibration patterns discuss some of the finer points about ensuring that any experiments performed using the simulation platform can be related to domain concepts and results. Finally, the *Amateur Science* and *Proof by Video* antipatterns warn against some problems that can occur when the underlying scientific purpose for building the simulations is forgotten.

4.1 High level patterns

Patterns are used to capture the overall high-level structure of the CoSMoS approach. The top level CoSMoS Simulation Project is summarised as:

- carry out the Discovery Phase
- carry out the Development Phase
- carry out the Exploration Phase

As noted earlier, not all these phases need be carried out in all projects. Variants and options allow different routes to be followed through the pattern language.

Drilling down one level, the Discovery Phase is summarised as:

- identify the Research Context
- define the Domain
- build a Domain Model
- Argue Appropriate Instrument Designed

The discussion accompanying these summaries captures the concepts overviewed in §2. These summaries, outlining what is required to achieve the pattern's intent, are to be read in a declarative, rather than sequential manner. The pattern says what needs to be achieved; the subpatterns say how to achieve each part; but there is no requirement (beyond certain dependencies) placed on the order these things need to be done.

As an example of a full high-level pattern, we present the Research Context.

Research Context

Intent

Identify the overall scientific context and scope of the simulation-based research being conducted.

Context

A component of the Discovery Phase, Development Phase, and Exploration Phase patterns. Setting (and resetting) the scene for the whole simulation project.

Discussion

The role of the research context is to collate and track any contextual underpinnings of the simulation-based research, and the technical and human limitations (resources) of the work.

The research context comprises the high-level motivations or goals for the research use, the research questions to be addressed, hypotheses, general definitions, requirements for validation and evaluation, and success criteria (how will you know the simulation has been successful).

The scope of the research determines how the simulation results can be interpreted and applied. Importantly, it captures any requirements for validation and evaluation of simulation outputs. It influences the scale and scope of the simulation itself.

Consideration should be made of the intended criticality and impact of the simulation-based research. If these are judged to be high, then an exploration of how the work can be validated and evaluated should be carried out.

Determine any constraints or requirements that apply to the project. These include the resources available (personnel and equipment), and the timescale for completion of each phase of the project. Any other constraints, such as necessity to publish results in a particular format (for example, using the ODD Protocol), should be noted at this stage. This helps ensure that later design decisions do not violate the project constraints. Ensure that the research goals are achievable, given the constraints.

As information is gathered during the project, more understanding of the domain and the research questions will be uncovered. For example, a Prototype might indicate that a simulation of the originally required detail is computationally infeasible. The Research Context should be revisited between the various phases, and also at any point where major

discoveries are made, in order to check whether the context needs to change in light of these discoveries.

Summary

- document the research goals
- Document Assumptions relevant to the research context
- identify the team members, including the Domain Expert, the Domain Modeller, and the Simulation Implementor, their roles, and experience
- agree the Simulation Purpose, including criticality and impact
- note the available resources, timescales, and other constraints
- determine success criteria
- revisit between phases, and at discovery points; if necessary, change the context, and Propagate Changes

Related patterns

The research context scopes what should go in the models and simulation: beware of modelling *Everything but the Kitchen Sink*.

It is important to identify if, when, why and how the research context changes throughout the course of developing and using the simulation. Beware of *Moving the Goalposts*.

4.2 Detailed patterns

Here we present two example related detailed patterns, that of the Data Dictionary, and that of Calibration as mentioned in the Data Dictionary.

Data Dictionary

Intent

Define the modelling data used to build the simulation, and the experimental data that is produced by domain experiments and the corresponding simulation experiments.

Context

A component of the Domain Model, Platform Model, and Results Model.

There is observational data that is present in the Domain Model. It needs to have instrumentation provided for in the Platform Model and the

Simulation Platform, to extract the analogous data from the simulation. This model is also used to capture the simulation outputs as part of the Results Model.

Discussion

The Domain pattern includes identification of the data sources that populate the Data Dictionary. There are two kinds of data in this model:

1. *modelling data (parameter values)*: used to parameterise the various models, by providing numbers, sizes, timescales, rates, and other system-specific values; this usually comes from the raw data from previous experiments, analysed and reduced using previous models and theories.
2. *experimental data*: comprising the input values and output results of the domain experiments and corresponding simulation experiments; this is broken into three parts:
   (a) Calibration data, for setting and tuning the platform parameter values
   (b) validation data, to allow the calibrated simulation platform to be validated against the Domain Model
   (c) unseen (predicted) data

The separate Calibration and validation data sets are analogous to the training and test data sets used in machine learning [18]. This approach ensures that the simulation is not so tuned that it "overfits" the calibration (training) data, but is generic enough to also fit the (unseen during calibration) validation data.

In some systems there may be insufficient experimental data to perform calibration and validation. If so, an argument should be used to demonstrate why this is not considered to be a problem.

Experimental data may be of varying quality. It may be a set of particular experimental values, with well-characterised errors and a measured statistical distribution. Or it may be more qualitative, such as "quantity A is bigger than B", "event C occurs before D". Different qualities of data will require different calibration comparisons.

If the simulation has high criticality (determined from the Research Context), it would be reasonable to require a further set of truly unseen validation data, to form the basis for an "acceptance test", before the system is used in any critical predictive capacity.

Domain values might be directly used in the platform model and simulation platform. For example, environmental parameters such as rainfall rates in an ecological simulation, or robot sensor data in an engineering simulation.

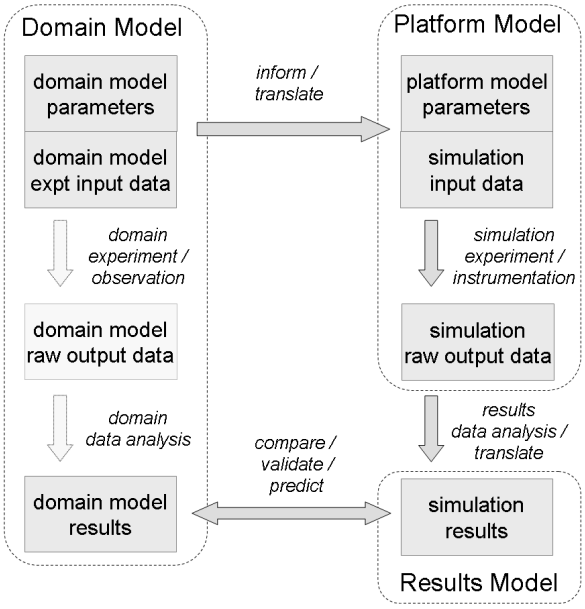


Fig. 3. The components of the Data Dictionary in the Domain Model, and how they relate to components in the Platform Model and the Results Model.

Domain model parameters and data values are not necessarily identical to the platform model parameters and data values, however. For example, a single value in a simulation could well be a proxy for a number of values in the domain. So there needs to be a well-defined translation mapping of these values between models, captured by the "informs" arrow in figure 3. Similarly, the data output from a simulation run, captured and analysed in the Results Model, needs to be translated into Domain Model terms. The form of these translations is guided by the translation of domain model concepts to platform model concepts, and the precise structure is determined by Calibration runs. Translation back from results model to domain model equivalents (interpreting output in real-world terms).

Once the simulator has been calibrated and validated, it can be used to generate data for novel scenarios, to make predictions; the domain model can potentially be augmented with new experimental data to test those predictions.

It is possible to extract much more information from a simulation than from a biological experiment, say, but if it is not observable (even

indirectly, through surrogates, or by investigating predictions) in the domain model, it is of little use.

The necessity for suitable data in the Results Model implies requirements on the Platform Model: it must be of a form that can produce the required data, and must be suitably instrumented to output the data.

This careful separation of modelling data (used to build the model) and experimental data (to be produced by the domain experiment or analogous simulation experiment) is important, in order not to *Program In the Answer*.

Summary

- build a model of the modelling data, used to build the simulation
- build a model of the experimental data that will provide the comparison between the Domain Model and the Results Model; include considerations of data quantity and quality
- determine whether the domain experimental data is of sufficient quantity and quality to provide adequate calibration, validation (and if critical) unseen acceptance test data sets
  - either: argue that the domain experimental data is sufficient
  - or: argue why apparently insufficient data is not a problem in this case

Related patterns

Visualisation Model, for presenting experimental output data to the user.

Calibration

Intent

Tune the Simulation Platform parameter values so that simulation results match the calibration data provided in the Data Dictionary.

Context

A component of the Simulation Platform pattern.

Discussion

Calibration is a standard part of the manufacture and deployment of any scientific instrument. It often refers to setting the correct zero point

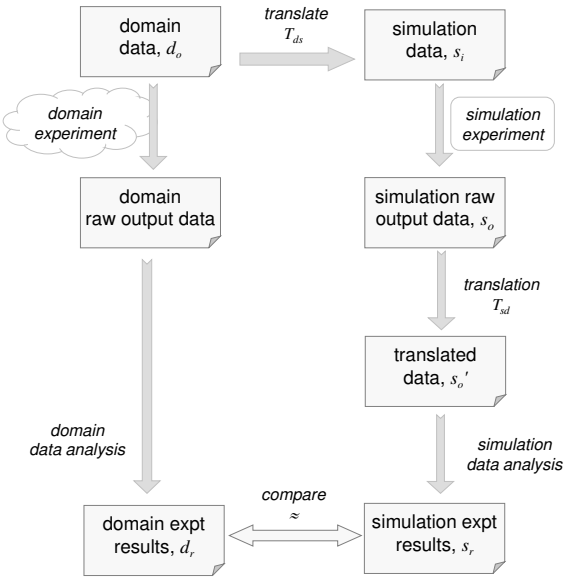


Fig. 4. Summary of the components of the data model in the Domain Model, how they relate to components in the Platform Model and the Results Model, and the relationship that calibration aims to achieve.

and scale. Physical scientific instruments may need to be recalibrated if environmental conditions change (such as temperature causing expansion of parts of the device). Simulation scientific instruments should only need to be calibrated once before use, but do need to be recalibrated if the simulation platform is changed in any way (see *Tweaking*).

Calibration is required in order to bring the Simulation Platform to an experimentation-ready state. Uncertainties in parameters (and potentially in sub-models) are addressed by exploring the parameter space (or trying different sub-models), in order to obtain from the simulation platform outputs in agreement with calibration data. Calibration can be performed through simple, manual adjustments or more elaborate fitting, e.g. GAs, gradient techniques.

The various kinds of data involved are part of the Data Dictionary. Figure 3 shows the various data components in detail. The calibration data is used to adjust the translations and parameter values until the Results Model data fits the Domain Model experimental results. Figure 4 shows a summary of this, indicating what the calibration exercise affects.

The Domain Model has input data d_i (comprising both parameter values and experimental data). A domain experiment based on this experimental data will produce raw output data. After the appropriate scientific data analyses, this yields the domain results data, d_r, conforming to the Results Model.

To move to the simulated world, the domain data needs to be translated to appropriate Simulation Platform values s_i, using T_{ds}. A simulation experiment given input data s_i (simulation parameters and experimental setup), will produce raw simulation data s_o. This needs to be translated back into domain world terms, using T_{sd}, and then similarly analysed into the Results Model, to yield the simulation results s_r.

The calibration exercise is to adjust the translation functions T_{ds} and T_{sd} to achieve $d_r \approx s_r$. The relationship between domain and simulated results need not be exact equality, but can be statistical similarity, or qualitative agreement; the achievable relationship depends on experimental data quantity and quality. The domain and simulation experiments are not functions in the mathematical sense, since different experimental runs on the "same" input data will yield different output data, due to variation, experimental error, and stochasticity.

If the parameters are time varying, there is also the need to translate from domain time to simulation time.

Calibration is a "data-fitting" process: translation function parameters are tuned so that the simulation adequately reproduces the calibration data. As such, common data-fitting issues such as "overfitting" need to be avoided. In particular, the *form* of the translation functions should not be arbitrarily fitted; their design should be constrained and guided by the kinds of changes made moving from domain to platform models.

The translation, T_{ds} may be relatively trivial (not much more than the identity transformation) if the domain and platform models are very similar. However, it might be sophisticated, if the platform model has introduced differences, such as surrogate entities standing in for multiple domain entities, change of dimension, non-trivial discretisation, and so on. The complexity of the back-translation, T_{sd} will mirror that of T_{ds}. If information is lost by T_{ds} that cannot be regained by some T_{sd}, then the Domain Model must be defined so as not to need this information, and the domain data analysis process will also lose it.

One technique that can be used to help calibrate surrogates is to express parameters in terms of dimensionless quantities (for example, the Rayleigh Number or the Reynolds Number) to minimise the effect of unit choices and other changes.

The calibration data has to be selected to ensure good calibration. It should be of broad enough span that the planned Simulation Experiments will not be using the Simulation Platform "out of calibration". The validation data should have a similar span, and be kept separate and independent of the calibration exercise to ensure a fair validation. The accompanying argument should cover the choice, span, and independence of this data.

Running experiments "out of calibration" (that is, in an area of experimental space not well covered by the calibration data) should be done with caution. One reason for doing so it to explore which might be the most fruitful areas for further domain experiments.

Calibration might not succeed: it might not be possible to tune the simulation parameters to make the Simulation Experiment results conform sufficiently to the domain experiment results. This could indicate a problem with the Domain Model, such as missing component or mechanism, or with the Platform Model, such as poor discretisation, or inappropriate approximations. In the simplest cases, the relevant model should be changed (for example, by making some components platform higher fidelity to better simulate the corresponding domain components), remembering to Propagate Changes, recalibrate, and reargue as appropriate. In more extreme cases, further domain experiments or hypotheses might be needed to gain a more adequate Domain Model.

Summary

- select the calibration and validation data
- perform calibration, to produce a calibrated Simulation Platform suitable for performing Simulation Experiments
  - determine the translation from domain data to simulation input data
  - determine the translation from simulation output data to domain data
  - run calibration Simulation Experiments, tuning parameter values until the platform results match domain results to the required accuracy
  - fix these tuned parameter values in the calibrated Simulation Platform
- use the validation data to ensure that calibration has not overfitted the Simulation Platform
- argue that the calibration is appropriate for purpose

Related patterns

Beware of *Living in Flatland*.

Do not confuse calibration with Sensitivity Analysis.

With these more detailed patterns, we see that not every item in the summary is a further, more detailed pattern. Eventually, we reach "primitive" tasks that do not require a pattern themselves, either because they are simple, or because they are well-known tasks for which there is an adequate literature.

4.3 Antipatterns

Here we present two example antipatterns, *Amateur Science*, which can happen at the early stages of a simulation project, and *Proof by Video*, which tends to happen later on.

Amateur Science

Problem

You do not engage with a domain expert, because you think you know the domain science well enough.

Context

Building the Domain Model; making simplifying assumptions in the Platform Model; performing platform Calibration; building the Results Model; running a Simulation Experiment.

Discussion

While modelling it can be easy to use your own understanding of the domain, rather than referring to the domain expert or relevant literature. This understanding is, however, nearly always oversimplified and at too shallow a level: even if a domain looks relatively straightforward from the outside, it can have hidden subtleties and traps. After all, if it really were that simple, there would be no need for a simulation instrument.

If you are finding it difficult to Document Assumptions about the Domain or Domain Model, you may be engaged in *Amateur Science*. The next step up in sophistication is to fall into the *Literature Only* antipattern.

Solution

Engage with the Domain Expert, who will soon make it clear that the real world domain "is more complicated than that". But beware of *Blind Trust* in the expert.

Proof by Video

Problem

The Visualisation Model is all there is.

Context

Building a Results Model during the Exploration Phase

Discussion

The visualised results from the simulation look superficially similar to those from the domain (be it a static figure or an animation), and so you judge the simulation to be a "success". But there is no quantification of the similarity of the results, so you cannot be sure the correspondence is more than an optical illusion, and you cannot make any quantitative statements or predictions.

Solution

Analyse the data from the simulation experiment, and compare the results quantitatively with domain experiment results, as specified by the Data Dictionary. Argue how the comparison validates the simulation results.

Not to be confused with Debug By Video.

It is worthwhile to keep these antipatterns in mind, to help guard against problems. The "related patterns" section of individual patterns can warn of potential antipatterns relevant to that pattern (as in the Calibration case). For an antipattern of the form "doing too little", it can also warn against the paired "doing too much" antipattern (as in the *Amateur Science* case), and vice versa.

5 Discussion and conclusions

The CoSMoS approach describes a collection of roles, artefacts, and arguments that go into developing a simulation as a scientific instrument. The pattern language outlined in this paper provides guidance for using roles and developing the artefacts and arguments. Three patterns and two antipatterns have been given in detail, and several more have been summarised (in the appendixes). The full pattern language [33] has around one hundred patterns and antipatterns, covering various levels of detail, phases of the simulation project, and variations on the approach.

Using a pattern language helps provide guidance for the simulator developer in "bite sized" chunks, and provides a universal vocabulary for talking about the development project. The patterns are based on the CoSMoS project partners' experience of developing a range of simulations used as scientific instruments.

Acknowledgements

This work is part of the Complex Systems Modelling and Simulation (CoSMoS) project, funded by EPSRC grants EP/E053505/1 and EP/E049419/1. I would like to thank my CoSMoS project colleagues, and pattern book [33] co-authors, for providing much of the raw material on which this paper is based. Thanks also go to the anonymous referees, whose comments have helped improve this paper.

CoSMoS project documentation of simulation, modelling and process descriptions [3, 29, 30], of validation and argumentation [2, 16, 17, 28], of various biological system simulation case studies [8, 10, 14, 15, 31, 32], of environment orientation [22, 23], of metamodels [4, 24], and of the CoSMoS workshop proceedings [34–37], is available from the CoSMoS project website www.cosmos-research.org

References

[1] Christopher Alexander, Sara Ishikawa, Murray Silverstein, Max Jacobson, Ingrid Fiksdahl-King, and Shlomo Angel. *A Pattern Language: towns, buildings, construction.* Oxford University Press, 1977.

[2] Paul S. Andrews, Fiona Polack, Adam T. Sampson, Jon Timmis, Lisa Scott, and Mark Coles. Simulating biology: towards understanding what the simulation shows. In Stepney et al. [34], pages 93–123.

[3] Paul S. Andrews, Fiona A. C. Polack, Adam T. Sampson, Susan Stepney, and Jon Timmis. The CoSMoS process, version 0.1: A process for the modelling and simulation of complex systems. Technical Report YCS-2010-453, Department of Computer Science, University of York, March 2010.

[4] Paul S. Andrews, Susan Stepney, Tim Hoverd, Fiona A. C. Polack, Adam T. Sampson, and Jon Timmis. CoSMoS process, models, and metamodels. In Stepney et al. [35], pages 1–13.
[5] Kent Beck. *Smalltalk Best Practice Patterns*. Prentice Hall, 1997.
[6] William J. Brown, Raphael C. Malveau, Hays W. "Skip" McCormick III, and Thomas J. Mowbray. *AntiPatterns: refactoring software, architectures, and projects in crisis*. Wiley, 1998.
[7] James O. Coplien and Douglas C. Schmidt, editors. *Pattern Languages of Program Design*. Addison Wesley, 1995.
[8] Alastair Droop, Philip Garnett, Fiona A. C. Polack, and Susan Stepney. Multiple model simulation: modelling cell division and differentiation in the prostate. In Stepney et al. [35], pages 79–111.
[9] Eric Evans. *Domain-Driven Design: tackling complexity in the heart of software*. Addison Wesley, 2004.
[10] Anton Jakob Flügge, Jon Timmis, Paul Andrews, John Moore, and Paul Kaye. Modelling and simulation of granuloma formation in visceral leishmaniasis. In *CEC 2009*, pages 3052–3059. IEEE Press, 2009.
[11] Martin Fowler. *Analysis Patterns: reusable object models*. Addison Wesley, 1997.
[12] Richard P. Gabriel. *Patterns of Software: tales from the software community*. Oxford University Press, 1996.
[13] Erich Gamma, Richard Helm, Ralph E. Johnson, and John Vlissides. *Design Patterns: elements of reusable object-oriented software*. Addison Wesley, 1995.
[14] Philip Garnett, Susan Stepney, Francesca Day, and Ottoline Leyser. Using the CoSMoS process to enhance an executable model of auxin transport canalisation. In Stepney et al. [36], pages 9–32.
[15] Philip Garnett, Susan Stepney, and Ottoline Leyser. Towards an executable model of auxin transport canalisation. In Stepney et al. [34], pages 63–91.
[16] Teodor Ghetiu, Robert D. Alexander, Paul S. Andrews, Fiona A. C. Polack, and James Bown. Equivalence arguments for complex systems simulations - a case-study. In Stepney et al. [37], pages 101–140.
[17] Teodor Ghetiu, Fiona A. C. Polack, and James L. Bown. Argument-driven validation of computer simulations – a necessity rather than an option. In *VALID 2010:*, pages 1–4. IEEE Press, 2010.
[18] Paolo Giudici. *Applied Data Mining: Statistical Methods for Business and Industry*. Wiley, 2003.
[19] Volker Grimm, Uta Berger, Finn Bastiansen, Sigrunn Eliassen, Vincent Ginot, Jarl Giske, John Goss-Custard, Tamara Grand, Simone K. Heinz, Geir Huse, Andreas Huth, Jane U. Jepsen, Christian Jørgensen, Wolf M. Mooij, Birgit Müller, Guy Pe'er, Cyril Piou, Steven F. Railsback, Andrew M. Robbins, Martha M. Robbins, Eva Rossmanith, Nadja Rüger, Espen Strand, Sami Souissi, Richard A. Stillman, Rune Vabø, Ute Visser, and Donald L. DeAngelis. A standard protocol for describing individual-based and agent-based models. *Ecological Modelling*, 198(1-2):115–126, 2006.

[20] Volker Grimm, Uta Berger, Donald L. DeAngelis, J. Gary Polhill, Jarl Giske, and Steven F. Railsback. The ODD protocol: A review and first update. *Ecological Modelling*, 221(23):2760–2768, 2010.

[21] Neil B. Harrison, Brian Foote, and Hans Rohnert, editors. *Pattern Languages of Program Design 4*. Addison Wesley, 2000.

[22] Tim Hoverd and Adam T. Sampson. A transactional architecture for simulation. In *ICECCS 2010: Fifteenth IEEE International Conference on Engineering of Complex Computer Systems*, pages 286–290. IEEE Press, 2010.

[23] Tim Hoverd and Susan Stepney. Environment orientation: an architecture for simulating complex systems. In Stepney et al. [37], pages 67–82.

[24] Tim Hoverd and Susan Stepney. Energy as a driver of diversity in open-ended evolution. In *ECAL 2011, Paris, France, August 2011*. MIT Press, 2011.

[25] Andrew Koenig. Patterns and antipatterns. *Journal of Object-Oriented Programming*, 8(1):46–48, 1995.

[26] Macmillan Dictionary. http://www.macmillandictionary.com/dictionary/british/fit-for-purpose.

[27] Robert C. Martin, Dirk Riehle, and Frank Buschmann, editors. *Pattern Languages of Program Design 3*. Addison Wesley, 1998.

[28] Fiona A. C. Polack. Arguing validation of simulations in science. In Stepney et al. [36], pages 51–74.

[29] Fiona A. C. Polack, Paul S. Andrews, Teodor Ghetiu, Mark Read, Susan Stepney, Jon Timmis, and Adam T. Sampson. Reflections on the simulation of complex systems for science. In *ICECCS 2010*, pages 276–285. IEEE Press, 2010.

[30] Fiona A. C. Polack, Paul S. Andrews, and Adam T. Sampson. The engineering of concurrent simulations of complex systems. In *CEC 2009*, pages 217–224. IEEE Press, 2009.

[31] Fiona A. C. Polack, Alastair Droop, Philip Garnett, Teodor Ghetiu, and Susan Stepney. Simulation validation: exploring the suitability of a simulation of cell division and differentiation in the prostate. In Stepney et al. [35], pages 113–133.

[32] Mark Read, Paul S. Andrews, Jon Timmis, and Vipin Kumar. A domain model of experimental autoimmune encephalomyelitis. In Stepney et al. [37], pages 9–44.

[33] Susan Stepney, Kieran Alden, Paul S. Andrews, James L. Bown, Alastair Droop, Teodor Ghetiu, Tim Hoverd, Fiona A. C. Polack, Mark Read, Carl G. Ritson, Adam T. Sampson, Jon Timmis, Peter H. Welch, and Alan F. T. Winfield. *Engineering Simulations as Scientific Instruments*. Springer, 2012. in preparation.

[34] Susan Stepney, Fiona Polack, and Peter Welch, editors. *Proceedings of the 2008 Workshop on Complex Systems Modelling and Simulation*. Luniver Press, 2008.

[35] Susan Stepney, Peter Welch, Paul S. Andrews, and Carl G. Ritson, editors. *Proceedings of the 2011 Workshop on Complex Systems Modelling and Simulation*. Luniver Press, 2011.

[36] Susan Stepney, Peter H. Welch, Paul S. Andrews, and Adam T. Sampson, editors. *Proceedings of the 2010 Workshop on Complex Systems Modelling and Simulation*. Luniver Press, 2010.

[37] Susan Stepney, Peter H. Welch, Paul S. Andrews, and Jon Timmis, editors. *Proceedings of the 2009 Workshop on Complex Systems Modelling and Simulation*. Luniver Press, 2009.

[38] John Vlissides, James O. Coplien, and Norman L. Kerth, editors. *Pattern Languages of Program Design 2*. Addison Wesley, 1996.

A Referenced patterns and their intent

Argue Appropriate Instrument Designed	Present the basis of consensus that the simulation as a scientific or engineering instrument is appropriate to its purpose and use
Calibration	Tune the Simulation Platform parameter values so that simulation results match the calibration data provided in the Data Dictionary
Data Dictionary	Define the modelling data used to build the simulation, and the experimental data that is produced by domain experiments and the corresponding simulation experiments
Debug By Video	Use a visualisation of the simulation results to help detect problems with the implementation
Development Phase	Produce a Simulation Platform, based on the output of Discovery Phase
Discovery Phase	Establish the scientific basis of the project, and build the Domain Model
Document Assumptions	Ensure assumptions are explicit and justified, and their connotations are understood
Domain	Identify the subject of scientific research: the real-world system and the relevant information known about it
Domain Expert	Identify the "owner", or single point of contact, for domain knowledge

Domain Model	Produce an explicit description of the relevant domain concepts
Domain Modeller	Identify those team members responsible for producing and maintaining the Domain Model
Exploration Phase	Perform Simulation Experiments to explore the scientific questions established during Discovery Phase
ODD protocol	Present the simulation details in conformance with the ODD protocol [19, 20]
Platform Model	From the Domain Model, develop a platform model suitable to form the requirements specification for the Simulation Platform
Propagate Changes	Ensure that changes in one part of the system propagate throughout, to ensure consistency
Prototype	Build an executable model to explore specific domain or implementation issues
Research Context	Identify the overall scientific context and scope of the simulation-based research being conducted
Results Model	Encapsulate the understanding of outputs and results from Simulation Experiments, in Domain Model terms
Simulation Experiment	Perform an *in silico* experiment using the Simulation Platform
Sensitivity Analysis	Discover how the uncertainties in the simulation output values depend on uncertainties in the input and modelling parameter values
Simulation Implementor	Identify those team members responsible for producing and maintaining the Simulation Platform
Simulation Purpose	Agree the purpose for which the simulation is being built and used, within the research context

Simulation Platform	Develop the executable simulation platform that can be used to run the Simulation Experiment
Visualisation Model	Visualise the Simulation Experiment results of the Data Dictionary in a manner relevant to the users

B Referenced antipatterns and their problem statements

Amateur Science	You do not engage with a domain expert, because you think you know the domain science well enough
Analysis Paralysis	You are spending too much time analysing and modelling the domain, trying to get everything perfect, and never getting to the simulation
Blind Trust	You accept everything the Domain Expert tells you, even outside their own expertise
Everything but the Kitchen Sink	You are putting irrelevant information or detail into a model, just because you can
Literature Only	You take the domain literature as the only input to the Domain Model
Living in Flatland	You are simulating a 2D space, and naively translating the results to 3D reality
Moving the Goalposts	You change the Research Context (for example, you pose a new research hypothesis), without checking that the models and validity arguments still hold
Program In the Answer	The results from the simulation are an inevitable consequence of the simulation programming, not an emergent consequence of the operation of the simulation

Proof by Video	The Visualisation Model is all there is
Premature Implementation	You start writing Simulation Platform code before having a proper understanding of the domain
Tweaking	You make a series of small, "unimportant" changes to the working Simulation Platform

Simulating the Effects of Anticoagulant Drugs Upon Blood Clotting Dynamics

Alexey Goltsov, Gregory Goltsov, and Adam Sampson

Centre for Research in Informatics and Systems Pathology (CRISP),
University of Abertay Dundee, Dundee, DD1 1XF, United Kingdom

Abstract. Linking scales in both modelling and visualisation is a key challenge in computational physiology. We have combined two existing simulations of blood clotting – a large-scale simulation of the physical interactions between platelets in the bloodstream, and a detailed simulation of the chemical signalling inside a platelet based on an accurate mathematical model – and linked them to an interactive 3D visualisation, allowing researchers to immediately see the tissue-scale results of changes to cell-scale models.

A The Original Simulations

This work is based upon two existing, mature simulations. The first simulation was constructed to explore the effects of combinations of anticoagulant drugs such as aspirin and celecoxib upon blood clotting [1]. These drugs work by inhibiting the production of prostaglandin H synthase (PGHS), a precursor of signalling molecules such as thromboxane, a key factor in the activation and aggregation of platelets. Our kinetic model of PGHS-1 catalysis within platelets was calibrated using in-vitro and in-vivo experimental data.

The second simulation was developed as part of the TUNA project to study the low-level platelet behaviours necessary for clotting to emerge [2] and implements an agent-based model of spatial interaction using concurrent techniques for multicore and distributed simulation, allowing experimentation and cross-validation at realistic scales. The model was later reworked using techniques developed by the CoSMoS project for improved scalability.

Merging these two approaches enables interactive experimentation with the effects of anticoagulant drugs in a realistic spatial environment. The first simulation provides a biologically-accurate model of the key

processes within a cell; the second provides the implementation technologies necessary to visualise the emergent effects of those processes at larger scales.

B The Combined Simulation

Starting with our existing model of platelet chemical signalling, we specified and calibrated a characteristic subset of the model using a hybrid Petri net approach. We designed a declarative embedded domain-specific language (EDSL) for hybrid Petri nets, and developed a new signalling simulator that could be embedded into the existing spatial simulation enabling fine-grained parallelisation. We extended the spatial simulation to model diffusion of multiple chemical signals with realistic concentrations, and linked it to the new signalling simulation. We also built a new immersive 3D visualisation for the simulation using the Cinder library, which allows easier navigation and exploration of the simulated system, interactive adjustment of parameter values, and more effective visualisation of cell properties (Fig. 1).

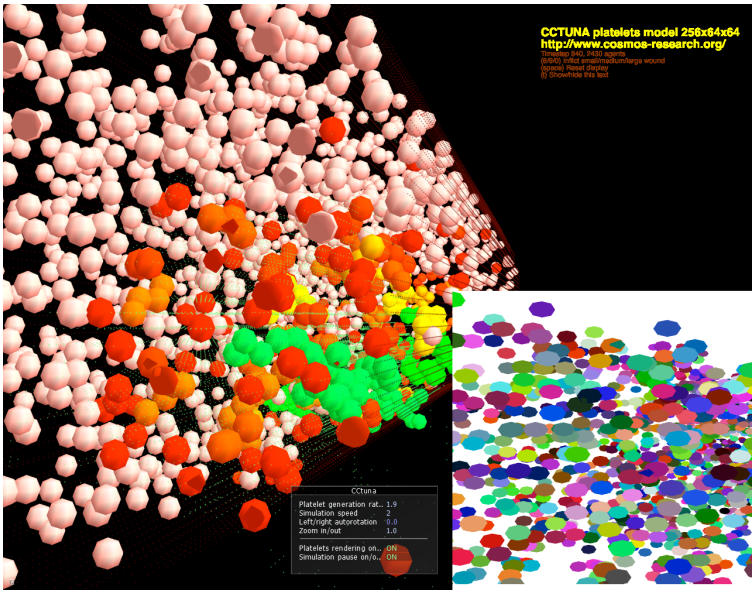


Fig. 1. Clotting in response to a wound in the combined simulator.

The resulting simulation correctly reproduces the required domain behaviours, and provides a number of reusable components for other work within CRISP, integrating equational models of cell signalling with CoSMoS-style models of spatial interaction. We are particularly interested in spatial aspects of cancer growth, and plan to couple the simulation built in this project to our group's biologically-accurate visualisation of cell signalling network dynamics, enabling the user to "zoom in" from a physical view of the world to a conceptual view of the signalling network within a chosen cell.

Acknowledgements

The authors would like to thank the Nuffield Foundation for supporting Gregory Goltsov's work on this project through an Undergraduate Research Bursary.

References

[1] Alexey Goltsov, Galina Lebedeva, Ian Humphery-Smith, Gregory Goltsov, Oleg Demin, and Igor Goryanin. In silico screening of nonsteroidal anti-inflammatory drugs and their combined action on prostaglandin H synthase-1. *Pharmaceuticals*, 3(7):2059–2081, 2010.

[2] Carl G. Ritson and Peter H. Welch. A process-oriented architecture for complex system modelling. *Concurrency and Computation: Practice and Experience*, 22:965–980, 2010.