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.

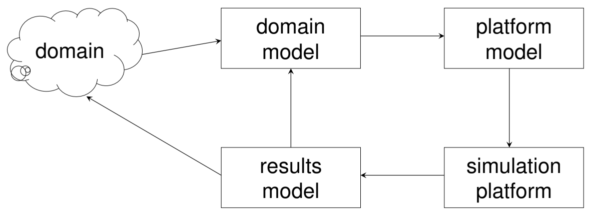

Fig. 1. Relationship between simulation components; arrows represent flows of information. These are all framed by the research context.

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).

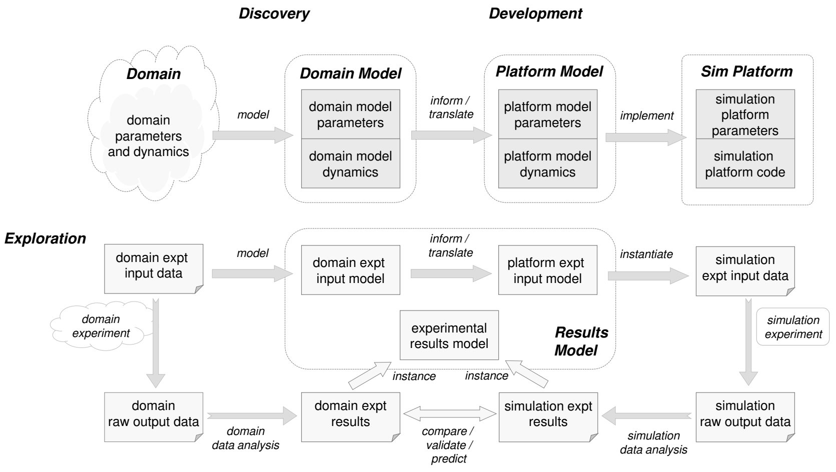

Fig. 2. The relationship between the various models and phases, and how the simulation platform is used to perform simulation experiments. See text for details.

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 simulation 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.

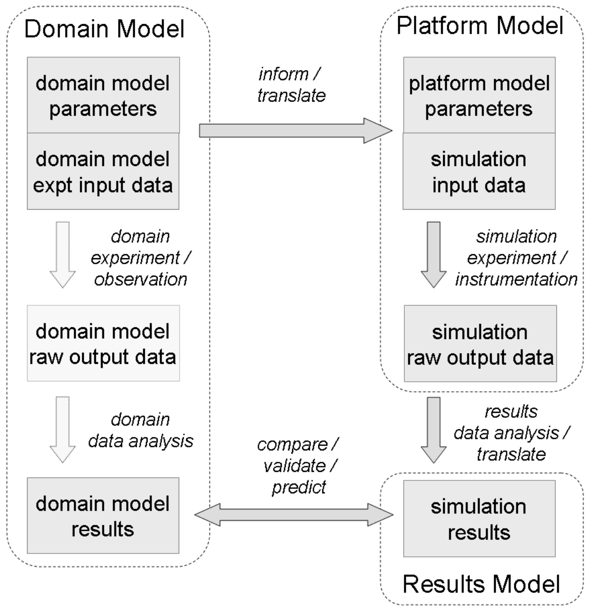
Domain Model
domain model parameters
domain model expt input data
domain experiment / observation
domain model raw output data
domain data analysis
domain model results
inform / translate
compare / validate / predict
Platform Model
platform model parameters
simulation input data
simulation experiment / instrumentation
simulation raw output data
results data analysis / translate
simulation results
Results Model

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

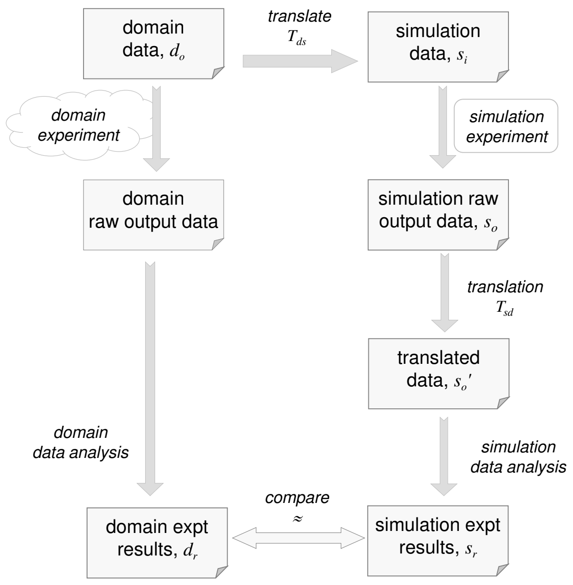

Fig. 4. Summary of the components of the data model in the Domain Model, how they relate to components in the Platform Model and the Results Model, and the relationship that calibration aims to achieve.

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