Object-Oriented Paradigms for Modelling Vascular Tumour Growth: A Case Study

Anthony J. Connor*, Jonathan Cooper*, Helen M. Byrne* †, Philip K. Maini‡ and Steve McKeever*
*Department of Computer Science
University of Oxford, Oxford, England.
†Oxford Centre for Collaborative Applied Mathematics
University of Oxford, Oxford, England.
‡Centre for Mathematical Biology
University of Oxford, Oxford, England.
Emails: anthony.connor@cs.ox.ac.uk, jonathan.cooper@cs.ox.ac.uk, helen.byrne@cs.ox.ac.uk,
maini@maths.ox.ac.uk, steve.mckeever@cs.ox.ac.uk.

***Abstract*—Motivated by a family of related hybrid multiscale models, we have built an object-oriented framework for developing and implementing multiscale models of vascular tumour growth. The models are implemented in our framework as a case study to highlight how object-oriented programming techniques and good object-oriented design may be used effectively to develop hybrid multiscale models of vascular tumour growth. The intention is that this paper will serve as a useful reference for researchers modelling complex biological systems and that these researchers will employ some of the techniques presented herein in their own projects.**

***Keywords*-multiscale; hybrid; object-orientation; vascular tumour growth; modelling.**

I. INTRODUCTION

Object-orientation was originally heralded as a "silver bullet" [1] for dealing with the software crisis, which was characterised by, amongst other things, the high complexity, low productivity, and poor reliability of software systems. Object-oriented programming (OOP) [2] promises to encourage code re-use, maintainability, understandability and extensibility for those systems which are amenable to object-oriented design. However, producing an object-oriented system or program with long term re-usability, extensibility and maintainability requires considerable investment of time and effort at the start of the project, for what at first may seem like little return; it is an art-form, often requiring years of experience and/or a time-consuming trial and error approach. Nevertheless, the potential benefits of a good object-oriented design are well worth the time and effort spent.

OOP today remains the most popular programming paradigm world wide, being used extensively by both software engineers and mathematical modellers. However, OOP is often used poorly or inappropriately, particularly by mathematical modellers, who may not have a background in computer science and little experience with OOP. This issue is compounded by the fact that modelling academics are under tremendous pressure to produce and publish results from implemented models quickly, before competing groups beat them to the punch. This inherently leads to little time and effort being put in to the design of model code, which might have ensured that the model implementation would be more understandable and could be more easily maintained and extended at a later date.

In this paper, we advocate the use of good object-oriented techniques for developing multiscale models of vascular tumour growth. Motivated by models proposed by Alarcón and co-workers [3], [4], [5], [6], [7], [8], we have developed an object-oriented modelling framework in which a range of hybrid multiscale models of vascular tumour growth are implementable. To date, the functionality of the framework focusses around the methodologies employed by Alarcon and co-workers. However, by employing an object-oriented framework, and designing it with re-usability and extensibility in mind, our framework could be extended to allow for additional functionality with relative ease.

Our paper focuses on the aspects of OOP which make it appropriate for developing models of complex biological systems, using the models of Alarcón and co-workers as a case-study. We exploit good object-oriented design principles, describing the models abstractly and decoupling the framework design from model implementations. These aspects of our work may have implications for increasing the trust-worthiness of model execution by remote users and improve the prospects for model re-use in the modelling community. By decomposing models into collaborating classes of biological entities and behavioural algorithms, we also enable the rigorous validation of models and a test-driven approach to model development to be implemented.

The remainder of the paper is structured as follows. In Sections II and III, we briefly introduce the application domain and some basic concepts of OOP. We also address what aspects of the paradigm make it appropriate for developing the complex multiscale biological models which we consider. In Section IV, we present the results of an exemplar simulation whose implementation was realised using the object-oriented framework we have developed. We then discuss the implications of our work and avenues of future work in Section V, before concluding in Section VI.

 ISBN: 978-1-61208-234-9

II. The application domain

The entry of a tumour into its vascular growth phase marks the transition from a phase in which the tumour is essentially harmless to one in which it is potentially fatal [9]. During avascular growth, tumours are limited in size because they rely on diffusion to obtain nutrients. As a tumour grows, parts of it become deprived of oxygen, resulting in some of the tumour cells becoming quiescent and expressing various angiogenic factors. These tumour angiogenic factors (TAFs) diffuse throughout the surrounding tissue and, upon reaching a blood vessel, stimulate the formation of new vessel sprouts which migrate towards the tumour. After flow has been established in the new vasculature the surrounding tissue has increased access to nutrients, allowing the tumour to continue growing and to invade the adjacent healthy tissue. This process is known as tumour-induced angiogenesis. Once vascularised, the tumour also gains its own transport network by which tumour cells may be transported around the body to form metastases in any part of the host organism.

There is a large body of literature devoted to modelling both avascular tumour growth and tumour-induced angiogenesis. Broadly speaking such models can be placed into three categories: continuous, discrete and hybrid. For extensive reviews, see [10], [11].

Multiscale modelling involves the integration of several biological models, each describing a certain process at a particular time and length scale. Furthermore, each process may be represented using different mathematical modelling methodologies. For example, coupled ODEs may be used to describe subcellular processes and protein interaction networks and PDEs may be used to describe the diffusion of nutrients or chemical signals through tissues, while cell-cell interactions may be modelled discretely using on- or off-lattice techniques. Multiscale models allow modellers to capture the interdependence of biological phenomena which occur at different biological scales. They offer a natural framework for studying biological phenomena, such as angiogenesis and tumour growth, which are inherently multiscale in nature, and thus appear to offer the cutting-edge with regards to potential predictive power and clinical applicability. As such, multiscale models of angiogenesis and vascular tumour growth have gained in popularity over the last decade and have begun to show promise at the clinical level. It is for these reasons that we too focus on multiscale models of vascular tumour growth in this work.

Alarcon, Byrne, Maini and co-workers were one of the first groups to develop a vascular tumour model which incorporated multiple biological scales in a systematic way [3], [4], [5]. Their hybrid multiscale model has continued to develop through contributions from Betteridge [6], Owen [7] and Perfahl [8], amongst others. This family of models couples biological phenomena that include vascular adaptation and remodelling, blood flow, nutrient and vascular endothelial growth factor (VEGF) diffusion throughout the extracellular space and the cellular and subcellular dynamics of normal and cancerous cells. The mathematics that underpins the model implementations is described in the original papers and, for brevity, is not repeated here. Other notable multiscale models of tumour growth are presented in [12], [13]; for a further review, readers may consult [14].

One of the best hopes for developing increasingly complex models of vascular tumour growth that span multiple biological scales is by re-using and extending existing models. However, the integration or extension of existing models of vascular tumour growth (or elements of those models) currently represents a substantial technical challenge in the field. At present, mathematical models of cancer are often implemented by hand, in the language of choice of a modeller, with little or no thought given to how code may be re-used or models extended at a later date. This means that modellers must be familiar with the code in order to manipulate and evaluate simulation runs. Additionally, modellers are met with significant issues when they wish to re-use, extend, or maintain model code. As such, the development of cancer models and the success of the cancer modelling community as a whole is severely hindered. These are the principal factors which have inspired our work.

III. Object-oriented design for *in silico* models of vascular tumour growth

The way in which we think about and describe the world usually involves looking at it in terms of objects and the interactions between those objects. For complex biological systems, such as a growing tumour, verbal modelling does not enable us to describe or communicate aptly the complex non-linear feedback interactions which characterise those systems [15]. For this, we need mathematics. OOP strives to describe systems in terms of objects and the interactions between those objects computationally, thus enabling the quantitative mathematical analysis of complex biological systems. By describing systems in a way with which we are familiar, OOP promotes both an understanding of the system which we are trying to model and also an understanding of the model code itself; OOP helps us to manage the *essential* complexity [1] associated with real-world modelling.

Many of the object-oriented techniques described in this section help modellers to accurately describe the world in a way which is natural and understandable to others. This is particularly important within the field of biological modelling. Not only is it desirable for other modellers and programmers to understand our code, so that they may easily use, re-use and extend our models, but it is extremely desirable for experimentalists to be able to understand the system described through the code. This is because systems biology and the development of accurate and validated complex multiscale models heavily rely on the co-operation between experimentalists and modellers. Making computa-

 ISBN: 978-1-61208-234-9

tional models easier to understand for experimentalists, who may have little programming experience, will promote the closer collaboration of experimentalists and modellers.

A. Objects, classes and modularity

Objects in the context of OOP are complex data structures which contain data fields (attributes) and a complete set of operational methods to which that object may respond (its interface). In this way, objects represent a convenient data-centric way of decomposing systems into understandable and manageable sets of modules. In OOP, one actually usually designs *classes*, not objects. These classes, then, form the templates from which multiple objects may be instantiated and exist concurrently inside a program at run-time. Classes may be developed and tested individually before being integrated into larger systems being developed by a team or, in our case, potentially a larger community of programmers. Designing classes appropriately may make possible their re-use in other applications.

Simply by enforcing boundaries and structure on a program or model, objects as modules increase the maintainability of code. In software design, *low coupling* and *high cohesion* are often highlighted by programmers as qualities which a re-usable, extensible and maintainable piece of software should possess. Broadly, coupling refers to the interdependencies between different modules and cohesion refers to how strongly related the functions within a module are. High cohesion is an important quality because it increases the understandability of code. Low coupling, where one module interacts with another through a stable and well-defined interface and independently of the internal structure of the other module, increases maintainability, re-use and extensibility of code. Designing a library of classes with these properties is technically demanding and time-consuming but the benefits are invaluable, especially later in the life-cycle of a project.

B. Encapsulation and information and implementation hiding

The wrapping up of operations and attributes into a class, so that those attributes may only be manipulated through or accessed via the operations provided by the class, is encapsulation. Good encapsulation hides the details of an object's internal attributes and implementation of operations from modellers using that object [2]. These techniques are known as information hiding and implementation hiding, respectively, and their use is essential for promoting the understandability of code in our framework.

Many complex algorithms are employed in the models implementable in our framework, none more so than the structural adaptation algorithms used to determine the pseudo-steady-state radii of vessels and haematocrit distribution in a vessel network (see [3] for details). However, by packaging the operations and parameters required by this algorithm into a single `StructuralAdaptationAlgorithm` class and providing a single operation by which the entire algorithm may be implemented, we ensure that modellers may easily run the structural adaptation algorithm on a simulated vessel network without necessarily knowing the detailed mathematics involved in the algorithm. By encapsulating all parameters associated with the structural adaptation algorithm in this class, we also reduce the possibility of introducing errors into model implementation. This is established by ensuring that these parameters may only be accessed from a single point in the code and thus may not inadvertently affect other parts of the model implementation. In this way, encapsulation facilitates model code maintainability and changeability as well as understandability.

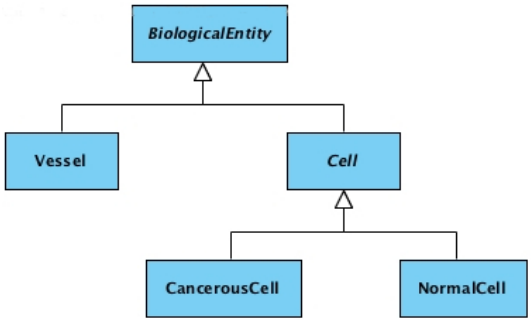


Figure 1: UML [16] diagram illustrating the class hierarchy for biological entities in our framework.

C. Class and containment hierarchies

In the real world, objects are often related by *is-a* type relationships. This type of relationship is also realisable in OOP languages by using class inheritance. For instance, in the models which motivate our work *vessels* and *cells* are the primary *biological entities* of interest. Additionally, we consider two types of cells: *cancerous* and *normal*. These relationships are represented in OOP using inheritance to define class hierarchies, such as that in Figure 1. The classes `CancerousCell` and `NormalCell` inherit from the class `Cell`, which in turn inherits from the `BiologicalEntity` class. The class of `Vessel` also inherits from the `BiologicalEntity` class. In this way, appropriate relationships, which mimic real-world relationships, are defined between some classes of objects in our modelling domain. Furthermore, class inheritance provides a powerful mechanism by which code may be re-used. By inheriting from an existing class, such as our `Cell` class, the subclasses (`CancerousCell` and `NormalCell`) automatically get all of the functionality (operations) and attributes defined in the `Cell` class. Moreover, new operations and attributes may be defined in the subclasses. In this way, inheritance allows programmers to define classes as incremental variations of more basic and abstract classes.

In OOP, inheritance also enables *substitutability*. In our example, this means that objects of type `CancerousCell`

 ISBN: 978-1-61208-234-9

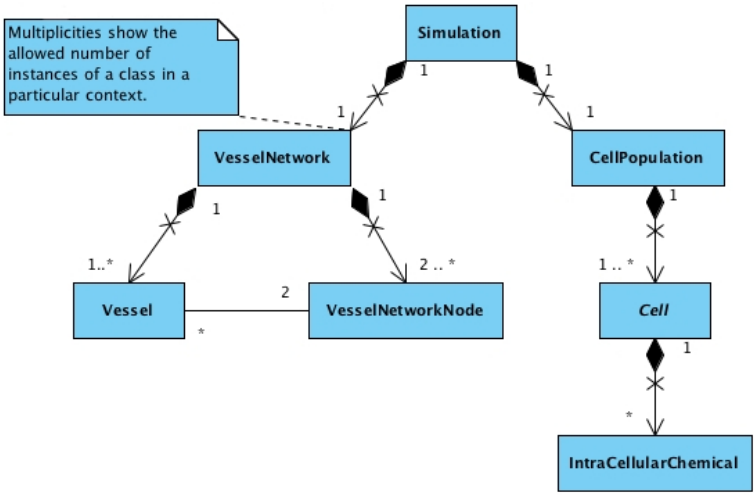


Figure 2: Containment hierarchy for major biological classes in our modelling framework.

or `NormalCell`, if defined appropriately, may be provided at run-time in any context where an object of type `Cell` is expected without affecting the correctness of the model implementation. This property of an object-oriented system imposes strict rules on class inheritance, namely that any class inheritance hierarchy must follow the principle of type conformance [2]. Similarly, the Liskov substitution principle [17] defines the appropriate use of subtyping relationships. The ability to substitute classes dynamically at run-time is hugely advantageous and we expand on this idea in Sections III-D and III-E.

OOP also allows objects to be contained within other objects. This is what is meant by a containment hierarchy. A simple example, implemented in our framework is shown in Figure 2. This hierarchy reflects the fact that vessel networks comprise many vessels which are connected at nodes in that vessel network and also that a cell population contains cells, each of which contain a set of intra-cellular chemicals, whose concentrations may dictate that cell's behaviour.

The construction of containment hierarchies is a useful technique, particularly for multiscale modelling. Objects contained within other objects may be used to represent nested levels of organisation, naturally mimicking the biologists' view of their systems. This technique also provides an intuitive mechanism for simplifying the implementation of individual submodels at the specific biological scale(s) with which that submodel is concerned. More complex containment hierarchies may be constructed to model biology at increased levels of detail, as demonstrated in [18].

In a wider context, certain classes of objects may be re-used by allowing instances of those classes to form components of more than one type of composite object. Class composition and inheritance may also be combined effectively in order to promote further re-use and extensibility of model code. This is best exemplified by the strategy pattern which is explained in Section III-E2.

D. Polymorphism

There are two distinctly different types of polymorphism in OOP: *static polymorphism* and *dynamic polymorphism*. Static polymorphism is also known as *overloading* and involves defining operations with the same name, but which take different types and/or numbers of input arguments. An appropriate example of overloading would be when requesting that a `CellPopulation` provide access to a `Cell` contained within that population. We may wish to access a `Cell` object by providing the `CellPopulation` object with either a unique id number, which each cell possesses, or with the location of the cell. In order to implement these requirements we define two operations inside the `CellPopulation` class, each called `GetCell`, which each return a pointer to the `Cell` object of interest:

1) `GetCell( int idNumber )` returns the `Cell` with `id` equal to `idNumber`.
2) `GetCell( coordinate cellLocation )` returns the `Cell` with `location` equal to `cellLocation`.

This technique aids in program design and model implementation by providing a uniform and intuitive interface by which object attributes may be accessed or manipulated [19].

Dynamic polymorphism, or *overriding*, is closely related to the concepts of inheritance and substitutability (Section III-C). We have already noted that inheritance guarantees that a subclass will contain at least the same operations that are defined in that class' superclass and sometimes more. OOP also allows the implementation of inherited operations to be redefined in a subclass: this is overriding. Calls to an operation which have been overridden in a subclass generally perform a slightly modified function. The functionality executed by a program then depends on what type of object the method is actually called on at run-time. This technique empowers modellers to create implementations which are very malleable and extensible

 ISBN: 978-1-61208-234-9

by allowing interchangeable classes within a class hierarchy to perform moderately different functions in response to the same operation call. Such polymorphism plays an important role in the design patterns discussed in the next section.

E. Design patterns

Whilst design patterns are not strictly a part of OOP, the terms object-oriented design and design patterns became almost synonymous after the "Gang of Four" published their book, *Design Patterns: Elements of Reusable Object-Oriented Software* [20], in 1994. Design patterns represent elegant and re-usable solutions to problems commonly encountered in software design. Simply using a design pattern can improve code readability and understandability, especially for programmers and modellers who are familiar with the pattern used. Utilising these tried and tested solutions to software development problems may prevent subtle issues that may not have been considered but which could cause problems at a later stage in the life-cycle of a development process. From a biological modelling perspective, certain design patterns, for example the visitor pattern (Section III-E1), enable programmers to separate out biological structure from algorithms which determine the behaviour of the biological entities in a modelling domain. Such design patterns further promote model understandability for biologists.

We do not intend to explain or provide an exhaustive list of design patterns here, but instead present two behavioural design patterns that are used extensively throughout our framework: the visitor pattern and strategy pattern. We have also made use of the factory pattern, the template method pattern and the null object pattern in our code but do not present full descriptions of these additional patterns here.

1) Visitor pattern

The main appeal of using the visitor pattern is that it allows programmers to add new algorithms which operate over a class hierarchy without having to modify that class hierarchy, i.e., by adding operations specific to that algorithm to each member of that hierarchy. Implementation of this pattern involves defining two class hierarchies: one for the classes of objects being operated on (*elements*) and one for *visitors* which define the operations on elements. A dynamically polymorphic `accept` operation is defined in the element hierarchy which accepts a visitor object as an argument. A visited element responds to an `accept` operation call by calling the `visit` operation on the provided visitor, giving itself as an argument. Generic `visit` operations defined in the visitor hierarchy are overridden and overloaded in each member of the hierarchy, in order to define the behaviour of each algorithm on each element class, respectively. By grouping together related operations on one class hierarchy within a separate class hierarchy, we simplify both the class of objects carrying out a specific behaviour and the implementations of the various algorithms which determine that behaviour. In our framework, this pattern is used extensively to implement models for the behaviour of the vasculature, diffusible substances and cells. We will describe the visitor pattern as implemented in the context of modelling cell behaviour to better illustrate its use.

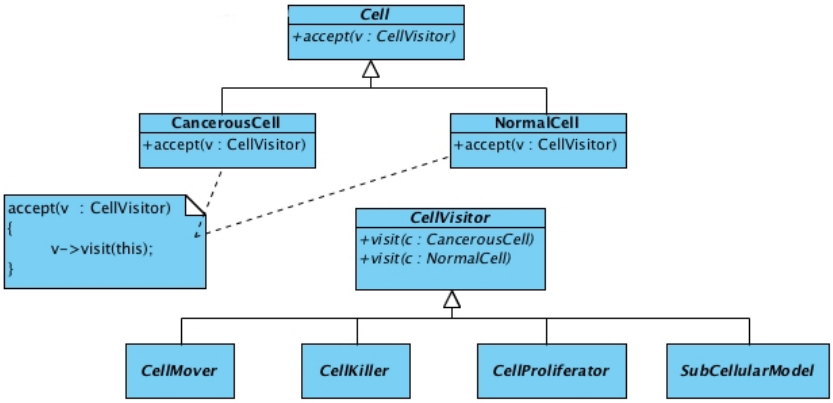
In our framework, populations of cells are modelled using an agent-based approach, where the behaviours of the cells are modelled by the application of rules, which may differ according to the type of cell to which the rule is being applied. We model three aspects of cell behaviour: movement, death, and proliferation. In addition we also model subcellular events. Models for subcellular processes include rules which dictate how the cell-cycle progresses; when cells should undergo apoptosis, when they should enter a quiescent state, when they should divide and how VEGF is produced and released. A number of subcellular models have been implemented in our framework, including a model in which division occurs after a fixed period of time [3], a simple oxygen dependent phase model [21], and a more complex subcellular model, first introduced in [4], involving the solution of seven coupled ODEs which represent how various intra-cellular chemicals evolve in time and account for the effect of local oxygen concentrations on cell-cycle progression. Similarly, different submodels dictate how cells should move and proliferate and also how and when a cell should die, e.g., in response to the presence of a drug [5].

Following the template outlined above, we define a `Cell` class hierarchy and a `CellVisitor` class hierarchy, whose members define the algorithms which determine various aspects of a `Cell` object's behaviour (Figure 3 (a)). Dynamic polymorphism allows the different types of `CellVisitor` classes to modulate the type of cell behaviour which the visitor executes by responding to the same `visit` operation calls. Static polymorphism allows the same `CellVisitor` to implement slightly modified behaviours depending on what type of `Cell` object that visitor is operating on. In this way, an algorithm used to determine the behaviour of multiple cell types may be encapsulated inside a single class. Additionally, algorithm specific data structures and parameters are encapsulated within the visitor classes thus making the code more understandable and maintainable.

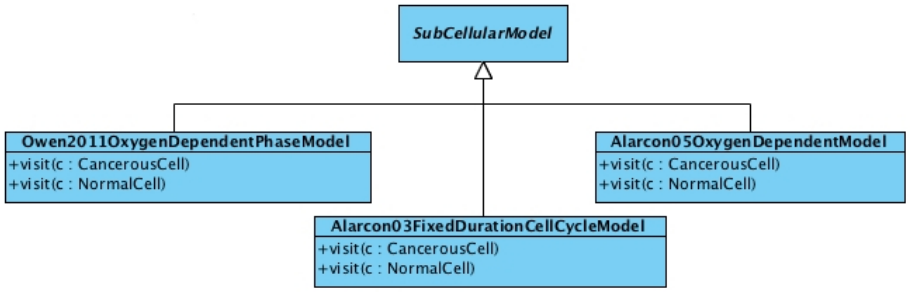
We extend the `CellVisitor` hierarchy in order to account for the different algorithms which may be used to model the same aspects of cell behaviour. For example, since we use several different types of subcellular model in our framework, we construct a hierarchy of `SubCellularModel` classes, each of which encapsulate a particular submodel, as shown in Figure 3 (b). At runtime, we may then choose which model we would like to implement very easily. We expand upon this idea in Section III-E2, where we present the strategy pattern.

It is particularly easy to vary and add functionality to our framework using this pattern. A new type of cell behaviour

 ISBN: 978-1-61208-234-9



(a) `Cell` and `CellVisitor` hierarchies.



(b) Hierarchy of classes that encapsulate the various subcellular models used in our framework.

Figure 3: Visitor pattern as implemented within the context of modelling discrete cell behaviour.

or a modified algorithm for, for instance, cell movement may be easily added to the framework by simply creating a new concrete class in the `CellVisitor` hierarchy. In this regard, this pattern makes extending certain aspects of the framework extremely easy and intuitive.

In general, the visitor pattern is applied only when the element class hierarchy is unlikely to change. Adding a new class to this hierarchy means that a new concrete implementation of the `visit` operation, which takes an instance of the new class as an argument, must be added to each concrete `CellVisitor` with an appropriate implementation. Thus, extending model code in this way can prove difficult. This issue may be partially negated by defining appropriate default implementations in the abstract `CellVisitor` class, however, this opens the door for potential bugs to creep in to model implementations. Nonetheless, this pattern is both adequate and appropriate for our purposes since we are currently interested primarily in changing the algorithms that determine the behaviour of biological entities rather than changing the types of biological entities that are present.

2) Strategy pattern

As well as modelling the behaviour of biological entities using the visitor pattern, we have also utilised the strategy pattern, which allows us to vary the precise algorithms which determine the various aspects of biological entity behaviour at run-time. Again, we clarify this statement with an example. At the highest level, for the running of simulations we utilize this pattern by defining a `Simulation` class. This class co-ordinates the events which occur during a model simulation, whilst delegating the responsibility of carrying out specific tasks to other classes. For example, a `Simulation` has a reference to the abstract `SubCellularModel` class. Different algorithms for this particular aspect of the system's behaviour are implemented as concrete subclasses, as shown in Figure 3(b). Due to substitutability, an instance of one such subclass may be assigned to this reference at run-time, providing the implementation as desired by a modeller for a particular realisation of a model. Figure 4 displays a class diagram which illustrates how a model simulation is constructed in our framework. Given that some simulations may not require a particular behaviour to be modelled, we complement the implementation of this design pattern by additionally implementing the *null object pattern* (an instance of this pattern involves defining an additional class within a class hierarchy which has an appropriate interface but possesses neutral behaviour). An example showing how we construct a simulation in our framework using this pattern is presented in the next section.

The strategy pattern promotes the understanding of model code by factoring out common functionality of algorithms into a suitable superclass, and promotes code maintainability and extensibility, allowing us to implement a large range of different models within our framework in an intuitive way.

 ISBN: 978-1-61208-234-9

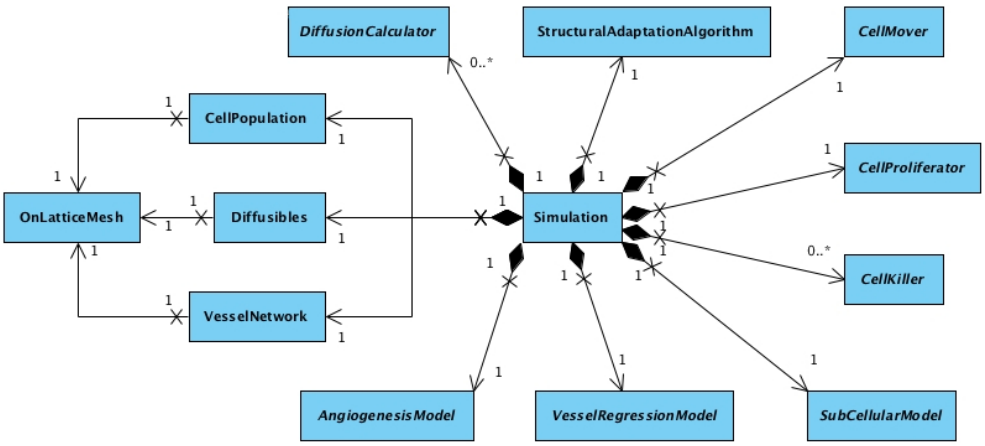


Figure 4: Class diagram showing the major classes involved in a simulation of vascular tumour growth in our modelling framework.

IV. Model Realisation

Due to its strongly object-oriented nature, a language supporting object-orientation is required to implement our framework. The nature of the models we consider also requires a language which supports pseudorandom number generation and multi-dimensional arrays. We chose C++ as our programming language because it fits these requirements, is relatively fast and is well-known and familiar to the scientific community. A wide range of open-source and reliable C++ libraries are also available which provide relevant trusted functionality and optimised algorithms. Several Boost [22] libraries are used extensively throughout our code, as is the Standard Template Library. We also employ the PETSc [23] library to solve large, sparse linear systems of equations. Simulations are visualised using Paraview [24].

Running a complex multiscale simulation of vascular tumour growth requires the co-ordinated interactions of many objects. One of the aims of our framework was to make the construction and running of model simulations intuitive and simple. An example of tumour growth in 2-D on an embedded vascular network is shown in Figure 5. To illustrate how such a simulation is set up in our framework we also provide a walk-through in pseudocode of the main steps involved in Algorithm 1. The first step in setting up this simulation is to initialise the spatial mesh. The spatial mesh is then used to initialise a `Diffusibles` object, a `VesselNetwork` object and a `CellPopulation` object. Diffusible species of interest are added to the `Diffusibles` object and the `CellPopulation` object is populated with cells, each of which contain various intracellular chemicals: proteins involved in the cell-cycle and species which diffuse throughout the model domain. A `StructuralAdaptationAlgorithm` object is then created and appropriately customised. Finally, a `Simulation` object is instantiated to which the various model elements are added before the simulation is run.

There is considerable scope for changing parameters in submodels and for removing and adding submodels. By employing C++'s class templating capabilities in our framework design, we have also ensured that simulations may be carried out in 3-D by varying a template parameter representing the dimensionality of the system. 3-D simulations of vascular tumour growth will be presented in future work.

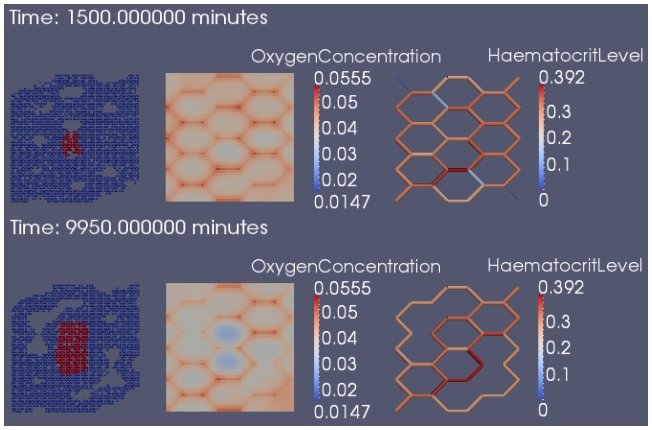


Figure 5: Two snapshots from a 2-D simulation of vascular tumour growth. Cell distributions are shown in the left-hand-side (cancerous cells are red and normal cells blue). The oxygen distributions, in terms of dimensionless concentrations, are shown in the middle panels and vessel networks in the panels on the right. Vessel colours correspond to the haematocrit level inside a vessel and the radius of a vessel is represented by its relative thickness.

A significant advantage offered by our object-oriented framework is the ability to isolate, analyse, test and validate submodels at the various biological scales which we consider. Figure 6 illustrates an example of data extracted from a simulation which considered only the progression of the Alarcón *et al.* 2005 cell-cycle model [4]. This functionality allows us to customise submodels to experimental and/or patient-specific data and facilitates further development.

V. Discussion

In order for the full potential of *in silico* models of cancer to be realised, the re-use of model code and the understandability of models must increase. In particular, attention

 ISBN: 978-1-61208-234-9

Algorithm 1 Example simulation pseudocode.

```
OnLatticeMesh(latticeSiteSize,domainSize_X,domainSize_Y)
Diffusibles(OnLatticeMesh)
Diffusibles.AddDiffusibleSpecies("Oxygen")
Diffusibles.AddDiffusibleSpecies("VEGF")
HexagonallyTesselatedVesselNetworkFactory(OnLatticeMesh)
HexagonallyTesselatedVesselNetworkFactory.CreateVesselNetwork()
CellPopulation(OnLatticeMesh)
for i = 1 .. numberOfCells
        Cell(location)
        Cell.AddIntraCellularChemical("Cdh1")

                ⋮

        Cell.SetMass(initialCellMass)
        CellPopulation.AddCell(Cell)
end for
StructuralAdaptationAlgorithm()
StructuralAdaptationAlgorithm.SetHaematocritCalculator(HaematocritCalculator())
Simulation(CellPopulation,VesselNetwork,Diffusibles)
Simulation.AddCellMover(Betteridge06OccupationBasedMover())
Simulation.AddSubCellularModel(Alarcon05OxygenDependentModel())
Simulation.AddDiffusionCalculator(OxygenCalculator())

                ⋮

Simulation.SetDuration(simulationRunTime)
Simulation.SetTimestep(timestep)
Simulation.Run()
```

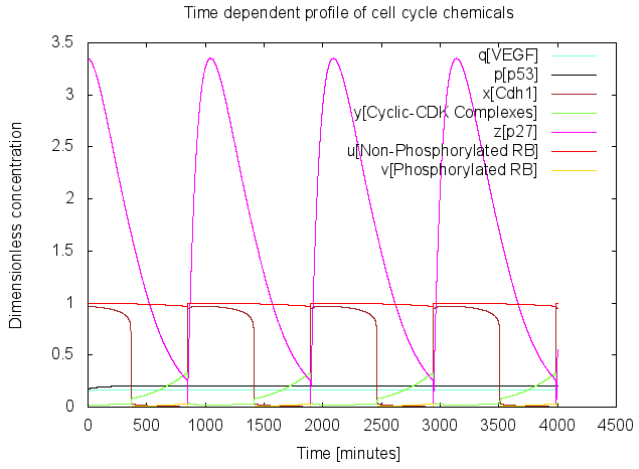


Figure 6: Time-evolution of intra-cellular chemical concentrations involved in the Alarcón *et al.* 2005 cell-cycle model [4] for a normal cell exposed to a dimensionless oxygen concentration of one.

must focus on closing the gap between experimentalists and modellers in order to encourage greater collaboration between them. An object-oriented approach to modelling cancer is ideally suited to these endeavours. That being so, motivated by the multiscale models of vascular tumour growth developed by Alarcón and co-workers, we have developed an object-oriented framework for developing and implementing multiscale models of vascular tumour growth which is both flexible and intuitive to use. Object-orientation has been used to describe elements of the application domain in a way which should be understandable to both experimental biologists and mathematical modellers with relatively little programming experience. We have provided a hierarchy of biologically-based classes which model populations of cells and vasculature, and several physics- and rule-based class hierarchies which describe how those biological entities behave and interact. We have focused on ensuring that code in our framework is understandable and maintainable, and that models implementable in our framework are easily extensible. This was accomplished by exploiting various object-oriented techniques, which have formed the focus of this paper, and by the application of several well-known design patterns, in particular, the visitor and strategy patterns. We have found that, by employing this design, we are able to move from one model implementation to another simply by defining a few new classes in pre-existing class hierarchies. This drastically reduces the development time required to produce a novel model implementation.

The development of our framework has been use-case driven, the primary use-cases being that the models of Alarcón and co-workers must be implementable in our framework. By basing our design around these models, we ensure that it will be more useful than a purely abstract design. Additionally, this family of models employs a diverse range of interchangeable algorithms for modelling various types of behaviour at multiple biological scales. This has

 ISBN: 978-1-61208-234-9

enabled us to explore the extensibility of model code in the context of existing models. The framework provides a convenient plug-and-play environment in which a variety of different models may be implemented. Furthermore, models may be easily deconstructed into their component submodels which may be individually tested, analysed, validated and further developed before being re-integrated into the larger models of vascular tumour growth. Thus, our framework simultaneously addresses what we consider to be the two main challenges in cancer modelling; it facilitates the validation of complex multiscale models of cancer and their extension to incorporate novel data and new functionality.

Other groups have adopted similar approaches to develop multiscale modelling frameworks. In [25], an object-oriented framework was constructed in C++ to support the simulation of avascular tumour growth using an agent-based hybrid approach, and, using another object-oriented framework, Gao *et al.* [26] implemented a hybrid model of avascular and vascular tumour growth. To the best of our knowledge, however, our investigation is the first to explicitly explore the extensibility of model code in the domain of *in silico* oncology.

In wider biological fields, CompuCell3D [27] and Virtual Cell [28] are both well established and widely used frameworks, which aim to facilitate the simulation of models and re-use of model code. Chaste is another object-oriented mathematical modelling and simulation framework whose development to date has focussed on cardiac physiology and multi-cell models of tissue growth and carcinogenesis in the intestinal crypt [29]. Chaste aims to provide a reasonably generic framework for modelling biological systems. The models considered herein could be implemented within Chaste, however, the implementation would not be easily reconcilable with the real-world system. In contrast, the bottom up approach we employ allows us to minimise the conceptual distance between our model code and the corresponding real-world systems. Nevertheless, we plan to exploit existing links with Chaste and other related projects to advance the development of our framework.

Our current study addresses the re-use and extensibility of model code for a set of hybrid models of vascular tumour growth. Whilst focussing on describing the models abstractly, we have not attempted to provide a standard for describing or implementing models of this type. Thus, the models implementable in our framework are not interoperable with similar models from the wider modelling community. We will address this issue by further abstracting concepts within our application domain and developing an XML-based description of hybrid models of vascular tumour growth. By describing the models in a way which is not tied to a specific programming language and by adopting a specific ontology, perhaps extending that under development by the National Cancer Institute [30], we aim to make our framework accessible to the wider modelling community and facilitate the interoperability of model components. By choosing XML as our language of choice we will also enable the easy incorporation of models described in similar XML-based standards such as CellML [31] and the Systems Biology Markup Language (SBML) [32].

This effort will prove highly complementary to the work undertaken by the Centre for the development of a Virtual Tumour (CViT), whose Digital Model Repository (DMR) is now live. The CViT DMR has made great steps forwards in the sharing of cancer models across the entire modelling community, however, the models contained within the DMR are not interoperable since they are not written to an agreed upon standard or even in a common language. The future re-use and interoperation of models within the repository relies heavily on the adoption of a coding standard for cancer modelling which we hope to help establish. Enabling cancer modellers to download markup language (ML) descriptions of models which could be executed on a locally available solver tool, as opposed to downloading model executables, also offers significant advantages with regards to the trustworthiness of a model's execution. Whilst executables downloaded from an online repository should all individually be verified, since they carry the potential risk of having a virus or trojan embedded in them, downloaded ML descriptions need not be verified in this way because they are not capable of supporting the embedding of such malicious materials. Instead only a single tool, used to resolve the ML descriptions and run the models, need be verified.

VI. Conclusion and Future work

We have described some of the OOP techniques which we have found useful in the construction of an object-oriented framework for modelling vascular tumour growth. We hope that this paper will serve as a useful reference for biological modellers who do not have large amounts of experience with OOP, but who may benefit from employing these techniques in their own projects. We have explained the merits of various techniques, outlining specifically how each one may help programmers to produce *in silico* models which are extensible, maintainable, re-usable and understandable.

Our framework provides a convenient and intuitive plug-and-play environment in which a variety of different models may be implemented. It has laid the ground work for the further development of an XML-based domain specific language for modelling vascular tumour growth. The development of this language will form the focus of our future work. We also plan to develop a GUI in which modellers may engage with our plug-and-play functionality directly.

Acknowledgments

AJC acknowledges the financial support of the EPSRC and Hoffman la-Roche. This work was funded in part by award KUK-C1-013-04 made by the King Abdullah University of Science and Technology and by Award Num-

 ISBN: 978-1-61208-234-9

ber U54CAI43970 from the National Cancer Institute. We would like to thank Alexander Manta and Eliezer Shochat (Hoffman la-Roche) for their invaluable feedback.

REFERENCES

[1] F. Brooks Jr, "No silver bullet: essence and accidents of software engineering," *Computer*, vol. 20, no. 4, pp. 10–19, 1987.

[2] M. Page-Jones, *Fundamentals of object-oriented design in UML*. Addison-Wesley, 2002.

[3] T. Alarcón, H. M. Byrne, and P. K. Maini, "A cellular automaton model for tumour growth in inhomogeneous environment," *Journal of Theoretical Biology*, vol. 225, no. 2, pp. 257–274, 2003.

[4] ——, "A multiple scale model for tumor growth," *Multiscale Model Simulation*, vol. 3, no. 2, pp. 440–475, 2005.

[5] T. Alarcón, M. R. Owen, H. M. Byrne, and P. K. Maini, "Multiscale Modelling of Tumour Growth and Therapy: The Influence of Vessel Normalisation on Chemotherapy," *Computational and Mathematical Methods in Medicine*, vol. 7, no. 2-3, pp. 85–119, 2006.

[6] R. Betteridge, M. R. Owen, H. M. Byrne, T. Alarcón, and P. K. Maini, "The impact of cell crowding and active cell movement on vascular tumour growth," *Networks and heterogeneous media*, vol. 1, no. 4, pp. 515–535, 2006.

[7] M. R. Owen, T. Alarcón, P. K. Maini, and H. M. Byrne, "Angiogenesis and vascular remodelling in normal and cancerous tissues," *Journal of Mathematical Biology*, vol. 58, no. 4-5, pp. 689–721, Apr. 2009.

[8] H. Perfahl *et al.*, "Multiscale modelling of vascular tumour growth in 3D: the roles of domain size and boundary conditions," *PLoS ONE*, vol. 6, no. 4, p. (17 pages), 2011.

[9] J. Folkman *et al.*, "Tumor Angiogenesis: Therapeutic Implications — NEJM," *New England Journal of Medicine*, vol. 285, no. 21, pp. 1182–1186, 1971.

[10] S. M. Peirce, "Computational and Mathematical Modelling of Angiogenesis," *Microcirculation*, vol. 15, pp. 739–751, 2008.

[11] H. M. Byrne, "Dissecting cancer through mathematics: from the cell to the animal model," *Nature Reviews Cancer*, vol. 10, pp. 221–230, Jan. 2010.

[12] A. L. Bauer, T. L. Jackson, and Y. Jiang, "A cell-based model exhibiting branching and anastomosis during tumor-induced angiogenesis," *Biophysical Journal*, vol. 92, no. 9, pp. 3105–3121, 2007.

[13] P. Macklin, S. McDougall, A. R. A. Anderson, M. A. J. Chaplain, V. Cristini, and J. Lowengrub, "Multiscale modelling and nonlinear simulation of vascular tumour growth," *Journal of Mathematical Biology*, vol. 58, no. 4-5, pp. 765–798, Sep. 2008.

[14] T. S. Deisboeck, Z. Wang, P. Macklin, and V. Cristini, "Multiscale Cancer Modeling," *Annu. Rev. Biomed. Eng.*, vol. 13, no. 1, pp. 127–155, Aug. 2011.

[15] S. Schnell, R. Grima, and P. K. Maini, "Multiscale modeling in biology," *American Scientist*, vol. 95, no. 2, pp. 134–142, 2007.

[16] M. Fowler, *UML Distilled*, 3rd ed. Addison-Wesley, 2006.

[17] B. Liskov, "Keynote address-data abstraction and hierarchy," *ACM Sigplan Notices*, vol. 23, no. 5, pp. 17–34, 1987.

[18] K. Webb and T. White, "UML as a cell and biochemistry modeling language," *BioSystems*, vol. 80, no. 3, pp. 283–302, Jun. 2005.

[19] A. Gupta, S. Chempath, M. J. Sanborn, A. Clark, and R. Q. Snurr, "Object-oriented Programming Paradigms for Molecular Modeling," *Molecular Simulation*, vol. 29, no. 1, pp. 29–46, 2003.

[20] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, "Design patterns: elements of reusable object-oriented software," *Reading: Addison Wesle*, 1995.

[21] M. R. Owen *et al.*, "Mathematical modelling predicts synergistic antitumor effects of combining a macrophage based, hypoxia-targeted gene therapy with chemotherapy," *Cancer Research*, vol. 71, no. 8, pp. 2826–2837, Apr. 2011.

[22] "http://www.boost.org," 14.09.2012.

[23] "http://www.mcs.anl.gov/petsc," 14.09.2012.

[24] "http://www.paraview.org," 14.09.2012.

[25] A. C. Abajian and J. S. Lowengrub, "An agent-based hybrid model for avascular tumor growth," *UCI Undergrad. Res. J*, vol. 11, 2008.

[26] X. Gao, M. Tangney, and S. Tabirca, "2D simulation and visualization of tumour growth based on discrete mathematical models," *Bioinformatics and Biomedical Technology (ICBBT), 2010 International Conference on*, pp. 35–41, 2010.

[27] J. A. Izaguirre *et al.*, "COMPUCELL, a multi-model framework for simulation of morphogenesis," *Bioinformatics*, vol. 20, no. 7, pp. 1129–1137, Apr. 2004.

[28] I. I. Moraru *et al.*, "Virtual cell modelling and simulation software environment," *Systems biology, IET*, vol. 2, pp. 352–362, 2008.

[29] J. Pitt-Francis *et al.*, "Chaste: A test-driven approach to software development for biological modelling," *Computer Physics Communications*, vol. 180, no. 12, pp. 2452–2471, Nov. 2009.

[30] J. Golbeck *et al.*, "The National Cancer Institute's Thésaurus and Ontology," *Journal of Web Semantics*, 2003.

[31] A. Garny *et al.*, "CellML and associated tools and techniques," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 366, no. 1878, pp. 3017–3043, Sep. 2008.

[32] M. Hucka *et al.*, "The Systems Biology Markup Language (SBML): Language Specification for Level 3 Version 1 Core (Release 1 Candidate)," *Nature Precedings*, Jan. 2010.

 ISBN: 978-1-61208-234-9