Multiple model simulation: modelling cell division and differentiation in the prostate

Alastair Droop, Philip Garnett, Fiona A. C. Polack, and Susan Stepney

YCCSA, University of York, UK, YO10 5DD
`Alastair.Droop|Fiona.Polack|Susan.Stepney@york.ac.uk;`
`Philip.Garnett@yccsa.org`

Abstract. We describe an approach to building a hybrid multi-scale model, using a Petri net model for the top layer, and object-oriented models at the lower layers, with a rigorous definition of how the layers compose. We apply this approach to building a model of prostate cell division and differentiation, with each model layer describing the processes at a suitable level of abstraction. This provides a systematic modular approach to modelling and to model validation, allowing use of diverse modelling techniques in developing multi-scale models with no loss of rigour or validity. We use this prostate model to develop a prototype simulation, and demonstrate that it is capable of simulating biologically-relevant numbers of cells.

Keywords: complex systems, agent based modelling, Petri nets, object models, prostate

1 Introduction

Simulations of complex systems have the potential to complement laboratory based biological research, providing a platform for repeatable experimentation, and for exploration of parts of biological systems that cannot be analysed directly in live organisms or dead tissue. Elsewhere [1, 11, 33–35], we consider how scientific simulations can be designed and validated.

In this paper we present the modelling and simulation of cell division and differentiation in the prostate. The results of this simulation will be used as hypotheses to guide laboratory experimentation by suggesting the cellular behaviours and pathways which are most likely to be involved in early cancer neogenesis.

Prostate cancer is a common disease, accounting for almost 25% of male cancers in the UK. As with many cancers, prostate cancer is a complex disease arising from the aberrant behaviour of a tissue. High-grade prostate cancer is characterised by a breakdown of normal cell differentiation behaviour, leading to a proliferation of terminally-differentiated cells. The study of prostate cancer therefore requires an understanding of the dynamics of a large population of cells. Many of the genomic events that lead to cancer formation are rare, stochastic events. As such, a continuous modelling approach (for example based upon ordinary differential equations) is unable to capture the rich behaviour necessary to model cancer neogenesis. The construction, parameterisation and analysis of the complicated models needed to study prostate cancer requires input from multiple domain experts. The work reported here has been the result of close collaboration between domain experts in the YCR Cancer Research laboratory and software developers in the York Centre for Complex Systems Analysis (YCCSA).

The prostate domain biology provides insight into cell transitions and differentiation, transition rates and proportions of each cell type that can be expected in normal, modified and cancerous prostate. However, as in all systems biology, there are significant gaps in the domain knowledge, and there are many features that cannot be accurately studied. In simulating the prostate cells, we seek to be able to (a) replicate the cell behaviours and proportions in a normal prostate; (b) replicate the modifications that biologists make – for example, knock-out experiments to determine causality – and to show that appropriate behaviours are simulated in the modified-prostate simulation; (c) to experiment with the various forms of point mutation that have been postulated as initiators of prostate cancer. The need to precisely measure rates and proportions, and, in particular, the need to accurately express and explore interventions and point mutations, dictates the use of an individual-based model; we use an agent-based simulation. In modelling, we use diagrammatic approaches, which are easy to explain to domain experts, as well as being familiar software engineering tools.

For the simulation, key features are the model of state change (differentiation) and cell creation (division) in the prostate. We need design notations that help us in validating the domain and conceptual models with our domain experts. These models must also admit continuity in development, and be verifiable. We also need to view the development process as part of the construction of an argument of validity, which requires us to systematically identify and record assumptions.

We follow the CoSMoS process: a principled, systematic development, in which domain experts co-operate closely with simulation developers. A *domain model* captures the domain experts' view of the part of the system to be simulated, and the desired scope and purpose of simulation. From the domain model, a *platform model*, suitable for simulation development, is produced. The *simulation platform* is then constructed using a systematic software engineering process. Validation is emphasised, both in relation to the biological bases of the simulation and the design decisions of the software engineering.

In [35] we note that the validation of the implemented simulation is facilitated if the software engineering models use modelling approaches that map seamlessly with the implementation platform and language(s). In simulating aspects of prostate cell processes, we demonstrate that appropriate modelling not only facilitates software engineering and validation, but can also facilitate modelling and validation of the biology underpinning the simulation and simulation experimentation.

1.1 Modelling notations for systems biology

Petri nets are widely used in modelling biological systems. Petri nets (see §2) are based on state-transition diagrams and the theory of automata, they have strong mathematical underpinnings, and can provide formal verification of properties. However, in computational modelling of biological systems most uses encounter the limitations of reactive-systems modelling. For example, for the modelling of biological pathways, it has been found necessary to extend Petri net notations with continuous features: Matsuno et al propose, and extensively use, *Hybrid Functional Petri Net with extensions* to accommodate continuous places and transitions [28, 31]. Such adaptations of Petri nets have the advantage (usually) of maintaining the strong formal basis. However, the resultant models are poorly structured and hard to read – this inhibits validation of models against the biology. The scale and density of the Petri net models also makes them hard to implement in current computer languages.

In 1987, Harel [14] noted the inappropriateness of conventional state-transition modelling for transformational systems. His widely-adopted answer is statechart diagrams, "a visual formalism for describing states and transitions in a modular fashion, enabling clustering, orthogonality (that is, concurrency) and refinement, and encouraging 'zoom' capabilities for moving easily back and forth between levels of abstraction." [14, p233]. Statechart-like notations are readily understood by many biologists [12], and have been used in biological modelling [22, 43]. However, statechart modelling also suffers from aspects of inadequacy. In systems with significant state (those where data are generated, stored and used, and values persist over time), statecharts cannot capture data structures or transactional (and reactive) structures adequately. Like Petri nets, attempting to use a single statechart to capture a heterogeneous biological system results in a complicated, dense model which is hard to validate and to implement from.

Some biological modellers turn to UML-like modelling notations. UML (Unified Modeling Language) [32] has many attractions: it provides notations for modelling static and transactional structures as well as statechart-like notations; there is conceptual unification across notations (through a metamodel-based language definition); there are lots of tools; some of the notations map readily to common programming languages (principally, Java). However, UML is a unified *object-oriented (OO) modelling* language, and the connotations of object-orientation are not always appropriate for biological systems. The most significant issues for modelling cell division and differentiation are that objects cannot reproduce and divide, and an object cannot change its type (class). Continuous concepts such as cell populations and concentrations do not map well to the instance-based OO modelling concepts (class, object, slot, value). These problems are usually ignored in biological modelling [11, 12, 38–40, 48, 49]: it is considered sufficient that the diagrams say something about the biological structures, and map to Java.

To summarise: Petri nets are a clear, elegant way to model reactive systems; statecharts allow clarity in the modelling of transformational systems; OO modelling notations such as UML provide useful and complementary visuals if we can ignore issues of semantics.

1.2 Hybrid models

Most modellers use one base notation and extend it, adapt it or abuse it to model the whole of a biological system. However, this approach loses much of the clarity that abstract modelling should bring to a problem.

There is some multi-notation modelling in biology. Harel et al [7, 8, 15, 16] address the inadequacies of statecharts for biological modelling in Reactive Animation, a proprietary modelling framework that uses statecharts for transformational aspects, object models for static structure, and live sequence charts for transaction structures. The notations are integrated through their roles in generating and maintaining a simulation implementation. Setty, Cohen, and Harel [16, 42] use Reactive Animation to model pancreatic organogenesis. Statecharts are used to model the individual cell lifecycle; other models capture the morphology of the system. The cell division process is captured only informally: "Proliferation ends when the `Cell` duplicates itself by creating an identical instance. In turn, a message is sent to the front end, which creates a new identical sphere corresponding to the new `Cell` at the proper location." [16, p9]. In Reactive Animation, models are complementary, and the aim is to progress as far as possible in the analysis *without* connecting the models [42].

The hybrid modelling approach that we use is not proprietary (or tied to the modelling notations used here): it focuses on finding an appropriate notation for each aspect of the system, and an appropriate way to integrate the component models, for example through input-output mappings. Whilst we anticipated that this would cause problems in implementation, it turns out to be a natural way to construct and structure simulation code. Appropriateness is determined primarily by the ability of the notation to express domain concepts in ways that the domain experts, as well as the simulation developer, can understand. However, it is also important that models map clearly to code concepts, to support software validation.

1.3 Structure of this paper

In this paper, we use multiple models, apply them to modelling cell differentiation in the prostate, and reflect on the advantages, opportunities and limitations of such an approach. §2 introduces Petri nets and Petri net terminology, and introduces a high-level model of cell differentiation and division. §3 introduces three notations (loosely based on UML) that are used to model the lower-level individual cell behaviours. The behaviour of a cell is determined by its place in the Petri net, and its environment. A key aspect is that, in moving between two places, a cell undergoes a change in its active behaviours, and its internal structure. §4 describes how the different models and layers can be combined. §5 discusses how our approach amounts to defining a domain specific language. §6 presents a prototype model of the prostate cell structure. §7 describes the simulation implementation: this prototype replicates the dynamic changes in cell-type proportions in a normal prostate. §8 outlines further work in developing the modelling approach, and in the prostate model itself. §9 presents our conclusions.

2 The high level system structure: Petri net modelling

The high level Petri net model captures the structure of the cell division and differentiation process, without considering the detailed internal structure of the component cells. Petri nets are widely used to model signalling and pathways in biological systems (see, for example, [5, 13, 27, 41]), and are attractive to many systems biologists. They provide a natural way to model cell division (one object becoming two objects) and cell differentiation (one object becoming a different type of object).

2.1 Petri net concepts and definitions

The (draft) Petri net ISO standard website [17, 18] defines a Petri net as "a formal, graphical, executable technique for the specification and analysis of concurrent, discrete-event dynamic systems". There are many flavours and varieties of Petri net; here we describe only the features necessary for our model. The interested reader is referred to the extensive bibliography accessible from the ISO standard website.

A Petri net is a bipartite directed graph. The two kinds of nodes are *places* and *transitions*. Place nodes have a *marking*: a set of *tokens* occupying the place (figure 1a). A transition can *fire* when all its input places hold a token: one token per input arc is *consumed* from its input place, and one token per output arc is *produced* in the respective output (figure 1b). Some descriptions instead talk of tokens flowing around the net; here we emphasise that tokens are consumed and produced by transitions, and so emphasise that their numbers and types need not be conserved, as the focus of our modelling is on the production of new cells and new kinds of cells.

When Petri net models get large, it can be helpful to structure them. Fusion places [19, ch.5], an extension to standard Petri nets, allow a Petri net to be presented in several pieces. A fusion place is marked on several nets, which means it should be identified across the nets (figure 2). We use fusion places to help structure the chain of cell differentiations. A fusion transition can be used to capture an entire sub-net; we could use a simple version to help capture the repeated pattern of cell divisions, but do not do so here (see §8.3).

2.2 Modelling cell interactions

In our Petri net model, each cell is modelled as a token, and each type of cell is modelled as a place. The way cells differentiate (change type) and divide is

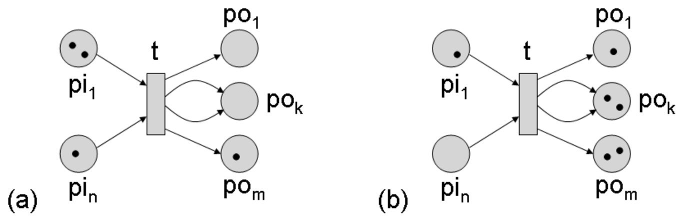

Fig. 1. Example Petri net. (a) A net with n places $\mathsf{pi}_1 \ldots \mathsf{pi}_n$, input to the transition t, which has outputs to m places $\mathsf{po}_1 \ldots \mathsf{po}_m$. Place po_k has two incoming arcs. The net's current marking is 2 tokens in place pi_1, one token in pi_n, and one token in place po_m. (b) The same net after transition t has fired. One token has been consumed from the place along each input arc, and one token has been produced in the place along each output arc. Hence two tokens have been produced in place po_k, one for each of its arcs.

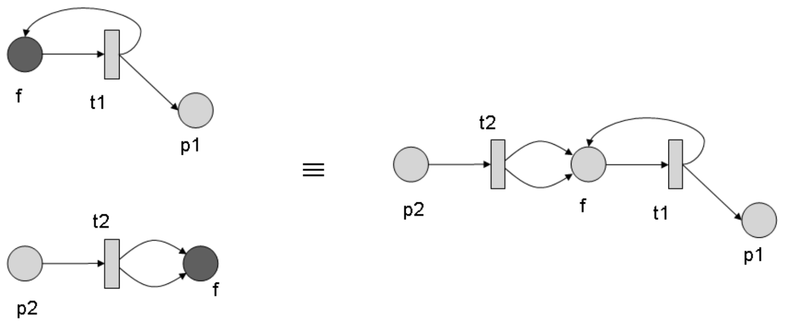

Fig. 2. Example fusion place. The two nets on the left have the fusion place f (dark grey). They are equivalent to the net on the right.

modelled by the transitions. We name all the places and transitions in the net; these names are used to link the Petri net model to the lower level models (§3).

Figure 3 shows a variety of differentiation and division processes modelled using a Petri net. These various possibilities can be captured in a single model of cell differentiation, figure 4. (Our actual model of prostate cell differentiation and division includes more detail; see §6.) There are two possible models. Figure 4a shows the case where the two daughter cells are distinguished separately. This allows asymmetric transition probabilities (with d1 being highly likely to remain a p1 cell, and d2 being highly likely to differentiate, for example). Figure 4b shows a simpler representation, that cannot capture the asymmetry at this level of modelling. As we include a lower layer of modelling, we use the simpler representation, and devolve handling the asymmetry to the lower level.

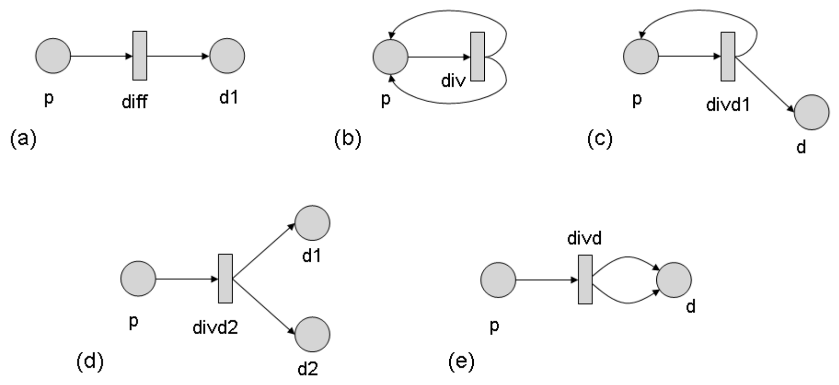

Fig. 3. Simple Petri net models of cell differentiation and division. (a) cell of type p differentiates into cell of type d1; (b) parent cell of type p divides into two daughter cells also of type p; (c) parent divides into two daughter cells, one also of type p, one differentiated into a daughter cell of type d; (d) parent divides into two cells, one differentiated into type d1, one into type d2. (e) parent divides into two cells, both differentiated into type d.

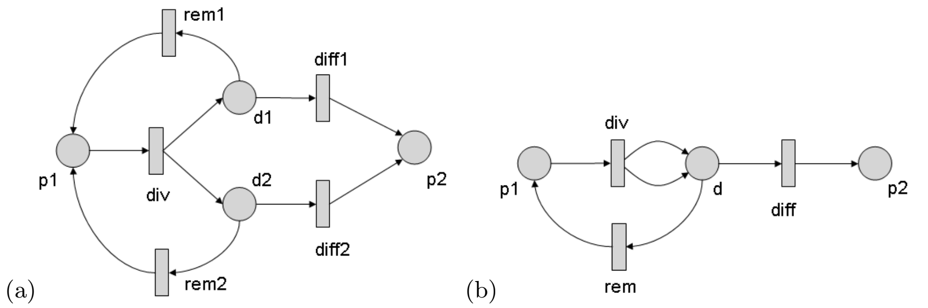

Fig. 4. A Petri net model of cell division incorporating all differentiation possibilities. (a) a cell p1 divides into two daughter cells d1 and d2, which each chose either to remain a p1 cell type (transition rem), or differentiate into a p2 cell type (transition diff). (b) a cell p1 divides into two daughter cells d, which each chose either to remain a p1 cell type (transition rem), or differentiate into a p2 cell type (transition diff).

3 The low level component structure

The low level component model captures the behaviour of an individual cell through its life cycle, and the details of the division process where one cell becomes two, and includes interactions with the environment. We use object-oriented modelling notations to capture this structure (see, for example, [9]).

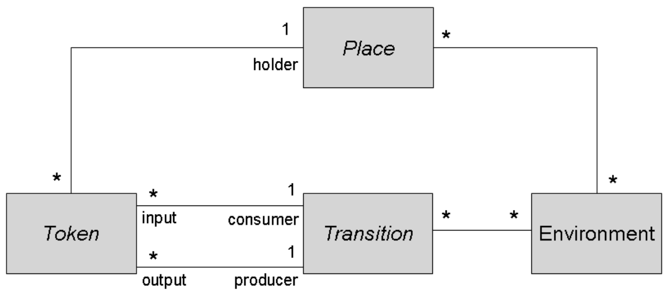

Fig. 5. The class model for the tokens. A token is in a place. Token and Place are abstract classes; there is one subclass of each for each Petri net place. A transition may consume many input tokens; a token is consumed by precisely one transition. The transition may produce many output tokens; a token is produced by precisely one transition. Transition is an abstract class; there is one subclass for each Petri net transition. Places and transitions may be associated with many environments, which can affect their behaviour.

3.1 Modelling cells: classes and objects

A cell, modelled as a Petri net token in the system layer, is modelled as an object (instance of a Token class) with state and behaviour in the component layer. Each place is modelled as a (singleton) instance of its Place class (every place is different), as it too can have state and behaviour. The behaviour of a transition is modelled by an instance of the Transition class, which *consumes* token objects from the input places, and *produces* token objects into the output places. We also allow the model to have environment(s) (instances of the Environment class or its subclasses), to provide inputs to the behaviours (for example, to the guards on state changes, or to the mutation rate on cell division). The relevant class model is shown in figure 5.

3.2 Modelling cell state: state diagrams

The behaviour of a token in a place is modelled using a state diagram; each place has a different kind of token (cell), and so has its own state diagram. This has one or more start states, labelled with a name (corresponding to a Petri net layer transition name, or to some initialisation), and one or more end states, also labelled with a transition name. For example, see figure 6.

3.3 Modelling cell transitions: sequence diagrams

In the Petri net layer, the transition is where a cell divides into two new cells, or differentiates into a new kind of cell. The low level modelling allows details of these processes to be specified, such as what information about the cell survives the transition, and what changes. We use a transient transition object for this, which ‘consumes’ the input cell, and ‘produces’ the new output cell(s). In some

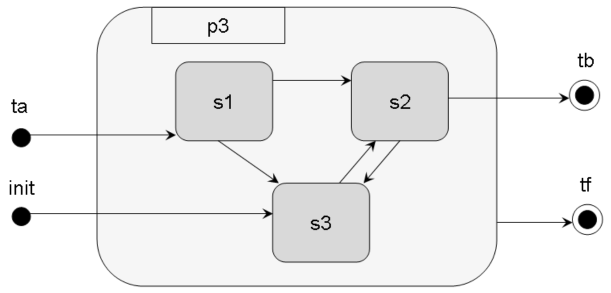

Fig. 6. An example state diagram. If the token in place p3 in the Petri net layer is produced from transition ta, it starts in state s1; if produced externally, it starts in state s3. The cell can exit to transition tb only from state s2; it can exit to transition tf from any substate.

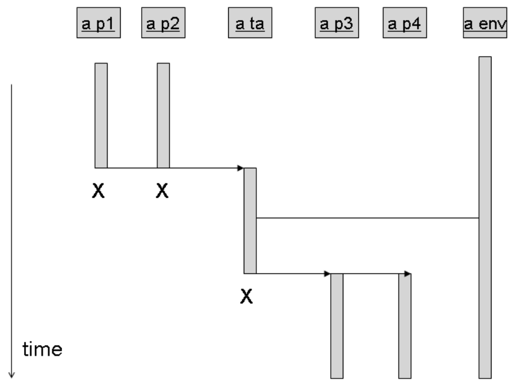

Fig. 7. An example sequence diagram. The input tokens a_p1 and a_p2 are consumed by the transition object a_ta (which can therefore have information about their state). The transition object produces tokens a_p3 and a_p4; it can initialise them with processed information from the inputs, and from the environment a_env. The transition object exists only for the duration of the interaction. X indicates object termination.

sense, this transition object can be thought of as 'being' a cell in its transition state of dividing or differentiating.

So, in object-oriented modelling terms, a transition object (figure 5) exists for the duration of the transition behaviour. It is initialised with any necessary information from the input token object(s) and the environment, performs appropriate behaviours, and initialises the output token object(s). The input tokens(s) terminate once the transition object is initialised. The transition object terminates once the output token(s) are initialised. The interactions can be modelled with a sequence diagram; a typical example is shown in figure 7.

3.4 Introducing the environment

Cell behaviour (maturation, differentiation, and division) is affected by environmental factors as well as internal properties. It is important to capture environmental effects in the overall model; that is the purpose of the Environment class in figure 5.

The cell model has multiple levels of detail and abstraction, and the environment can interact with all these. Environmental effects may be global (for example, a global irradiation of tissue), or local (for example, specific to a particular spatial region or type of cell). The environment can affect rates (transition rates within a state model, transition rates within the Petri net model) and behaviours (choice of transition within a state model due to guard values). The environment can itself be affected by the cells (for example, cells excreting chemical signals that are transported by the environment to other cells).

As much, or as little, should be put into the environment as is needed for the particular modelling purpose. This enables environmental effects to be modelled explicitly and rigorously within the same framework as the cell modelling, but cleanly separated from the individual cell models.

4 Combining the layers

The high level Petri net model and the detailed object-oriented models represent the layers of the system. These layers are combined by linking the Petri net place and transition names to the component level class names and state diagram labels. The token classes are named after the Petri net place names, the place classes are named Place_X, where X is the relevant token class name. The start and end points on the state diagram are labelled with the relevant Petri net transition names. The transition classes are named after the Petri net transition name.

This approach provides a simple and straightforward, but nevertheless explicit and rigorous, linking of the two layers of the model. The cell state behaviour, cell differentiation and division, and environmental effects are all modelled. These models make sense individually, at suitable levels of abstraction, and provide a modular approach to modelling and to model validation. How the models are related is clearly defined through this linkage process, and so there is a clear mapping from the model to the code, further easing validation.

Systems biology aims to analyse biological phenomena as dynamic, interacting systems rather than individual components [20, 24]. This high-level approach allows us to address the complexity inherent to biological systems in a way that is impossible using conventional, reductionist molecular biological approaches. A major limitation to the utility of systems approaches to biology is that very many components across multiple scales need to be simultaneously modelled. Building single models that capture multiple levels of biological complexity is extremely difficult. The modelling strategy proposed here provides an intuitive framework for combining models at different biological scales (in this case a tissue differentiation level and a cellular level) in such a way that each level can be treated separately. This separation of levels allows for the construction of manageable models, whilst providing the ability to build multi-scale models.

5 A Domain Specific Language

Our use of different models and notations effectively amounts to a domain specific language (DSL) [10, 29] for modelling cell differentiation and division. As with any DSL, we develop the DSL by extending and adapting existing languages. There are three principle extensions and adaptations.

For the detail of cell behaviours, the models are based on object-oriented modelling notations. However, we apply an agent semantics, rather than the object semantics defined in, for example, UML2.x [32]. The agent semantics means that an object is instantiated to an active state and allocated its own execution thread. The object persists until it is consumed, and its thread terminates. This alternative semantics allows us to map cleanly from the low-level specification to an agent-based implementation, aiding software engineering validation. Furthermore, this semantics allows us to express concepts in a biologically meaningful way, supporting biological validation.

For the system level, we use Petri nets. However, we modify the semantics of transition firing. In standard Petri nets, a transition fires when a token is in a place. In our usage, the firing of a transition is determined by the state diagram of the relevant place: a token (representing a cell) remains in a place until the conditions are achieved for firing a transition. This may depend on the environment, or simply on passage of time to maturity. Again, this semantic adaptation allows a biologically-meaningful representation of cell behaviours. Note that a common alternative approach is to introduce a timing distribution to control transitions; even if based on biological observation, this is a coarser solution, providing a less biologically-realistic determination of transition rates.

The third key element of our DSL approach is to define the linking between layers (§4). Like UML syntax, concepts in our models are named, and the names are used to link the detailed and system level models, as described above.

The DSL approach opens several novel directions for our research (see also §8). Firstly, we can exploit the metamodelling approach to DSL syntax and semantics, to produce tool support for our models. Secondly, we can exploit the formal underpinnings of Petri nets, and the fact that state diagrams and other OO modelling notations can be represented as Petri nets [30, 45], to support formal analysis of the cell behaviours. The potential of the DSL approach differentiates our simulation and modelling from other multi-model solutions (for example, the work of Harel et al discussed in §1.1), where model integration is done at the implementation level, through a computational framework.

6 Domain model of prostate cell division

6.1 Biological domain

Cancer is a phenotype arising as the result of aberrant interactions between many individual cells. The study of cancer is therefore the study of cell population dynamics. Prostate cancer is the most commonly diagnosed cancer in UK men, accounting for almost 25% of all male cancers in the UK [46]. The UK lifetime risk for getting prostate cancer is approximately 1 in 10 men [46]. Mutations in a wide range of genes (*oncogenes*) are known to increase the risk of cancer [4]. Although the presence of mutations in oncogenes confers an elevated risk of cancer, there is a high level of variability in the timing and exact genotype of cancers. This stochastic element makes the analysis of cancers at the expression level very difficult. Stochasticity and genetic variability make cell population modelling a very attractive tool for the study of cancer neogenesis.

Several cell populations are present in the prostate (figure 8). The majority of prostatic tissue is composed of stromal cells which consist of connective tissues and blood vessels. The tissue of interest with respect to prostate cancer is the prostatic epithelium, which consists of basal, secretory and neuroendocrine cells [6]. The secretory cell population consists of terminally-differentiated columnar cells. The basal cell compartment contains less-differentiated cells that are still in contact with the basement membrane.

The presence of small numbers of self-renewing stem cells in most tissues is now widely accepted. The stem cell population is able to replace dead cells in the tissue by the processes of division and differentiation. Stem cells are able to undergo two types of division: symmetric division in which a single stem cell divides to yield two similar daughter stem cells; and asymmetric division in which a single stem cell divides to give rise to a daughter stem cell and a daughter cell of a more differentiated phenotype.

The role of stem cells in the formation and maintenance of tumours is not well understood. Viable mutations in a stem cell will be passed on to its large number of progeny by the normal processes of cell division and differentiation. The cancer stem cell model [25] suggests that, if these stem cell mutations confer a malignant phenotype, then these progeny will form a cancerous population (a tumour). If this model is correct, then the stem cell population is of utmost importance in cancer treatment; the common therapy of removal of the bulk tumour mass will not impact the long-term patient survival rate, whereas ablation of cancerous stem cells would allow successful treatment.

Here we are studying a population of cells moving through a differentiation topology (modelled by the high-level Petri net). The decision-making processes and the underlying molecular biology are encapsulated in the lower level models. The two-level modelling allows us to easily create appropriate agent-based simulations of cell population dynamics. The cell types used in the model are illustrated in figure 8.

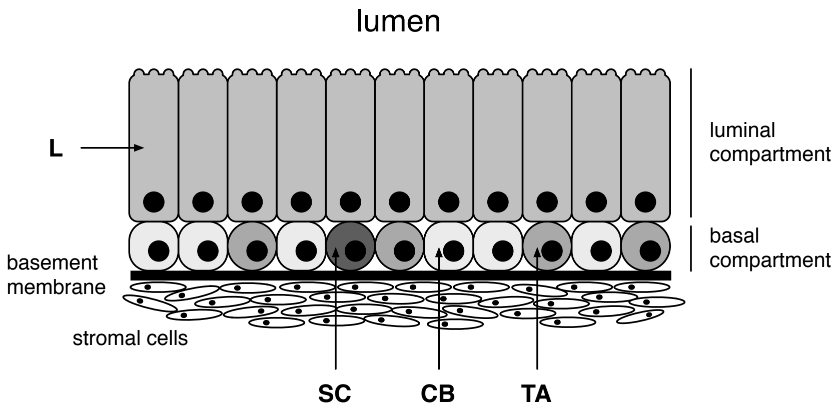

Fig. 8. The major cell types present in the prostate. The stromal cell compartment contains various structures, for example blood vessels, and is not modelled in our work. **SC**: stem cells; **TA**: transit amplifying cells; **CB**: committed basal cells; **L**: luminal cells. The transit amplifying cells in our model include both non-stem basal cells and transit amplifying cells. Figure adapted from [26].

6.2 The division pathway model

Figure 9 shows our Petri net model of prostate stem cell division and differentiation. A stem cell can divide; its daughters can each either remain a stem cell, or differentiate to a Transit Amplifying cell TA. In figure 9 the place TA is shown as a *fusion place*, indicating it occurs in another Petri net in this model (here, in figure 10). A stem cell can also differentiate directly to a TA cell without division, or can undergo apoptosis, and become dead. A TA cell can revert to a stem cell.

Figure 10 shows our Petri net model of TA cell division and differentiation. The pattern is similar to the SC diagram, except that a committed basal cell (CB) cannot revert to a TA cell (there is no analogue of the rev_{TA} transition). Note that the dead place is not modelled as a fusion place: each cell type has its own dead place. This is because a dead cell is still present in the tissue, taking up space and potentially influencing different locations of the environment in different ways (for a while at least; a more sophisticated model could have a further transition to a Decayed place).

Figure 11 shows our Petri net model of CB cell division and differentiation. The logic of division and differentiation is identical to the TA cell model in figure 10, but with different cell types and transition types.

Figure 12 shows our Petri net model of luminal cell division. A luminal cell can divide into two luminal cells (although this happens at a very low rate in normal tissue, it is included here because it occurs in the cancerous case). Note we do not need the intermediate daughter cell places in this model, as biologically there is no “remain or differentiate” choice in this division process. A luminal cell can also undergo apoptosis, and and become dead.

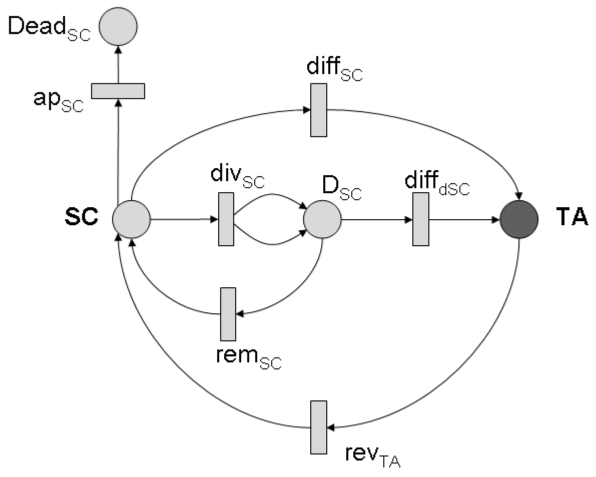

Fig. 9. A Petri net model of stem cell (SC) division and differentiation. A stem cell can divide (div_{SC}); its daughters D_{SC} can each either remain a stem cell (rem_{SC}), or differentiate (diff_{dSC}) to a Transit Amplifying cell TA. A stem cell can also differentiate (diff_{SC}) directly to a TA cell without division, or can undergo apoptosis (ap_{SC}), and become dead (Dead_{SC}). A TA cell can revert (rev_{TA}) to a stem cell.

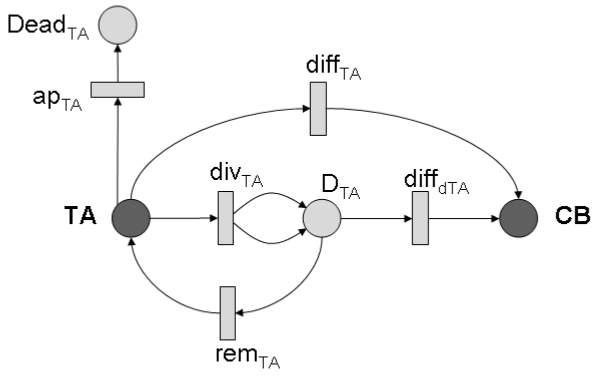

Fig. 10. A Petri net model of transit amplifying (TA) cell division and differentiation. The pattern is similar to the SC diagram, except that a committed basal cell (CB) cannot revert to a TA cell.

6.3 The cell state model

Figure 13 shows the class model of the place tokens, with a subclass for each place type (the daughter cell subclasses and dead cell places are omitted for clarity). This is an instantiation of figure 5, and as such could mostly be automatically derived by a tool. We have chosen to bundle together the similar D classes and the Dead classes into abstract classes, and not show the individual subclasess, for

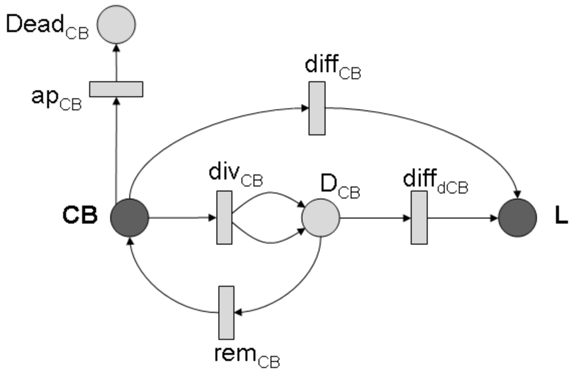

Fig. 11. A Petri net model of committed basal (CB) cell division and differentiation. The logic is identical to the TA cell model in figure 10.

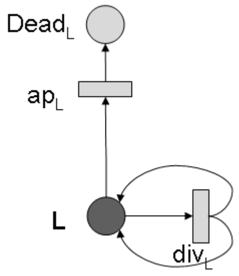

Fig. 12. A Petri net model of luminal (L) cell division. A luminal cell can divide (div$_L$) into two luminal cells, or can also undergo apoptosis (ap$_L$), and and become dead (Dead$_L$).

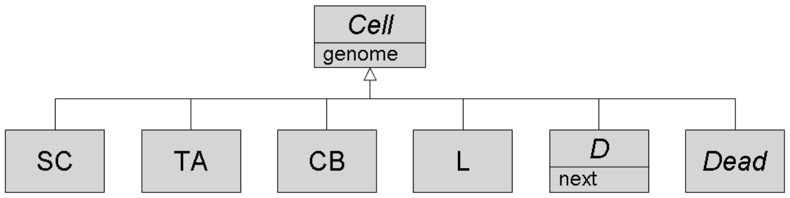

Fig. 13. The class model of the cell tokens (the tokens in the various places). Apart from the Cell and D instance variables, this can be automatically generated from the Petri net model. There is a parallel class model for the places themselves.

brevity. The only addition compared to figure 5 is the instance variable genome in the abstract Cell class, and the instance variable next in the abstract D class.

The next instance variable indicates whether the daughter cell should preferentially remain as its originator type, or differentiate into the next type of cell. This captures the asymmetry of the cell division, and leaves enough flexibility for

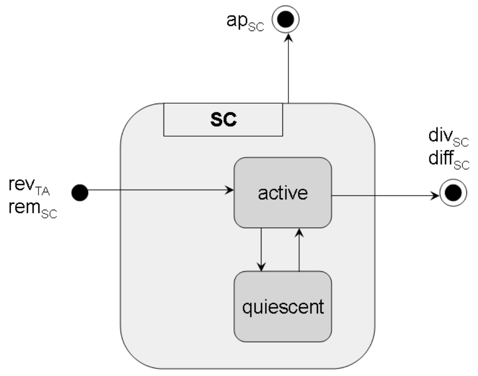

Fig. 14. A state model of a stem cell (a token in the SC place). It has two sub-states: quiescent and active. It is produced in the active state, and it exits to differentiate or divide from the active state. It can apoptose from any state.

the actual transition taken to be influenced by environmental factors in addition to the value of this instance variable.

The genome instance variable encodes an individual cell's transition rates and mutation rates. The genome is the part of the state that is preserved (possibly slightly modified) as a cell transitions between places. Since the genome is modified at certain points (particularly on division due to copying errors, but also more slowly at at other times), we can model the fact that mutations change an individual cell's transitions rates, which potentially initiate cancerous behaviours. Thus our model does not have a 'flag' that says whether the cell is cancerous or not (cancer is a phenotypical property of a collection of cells, not a property of a single cell), but is a model of a population of cells with varying properties, which allows investigation of the situations that lead to the *emergence* of a cancerous phenotype.

Figure 14 shows our state model of a stem cell SC. It has two sub-states: quiescent and active. It is produced (via reversion, or from a divided daughter cell) in the active state, and it exits to differentiate or divide from the active state. It can apoptose from any state.

Figure 15 shows our state model of a TA cell. It has three sub-states: juvenile (just produced), excreting (active)[1], and mature (ready to divide or differentiate). A TA cell is produced (via differentiation of a stem cell, or from a divided daughter cell) in the juvenile state. It can revert to a stem cell from this juvenile state; it can differentiate or divide from the mature state. It can apoptose from any state.

We omit discussion here of the CB and Luminal L cell state models, for brevity. The state models of the various daughter cells are essentially trivial

[1] The term *excreting* is used here to refer to a metabolically active cell before it commits to differentiation, division or death. Excreting cells in this model can emit signals to the environment.

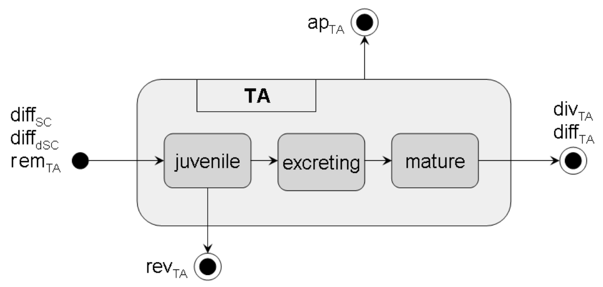

Fig. 15. A state model of a TA cell (a token in the TA place). It has three sub-states: juvenile, excreting, and mature. A TA cell is produced in the juvenile state. It can revert to a stem cell from this juvenile state; it can differentiate or divide from the mature state. It can apoptose from any state.

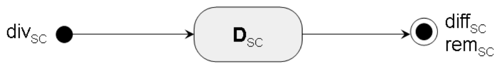

Fig. 16. A state model of a daughter cell (a token in the D_{SC} place). The model for the other daughter cell places has identical logic.

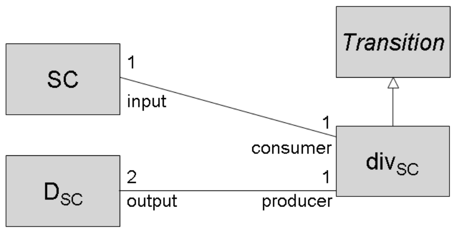

Fig. 17. A class model of the div_{SC} transition. This can be automatically generated from the Petri net model.

(figure 16), having no substates. The state models of the various dead cells are even more trivial (no substates, one input, no outputs).

All the transitions in these state models have conditions and rates associated with them, some affected by environmental conditions. We omit discussion of these, again for brevity.

6.4 The cell transition model

There is a subclass of Transition for each transition in the Petri net (not shown). The subclassing is an instantiation of figure 5, and automatically generatable from the Petri net model. Figure 17 shows our class model of a div_{SC} transition, an (automatically generatable) instantiation of figure 5.

The operation of the specific transition object is to take the genome of the parent stem cell, and 'replicate' it, subject to an environmentally and genomically specified mutation. So, in pseudocode:

```
div_SC()
    output1.genome = mutate(input.genome, env)
    output2.genome = mutate(input.genome, env)
```

This gives two potentially different resulting genomes, as the `mutate()` operation is stochastic.

6.5 The full combined model

All the separate models are combined automatically by matching cell layer and Petri net layer names, as described in §4.

6.6 Model and simulator validation process

Validation is a continuous review process between the biological domain experts, the modellers, and the simulation developers [33]. The validation process for the models presented in this paper comprised walk-throughs of the diagrammatic models, with highlighting and discussion of the appropriateness of assumptions and abstractions. Meeting notes are recorded in the project wiki.

In the prostate cancer modelling, our domain expert is a group of researchers from Maitland's Yorkshire Cancer Research lab at the University of York. One of the development team is also a member of this lab; his roles are: to identify biological issues as they arise; to provide background and interpretation of the biology for the developers; and to set up review meetings at which both developers and lab members are represented. In producing the first prototype simulator, there were four major review meetings, at which diagrammatic models were discussed in detail, and changed to better represent the biological understanding of the laboratory researchers. The developer team comprised an implementation expert, two modelling experts and the link-biologist, who has experience of modelling and simulating biological systems. Many of the advances in modelling the prostate cell behaviours occurred in regular discussions among the team.

A validation argument for the models and simulator has been captured in a rigorous form after the modelling [36]. The information needed for this argument was captured during the development process.

7 The platform model and prototype simulator

In moving from the domain model to the platform model, we focus on the implementation detail of the simulator. This requires design decisions about the implementation of the concepts in the domain model.

At the time of writing, we have completed a first prototype simulator, using a platform model developed from the domain model in §6, which was fully validated with YCR biologists before implementation started. The prototype simulator is being calibrated, and we are testing its scalability properties and performance.

7.1 The platform model derived from the domain model

To implement the prostate cell behaviours, we chose to use a process-oriented programming language, since this provides a natural implementation for the state-transition structure of the high-level model in §6. For prototyping, we use the JCSP (Java Communicating Sequential Processes) class library for Java: this also gives us a seamless development from the low-level OO models.

JCSP[2] provides a natural way to capture the agent semantics. Any class that implements the JCSP class CSProcess has instances that each run in its own separate Java thread. The Petri net domain model places and transitions map to platform model CSProcess classes, and the Petri net arcs map to communication channels between the processes.

The use of JCSP requires the addition of structures to implement process synchronisation: JCSP barriers to synchronise operations, so that all parts of the simulation experience the same number of time-steps, at the same time.

The prototype simulation is developed from the stem cell division and differentiation domain model (figure 9), with the addition of transit amplifying cell apoptosis (figure 10). The platform model is summarised in a Java class diagram (figure 18). Each place and transition from the domain model is a singleton Java class that implements the JCSP class CSProcess. Because places and transitions have common characteristics, inheritance hierarchies are used (not shown in figure 18).

The current prototype simulator includes stem cells (SCs), daughter stem cells, TA cells, and dead stem and TA cells. Stem cells move between the substates active and quiescent, and can apoptose, differentiate or divide (figure 14). The TA cells have only limited behaviour: they are either active or resting, and can only revert to a stem cell or apoptose.

Cell division and differentiation in the platform model is derived from the Petri net structures, with the detailed actions of each behaviour defined in the state diagrams of the domain model. The sequence diagrams in the domain model (not illustrated) provide the interaction detail for the implementation.

The cell state and operations are defined on the hierarchy of Cell classes (implementing figure 13) Each cell type has appropriate methods for changing its state, and an instance of the Genome class. The design decision to extract the genome as a Java class supports the operations needed to enact the state-diagram state changes (such as the move from active to quiescent states of the stem cell, figure 14). The genome stores and sets probabilities for different cell events, which gives the necessary control over state change for biological experimentation with the simulator; as in the cell biology, the genome representation allows differentiation to change the ‘active genes’ in the genome.

[2] http://www.cs.kent.ac.uk/projects/ofa/jcsp/

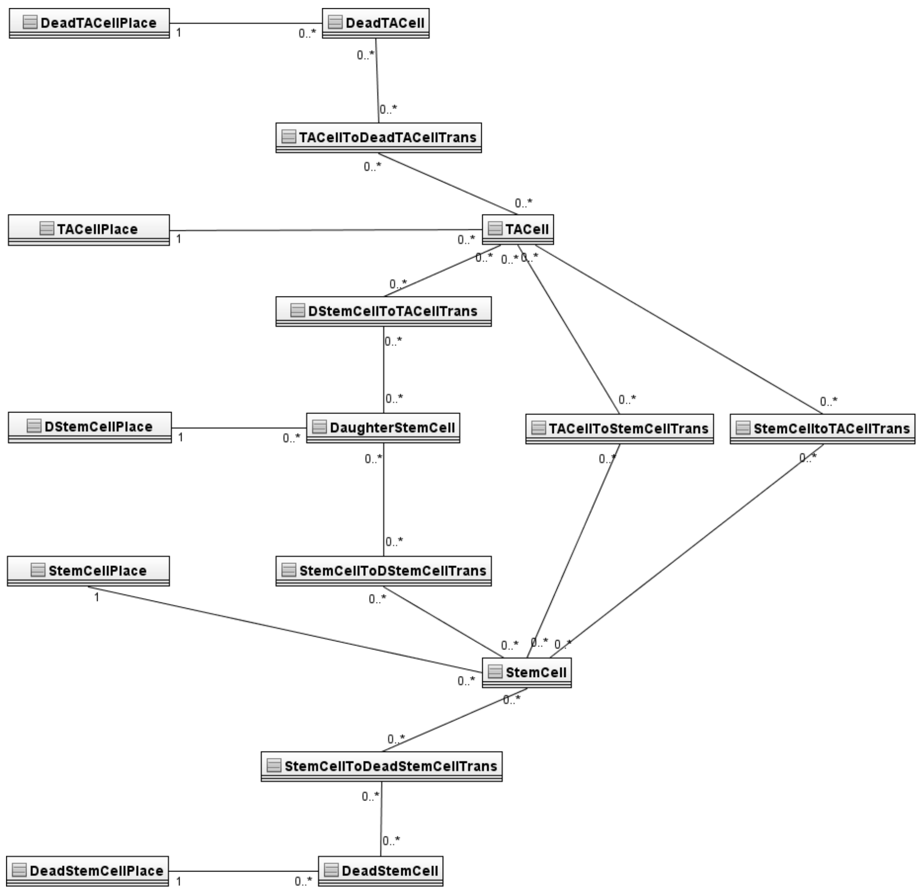

Fig. 18. Platform model of places, transitions, and cells.

The platform model also has a basic Environment class. This is used to represent a global environment, the simulated prostate tissue. Cells can poll the environment at any time, and the results affect the specific behaviour of each cell. The interaction of the cells with the environment will be used to simulate global mutation rates, and ultimately potential treatments that could alter the mutation rate of the whole prostate tissue. There is significant further development to be done, of which the refinement of the environment, and of the behaviour and regulation of the genome during the life of a cell are perhaps the most interesting and potentially challenging.

7.2 The platform model implementation additions

The platform model needs extra implementation specific features not derived from the domain model: buffers between places and transitions; JCSP barriers to synchronise processes; a global clock to simulate time.

JCSP multi-threading provides better performance than a purely-sequential implementation. Threads are used for places and transitions, whilst individual cells (tokens in each location) are implemented as Java objects (not CSProcess objects, due to the limited number of threads available), with relevant behaviours invoked each simulation timestep. This means that the cells in a place or transition can all be processed during each time-step. This design gives the potential to support many thousands of cell objects per place, which is the biological requirement for the eventual cancer simulations.

The Place and Transition JCSP CSProcess classes provide access to both the read and write CSP channels (and barriers) as well as interaction with the GlobalClock and the Environment singleton classes. Specific places and transitions in the platform model extend the Places and Transition classes: for example, TACellPlace extends Place implements the domain model place TA in figure 9, and TACellToStemCellTrans extends Transition implements revTA in figure 9.

In the implementation, cells are passed between places and transitions via JSCP InfiniteBuffers, which are used to control synchronisation of channels connecting places and transitions: this means that there is one InfiniteBuffer for each arc in figure 9. InfiniteBuffers do not impose read or write locks on the threads accessing them (unlike other JSCP channels); this means that threads can write data into the buffer and proceed without waiting for another process to read the data. In the simulator, this is of particular importance when there are no cells arriving at a particular location in a particular time-step: the place can process any cells that are already there, rather than be suspended waiting for cells that are not coming.

To keep track of the number of timesteps that the simulation has completed, the GlobalClock is implemented as a JCSP CSProcess. The clocking allows state readings to be regularly output during simulation, and allows the setting of an end point for the simulation run. The GlobalClock synchronises on the same barrier as all the places and transitions.

The barrier design and implementation is critical in ensuring that all places and transitions experience the same sequence of timesteps and remain synchronised. Before any place or transition can accept cells from a buffer to which it is connected, all places and all transitions must be synchronised on a read barrier. Once synchronised all cells are removed from the buffers and are processed for that timestep. Once the cells are processed all places and all transitions that output cells (that is, all locations except DeadPlaces) must synchronise on a write barrier. Once synchronised, the locations move the relevant cells to the correct buffer, and then synchronise on the read barrier. This ensures that all cells are read before any are processed, and all cells are written correctly before any are read. The additional development burden of correct barrier design can be verified by formal CSP analysis if required.

All the places and transitions are modelled as singleton classes so that they can access each other as the program requires, supporting essential feedback mechanisms in the simulation. For example, in the prototype model, the probability of a TA cell reverting to a stem cell increases if there are no stem cells in the stem cells place, and stem cells remain quiescent longer as the number of TA cells rises; these rate changes act as a surrogate for spatial crowding in the environment (see [36] for further discussion).

7.3 Initial Results

The prototype simulator has been used to develop an approach to implementation from multiple linked models. The simulator is subject to testing, particularly of the logic of the design and performance. It has not yet been used to run biologically-relevant experiments.

The implemented genome was designed to calibrate the prototype simulator, to make it produce large numbers of TA cells and maintain relatively small numbers of stem cells, with low death rates. This will replicate the normal behaviour of the prostate in biological experimentation.

The prototype simulator has completed testing with total cell numbers ranging between 100 000 to 600 000. Typical performance is illustrated by initialising a simulation run with 2 stem cells; the number of cells quickly grows as stem cells divide and differentiate; the expansion then slows as the TA population gets large, due to the surrogate crowding effects; the result after 100 000 time-steps is 851 stem cells, 598 616 TA cells, 56 dead stem cells, and 28 153 dead TA cells. (These numbers are not being represented as biologically relevant, since the rates have not been calibrated, but they do give an indication of the achievable scale.) This simulation took 23 minutes on a 3.2GHz AMD Phenom II computer. Whilst this is not particularly fast, it is a significant improvement on the time taken to conduct the equivalent laboratory tests on prostate cell behaviour (weeks or months of expensive wet-lab procedures).

Simulation performance is likely to decrease as simulation complexity increases but it should be possible to simulate populations of cells well into the millions within realistic timeframes. Further work on the implementation architecture to capture more of the natural parallel structure of the model (for example, by increasing the implementation parallelism within individual places and transitions) would support distribution across clusters or cloud platforms. The full process-oriented design, with each individual cell implemented as a process, would be transferable to high-performance parallel languages such as occam-π [50] not limited by the number of available threads.

8 Further work

8.1 Further work on the prostate model

The prostate is a complex organ composed of many cell types. The model we outline in this work is an excellent foundation for an iterative systems biology programme to analyse prostate cancer neogenesis.

Parameterisation & Initial Conditions. As with any model, we have a large number of possible variables that need to be parameterised. Transition rates in the Petri net layer need to be defined as accurately as possible. Similarly, the initial conditions for the model need to be sensibly defined. These parameter values come from a variety of sources: literature searches; close collaboration with domain experts when choosing suitable *in silico* proxies; focussed wet-lab experimentation to generate the required data.

Cell Genome. Currently, each cell type has a minimal genome. As the model is refined, suitable variables can be placed upon the genome, allowing for mutable cell phenotypes. It is tempting to add as much complexity to the model as possible, thus initially adding all the genes that we can think of to the cell's genome; however, we should attempt to keep the model as simple as possible whilst retaining the ability to address real biological hypotheses. The addition of genes to the cell genome and the parameterisation of their rates is a major challenge for the next stage of this project.

New Cell types. Castration-resistant prostate cancer frequently shows a neuroendocrine like phenotype [47]. Currently, our model does not include stromal or neuroendocrine cells, with the interactions between these cell types and the modelled cells being abstracted to 'global' signals. If necessary, the inclusion of stromal and neuroendocrine cells would be possible, but this extra complexity would have to be justified by added ability of the model to address specific biological hypotheses.

8.2 Tool development

In this paper, the way the models in the two layers are combined is described informally. The approach could readily be made rigorous using model-weaving techniques from model driven engineering [2, 3, 21], allowing tool support (see also §5).

The (draft) Petri net ISO standard [17, 18] defines the notation in terms of a UML class diagram, or metamodel. This would make it possible to integrate, at a deeper semantic (or at least abstract syntax) level, the Petri net notation and UML state diagrams (as well as other UML modelling notations). Two motivations for a language integration would be semantic consistency, and integrated tool development. Support tools would obviously help the use of combinations of models. Semantic consistency is theoretically desirable; however, even if we unify the semantics of two diagram notations, there is no way of guaranteeing that the user intends their diagram to follow the semantics defined by the language developer.

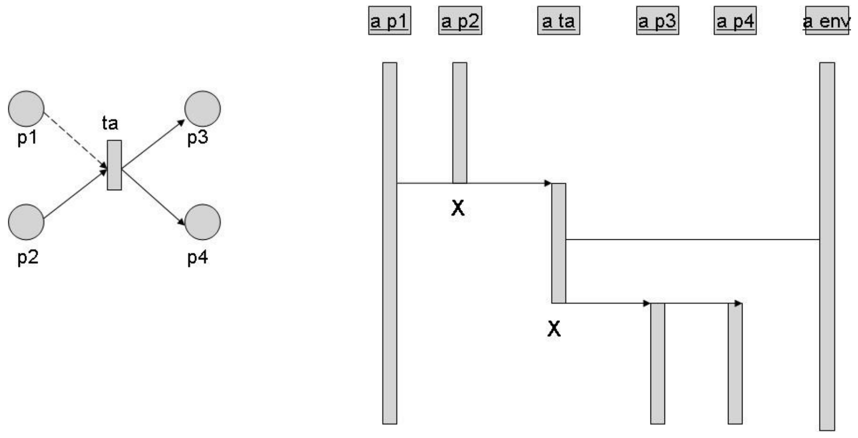

Fig. 19. An example catalytic arc. This is a small variant of figure 1, with the arc from `p1` to `ta` here a catalytic arc (left). The only change to the overall model is in the sequence chart (right), where the token `p1` is not consumed by the transition.

8.3 Extensions to this modelling approach

There are several extensions to this Petri net plus object modelling approach that are needed to make this more generally applicable to biological modelling. We outline a few here.

Petri nets can also be used to model biological processes of inhibition and catalysis, by suitably interpreting the relevant arcs. Inhibitory arcs need no change to the model described: they have the Petri net semantics of not permitting a transition to fire if there is a token in the relevant place. Catalytic arcs[3] need one change to the sequence diagram: the catalytic token object is not terminated (not consumed) by the transition. For example, see figure 19.

Larger models will need more structuring capabilities. We have used fusion places (figure 2) to split our Petri net into digestible chunks. Fusion transitions could be used to capture an entire sub-net, for example, to capture the identical logic of the various cell division stages. This requires further work to determine naming conventions to allow the individual models to be combined into a well-defined whole.

8.4 Incorporating other approaches

There is nothing specific to Petri nets and object models in our general approach; they are simply one way of capturing some high level global structure, and lower

[3] Catalysis is sometimes denoted by a double-headed solid arc in the Petri net, rather than the dashed arc of figure 19a. In our approach, this would imply that the token is consumed, and a new different token produced to replace it. The double-headed arc fits more readily into the 'token flow' interpretation (see §2.1), rather than the 'production and consumption' interpretation.

concept	Petri net	L-System
type of component	place	symbol
change of component	transition	production
instance of component	token	instance of symbol
current state	marking	current generation

Fig. 20. Relationship between high level Petri net models and L-System models

level component behaviours. This choice is a natural fit to a cell division model, but other biological processes may be better fit by other modelling techniques.

For example, L-Systems [37] have been designed to model plant-like growth process. Preliminary work suggests that they can be used in a hybrid model in an analogous way.

At the highest level, an L-System component (called a *module*) is just a symbol; and *productions* (rewrite rules) define how symbols 'grow' (are rewritten into sequences of symbols).

Descending levels one can add detail. Symbols can have parameters whose values are determined by the productions. As one adds more detail, full blown object/simulation models can be added to give the symbols and rewrites richer behaviours. This is the approach that the L-Studio tool [23] implements (by calling out to C functions).

We can use our approach to make this interaction of rewrite rules and object models fully rigorous: a high level L-System growth model can be combined with a lower level object-based module model. The correspondence between the Petri net approach and the L-System approach is shown in figure 20 (and hints at the possibility of a unifying metamodel). We can use the same approach to develop an object model that describes complicated behaviour in components and in component transitions. The 'glue' between the L-System and object model layers would need to be a little different from our Petri net/object model glue described above, however, as it would need to incorporate communication between the low level objects (between symbol instances), eg to implement hormone transport, during the non-growth part of the iteration.

9 Summary and Conclusions

We have presented a rigorous hybrid modelling approach combining the strengths of Petri nets and object-oriented models. These models make sense individually, at suitable levels of abstraction, and provide a systematic modular approach to modelling and to model validation, allowing us to use diverse modelling techniques in developing multi-scale models with no loss of rigour or validity. This hybrid modelling approach is not specific to Petri nets and object modelling; we outline a similar approach to combining L-Systems models with other modelling approaches, and discuss how the hybrid approach effectively amounts to the use of a domain specific language.

We have developed our hybrid modelling approach in order to model cell division and differentiation in the prostate. In building the prostate cell model, we have followed the CoSMoS process, with close cooperation between the biological domain experts and the modellers and simulation developers, to ensure that the model makes both biological and computational sense. In a companion paper [36] we have argued the validity of our model. We have developed a prototype simulation of the prostate cell model, and demonstrated that it is capable of simulating biologically-relevant numbers of cells.

In future work we will develop the hybrid modelling approach as a generic technique for building multi-scale models, along with tool support. We will also develop the prostate cell model and simulation further, and use it to investigate the onset of cancerous phenotypes.

Acknowledgements

This work is supported by Program Grant support (to N. J. Maitland) from Yorkshire Cancer Research, by TRANSIT (EPSRC grant EP/F032749/1) through the York Centre for Complex Systems Analysis, and by CoSMoS (EPSRC grant EP/E053505/1).

We thank the members of the Cancer Research Unit in York for their invaluable input to the project. Thanks to the anonymous referees for helpful suggestions to improve the paper.

References

1. Paul S. Andrews, Fiona A. C. Polack, Adam T. Sampson, Susan Stepney, and Jon Timmis. The CoSMoS process, version 0.1: A process for the modelling and simulation of complex systems. Technical Report YCS-2010-453, Department of Computer Science, University of York, March 2010.
2. Jean Bézivin, Nicolas Farcet, Jean-Marc Jézéquel, Benot Langlois, and Damien Pollet. Reflective model driven engineering. In *UML 2003 – The Unified Modeling Language*, volume 2863 of *LNCS*, pages 175–189. Springer, 2003.
3. Jean Bézivin, Frédéric Jouault, Peter Rosenthal, and Patrick Valduriez. Modeling in the large and modeling in the small. In *Model Driven Architecture*, volume 3599 of *LNCS*, pages 901–901. Springer, 2005.
4. L. C. Cantley, K. R. Auger, C. Carpenter, B. Duckworth, A. Graziani, R. Kapeller, and S. Soltoff. Oncogenes and signal transduction. *Cell*, 64:281–302, 1991.
5. Claudine Chaouiya. Petri net modelling of biological networks. *Briefings in Bioinformatics*, 8(4):210–219, 2007.
6. A. T. Collins and N. J. Maitland. Prostate cancer stem cells. *European Journal of Cancer*, 42:1213–1218, 2006.
7. Sol Efroni, David Harel, and Irun R. Cohen. Reactive animation: Realistic modeling of complex dynamic systems. *Computer*, 38:38–47, 2005.
8. Sol Efroni, David Harel, and Irun R. Cohen. A theory for complex systems: reactive animation. In Ray Paton and Laura A. McNamara, editors, *Multidisciplinary Approaches to Theory in Medicine*, volume 3 of *Studies in Multidisciplinarity*, pages 309 – 324. Elsevier, 2005.

9. Martin Fowler. *UML Distilled: brief guide to the standard object modeling language.* Addison-Wesley, 3rd edition, 2004.
10. Martin Fowler. *Domain Specific Languages.* Addison-Wesley, 2010.
11. 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. [44], pages 9–32.
12. Philip Garnett, Susan Stepney, and Ottoline Leyser. Towards an executable model of auxin transport canalisation. In Susan Stepney, Fiona Polack, and Peter Welch, editors, *Proceedings of the 2008 Workshop on Complex Systems Modelling and Simulation*, pages 63–91. Luniver Press, 2008.
13. P. P. Glory, N. G. David, and J. D. Emerald. Petri net models and non linear genetic diseases. In *Bio-Inspired Computing: Theories and Applications (BIC-TA), 2010*, pages 1466–1470. IEEE, 2010.
14. David Harel. Statecharts: a visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987.
15. David Harel, Sol Efroni, and Irun R. Cohen. Reactive animation. In *Formal Methods for Components and Objects 2002*, volume 2852 of *LNCS*, pages 136–153. Springer, 2002.
16. David Harel and Yaki Setty. Generic reactive animation: Realistic modeling of complex natural systems. In *Formal Methods in Systems Biology*, volume 5054 of *LNBI*, pages 1–16. Springer, 2008.
17. High-level Petri Nets - Concepts, Definitions and Graphical Notation. International Standard ISO/IEC 15909, 2000. Final Committee Draft: www.petrinets.info/docs/pnstd-4.7.4.pdf.
18. Software and Systems Engineering High-level Petri Nets Part 2: Transfer Format. International Standard ISO/IEC 15909, 2005. WD 15909-2:2005(E): www.petrinets.info/docs/ISO-IEC15909-2.WD.V0.9.0.pdf.
19. Kurt Jensen and Lars M. Kristensen. *Coloured Petri Nets.* Springer, 2009.
20. H. Kitano. Systems biology: a brief overview. *Science*, 295(5560):1662–1664, 2002.
21. Dimitrios Kolovos, Richard Paige, and Fiona Polack. Merging models with the Epsilon Merging Language (EML). In *Model Driven Engineering Languages and Systems*, volume 4199 of *LNCS*, pages 215–229. Springer, 2006.
22. Hillel Kugler, Antti Larjo, and David Harel. Biocharts: a visual formalism for complex biological systems. *Journal of The Royal Society Interface*, 7(48):1015–1024, 2010.
23. L-Studio. Plant modeling with CPFG virtual laboratory. algorithmicbotany.org/lstudio/flyer.pdf.
24. Y. Lazebnik. Can a biologist fix a radio? — or, what I learned while studying apoptosis. *Cancer Cell*, 2:179–182, 2002.
25. N. J. Maitland and A. T. Collins. A tumour stem cell hypothesis for the origins of prostate cancer. *BJU International*, 96(9):1219–1223, 2005.
26. N. J. Maitland and A. T. Collins. Prostate cancer stem cells: a new target for therapy. *Journal of Clinical Oncology*, 26(17):2862–2870, 2008.
27. Wayne Materi and David S Wishart. Computational systems biology in cancer: Modeling methods and applications. *Gene Regulation and Systems Biology*, 1:91–110, 2007.
28. Hiroshi Matsuno, Masao Nagasaki, and Satoru Miyano. Hybrid Petri net based modeling for biological pathway simulation. *Natural Computing*, pages 1–22, 2009. doi:10.1007/s11047-009-9164-6.
29. M. Mernik, J. Heering, and A.M. Sloane. When and how to develop domain-specific languages. *ACM Computing Surveys*, 37(4):316–344, 2005.

30. H. Motameni, A. Movaghar, M. Siasifar, M. Zandakbari, and H. Montazeri. Mapping state diagram to Petri net : An approach to use Markov theory for analyzing non-functional parameters. In *Advances and Innovations in Systems, Computing Sciences and Software Engineering*, pages 185–190. Springer, 2007.
31. M Nagasaki, A Doi, H Matsuno, and S Miyano. Petri net based description and modeling of biological pathways. In *Algebraic Biology 2005*, pages 19–31. Universal Academy Press, 2005.
32. OMG. Unified Modeling Language (UML). www.omg.org/spec/UML/.
33. Fiona A. C. Polack. Arguing validation of simulations in science. In Stepney et al. [44], pages 51–74.
34. 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.
35. 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.
36. Fiona A. C. Polack, Alastair Droop, Philip Garnett, Teodor Ghetiu, and Susan Stepney. Simulation validation: exploring the suitability of a simulation of cell division and differentiation in the prostate. In Susan Stepney, Peter Welch, Paul S. Andrews, and Carl G. Ritson, editors, *Proceedings of the 2011 Workshop on Complex Systems Modelling and Simulation, Paris, France, August 2011*. Luniver Press, 2011.
37. Przemyslaw Prusinkiewicz and Aristid Lindenmayer. *The Algorithmic Beauty of Plants*. Springer, 1990.
38. Mark Read, Paul S. Andrews, Jon Timmis, and Vipin Kumar. A domain model of experimental autoimmune encephalomyelitis. In Susan Stepney, Peter H. Welch, Paul S. Andrews, and Jon Timmis, editors, *Proceedings of the 2009 Workshop on Complex Systems Modelling and Simulation, York, UK, August 2009*, pages 9–44. Luniver Press, 2009.
39. Mark Read, Paul S. Andrews, Jon Timmis, and Vipin Kumar. Using UML to model EAE and its regulatory network. In *8th International Conference on Artificial Immune Systems (ICARIS)*, volume 5666 of *LNCS*, pages 4–6. Springer, 2009.
40. Magali Roux-Rouquié, Nicolas Caritey, Laurent Gaubert, and Camile Rosenthal-Sabroux. Using the Unified Modelling Language (UML) to guide the systemic description of biological processes and systems. *BioSystems*, 75:3–14, 2005.
41. Derek Ruths, Melissa Muller, Jen-Te Tseng, Luay Nakhleh, and Prahlad T. Ram. The signaling Petri net-based simulator: A non-parametric strategy for characterizing the dynamics of cell-specific signaling networks. *PLoS Comput Biol*, 4(2):e1000005, 2008.
42. Yaki Setty, Irun R. Cohen, Avi E. Mayo, and David Harel. On using divide and conquer in modeling natural systems. In Anne Condon et al., editors, *Algorithmic Bioprocesses*, pages 661–674. Springer, 2009.
43. Yong-Jun Shin and Mehrdad Nourani. Statecharts for gene network modeling. *PLoS ONE*, 5(2):e9376, 2010.
44. Susan Stepney, Peter Welch, Paul S. Andrews, and Adam T. Sampson, editors. *Proceedings of the 2010 Workshop on Complex Systems Modelling and Simulation, Odense, Denmark, August 2010*. Luniver Press, 2010.
45. Ivana Tričković. Formalizing activity diagram of UML by Petri nets. *Novi Sad J. Math.*, 30(3):161–171, 2000.
46. Cancer Research UK. Prostate cancer — UK incidence statistics. info.cancerresearchuk.org/cancerstats/types/prostate/incidence/. accessed 18/01/2011.

47. N. Vashchenko and P. A. Abrahamsson. Neuroendocrine differentiation in prostate cancer: implications for new treatment modalities. *European Urology*, 47:147–155, 2005.
48. Ken Webb and Tony White. Combining analysis and synthesis in a model of a biological cell. In *Proceedings of the 2004 ACM symposium on Applied computing*, SAC '04, pages 185–190. ACM, 2004.
49. Ken Webb and Tony White. UML as a cell and biochemistry modeling language. *BioSystems*, 80:283–302, 2005.
50. Peter H. Welch and Frederick R. M. Barnes. Communicating mobile processes: introducing occam-pi. In *25 Years of CSP*, pages 175–210. Springer, 2005.