Towards an Executable Model of Auxin Transport Canalisation

Philip Garnett[1], Susan Stepney[2], and Ottoline Leyser[1]

[1] Area 11, Department of Biology, University of York, YO10 5YW, UK
prg500@york.ac.uk
[2] Department of Computer Science, University of York, YO10 5DD, UK

Abstract. We describe our use of a modelling and development process to specify and implement biological simulations that involves the development of several different UML models to capture different perspectives on the system being modelled, in particular the investigation of various emergent properties. We use this process in the case of an auxin canalisation simulation, investigating the processes of auxin transport as guided by PIN proteins, in a developing plant. We discuss our preliminary results of investigating one hypothesis of PIN protein placement that fails to demonstrate canalisation in simulation.

1 Introduction

The simulation of biological systems presents a significant challenge requiring knowledge from all branches of science to capture all the relevant aspects of the biological, chemical and physical processes occurring. The challenge is made more difficult by the complex nature of biology: it is hard to make good assumptions about how a particular process is regulated. The quality of information available is important: the best solution based on the information at hand may not be the real solution, but more a reflection of how insufficient the current data are. Or the data may be excellent but missing a part of the picture altogether. Also the connectivity of processes in biology is often very high, therefore the question of the level of abstraction and simulation complexity is important. If the abstraction level is too high we risk ruling out simulations producing emergent behaviour; too low, and the simulations produced could be difficult to work with. So the decisions made when producing a simulation are important, as a balance must be sought between complexity and the all important emergent behaviours.

Here we present how we are using the CoSMoS lifecycle [3] to develop abstract models and executable simulations of a biological system, in order to test different hypotheses of how certain biological processes may work.

We are using this approach to investigate auxin canalisation in the plant *Arabidopsis*. Auxin is a plant growth hormone. The process of auxin canalisation occurs between auxin production sites and auxin sinks; in the plant stem, it results in the development of vascular tissue between new sites of auxin production and existing vascular tissue. We are interested in understanding how this complex self-organising process is regulated in the cells of the plant. Due to the complexity of canalisation, and some gaps in the biological knowledge of how and what is causing the canals to form, we are developing executable computer simulations in order to test multiple hypotheses. We are using UML (Unified Modelling Language) [25] to model the biology as we understand it, the simulated biology, and the details of the implementation. The hope is that the hypotheses tested by the simulations might indicate experiments that can then be tried in the lab to further our understanding, and to drive new simulations.

We find that the combination of UML and object-oriented programming maps naturally to the kind of biological processes that we are modelling. Biological entities, such as proteins and cells, map directly to objects in the UML models, which are then implemented as objects in the program code. The interactions between these biological entities similarly map directly to associations between objects in the UML models, which are then implemented as communications between objects in the program code. This allows us to build models containing the biological entities that we believe to be involved in canalisation, and then produce simulations that we can use test various hypotheses about the biological processes of interest. If an hypothesis is correct we should see emergent behaviour that is consistent with the real biological behaviour when the simulation is run; if not we can then return to the UML models and implement our next hypothesis. This provides a process to assist us in determining if our simulated biology is consistent with the real biology. Additionally, the UML diagrams are relatively accessible to biologists, allowing them to provide input to the model of the simulation without the need to understand the code.

This is a report of a work in progress on the modelling and simulation of a biological system. In section 2 we discuss the use of UML as a suitable modelling language. In section 3 we overview the process we are using for modelling, designing, and implementing biological system simulations. In sections 4, 5, and 6 we present our initial Domain,

Software, and Refined Software Models of auxin canalisation, and in section 7 we discuss some of the issues of building the resulting simulator. In section 8 we present some preliminary results on the auxin canalisation hypotheses, and conclude with a discussion of our experiences in section 9.

2 Modelling biology and simulations with UML

UML [25] is a modelling language comprising a suite of diagramming notations. It was originally developed to provide a standard concrete syntax for software modelling.

There are a large number of ways of developing software with UML. For example, the process can start by looking at high level interactions in the system being described [25]. This includes the ways that various external actors interact with, or use, the system (where these external actors include users and other systems). This is normally modelled in a Use Case Diagram supported by textual usage scenarios. The high level model is gradually refined by adding detail. Further diagrams are used to drill down details of how the system is built, what the objects are, what information they exchange and when those exchanges of information need to take place. This eventually results in a code skeleton being created for the objects with the attributes in place and the methods waiting to be implemented.

The process of developing biological models with UML is similar to the processes used for development using the Systems Biology Markup Language (SBML) [15, 10, 14]. However, as we are implementing our programs in object-oriented (OO) programming language Java, the ability to produce code skeletons from UML easily and flexibly with tools such as Rational Rose[16] is an advantage. UML is considered to be platform independent, as the diagrams can be transformed into a wide variety of different outputs.

As well as classic object-oriented technologies, UML is well suited to agent-based modelling [24] (where an agent can be thought of as an object with its own thread of control, allowing highly parallel systems of multiple agents). Biological "agents", such as cells and proteins, can be modeled as UML agents. There are many processes in cells acting in parallel. Some of these processes are individually sequential, such as expression of proteins in response signals detected in the cell. The signals might cause a number of events such as protein expression, which then in turn causes more events to occur. This sequential behaviour is often called a cell pathway. The parallel behaviour comes from this type of process occurring in a number of different pathways in one cell of the

plant at the same time, and in many cells at the same time throughout the plant.

We are not the first to apply UML and related modelling notations to the modelling of biology. There are a number of published cases where these have been successfully used to produce biological models [8, 17, 35].

We have taken a simple iterative approach, where UML models are developed, turned into the simulation code and tested. Then any necessary alterations are fed back into the beginning of the process, to ensure the models and code are consistent. We have not found it necessary to use all of the multifarious diagrams available in UML; we describe only the ones that we have found to be of greatest use.

3 Overview of the modelling lifecycle

In [8] we identify a conceptual framework for developing bio-inspired algorithms that takes a principled approach of building models of the biological system of interest, developing abstract computational models from this, then instantiating these computational models to produce bio-inspired algorithms. This framework can be adapted to producing *simulations* of complex (biological) systems, by implementing the abstract computational models to produce simulators.

Fowler [12, p.5] identifies two perspectives that can be used when building models: the *conceptual* perspective, representing "a description of the concepts of a domain of study", and the *software* perspective, where "the elements of the UML map pretty directly to the elements in a software system".

The CoSMoS (Complex Systems Modelling and Simulation) project[3] is developing a complex systems simulation development infrastructure (preliminary work is reported in [2, 28]). The CoSMoS lifecycle [3] has grown from the conceptual model and Fowler's perspectives, and adds the concept of an Analysis Model. It identifies the following components (summarised in figure 1):

Domain Model: a "top down" conceptual model of the real world system to be simulated, derived from the domain experts, from the literature, and (possibly) from further observations and experiments needed to provide sufficient data for modelling. Some modelling decisions about what to put in and what to leave out are made here. The model may explicitly include various emergent properties, since from a top down perspective it may not be obvious that these are emergent; or, if we are

[3] http://www.cosmos-research.org/, EPSRC grants EP/E053505/1, EP/E049419/1

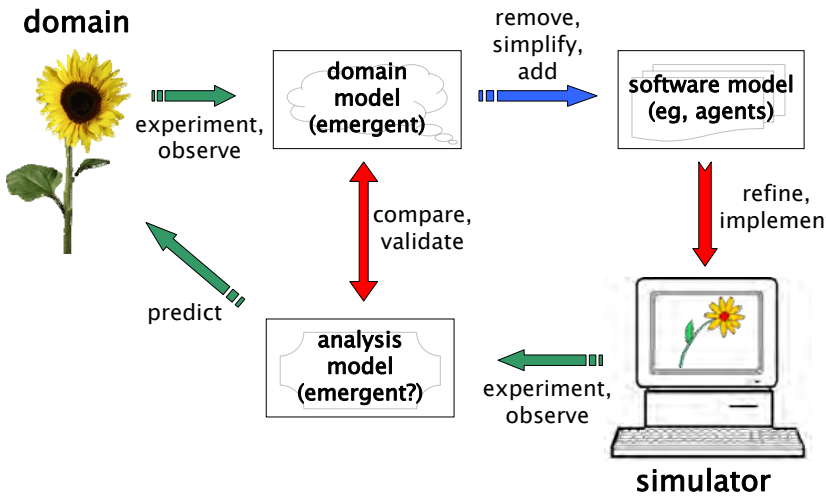

Fig. 1. The components of the CoSMoS basic lifecycle (after [3]).

aware of the emergent properties, it may not be obvious what low level processes produce them.

Software Model: a "bottom up" model of how the real world system is cast as a simulation. This includes: a definition of the system boundary (what parts of the Domain Model are being simulated); simplifying assumptions and abstractions; removal of emergent properties and replacement with the local interactions that are hypothesised to result in them; extra simulation-only concepts, such as "physics" engines to implement real world processes in possibly unnatural ways, user interfaces to view and control the simulator, and "probes" to produce output data for analysis.

Simulator: the executable implementation. The development of the Simulator from the Software Model is a standard software engineering process.

Analysis Model: a "top down" conceptual model of the *simulated* world, derived from observations and experiments on the simulation. The model may explicitly include various observed emergent properties. This model is compared to the Domain Model in order to test various hypotheses, such as the validity of the simplifications used to derive the Software Model.

In a large simulation development this basic lifecycle can be augmented with extra steps [3], such as the development of a **Refined Software Model**, describing the detailed simulator design and platform-

specific implementation details, which is a a refinement of the Software Model used as the basis for producing code.

The CoSMoS lifecycle is neutral in its choice of modelling language(s). For example, it could use a mix of text, biological "cartoons", Soft Systems' Rich Pictures [7], and mathematical equations to describe the Domain Model, and any standard software engineering technique to define the Software Model. Here we use mostly UML supported with text.

This process allows us to separate implementation details from the biology being simulated. This offers a number of advantages. It makes the individual models and accompanying diagrams simpler, as they are focussed on specific perspectives. As we are partly using UML as a communication tool it is advantageous for the diagrams to be as simple as possible. Different groups are more interested in certain perspectives: for example, biologists are probably more interested in a clear representation of the biology, rather than how the data I/O and GUI work.

4 Domain Model: auxin transport

4.1 Modelling

The process of auxin transport canalisation is complex, and is not fully understood. This makes it a natural target for modelling, but also a challenge. Our eventual goal is to produce models and simulations of shoot branching, but in order to do this we first have to understand the mechanism of auxin canalisation.

We have therefore started producing an executable model of the canalisation process. We are using the CoSMoS approach, in UML, starting from our background biology derived from the literature, and from wet lab experiments by Leyser and her group (summarised below). We use that to develop a UML Domain Model that includes all the necessary biology for the model to function, but keeps the model as simple as possible. We then develop a UML Software Model, and refine it to develop the simulator program itself, via the production of code skeletons. This helps us to ensure that the reasons for how and what is in the various models is carefully considered, and we are able to make comparisons with how the model simulates biology compared with how we think the biology works. Finally, we analyse the outputs of the simulator.

We produce abstract UML models of cells assisted by UML-based software development tool Rational Rose [16], and implement the simulator in Java.

4.2 Overview of biological domain

We summarise the background biology here to provide context; more detailed information than given here informs out full model.

There are many mathematical models of nearly all aspects of plant development [29], and many concerned with auxin transport [19]. We are interested in producing executable models as we believe that this type of modelling lends itself well to biological systems and might offer an alternative perspective that yields results [11].

We are developing models of the formation of auxin transport canals, particularly the transport canals that form in the stem of plants and often go on to differentiate into vascular tissue. This process is known as auxin transport canalisation and its regulation is not clearly understood.

We are looking at the regulation of shoot branching, where lateral axillary buds on the main stem of a plant activate and start to grow. Our eventual aim is to model the canals that form from a newly activated bud and progress into the existing main vascular tissue of the stem. Experiments indicate a correlation between the activation and growth of a bud and its ability to transport auxin out into the stem vascular tissue [5]. Therefore, the regulation of canalisation in the bud is linked to the regulation of shoot branching.

Auxin transport canals form between sites of auxin production and auxin removal: sources and sinks. In order for a canal to form between the bud (the source) and the stem vasculature (the sink), the stem vasculature must behave as a relatively strong sink compared to the surrounding tissue. If the stem vasculature is already transporting large amounts of auxin, it has no further capacity for auxin transport, so its sink strength is relatively reduced, and the canal does not form. Therefore the bud does not activate, and does not grow into a branch. However, if there is spare capacity for auxin transport in the vasculature, then it is a relatively stronger sink. This allows auxin canalisation to occur, so auxin can be transported out of the bud, allowing the bud to develop into a new branch [5, 26].

The formation of the canals between the sources and sinks is due to aspects of the biology of auxin transport (figure 2). Auxin is a weak acid and is able to enter cells passively from the more acidic apoplast (intercellular space) by crossing the cell membrane. However once in the pH-neutral cytoplasm of a cell it becomes deprotonated and therefore is not able to recross the cell membrane passively. The auxin is able to leave the cell only with the assistance of transport proteins. One particular family of auxin transport proteins, PIN proteins, are polarly localised in cells, often to only one face of the cell [23].

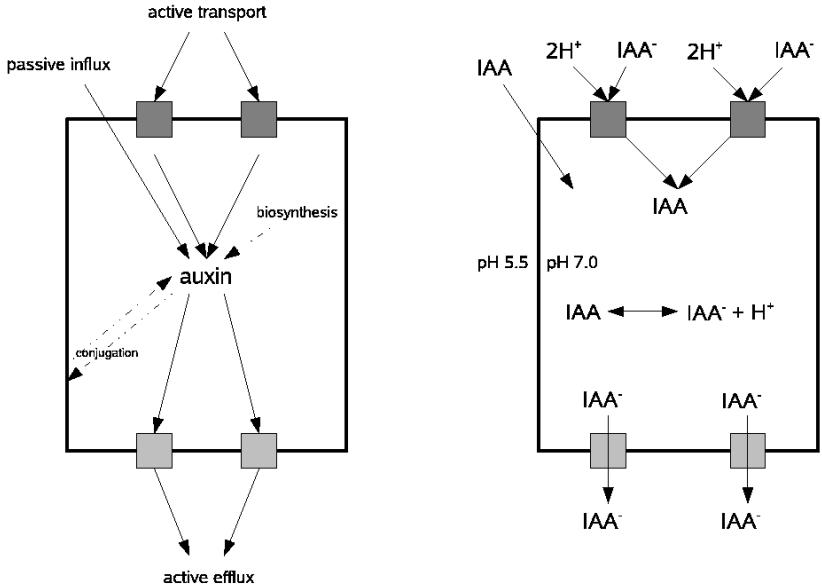

Fig. 2. A "cartoon" (informal picture) of the biology of auxin transport into cells. Auxin (IAA) can enter cells passively, or it can be actively transported by AUX/LAX proteins. Once in the cell its only method of escape is active transport by proteins like PIN [23].

In their capacity as auxin exporters, PIN proteins are important to the process of canalisation. The exact details of how PIN localisation is regulated are unknown, but increasing amounts of data and some prior modelling work are providing possible mechanisms. One hypothesis, by Sachs, suggests that auxin facilitates its own flow: both the ability of a cell to transport auxin and the polarity of the auxin flow increase with the amount of auxin being transported [31]. Therefore as the transport capacity increases the cells in the canal become better sinks and draw in more auxin. This process has been modelled by Mitchson and he showed it to work [21, 22], but the model has a few problems. Firstly, it predicts canals of high flow and low concentration, whereas experimental evidence suggests that there is both high flux and *high* concentration [34, 4]. Secondly, both Sachs' and Mitchson's models require the cells to be able to measure the flux of auxin; as yet a mechanism to do this has not been identified in the plant, although that does not mean that one does not exist.

More recent data on some aspects of the processes provides more details of what should be in the models. Auxin is up-regulating its own transport by increasing the amount of PIN protein available to transport auxin [27]. If the negative regulators of PIN accumulation, such as the MAX pathway, are removed, transport increases and the vasculature is able to transport more auxin [5]. Thus the more auxin in a cell, the more it can transport. In *Arabidopsis* the mutation of genes in the MAX pathway results in a phenotype of increased branching [33], which supports the theory that bud activation, and therefore branch number, is linked to the stem auxin sink strength.

PIN proteins are important to the canalisation process as they export auxin out of the cells, and their polar localisation patterns are responsible for complex transport patterns in a number of plant tissues [30, 9, 13]. Improved knowledge of PIN has also enabled development of simulations of auxin transport systems with high concentration and high flux [13, 18]. However, what directs the PIN in the cells into the polar patterns that are seen remains an important problem: if PIN is positioned by detection of auxin flux, as Sachs suggests [31], then what is the mechanism in cells that is detecting auxin flux?

4.3 Domain Model Use Cases

We start building our UML Domain Model from the background biological material. One approach to building UML models is to start with use cases, that capture the user requirements or how they want to use the system. That is not appropriate here, as we are not modelling what is wanted, but what is. So instead, in the domain model, we take the view that use cases capture a high level view of what the system "does", such as regulate proteins and transport auxin (figure 3).

This approach maps well to the ways biologists describe their domains. We have not found a need to develop this model further, by developing the more detailed usage scenarios necessary in a traditional system requirements elicitation process. Such scenarios might be found necessary for capturing more detailed behaviours ("so, *how* does it regulate proteins?"), and would presumably need to be modified from their traditional format.

4.4 Domain Model class diagram

We model the biological entities of interest as objects and classes. This approach works well here, because much of cellular biology can be thought of as interactions of discrete objects that result in complex behaviours.

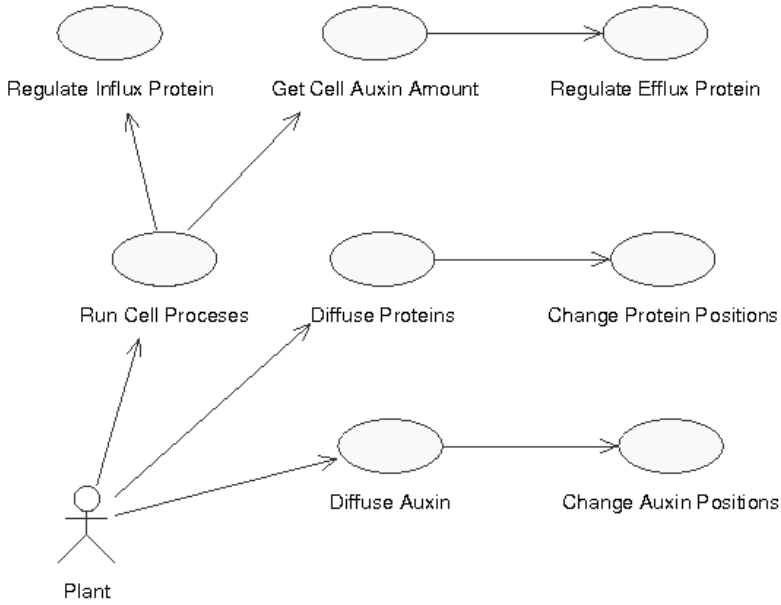

Fig. 3. The Domain Model Use Case diagram. This captures what the Plant does.

However, it should be noted that the class diagram, although maybe the most natural starting point for a object-oriented software developer, is not the most natural way for biologists think about and describe their domain, particularly in identifying associations between classes.

We consider the different parts of the cells, such as the cell membrane and vacuoles (cellular compartments), and the proteins like PIN and hormones like auxin, all as objects. One of the objects of interest at this level of model is the auxin canal. This is an emergent property of the lower level interactions. We model it explicitly here to capture its biological properties so that later simulation outputs can be related to it.

Figure 4 shows our Domain Model class diagram of the biologically relevant parts of the system (deciding what *is* biologically relevant is also part of the modelling process). In detail, it shows the following classes (type of objects) and relationships between the objects[4]:

[4] This detailed description is provided here to help biologists to interpret the diagram; there is nothing in the explanatory text not included in the diagram.

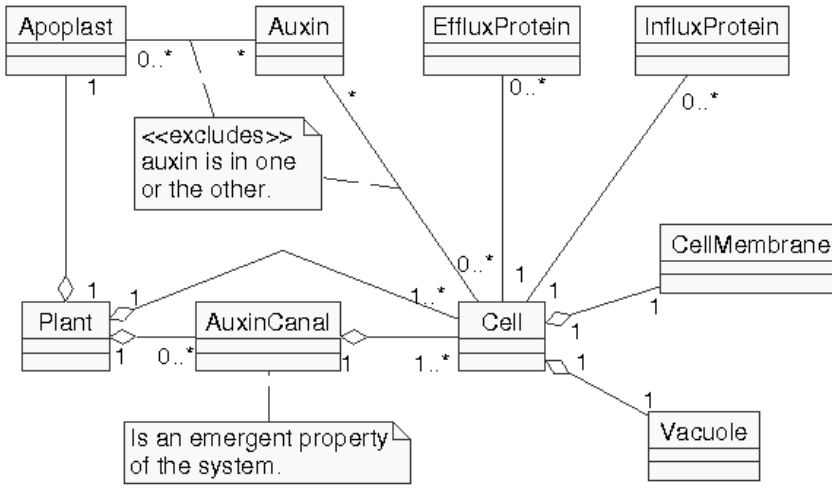

Fig. 4. The Domain Model class diagram.

- A Plant has one Apoplast (the space between cells), one or more Cells, and an optional Auxin Canal. (It also has other components, but these are not being modelled, even at the system level.)
- An Apoplast is part of one Plant, and has zero or more Auxin molecules.
- An Auxin molecule is in the Apoplast or in a Cell. (The relationship lines say that it may be in an Apoplast and it may be in a Cell; the excludes condition says that it is one or the other).
- An Auxin Canal is part of one Plant, and has one or more Cells.
- A Cell may be part of an Auxin Canal; it is part of a *Plant*. It has zero or more Auxin molecules, zero or more Efflux Proteins, and zero or more Influx Proteins. It has one Membrane and one Vacuole.
- An Efflux Protein is in one Cell; an Influx Protein is in one Cell.
- A Membrane is part of a Cell; a Vacuole is part of a Cell.

We impose an extra condition on the loop of relationships[5] containing the Plant, Auxin Canal, and the Cells: Consider a Cell that is part of an Auxin Canal that is part of a Plant, that Cell is also directly part of *the same* Plant. (There is no loop of relationships containing the Plant, Auxin Canal, Cells, Auxin, and Apoplast: the apparent loop is broken by the excludes condition.

[5] By "loop" we are referring to the two possible paths from Cell to Plant, one via the Auxin Canal, and the other directly.

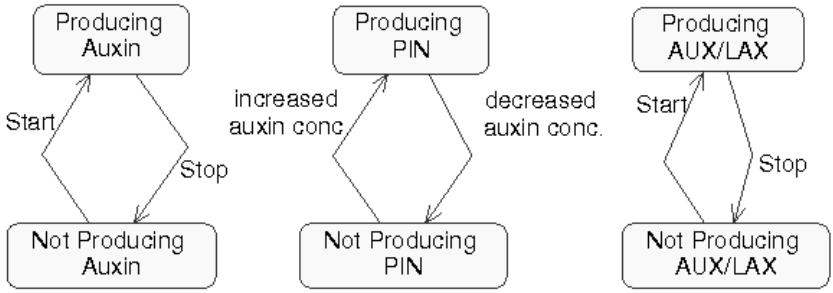

Fig. 5. State diagram for the Cell. (Since we are not explicitly modelling cell birth or death, no start or end states are needed.)

Note that there is no use of inheritance in figure 4, despite the fact that Efflux Protein and Influx Protein are both kinds of protein, for example. We find that we make relatively little use of inheritance in our Domain Models, because it is not necessary in order to understand and capture the biological domain, which is more interested in detailed differences than abstracted similarities. The Software model, on the other hand, exploits inheritance to provide abstraction and code reuse.

4.5 Domain Model state diagrams

The use case diagram and class diagram help with the identification and organisation of the different objects of the model, but provide little information about how those objects behave and how they interact.

Interactions are often captured in UML using sequence diagrams, and these show the passage of information between objects over time. In biology the order and direction of interactions is less clearly defined: the next step in the interaction sequence might not occur; the process might back up to the previous step in the sequence. For example, there is no guarantee of progression down a biochemical cell pathway. This makes capturing timing of events difficult with sequence diagrams.

UML has another way to capture how objects change over time: state diagrams. These diagrams show the different states an object can be in, and how the object moves from one state to another in response to an event. Such state diagrams also appeal to biologists, as they appear to map closely to the way that biological processes are understood.

State diagrams for the state changes associated with a Cell are shown in figure 5, for Auxin hormone in figure 6, and for PIN proteins (a kind of EffluxProtein) in figure 7. The states of these objects are linked, and a change in the state of one object is linked to that of the others. The state

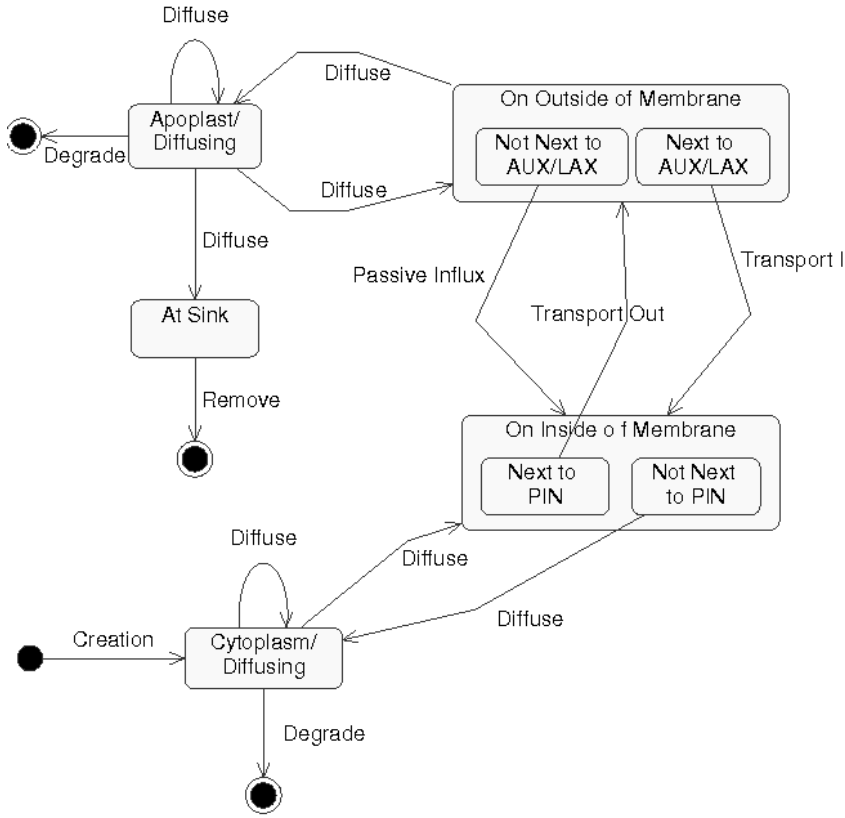

Fig. 6. State diagram for Auxin. Defining and expressing this type of complicated behaviour is where state diagrams can prove useful. The figure shows the different states auxin can progres through, and the events that can trigger those state changess.

of a cell is defined by the what is happing in that cell, which is determined by what the proteins and hormones are doing. At the moment, we are performing this linking by textual annotations.

Figure 5 shows that a Cell can be producing Auxin, PIN protein (an EffluxProtein), and AUX/LAX protein (an InfluxProtein). Not all cells are capable of making auxin, but they are all capable of producing PIN, which they do in response to the amount of auxin they have, so PIN production might be on even if there is no auxin production.

Figure 6 shows the state diagram for Auxin. It can be in four main states:

- in the cytoplasm (the inside part of the cell that is not vacuole). It is created here, and may degrade (be destroyed) here. It is in its deprotonated form. It is diffusing around, which either leaves it in the cytoplasm, or moves it to be:
- on the inside of the cell membrane, where it is in one of two substates, next to PIN, or not next to PIN. If it is not next to PIN, it diffuses back in the cytoplasm. If it is next to PIN, is transported out of the cell to be:
- on the outside of the cell membrane, where it is in one of two substates, next to AUX/LAX, or not next to AUX/LAX. If it is next to AUX/LAX, is transported into the cell to be on the inside of the cell membrane. If it is not next to AUX/LAX, is can passively influx into the cell to be on the inside of the cell membrane, or it can start diffusing to be:
- in the apoplast. It is in its protonated form. It is diffusing around, which either leaves it in the apoplast, or moves it to be on the outside of the cell membrane, or moves it to be at auxin sink, where it is removed from the system. Or it may degrade (be destroyed) here.

In reality, an auxin in the cytoplasm may be no different from an auxin adjacent to a cell membrane: it has no "sense" of where it is. Therefore the auxin may not have a different *biological* state when it is in these different situations. But we can *model* the biology in terms of such states.

State diagrams can be used to model alternative hypothesised behaviours. Simulations based on these different hypotheses can be compared. For example, figure 7 shows a state diagram for one hypothesis of PIN protein (an EffluxProtein) behaviour, and figure 8 shows the state diagram for a slightly different hypothesis for its behaviour. In the latter case, the PIN protein is allowed to move around on the cell membrane if it is not transporting auxin. Therefore the moving state is different from the transporting state when on the membrane of the cell. (Proteins are able to move around on cell membranes [32] and it is theoretically possible that a conformational change in response to actively transporting auxin might stop it from moving.)

When we are considering proteins, the different states in the Domain Model of the biology correspond more closely to real biological states than in the case of auxin. Proteins are active molecules, and can undergo conformational and other changes in response to events. Auxin however is a very simple molecule, and more of its behaviour is a passive response to its environment.

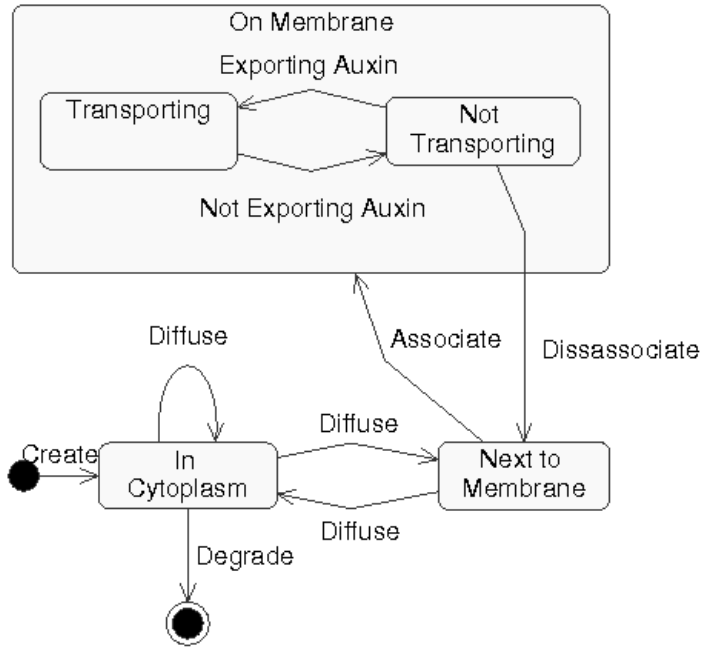

Fig. 7. State diagram for the PIN Efflux Protein. The PIN is associated with the membrane. It can either be actively transporting auxin or sitting idle. If it is not transporting auxin there is the possibility that it might disassociate and return to the cytoplasm. If it is transporting then it remains attached to the membrane.

5 Software Model

5.1 Overview of additional functionality

The Software Model includes the things that our simulation must do to be able to run. These include the setup procedures required to get the simulation to a starting point, including things like instantiating cells, and detecting the internal environment in the cell in order to produce proteins and hormones. This functionality can be considered from three points of view: that of the biological plant (captured in the Domain Model); the simulation of the biology that is required to be there but is not simulated in a particularly biological way; and the things that need to be there to produce a successful simulation but are not part of the biological Domain Model.

As our UML and simulations have developed, the simulated biology has come to represent the real biology, as currently understood, more

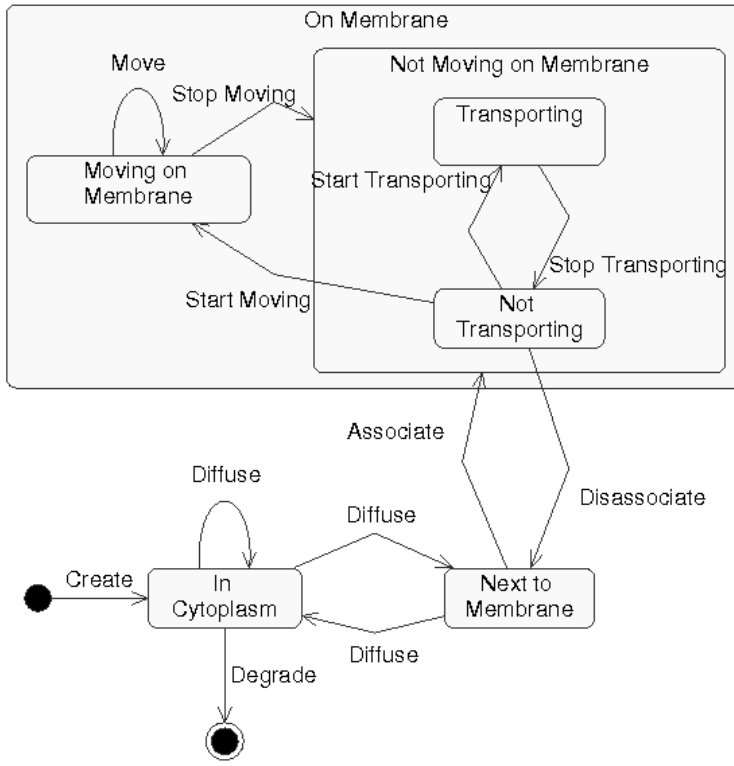

Fig. 8. State diagram of an alternative hypothesis for the PIN Efflux Protein, with different behaviour on the cell membrane: movement on the membrane is allowed.

closely. However, there are still a few areas where this has not been possible, or desirable, to achieve. For example, when a cell is created in the simulation it is necessary to create the cell and then create its membranes, a vacuole and a starting amount of proteins. In reality membranes partly define a cell: a cell cannot exist without a membrane. Therefore our simulation is not doing cell creation, or growth, in a particularly biologically realistic way. It would be more realistic for a cell be the outcome of a particular arrangement of cell membrane, vacuole and other cell elements. The increased flexibility of such an approach could in the future allow for simulation of growth, the lack of which is currently a limitation.

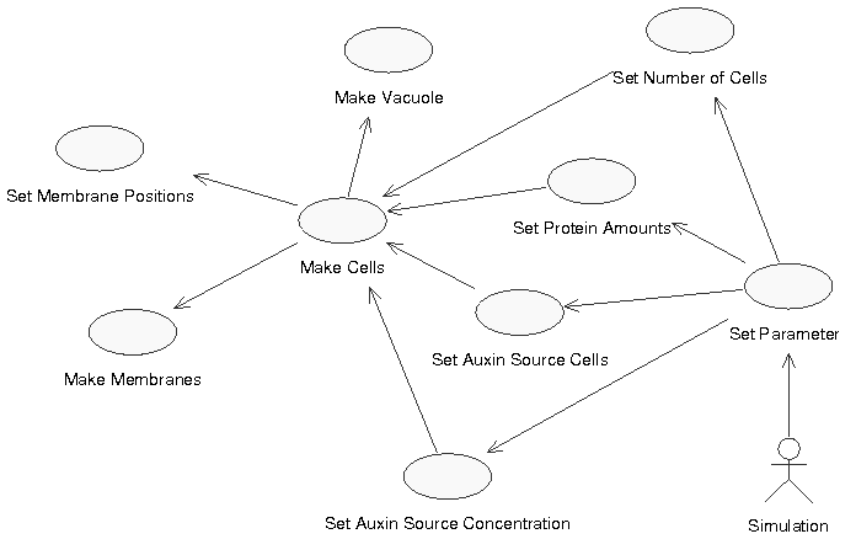

Fig. 9. The Software Model Use Case diagram, with the Simulation user actor.

The Software Model also includes things like the graphical user interface (GUI), and what I/O the simulation needs to do to provide the user or external systems with results.

5.2 Software Model Use Cases

A use case diagram is used to capture the user requirements for using the simulator, in the traditional way (figure 9).

5.3 Software Model Class diagram

The Software Model class diagram is produced from the Domain Model class diagram (figure 4) and the Software Model use cases. Figure 10 shows only the biologically relevant parts of the Software Model class diagram (to improve readability, it omits things like the data and visual output objects).

Certain classes are removed: we decide at this stage not to model the apoplast explicitly in the simulation. It appears as the gap between the cells in the visualisation (later). We also remove the explicit Auxin Canal: this is the emergent property that we desire the simulation to exhibit.

Certain classes are added: we include some inheritance. Proteins and Hormones share some common features, and we model them as subclasses

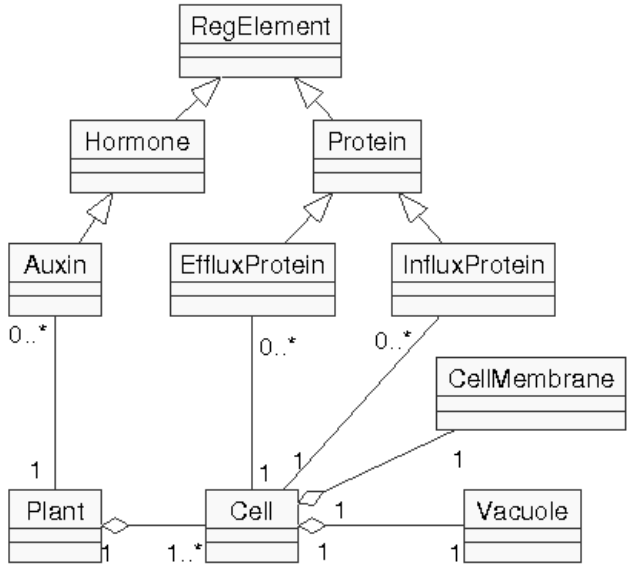

Fig. 10. The Software Model class diagram, showing only the biological part of the model.

of the RegElement (regulatory element) class. Classes that support user interaction are also added (not shown here).

Certain relationships are removed: Auxin is no longer related to Cells. This highlights a difference between the Software Model and the biology. In the Software Model, the Auxin is related only to the Plant, as are the Cells. For ease of implementation with regard to the diffusion of auxin inside and outside cells, the model records the position of the auxin in the simulation space (part of the Plant, and the Cells can query the Plant to discover how much of that auxin is internal to them. This is a suitable implementation strategy, even though it is not a good model of reality; it shows how the "same" objects in the Software Model can be quite different in structure from those in the Domain Model.

A future revision of the simulation will do this differently, by explicitly modelling the apoplast (the space between cells that the auxin is in when not in a cell).

5.4 Software Model State diagrams

The Software Model state diagrams follow the Domain Model, except that the production of AUX/LAX is left out, and expressed at a fixed amount and not (currently) regulated.

6 Refined Software Model

6.1 Refined Software Model class diagram

The Refined Software Model class diagram (figure 11) adds implementation detail to the Software Model class diagram (figure 10). It has all the methods and attributes of the objects (not shown here) and it is use to generate the code skeleton.

The Refined Software Model class diagram includes further details of how some of the biological processes are implemented. For example, the positions of components are held by Position objects and the singleton Hashmap object. It may be advantageous to split things in even more detail if the diagrams become overly complicated, as certain parts of the implementation are more important than others. Things like the implementation of diffusion and how positions of cells and hormones are stored are of greater interest than how the visual output is achieved.

This separation will be even more worthwhile when things like growth are implemented, as they are likely to be complicated and require detailed diagrams. Also, the implementation of such things is much more difficult than conceptualising them, and therefore should be open to more detailed scrutiny to ensure that it is done in a valid way.

The full class model has all the methods and attributes of the classes added, and it is this that is used to generate the code skeleton. Once the biology has been produced in both the class diagram and the state diagrams this is often enough to produce a code skeleton from the UML for the model. It might be necessary to define more clearly the interactions between the objects using sequence and collaboration diagrams if the model is large and complicated.

6.2 Refined Software Model state diagrams

The Refined Software Model links the Software Model state diagrams of different objects, particularly the overlapping parts of the auxin, PIN and cell state diagrams. This shows how a cell producing auxin influences its own state as it responds to the change in auxin concentration by making more PIN protein, and how a cell that does not make auxin, but which detects that there is auxin in its cytoplasm, responds by starting to increase production of PIN.

State diagrams are linked by shared events. The Auxin event of entering a cell, either passively or via a protein, is linked to the Cell event of detecting a change in auxin concentration, which causes the cell to enter into a PIN production state. At the moment, we are performing this linking by textual annotations.

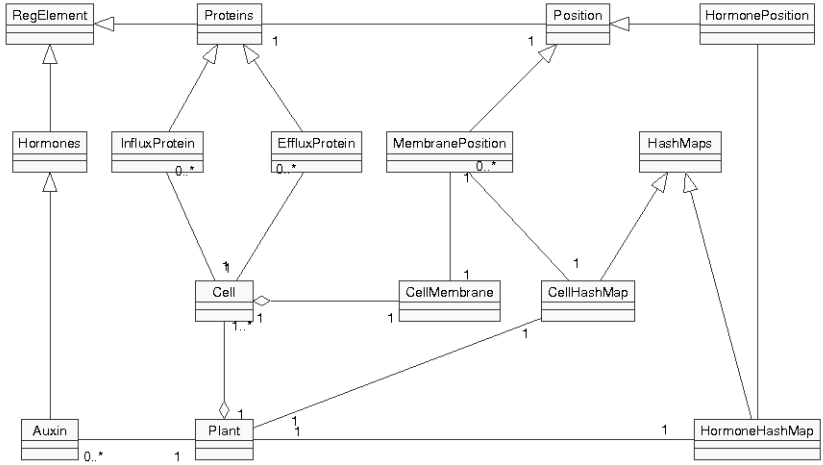

Fig. 11. A simplified implementation class diagram showing more detail of the underlying implementation of some of the biology in the model. It gives more detail on how the positions of different objects are controlled in the model. The full implementation class diagram shows the classes controlling diffusion and how the threading of the program is controlled. Some classes have been omitted for clarity.

This linking of states allows the interactions of the biological objects to be modelled at a higher level than sequence or collaboration diagrams, which are more useful for giving details of how the simulations are going to run. Linking could also be useful to include a notion of space in UML diagrams like state diagrams; for example, states of different objects may influence each other in different ways inside or outside a cell.

7 Simulator: Auxin transporter

We have taken the code skeletons produced from the Refined Software Model UML, and added the remaining code necessary to implement the executing simulator. Some implementation details are discussed here.

7.1 Molecule diffusion

We have implemented two versions of diffusion for the auxin (the work to compare the two has not yet been done).

The first to be implemented is the more simple, and will probably prove to be slightly faster in execution. The method follows an agent

based paradigm, modelling (collections of) auxin molecules as moving agents.

We need to estimate how much auxin is in a cell; this is a difficult estimate to make. The biological data for the amount of auxin in a cell is determined by crushing a section of a plant and measuring how much auxin there is in total, then dividing that value evenly amongst the individual cells. This assumes an equal distribution among the cells, which we believe not to be true, neither are the cells the same size. This method of estimation results in values for the auxin concentration in one cell that can vary by about two orders of magnitude.

In our Simulator, this auxin concentration is divided up into *auxin unit*s, the amount that would occupy 1 unit of space in the model (which is 1 square micron for the 2D model; one cubic micron for the 3D model). From the biological data, this corresponds to 20–2000 auxin molecules in each auxin unit [19, 20].

These auxin units diffuse around the model space. We are testing different ways of implementing this diffusion; currently we are using a random walk. In this approach, a "clump" of auxin molecules (all those corresponding to an auxin unit) move about together as one agent.

To test if this "clumping" is a problem, and whether it might be affecting the results of the simulation, we are currently implementing a second system of diffusion. This is a more continuous style model: every unit of space in the simulation has a number of auxin molecules associated with it. This representation allows a portion of the auxin to move into a neighbouring space, and also allows different areas of the simulation space to have different rules, allowing the rate of diffusion to be altered in different parts of the simulation. This is much more flexible, but is expected to be more costly in computing power. Having both systems is useful as it allows us to see if the more flexible but costly system is necessary, or if we can get enough information out of the simpler faster system, to determine if our hypotheses for auxin canalisation are correct.

We are interested to see if some of the features of the simpler system, that have been tested more, like intra-cellular gradients of auxin, are present in the newer system; and, if they are, can we see any differences.

We are implementing only one diffusion method for the proteins. This is the same as the simple agent based auxin diffusion system. Proteins are much larger than small molecules like auxin, and therefore each protein unit contains fewer protein molecules, and thus the approximation is less problematic.

7.2 2D and 3D simulations

The simulation has been designed to use much of the same code to model either 2D or 3D space. This means that we have 2D and 3D simulations and we know that the underlying algorithms and code is essentially the same. This is important because if we see significant differences in the behaviour of the different simulations for a given hypothesis or set of parameters we can be fairly sure that difference it due to the extra dimension, as opposed to differences in code.

We want a 2D simulator for performance reasons. A 3D simulation of 500 cells in an arrangement of $5 \times 5 \times 20$ cells high (representative of a section of plant stem exhibiting canalisation) might take a few days to run to a point where it can be considered finished (our simulations do not have end points unless some sort of target state is defined, otherwise the existence of a source and a sink allow the simulation to run forever). A 2D simulation of a similar arrangement of 5×20 cells high (representative of a longitudinal cross section) might be finished in less than an hour and require two orders of magnitude less memory (10s instead of 1000s of Megabytes: not only does the 3D simulation have more cells, but each 3D cell is bigger: $50 \times 50 \times h$ voxels, rather than $50 \times h$ pixels), and is comparatively very slow. If we find that a 3D model is necessary for realistic results, we will then investigate more efficient implementation approaches.

An important factor in the 2D simulation is the sizes of intercellular structures like vacuoles. A vacuole in a plant cells takes up a large amount of the space, squashing the cytoplasm to the inside of the cell membrane. Plant cells are 3D, so in our 3D simulation the ratio of cell size to vacuole size can be similar to what we measure in plant cells. What about 2D cells? Should a 2D cell look like a 2D cross section of a 3D cell? If so, should it be a slice through the centre, including the vacuole, or a slice near the membrane, missing the vacuole? Or should the ratio of 2D areas be the same as the ratio of the 3D volumes? Also, in a 3D cell, auxin can go round the vacuole along two axis, but in 2D it is limited to one. Should, therefore, the 2D vacuole have greater permeability to auxin than a 3D cell to account for the loss of a whole route to the other side of the cell?

We are investigating these questions in terms of the various timescales and scaling factors. However, having comparable 2D and 3D simulations based on the same implementation will allow us to validate our answers to these questions.

7.3 User interface

Biologists interact with the simulator through its user interface. In the current version of our simulator, altering model parameters and setting initial conditions can be a difficult process, particularly when defining the spatial arrangement of cells that the simulator uses.

Therefore we have started to develop a Little Language [6] (domain-specific language) to provide an easier interface to setting up the models via an interpreted language. This language provides an interface to two parts of the simulation. Firstly it can be used to set starting parameters for things such as the relationship between auxin and the expression of PIN protein, and the size of cells. Secondly, it allows the user to define what the layout of the cells in the model is, by defining cellular subunits (groups of similar cells) and then defining how the different subunits are arranged in the cell space.

8 Analysis Model: preliminary results

Eventually, will will build an analysis model of the simulator output that is analogous to the domain model. For now we conduct our analysis more informally.

The visual output from the simulation is shown in figure 12. The space between the cell membrane and the vacuole is the cytoplasm where the auxin (black dots) and proteins (gray dots) are synthesised. To avoid questions of sites of synthesis for now there is simply a certain probability that any position in the lattice produces a protein or auxin molecule when required. As the size and space of cells that canalisation occurs in varies quite widely, it is therefore important that a working hypothesis for canalisation in the simulation is not too dependent on size and shape. Therefore it is simple to vary the cell size and the size of the apoplast. It is also possible to have different sizes of cells in the same model, so long as care is taken to ensure that the cells align in a sensible way, without large gaps.

We have tested our first hypothesis for the regulation of auxin canalisation (figure 7) with the model as described here. The first hypothesis is simple and unlikely to be the full story, but it is based on ideas that might form part of the final hypothesis. The hypothesis tested concerns the positioning of the PIN proteins. PIN proteins can diffuse in the cytoplasm. When they associate with the cell membrane they are fixed in one position, unable to move. If the PIN protein exported some auxin on the timestep, it cannot disassociate from the membrane. If it did not do any transport, there is a certain probability that it can disassociate from

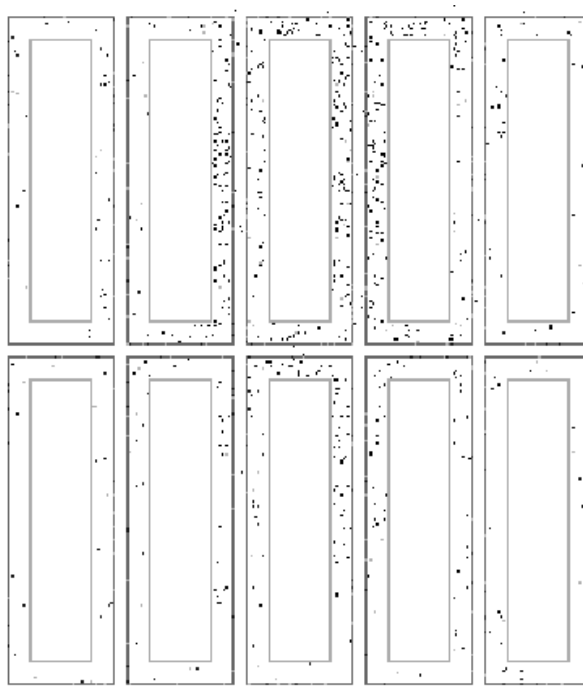

Fig. 12. Screen shot from a running simulation. The darker lines are the cell membranes, and the light lines are the vacuoles. The black dots are *auxin units* and the grey dots are either PIN or AUX/LAX proteins (darkest are the PIN). The middle cell on the top row is the only one producing auxin. The light gray line in the centre at the bottom of the figure is the auxin sink. This is where auxin leaves the model. One dot, such as an auxin unit, is 1 micron in size. To produce a clear figure, the apoplast is increased to 6 microns, from a usual 2 or 3. The cells are 147 microns high and 50 microns wide. For convenience only a few rows of cells are shown: the full simulation currently uses a 5×20 array of cells.

the membrane. The idea is that the PIN would naturally congregate on a membrane where they were able to export auxin.

Simulation of this hypothesis did *not* produce canals in the experiments carried out so far. This could be due to there being no feedback between the extra cellular auxin concentration and the PINs pumping auxin into that space. Therefore only the internal cell conditions stop the PINs from moving around randomly, as the PINs can respond only to the internal auxin concentrations of the cytoplasm, and not to the

auxin in the apoplast. Uneven distribution of auxin suggest that PINs could be responding to these internal conditions.

We intend to experiment further with this idea. We will also carry out more testing with the current hypothesis, as there maybe parameters that allow for the internal gradients of auxin to be enough to position PIN to produce canals.

Next, we will develop our model of auxin canalisation into a more complete model of shoot branching, to inform new models, and to form part of a multilayered approach to modelling shoot branching. Each layer can use an approach to modelling at the level of abstraction most suited to build a more complete picture.

9 Discussion

Our UML assisted development process has provided a number of advantages to our simulations.

The diagrammatic nature of UML as tool for producing various levels of models, including descriptions of program code, has helped produce simulations that not only work in an intuitive way, but that are built intuitively. Biology maps to UML objects in a way that can be understood by developers and biologists alike.

The resulting simulations are flexible. By concentrating on building the biological components and their interactions into the simulations we are able to test various hypotheses for the regulation of auxin transport. The results should be reflections of truly emergent behaviours, rather than due to those behaviours having being hard-coded into the simulation.

The use of different models to capture domain, software, and implementation details has helped produce conceptually cleaner models.

The Domain Model looks purely at the biology. Here class diagrams and state diagrams are of greatest use. The class diagrams allow us to look at the static structure of the model, and how the different parts are connected together. They can include the emergent properties of interest, so that we have these properties captured rigorously in a model. State diagrams provide detailed information of how the objects change in response to events. They are normally produced by thinking about the known biology of the different biological elements.

The Software Model does not explicitly include the emergent properties of the Domain Model: these should emerge from the interactions of the lower level simulated components, and can be compared against the Software Model for plausibility. At this stage inheritance is added to class diagrams, to indicate classifications and generalisations, and to

be used in implementation to reuse code and reduce duplication. The Software Model can also transform biological components to behave in non-biological ways that are more readily simulatable. For example, we may require more states to capture events than are provided in the Domain Model. The existence of the two models highlights areas where the simulation is breaking with the biology.

The biological literature gives details of how the experiments that produced the data were carried out in a lab. It should also be the case that how a simulation works and produces its data should be equally well explained, in order to allow independent validation of results and the sharing of methods and techniques among the modelling community. The increased use of modelling and the complexity of the simulations produced make this a more pressing need. UML can provide an effective way of developing and communicating simulations.

In the long term, UML could be used as an interface into models for biologists to use directly, to extend and develop models themselves. Our current work on the use of a Little Language to provide easier access to some of the deeper parts of the simulations could be extended, for example, to allow the addition of new proteins and their behaviours without the need to delve into Java code. (Of course, there is always the danger that the Little Language itself grows until it is of the complexity of Java. However, the intention is that it should be couched in biological domain specific terms, not generic programming terms.)

Eventually, biologists should be able to draw UML diagrams of these new proteins (or other objects), associate them to other biological objects, and link them to implementation objects that allow them to function. The links with the biology would confer the biological behaviour, and the links with implementation would handle diffusion, positioning, I/O etc. Or alternatively UML could be used as a simulation code navigational aid to allow direct access important parts of a simulation to allow biologists to tailor it to their own needs or add new functionality. UML could allow them to visually locate the part of the simulation that requires editing without looking though large amounts of code and needing to be able to decypher the way the simulation is constructed. The models produced would be more general in the capabilities and allow for more hypotheses to be explored.

Acknowledgements

This work is supported by a BBSRC/Microsoft Research CASE studentship. Thanks to Fiona Polack for discussions about UML models, and to the anonymous referees for their detailed comments. Thanks also to Lauren Shipley for proofreading the paper.

References

[1] *ALife XI, Winchester, UK, September 2008*. MIT Press, 2008.

[2] Paul S. Andrews, Adam T. Sampson, John Markus Bjrndalen, Susan Stepney, Jon Timmis, Douglas N. Warren, and Peter H. Welch. Investigating patterns for the process-oriented modelling and simulation of space in complex systems. In [1].

[3] Paul S. Andrews, Adam T. Sampson, Fiona Polack, Susan Stepney, and Jon Timmis. CoSMoS development lifecycle, version 0. Technical report, University of York, 2008. (in preparation).

[4] Eva Benková, Marta Michniewicz, Michael Sauer, Thomas Teichmann, Daniela Seifertová, Gerd Jürgens, and Jirí Friml. Local, efflux-dependent auxin gradients as a common module for plant organ formation. *Cell*, 115(5):591–602, Nov 2003.

[5] Tom Bennett, Tobias Sieberer, Barbara Willett, Jon Booker, Christian Luschnig, and Ottoline Leyser. The arabidopsis MAX pathway controls shoot branching by regulating auxin transport. *Curr Biol*, 16(6):553–563, Mar 2006.

[6] J. Bentley. Programming pearls: little languages. *Communications of the ACM*, 29(8):711–721, 1986.

[7] Peter Checkland and Jim Scholes. *Soft Systems Methodology in Action*. Wiley, 1990.

[8] S. Efroni, D. Harel, and I. R. Cohen. Toward rigorous comprehension of biological complexity: modeling, execution, and visualization of thymic T-cell maturation. *Genome. Res.*, 13(11):2485–97, jul 2003.

[9] François G. Feugier and Yoh Iwasa. How canalization can make loops: a new model of reticulated leaf vascular pattern formation. *J Theor Biol*, 243(2):235–244, Nov 2006.

[10] A. Finney and M. Hucka. Systems biology markup language: Level 2 and beyond. *Biochem Soc Trans*, 31(Pt 6):1472–1473, Dec 2003.

[11] Jasmin Fisher and Thomas A. Henzinger. Executable cell biology. *Nat Biotechnol*, 25(11):1239–1249, Nov 2007.

[12] Martin Fowler. *UML Distilled*. Addison-Wesley, 3rd edition, 2004.

[13] Verônica A. Grieneisen, Jian Xu, Athanasius F. M. Marée, Paulien Hogeweg, and Ben Scheres. Auxin transport is sufficient to generate a maximum and gradient guiding root growth. *Nature*, 449(7165):1008–1013, Oct 2007.

[14] M. Hucka, A. Finney, S. Hoops, S. Keating, and N. Novere. Systems biology markup language (SMBL) level 2: Structures and facilities for model definitions. *Nature Precedings*, 58(2), 2007.

[15] M. Hucka, A. Finney, H. M. Sauro, H. Bolouri, J. C. Doyle, H. Kitano, A. P. Arkin, B. J. Bornstein, D. Bray, A. Cornish-Bowden, A. A. Cuellar, S. Dronov, E. D. Gilles, M. Ginkel, V. Gor, I. I. Goryanin, W. J. Hedley, T. C. Hodgman, J-H Hofmeyr, P. J. Hunter, N. S. Juty, J. L. Kasberger, A. Kremling, U. Kummer, N. Le Novère, L. M. Loew, D. Lucio, P. Mendes, E. Minch, E. D. Mjolsness, Y. Nakayama, M. R. Nelson, P. F. Nielsen, T. Sakurada, J. C. Schaff, B. E. Shapiro, T. S. Shimizu,

H. D. Spence, J. Stelling, K. Takahashi, M. Tomita, J. Wagner, and J. Wang. The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. *Bioinformatics*, 19(4):524–531, Mar 2003.

[16] IBM. Developer of Rational Rose UML editing software. *http://www-306.ibm.com/software/awdtools/developer/rose/index.html*, 2008.

[17] N. Kam, I. R. Cohen, and D. Harel. The immune system as a reactive system. In *Proc Visual Lang. and Formal Methods*. IEEE, 2001.

[18] Eric M. Kramer. PIN and AUX/LAX proteins: their role in auxin accumulation. *Trends Plant Sci*, 9(12):578–582, Dec 2004.

[19] Eric M. Kramer. Computer models of auxin transport: a review and commentary. *J Exp Bot*, 59(1):45–53, Apr 2008.

[20] Eric M. Kramer. Mode parameters: the devil in the details. *The understanding and modelling of auxin transport in plants. Advanced Workshop, Nottingham University*, page 16, 2008.

[21] G. Mitchison. A model for vein formation in higher plants. *Pro. R. Soc. Lond.*, 207:79–109, 1980.

[22] G. Mitchison. The polar transport of auxin and vein patterns in plants. *Phil. Trans. R. Soc. Lond.*, 295:461–471, 1981.

[23] D. Morris, J. Friml, and E. Zamimalova. Auxin transport. In P. Davies, editor, *Plant Hormones: Biosynthesis, Signal Transduction, Action!*, chapter E1, pages 437–470. Kluwer Academic Publishers, 2004.

[24] J. Odell, H. Parunak, and B. Bauer. Extending UML for agents. In *AOIS Worshop at AAAI*, 2000.

[25] OMG. Maintainer of the UML standards. *http://www.omg.org*, 2008.

[26] Veronica Ongaro and Ottoline Leyser. Hormonal control of shoot branching. *J Exp Bot*, 59(1):67–74, Aug 2008.

[27] Tomasz Paciorek, Eva Zazímalová, Nadia Ruthardt, Jan Petrásek, York-Dieter Stierhof, Jürgen Kleine-Vehn, David A. Morris, Neil Emans, Gerd Jürgens, Niko Geldner, and Jirí Friml. Auxin inhibits endocytosis and promotes its own efflux from cells. *Nature*, 435(7046):1251–1256, Jun 2005.

[28] Fiona A. C. Polack, Tim Hoverd, Adam T. Sampson, Susan Stepney, and Jon Timmis. Complex systems models: Engineering simulations. In [1].

[29] Przemyslaw Prusinkiewicz and Anne-Gaëlle Rolland-Lagan. Modeling plant morphogenesis. *Curr Opin Plant Biol*, 9(1):83–88, Feb 2006.

[30] Didier Reinhardt, Eva-Rachele Pesce, Pia Stieger, Therese Mandel, Kurt Baltensperger, Malcolm Bennett, Jan Traas, Jirí Friml, and Cris Kuhlemeier. Regulation of phyllotaxis by polar auxin transport. *Nature*, 426(6964):255–260, Nov 2003.

[31] T. Sachs. The control of the patterned differentiation of vascular tissues. *Advances in Botanical Research Incorporating Advances in Plant Pathpology*, 9:151–262, Jan. 1981.

[32] S. J. Singer and G. L. Nicolson. The fluid mosaic model of the structure of cell membranes. *Science*, 175(23):720–731, Feb 1972.

[33] Petra Stirnberg, Karin van De Sande, and Ottoline Leyser. MAX1 and MAX2 control shoot lateral branching in Arabidopsis. *Development*, 129(5):1131–1141, Mar 2002.

[34] C. Uggla, T. Moritz, G. Sandberg, and B. Sundberg. Auxin as a positional signal in pattern formation in plants. *Proceedings of the National Academy of Sciences of the United States of America*, 93:9282–9286, 1996.
[35] K. Webb and T. White. UML as a cell and biochemistry modeling language. *BioSystems*, 80:283–302, 2005.