A virtual lab for exploring the $[\mathbf{PSI}]^+$ yeast prion

Jacqueline L. Whalley*, Mick F. Tuite† and Colin G. Johnson*
*Computing Laboratory and †Department of Biosciences
University of Kent at Canterbury
Canterbury, Kent, CT2 7NF, England

Abstract

This paper discusses ongoing work on simulating the propagation within the cell of a prion protein in yeast. The biological background to the project is outlined, and a number of questions about this system are posed. The paper then discusses how computer simulation is being used to provide a "virtual laboratory" for this system, which will be employed to support and understand real experiments. Furthermore, the development of the application is detailed with emphasis on the challenges encountered to date.

1 Biological introduction

Certain proteins within cells are capable of folding into abnormal three-dimensional structures without any change in their amino-acid sequence. Such conformational changes are interesting, but rarely of clinical significance because there is no information at the sequence level which would cause the change to be replicated and thus have a large-scale cellular effect.

However a small number of such changes have an extraordinary "infectious" property, whereby proteins in the normal conformation can be changed into the unusual, infectious form by contact with a protein in the abnormal form. It has been postulated that these abnormal proteins act as a template for this transformation process. Abnormal forms of proteins which have this infectious property and known as *prion proteins* [9, 10, 14].

Prion proteins have come to the forefront of medical and veterinary research in the last couple of decades because they are believed to play a role as an infectious agent in a number of diseases, such as scrapie, BSE and Creutzfeld-Jakob disease [14]. However prions can also be found in simpler organisms such as yeasts [20], which make an ideal subject for laboratory study of the prion phenomenon [1]. The particular system we are interested in in this paper is the protein *Sup35p* which is present in yeast cells of species *Saccharomyces cerevisiae* [17]. It is produced in the cytoplasm of the cell and, after combining with *Sup45p* proteins, is utilised by the *ribosome* where it plays a role in RNA translation.

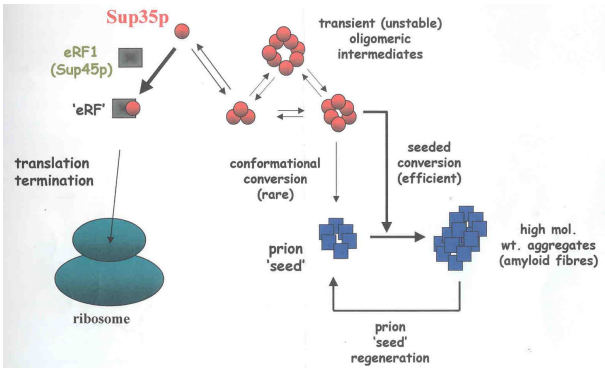

Figure 1. Normal and abnormal behaviour of *Sup35p* and related proteins within the cell.

The hypothesised behaviour of the yeast prions is illustrated in figure 1. *Sup35p* molecules are created within the cell and, depending on the other molecules that they encounter in their random walk around the cell, take one of two routes. The first is that they bind with *Sup45p* to form the protein known as *eRF*. This is used by *ribosomes*, which are structures that translate RNA sequences (ultimately derived from the DNA in the cell *nucleus*) and assemble the proteins needed for the cell to function. The second fate is that unstable clusters (*oligomers*) of *Sup35p* form. Whilst *Sup35p* molecules are part of a cluster, they cannot bind with the *Sup45p*. However the clusters can break apart and so this route is reversible, unlike the other route.

These clusters are believed to be the structures which can be transformed into the prion protein. During a very rare cellular event *Sup35p* clusters are thought to undergo a spontaneous change of conformation to form the abnormal infectious protein. Once this structure is formed it will act as a seed on other *Sup35p oligomers* by a complex and little-understood process that converts them into the prion

form. These prion clusters group together into large, sticky structures called *amyloid plaques*. When a number of these have been formed, they interfere with the cell's functioning to such an extent that it can no longer survive. It is these plaques which cause brain degeneration in prion brain diseases. This process can be readily studied in the laboratory by the insertion of a number of prion seeds into the cell.

There are a number of questions which we would like to understand about this process.

- When proteins within the cell come close to each other, what is the probability that they will bind to form an aggregate structure? Can we use a model to deduce likely values for these probabilities from the global behaviour observed?
- Are the prion seeds and aggregates moving around within the cytoplasm of the cell?
- How does the rate of production of the *Sup35p* and *Sup45* proteins affect the state of the cell?
- How do the prion aggregates dissociate? Do they break apart into large pieces or do individual molecules dissociate from the edge of the cluster? What effect does this have on the system? Does this differ from the way the non-prion clusters dissociate?
- Why do some processes (e.g. the inclusion of guanidine into the cell [2]) stop the process of prion propagation? How can we distinguish between various hypotheses to explain this phenomenon?
- It has been observed that almost all daughter cells of a cell which contains prion structures are themselves infected with the prion protein. Is this explainable by diffusion of prion seeds into the daughter cells whilst the daughter cell is budding off from the parent cell, or is another mechanism required to explain this?
- A number of structures in the cell (the nucleus, the mitochondria, et cetera) restrict the movement of proteins in the cell. Is the presence of these significant in understanding the propagation of the prion throughout the cell?

2 Understanding cellular dynamics using computer simulation

We are currently constructing computer simulation of the propagation of the prion protein within the cell. There are a number of reasons for approaching this problem through computer simulation.

The first reason is that it is possible to "look inside" virtual cells in a way which cannot be done with real cells. When a simulation of a system is constructed, it is necessary to specify the various parameters required. This provides a mechanism to discover feasible values for system parameters, which can be used to drive future experimental work. One way to to this is as follows. Measure a number of characteristics of the real system; for example in this system it would be possible to measure the time between the seed being inserted and cell death, a time series of number of seeds found at certain sample time-points, and the response of the system to various interventions. Similar characteristics could then be measured for instances of the simulated system, and the parameters adjusted (using an optimization techniques such as simulated annealing [11, 15], tabu search [4, 5, 6], or genetic algorithms [13]) until a set of parameters is found for which the simulated characteristics match the real characteristics. Clearly it is not possible to conclude directly that the underlying parameters are the same in the simulated and real systems, as there could be multiple parameter settings giving rise to the same characteristics; nonetheless such conjectures are useful for suggesting where future experimental work may be most fruitful.

More generally, the ability to search the space of possible variants on a system is a powerful reason for using simulation. In addition to searching the parameter space of a system for parameters which match reality, it may also be possible to drive the search to look for "critical points" in the system at which a particular intervention makes a significant change in system behaviour. One of our current projects is using this type of search to look for intervention targets in complex cellular systems for drug development.

Another way in which a "virtual lab" can be used as part of the scientific process is in distinguishing between various hypotheses. If we have a choice of hypotheses with which to explain a phenomenon, then we can use the simulation to create models which assume each hypothesis in turn, and by comparing simulation and experiment we can show which hypotheses are sufficient to explain the phenomenon at hand. We are about to use the system below in this way to investigate various hypotheses for the "curing" of proteins by guanidine hydrochloride.

This drawn upon a powerful feature of simulations—that they can demonstrate what is *sufficient* to produce a particular behaviour. If a certain simplified simulation of a system can reproduce the behaviour of interest, then it is likely that other components of the system are irrelevant to producing that behaviour. This underpins much of the work in the "artificial life" area [12], an important aim of which is to show how complex behaviour can arise from fairly simple systems.

3 Details of the simulation

We have created an individual-based object-oriented model of the above system. The object-oriented framework provides a powerful way of modelling systems such as these, as there is no need to directly model the complex behaviour of the system as a whole; individual components and the ways in which they interact can be modelled and the complex behaviour which emerges from this can be studied. The first step was identifying the various components of the system and their interactions. The objects within the cell which were included in the model were the following:

- Sup35p molecules
- Sup45p molecules
- Sup35p/Sup45p dimers (joined pairs of one of each molecule)
- Transient unstable Sup35p oligomers
- Prion seeds
- Prion aggregates
- Ribosomes
- The nucleus

Classes to represent each of these were created, with appropriate behaviours. The main behaviours were the ability of the structures to move around in the cell (implemented by using Brownian motion movement to simulate the complex environment within the cell) and interactions and binding with other components. The various relationships between the components in the cell and the `CellSegment` class (the main object which holds the above components and controls the model) is illustrated in figures 2 and 3 in the form of a UML [3, 18] class diagram.

The `CellSegment` class represents a cuboid extracted from a typical point in the cytoplasm of the cell (figure 4). It is a reasonable assumption that the flux of molecules out of the cell segment is equal to the inward flux of molecules into that segment. This is approximated by giving a toroidal topology to the cuboid, so that particles leaving one face of the cuboid re-enter at the opposite face.

An object of this `CellSegment` class drives the simulation, by providing a discrete-timestep clock and therefore sending "move" instructions to each object in the simulation at each timestep, and by sending out collision-check requests and managing the results thereof.

The problem of detecting precisely when two objects overlap comes up in many computational contexts. It is a crucial step in studies of molecular docking, such as simulations of how two proteins bind to each other. The problem of the detection of the collision of two proteins in this simulation was simplified to the detection of collisions between two spheres. When two spheres representing proteins collide, the decision as to whether they form an aggregate or not is then decided by giving a probability that the two proteins are in a position in which they can bind together. The problem of detecting collisions between spheres is known in computational physics as the *billiard ball problem*. It has been fairly well studied; see [8, 16] for surveys.

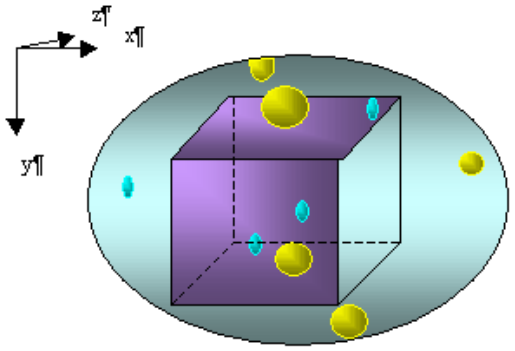

Figure 4. Sketch of the cell-segment as used in the simulation.

It was decided to implement (at least in the initial simulation) a naïve algorithm to detect such collisions. Namely, whenever a `Protein` is moved to a new position, the distance between this protein and all the other `Proteins` was calculated. If the distance, measured from centre-to-centre, was less than or equal to the sum of the two spherical proteins' radii, a collision has occurred and the `Proteins` must either dock/bind or bounce off one another.

There are a number of drawbacks with employing this algorithm. Efficiency is an issue. The collision detection algorithm is an $O(n^2)$ algorithm—because each `Protein` might conceivably collide with any other ball, the effort rises as the square of the number of `Proteins` in the system. Doubling the number of `Proteins` brings a four-fold increase in computational labour. Additionally it may be argued that this method of detecting collisions is artificial, as by the time a collision is detected the spheres may already have penetrated each other's volume.

A more serious issue is that in some cases the algorithm may miss a collision altogether. If the `Proteins` are not touching at time t and they are not touching again at $t + \Delta t$, there is no way of knowing that they passed through each other at some moment between these times. One potential solution would be to dynamically adjust Δt so that each move is always to the time taken to the earliest next collision. Due to these problems, we chose to reengineer the heart of the simulation by superimposing a grid in the volume and associating each `Protein` with a one or more points on that grid.

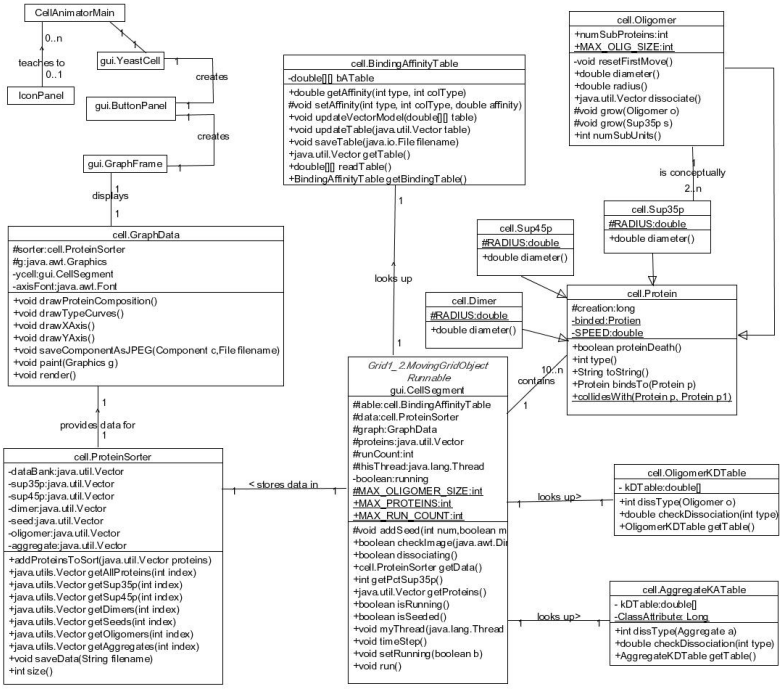

Figure 2. First part of a UML class diagram illustrating the various classes in the simulation and their relationships.

This grid model will not only simplify the problem of collision detection but also speed up the simulation. Adopting a model developed earlier at the University of Kent [7], we used a system that exploits the ability of OO systems to link two pieces of information together so that the two pieces are mutually aware of each other. The main representation is as before with the *Proteins* moving in three dimensions within a cubic volume with their respective centre positions monitored as a global floating point values. In addition to this space a three dimensional grid of cubes spans the space. Each point (which may be a centre point or position of a `Protein`) on this Grid "owns" the cubic volume represented by a `GridCube` object that is itself referenced by its left-upper-front corner. At each crossing point on the `Grid` there is a variable length list (such as a Java `Vector`) of `GridCube` references (C-style pointers) to objects that are moving in the `CellSegment`.

When the simulation is started we iterate through the objects in the `CellSegment`, working out which regions they belong to, and then add a reference from the appropriate `GridCube` point to the object. Then a list pointers to the objects contained within each `GridCube` is passed back to the respective `GridCube`. Thus each `Protein` knows which `GridCube`(s) it occupies and each `GridCube` knows which `Protein`(s)occupy its volume (figure 5).

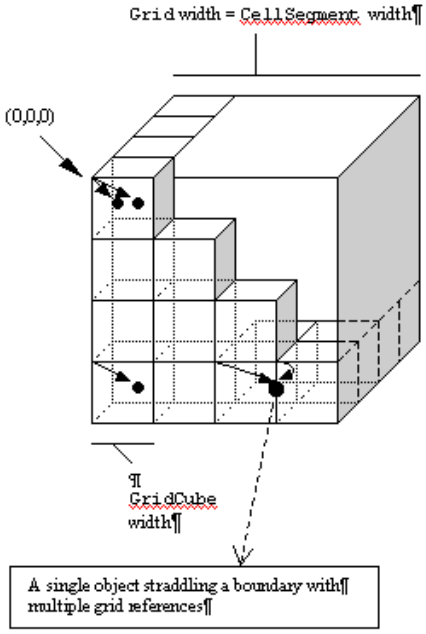

Figure 5. Clamping the objects to the corners of the grid cubes that the objects occupy.

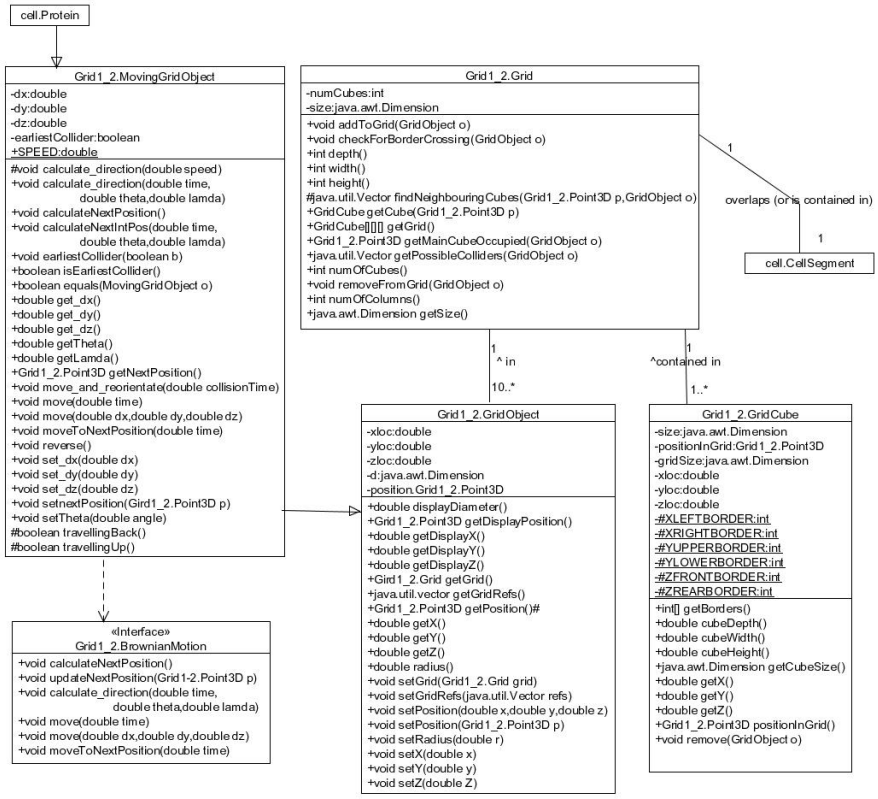

Figure 3. Second part of a UML class diagram illustrating the various classes in the simulation and their relationships.

Once this data is established checking for collisions is easy. We take the object we are interested in, predict its next position and iterate through its list of grid points. This gives us a list of regions the object occupies. Assuming that all the objects have a speed of less than half the width of a *GridCube*, we simply check the nearest neighbouring *GridCube*s of this region for objects that could possible collide with our object during this timestep. As we know the direction of travel of this object we can trim this algorithm further by only checking neighbouring cubes surrounding and in the path of object. Once all the possible collisions have been identified we then carry out an intersection check. The exact time when each of the possible colliding pairs just touch is calculated. If the time of collision is between zero and one this means that the collision occurs within the current time step. The earliest collision time within the timestep is found and all the objects in the `CellSegment` are moved to this earliest collision point (this idea was introduced in [16]). The two objects that have collided are re-orientated or undergo binding to form a new type of `Protein`. Then the process is repeated after updating the grid references and predicting the next position of each object. When the total time is one, a new timestep starts and the molecules have their direction of travel calculated according to the Brownian motion algorithm. If the total time taken plus the next earliest collision time is greater than one then all the objects are moved to a time of one and a new timestep started.

The binding of two or more proteins is a process that is in equilibrium. Their affinity for each other is a balance between their tendency to want to stick together (measured as their association constant, kA) and their tendency to come apart (known as their dissociation constant, kD). A high kA and a low kD mean that proteins have a will tend to stick together, the reverse is true if they have a low kA and a high kD.

In biological systems two colliding proteins do not necessarily result in a successful binding. The `Proteins` must collide or meet each other at the precise angle and position to allow successful docking, much like the space shuttle docking onto a space station. In order to mimic this behaviour using the simplified sphere protein model it was necessary to introduce binding affinities.

A new class `BindingAffinityTable` was created. This provides a two dimensional array of values for the kA of all the `Protein` types. Binding affinity values between zero and one stored in the 2D array indicate the probability of a successful binding. On collision a random probability (between zero and one) is generated and the `BindingAffinityTable` looked up to find the kA based on the type of `Proteins` involved in the collision. If the probability generated is less than the kA then binding occurs otherwise the two `Proteins` are reorientated and hence on the next move bounced off each other.

Both `Oligomers` and `Aggregates` dissociate but the actual implementation of this behaviour is different. An `Oligomer` is a highly unstable type of `Protein` which may break-up readily into smaller `Oligomers` and to individual `Sup35p` proteins. In contrast `Aggregates` are highly stable and believed to fragment to the prion `Seed`; this process propagates `Aggregate` growth. At each timestep every Oligomer and Aggregate is checked for dissociation.

In order to predict the probability of an `Oligomer` fragmenting a `kDTable[]` was introduced which determines if the `Oligomer` will fragment based upon size. The `kDTable` is looked up to determine if an `Oligomer` will fragment. As fragmentation occurs carried out the position of each fragment is calculated randomly using the `Oligomers`' position as a starting point and the newly created `Proteins` are added to the `CellSegment`. A simpler, but similar, approach was used to model the dissociation of the `Aggregate` by incorporating an `AggregatekDTable`. The implementation of the `AggregatekDTable` was almost identical to that for the `OligomerkDTable`. A random `double` was generated in the range $0.0 - -1.0$ and this value used to calculate the number of `Seeds` that would be shed if the `Aggregate` dissociates. This `Aggregate` may be switched on or off for a particular run of the simulation.

The final major component of the system is a visual interface. This is illustrated in figure 6. This visualises a two-dimensional projection of the cell segment onto the back face of the cuboid. Current work is using Java3D technology to provide a three-dimensional visualization of the simulation. The visual component is important for a number of reasons. Firstly it provides a tool by which scientists who are accustomed to seeing images of this system through microscopy can compare their intuitions about the behavior of the system with the simulation. This provides a way of looking for errors and inaccuracies in the model. Secondly the information that can be gained from a visual interface is unbounded-the visual interface can be used as a tool for "what if" type experiments, and the results gained are not limited to a small number of possible observation statistics. Once these features have been identified they can be investigated further by developing statistical measures that allow comparison with the same features in real experiments.

4 Notes and acknowledgements

Further details about the early part of this work can be found in the first author's masters thesis [19]. The authors would like to thank Jacki Goldman for her help and advice during the development of this work.

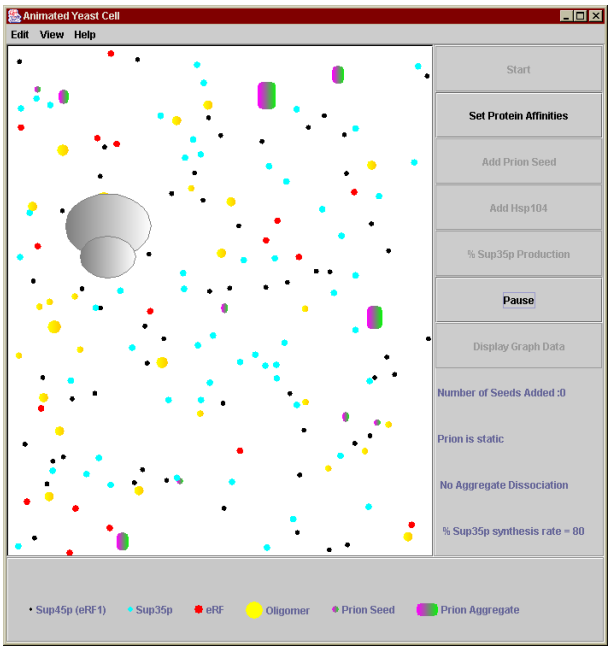

Figure 6. A two-dimensional view of the yeast cell

References

[1] B. Caughey. Transmissible spongiform encaphalopathies, amyloidoses and yeast prions: Common threads? *Nature Medicine*, 6(7):751–754, 2000.

[2] S. S. Eaglestone, L. W. Ruddock, B. S. Cox, and M. F. Tuite. Guanidine hydrochloride blocks a critical step in the propagation of the prion-like determinant $[psi^+]$ of *saccharomyces cerevisiae*. *Proceedings of the National Academy of Sciences*, 97(1), 2000.

[3] M. Fowler. *UML Distilled*. Addison-Wesley, 1997.

[4] F. Glover. Tabu search—part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[5] F. Glover. Tabu search—part II. *ORSA Journal on Computing*, 2(1):4–32, 1990.

[6] F. Glover, E. Taillard, and D. de Werra. A user's guide to tabu search. *Annals of Operations Research*, 41:3–28, 1993.

[7] J. P. Goldman, W. J. Gullick, D. Bray, and C. G. Johnson. Individual-based simulation of the clustering behaviour of epidermal growth factor receptors. In G. Lamont, editor, *2002 ACM Symposium on Applied Computing*. ACM Press, 2002.

[8] B. Hayes. The way the ball bounces. *American Scientist*, pages 331–335, July-August 1996.

[9] A. Horwich and J. Weissman. Deadly conformations–protein misfolding in prion disease. *Cell*, 89:499–510, 1997.

[10] G. S. Jackson and A. R. Clarke. Mammalian prion proteins. *Current Opinion in Structural Biology*, 10:69–74, 2000.

[11] S. Kirkpatrick, C. Gellat, and M. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[12] C. G. Langton, editor. *Artificial Life: An Overview*. MIT Press, 1997.

[13] M. Mitchell. *An Introduction to Genetic Algorithms*. Series in Complex Adaptive Systems. Bradford Books/MIT Press, 1996.

[14] S. Prusiner, editor. *Prion Biology and Diseases*. Cold Spring Harbor Press, 1999.

[15] C. R. Reeves, editor. *Modern Heuristic Techniques for Combinatorial Problems*. Blackwells, 1993.

[16] H. Sigurgeirsson, A. Stuart, and J. Wan. Collision detection for particles in a flow. *Journal of Computational Physics*, 172:766–807, 2001.

[17] H. E. Sparrer, A. Santoso, F. C. Skoza Jr., and J. S. Weissman. Evidence for the prion hypothesis: Induction of the yeast $[psi^+]$ factor by in-vitro-converted Sup35 protein. *Science*, 289:595–599, 2000.

[18] P. Stevens. *Using UML*. Addison-Wesley, 2000. Original edition 1999.

[19] J. L. Whalley. Object oriented computational models of prion propogation. Master's thesis, University of Kent, 2001.

[20] R. B. Wickner, K. L. Taylor, H. K. Edskes, M.-L. Maddelein, H. Moriyama, and B. T. Roberts. Prions of yeast as inheritable amyloidoses. *Journal of Structural Biology*, 130:310–322, 2000.