From biophysics to behavior: Catacomb2 and the design of biologically plausible models for spatial navigation

R. C. Cannon[a,b], M. E. Hasselmo[c] and R. A. Koene[c]
[a]Theoretical Neurobiology, Born-Bunge Foundation
University of Antwerp, Universiteitsplein 1, Antwerp, Belgium
[b]Institute for Adaptive and Neural Computation
Division of Informatics, University of Edinburgh
5 Forrest Hill, Edinburgh EH1 2QL, Scotland
[c]Department of Experimental Psychology,
Boston University, 64 Cummington Street, Boston, MA 02215

June 10, 2002

Abstract

A variety of approaches are available for using computational models to help understand neural processes over many levels of description, from sub-cellular processes to behavior. Alongside purely deductive bottom-up or top-down modeling, a systems design strategy has the advantage of providing a clear goal for the behavior of a complex model. The order in which biological details are added is dictated by functional requirements in terms of the tasks the model should perform. Ideas from engineering can be mixed with those from biology to build systems in which some constituents are modeled in detail using biologically realistic components, and others are implemented directly in software. This allows the areas of most interest to be studied within the context of a behaving system in which each component is constrained both by the biology it is intended to represent and the task it is required to perform within the system. The Catacomb2 modeling package has been developed to allow rapid and flexible design and study of complex multi-level systems ranging in scale from ion channels to whole animal behavior. The methodology, internal architecture, and capabilities of the system are described. Its use is illustrated by a modeling case study in which hypotheses about how parahippocampal and hippocampal structures may be involved in spatial navigation tasks are implemented in a model of a virtual rat navigating through a virtual environment in search of a food reward. The model forms a framework for how theta rhythm may be involved in performance of spatial navigation tasks and yields testable predictions about the phase

relations of spiking activity to theta oscillations in different parts of the hippocampal formation at various stages of the behavioral task.

1 Introduction

Two tasks at which the capabilities of computers far exceed those of human researchers are the management of very large homogeneous volumes of data, and the numerical calculation of the behavior of complex systems based on precise and complete formulations of their constituent parts.

Both of these capabilities are of great potential benefit to research in neuroscience and yet the uptake of database technology and the growth in the use of models has been very much slower than in many other sciences. This may be ascribed, on one hand to the extreme heterogeneity of the information to be stored and processed, and on the other to the total absence of "precise and complete formulations" of the behavior of elementary constituents of neural systems. That is, heterogeneity is as much a problem in computing collective behavior as it is in storing and handling data. Indeed, ambitious recent developments such as CellML (*www.cellml.org*) or Neurospaces(Cornelis and De Schutter, 2003) intentionally blur the distinction between representations of the biological structure and of the mathematical properties of a system. They treat the mathematical formulation simply as more first-order information to be processed along with a system's logical and spatial structure, in order to derive higher level properties.

The impact of heterogeneity on the applicability of computational methods in neuroscience has often been underestimated, as characterized by the view that if we just work a little harder and make a bit more effort, then the methods that are so effective in, for example, theoretical physics, will yield equally fruitful and compelling results in neuroscience. This view neglects, or denies, a fundamental difference between physics and life sciences which Schroedinger(Schroedinger, 1956) describes as the difference between "hot" and "cold" systems. In a "hot" system (almost everything treated by theoretical physics) as you include more and more elementary units, their collective behavior begins to be independent of the detailed properties of the units themselves. Consequently, mathematical approximations get better and better for larger and larger systems. Cold systems (which includes all living things) behave in the opposite manner, more analogous to a machine than to a statistical equilibrium, with the variety of realizable behaviors growing with increasing size.

A consequence of this is that although abstract mathematical models and detailed physics-style bottom up models can both be useful, there is a whole domain of computational applications in neuroscience that is simply not represented in other scientific fields. These are the techniques appropriate to the study of complex "cold" systems in terms of information management

and software engineering. They form a major constituent of the emergent field of neuroinformatics.

Because of the closer analogy of neural systems to machines (mechanical, computational or economic) than to perturbations of statistical equilibria, it is to be expected that much of the methodology and technology of neuroinformatics should owe more to engineering or commerce than it does to applied mathematics and theoretical physics. Thus, for example, the software systems of choice in computational neuroscience include C^{++}, Java and XML (Goddard et al., 2001a,b; Forss et al., 1999), as in the business community, whereas these systems have only achieved minimal penetration among physicists, presumably because they are not particularly useful for most research problems in physics. Likewise, neuroscience database technologies(Cannon et al., 2002) are more closely related to distributed shopping systems than to their highly centralized astronomical counterparts(Wenger et al., 2000). One exception to the correspondence with business software is the extensive use of Lisp, the preferred language of artificial intelligence research, in the Surf-Hippo Neuron Simulation System(Borg-Graham, 2001). The motivation for this choice includes features such as platform independence and access to the interpreter which have been present in Lisp for many years. The eventual appearance of these features in the C-family of languages is a large part of the appeal of Java for neuroinformatics applications.

Good sources of inspiration in using computers to study the integrative behavior of neural systems can be found in various domains of software and hardware engineering, including computer-aided design, e-commerce applications, computer games(Funge, 1999) and, indeed, almost any modern application of object-oriented design(Gamma et al., 1995). In conjunction with more conventional areas of numerical analysis, these have been primary influences in the development of the present modeling system via various intermediate stages(Cannon, 2001a; Cannon et al., 1998; Hasselmo et al., 2002b).

The next section explores the advantages of the design-based approach to modeling in comparison with conventional deduction-based methods. It is followed in section three by an examination of how best to represent multi-level biological models. Each section begins with an analysis of the problem and works through possible solutions, ending with the details of the particular choices made in Catacomb2. The software architecture of the system is described in section four, and the main biological and non-biological components that are currently available for use in models are presented in section five. Sections six, seven and eight cover three of the most important requirements of the current generation of modeling systems: meta-modeling facilities such as sensitivity analysis and optimization; infrastructure for model sharing and publication; and mechanisms which allow the system to be extend or used in conjunction with other tools. Throughout the text, concepts are illustrated with examples taken from the first major model

built with Catacomb2. This model involves spatial navigation by a virtual rat in a virtual environment guided by spiking activity of populations of cells representing parahippocampal and hippocampal structures(Hasselmo et al., 2002b,a). Scientific and technical aspects of these models are presented as a case study in section nine. Finally, section ten reviews recent progress and discusses future directions for the software and modeling work.

2 Modeling philosophy

Neural systems have functionally important features on a wide range of spatial and temporal scales from spine morphology to system connectivity, and from receptor kinetics over a period of milliseconds to permanent morphological changes. Building an artificial system which encompasses all, or even some part, of this range involves constructing a path through the space of possible models which connects structure at the smallest scale present to that at the largest scale via all the intermediate levels(Kötter et al., 2002; Borg-Graham, 1999). In some respects, this can be likened to solving a differential equation where constraints, or boundary conditions, are imposed on the solution at both ends of the domain. The historical development of numerical methods for solving differential equations can therefore be used to frame corresponding ideas about how to tackle multi-level modeling problems.

2.1 Neither top-down nor bottom-up

One of the earliest and simplest approaches to solving two-point boundary value problems is known as the "shooting" method. It takes the known conditions at one end, guesses any unknown quantities that are needed, and propagates the solution across the domain to the other end. In general, the resulting path will not meet the desired boundary conditions, so the initial guesses are modified and the process is repeated. If the equations are well-behaved such that the point at which the path meets the far end varies systematically with the initial guesses, then the right solution can eventually be found by judicious adjustment of the initial guesses. Both the bottom-up (working up from biophysics) and top-down (working down form behavior) methodologies of computational neuroscience are analogous to shooting methods. Unfortunately, it is a well established result of numerical analysis that shooting is only a successful strategy for relatively simple systems comprising no more than a few, nicely-behaved, equations. It is therefore unsurprising that these methods are very difficult to apply successfully in neuroscience, where the boundary conditions are complicated and the integrative behavior even between adjacent levels rarely obeys simple rules. Fortunately, the differential equation literature contains many methods that have been developed to work in more complex environments. These form the

basis of much day-to-day numerical work in theoretical physics (e.g., Eggleton (1971); Lattanzio et al. (1997)) and suggest that analogous methods may prove useful in neuroinformatics.

Among the methods developed for two-point boundary value problems, perhaps the most widely used and generally useful are relaxation methods(Press et al., 1993). The strategy here is to start with a complete path across the domain, that, of course, is not a solution, but that *does* meet the desired conditions at the two ends. The path at least provides a value, albeit wrong, for the function at all intervening point. An iterative procedure is then used to gradually adjust all these points together in order to bring the path closer and closer to the correct solution - a process termed "relaxation". An alternative version has the starting configuration be the correct solution to a different, simpler, problem and then gradually move towards the real problem in small steps such that the solution can be kept up to date for each change(Cannon, 2001b). The challenge in this method is to find an acceptable starting solution, or simple equivalent problem, and then to come up with an iteration scheme which does indeed bring it nearer to the desired result. Relaxation methods have been applied to many problems which are intractable by the shooting method, and they are responsible for the vast majority of our knowledge about certain types of system including, for examples, the internal structure and evolution of stars(e.g., Faulkner (1968); Eggleton (1971); Lattanzio (1986)).

Based on these observations from numerical analysis, one of the design goals in developing Catacomb2 has been to facilitate model development by relaxation, rather than only by shooting. It should be stressed that the correspondence to numerical analysis is strictly an analogy. Internally models often require the solution to differential equations by a variety of implementation-dependent methods, which may indeed involve relaxation methods, but this is a choice of a particular implementation. The intended analogy with relaxation is that it can also guide the way a modeler interacts with a modeling package. The modeler's goal is to achieve a plausible computational equivalent to a biological system, and the relaxation approach is to start with an implausible model that at least covers several parts of the problem domain and then gradually refine its internal components. Thus, Catacomb2 aims to help build approximate models which show complex behaviors, rather than biologically very realistic models which only reach simple integrated behaviors. It does this by allowing a wide variety of software components in the chain linking biophysics to behavior, some implemented with biologically realistic components, others with algorithmic "black boxes". So, for example, besides models of cells, it also has logical components capable of performing tasks such as computing a direction of motion from current and desired position. A behaving model can contain any mixture of high-level logic with plausible neural circuitry. The aim is first to build a system which performs the task in question, then gradu-

ally to refine (relax) its implementation to use more realistic components. The advantage is that at every stage there is input from both ends — what nature actually uses to achieve particular functional goals, and what algorithms these components may be implementing. Intuitively, this is also the reason why relaxation methods are so successful in numerical analysis: they allow the update step at a particular point to be influenced by information propagating from all the constrained points of the system.

A correlate to this approach is that there are no pre-defined "correct" structures to use in an intermediate model — any structure which helps the model work is legitimate, and is a potential basis for subsequent refinement by the relaxation process. Models can exploit concepts from a variety of domains, including continuous-time and discrete-time processes, as in the growing field of "heterogeneous modeling"(e.g., *www.ptolemy.org*). This eliminates many of the somewhat arbitrary boundaries which sometimes appear in the modeling literature, such as the distinction between channel-based cell models and integrate-and-fire cell models. In Catacomb2, for example, there is no barrier to constructing a cell model which fires a spike and is reset after reaching a specified threshold, but which also contains membrane ion channels to generate subtle sub-threshold behaviors. Indeed, there are many numerical advantages to separating out the spiking behavior as an all-or-none event while working with the channel kinetics at a slower time scale appropriate to other sub-cellular processes. Five examples of cell models at different levels of complexity are shown in figure 1. The first behaves as a simple integrator. Features are added one by one to create a range of models: any model can be used in a network according to the functional needs in a particular context. Further examples of cells designed to play specific roles within a network are presented in section nine.

2.2 Constraining possible models

One of the first problems in using a general purpose differential equation package to model a biological system is that it provides too much freedom both in the choice of mathematical formulation and in parameter values. A central goal of domain-specific software systems is therefore to restrict this freedom so that the parameter space reflects the plausibility of particular models in the domain. Users wish to be presented with options which allow them to build working systems, not options where the vast majority of configurations yield meaningless results.

Script-based modeling systems such as Neuron(Hines and Carnevale, 2001) and GENESIS(Bower and Beeman, 1994) improve on general-purpose packages by providing a set of biologically meaningful constructs which can be augmented, where necessary, with general purpose code. The approach taken in Catacomb2 has been to accentuate this distinction by providing a wide range of predefined components that are intended to work together

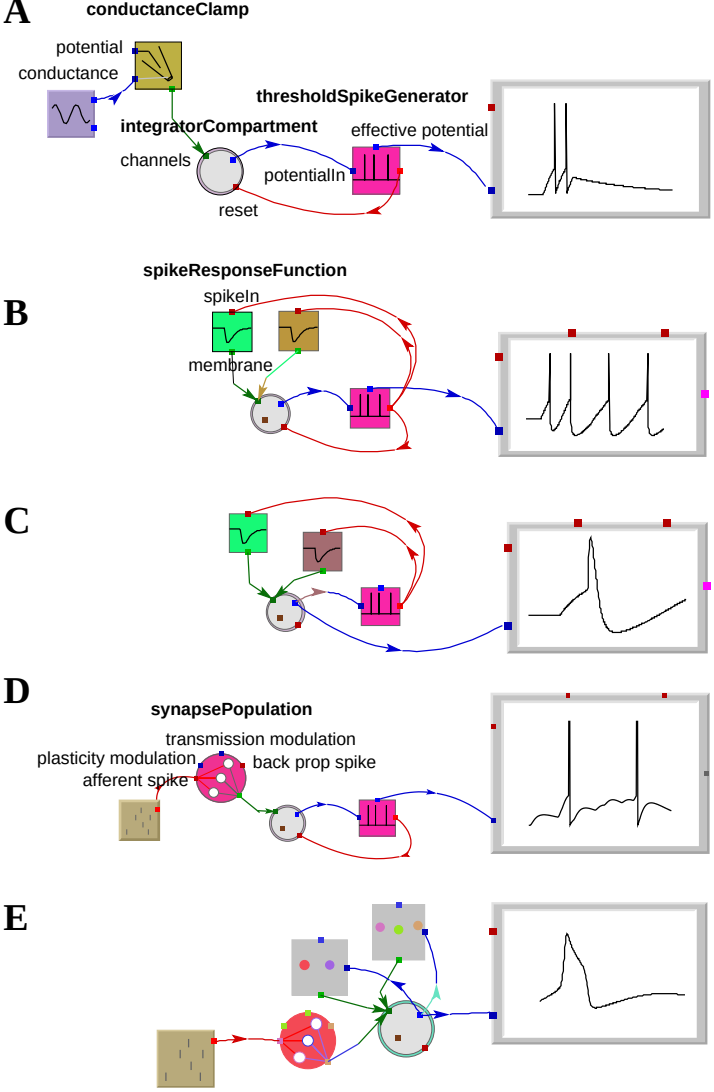

Figure 1: Component-based single cell models. The components on the left are used for stimulation, either by imposing a transient conductance change on the membrane (A,B, and C) or by sending a train of events to a population of synapses (D, and E). The right-most component shows the membrane potential of the cell. All figures are generated directly from the software and appear much as they do through the user interface. A. Simple integrate-and-fire cell with a leaky integrator compartment and a spike generator which fires an event whenever the cell exceeds a threshold. The spike generator reads the potential of the cell and causes the potential to be reset when it fires an event (link going back to the "reset" port on the integrator compartment). B. As case A, but with the addition of two spike response functions that are triggered by events from the spike generator. One is set to produce a rapid after-hyperpolarization, and other to give a slow after-depolarization which causes the cell to continue firing periodically after the stimulus finishes. C. As B but without the connection from the spike generator to the compartment reset port. The cell is repolarized by the spike response function and has slightly more realistic spike shape. D. A simple integrator with complex synaptic input. The component on the far left delivers a sequence of events to the synapse population, each of which causes a biexponential conductance change on the compartment. E. Spikes generated by ion channel models. There are two populations of ion channels loosely representing sodium and potassium channel kinetics. Together they generate a wide spike in response to input from the synapse population.

in almost any configuration which the system lets the user assemble. More complex logic can be implemented by writing new code modules or scripts, but for many applications this should not be necessary. It encourages the view that script or module writing is a slow one-off activity which should be carefully planned and revised, whereas model construction, testing and modification is an everyday activity to be made as quick and as easy as possible.

The design of a system which permits only interesting models to be built, or rather, one with a parameter space which is at least densely populated by useful models, remains a major challenge in computational neuroscience. With such a system, for example, genetic algorithms or other optimization techniques could be used on points in the model space itself. At present, a certain part of the instantaneous structure of biologically plausible models is captured, but there is no mechanism for describing the slower regulatory processes which would govern many of the quantities in a real system. The resulting models suffer the same fragility with respect to their parameters as is familiar from many other systems. In effect, although the parameter space does contains interesting models, it fails to satisfy the natural corollary that it should also be relatively smooth and that each model should occupy at least some minimal volume of the total space. There is a simple mechanism for testing this with a `perturber` component that goes through any model and introduce random changes, according to specified probability density functions, in all the parameters. Ideally, if a model covers a significant part of the space, then introducing random multiplicative changes of a few percent should not change the gross behavior. Perturbation of complex models, however, generally changes the behavior substantially, indicating that there is scope for improvement in the choice of parameterizations. This is considered further in section ten.

2.3 Model development

Computers are often used to perform numerical calculations in physics in a single step: a program is written which incorporates the physical constants and input conditions and generates results. In some cases, where the program is used repeatedly, there are two steps: first the numerical method is implemented, then it is run repeatedly on different data sets. Increasingly now, it is a three step process, with the first step being the implementation of a set of general-purpose mathematical procedures by professional software engineers as in MATLAB (*www.matlab.com*) or Yorick (*ftp://ftp-icf.llnl.gov/pub/Yorick/doc/index.html*). These are then used by researchers in much the same way as they would use a low-level programming language. Methodological development is clearly in the direction of multi-layer systems where complex problems are divided according to the available expertise. While the efficiency gains, and therefore design pressure in this

direction remain modest in many areas of applied mathematics, the complexity of implementing brain-like systems makes it an absolute necessity in neuroinformatics. Indeed, the ideal number of levels in neuronal modeling may be many more than three, and systems which allow this separation and specialization may make more effective research tools than those that do not. For example, a single modeling project may involve writing an efficient numerical implementation of an ion channel, building a biologically plausible single-cell model, designing a network to implement a given algorithm and setting up a realistic behavioral experiment. These tasks require quite different expertise and can best be done by different researchers working together.

At present, Catacomb2 tries to achieve this separation first by starting with the constructs provided by Java, an object-oriented language with extensive libraries for managing data structures, building graphical user interfaces and accessing the Internet. It then has hard-coded modular implementations of many basic constructs from neuroscience including ion channels, synapses, isopotential compartments, projection patterns and populations. These can be assembled into functional units through a graphical interface, and such units can then be treated as single components for reuse in more complex models and so on up to the level of modeled behavior. In addition, layers can be represented as coarsely (using simple algorithmic components) or finely (with detailed biologically-oriented models) as required so that a complete multi-layer system is available early in the development process. This allows a single layer to be refined within its wider functional context even before realistic implementations of other layers are available. This approach does achieve part of the goal of multi-layer modularity, but, nevertheless, it remains purely structural. A more efficient and robust development strategy might allow even coarser implementation of some parts of a model by defining the tasks a section should perform, or the concepts it implements. This would broaden the range of possible models and make a more direct connection with much current top-down modeling work.

3 Model structure

The development of NeuroML(Goddard et al., 2001b) and of similar "MLs" (Markup Languages) in other domains has occasionally been misunderstood as an almost magical solution to problems of compatibility between modeling systems, as though it would one day be sufficient to click "save as neuroml" to export a GENESIS model in a form which could be read by Neuron. This seems very unlikely to happen for any but small parts of existing models, primarily because of fundamental differences in the ways in which these systems describe models internally. The real advantage to community agreement on a standard such as NeuroML is that it carries with

it a particular way of describing models of biological systems. If there is agreement at this level, then the details of taxonomy and file formats are no more than a software problem that can easily be overcome.

There is little dissension to the idea that purely declarative model descriptions are a good thing(Beeman et al., 1997). That is, that a description of a model should contain only statements of the model structure and parameter values, essentially a set of grouped "name=value" statements. This is to be contrasted with procedural model descriptions which resemble a computer program, possibly with conditional statements and looping constructs. Only by following through the program can the full model be reconstructed. The advantages of a declarative description include readability, since each statement should be meaningful on its own, and portability since it is not dependent on any particular execution environment. The distinction, however, is not always clear as for example in describing a neuron with spines distributed along its dendrites. A declarative description of such a cell might include the individual positions of all the spines. This would be fine if the spine positions came from a detailed morphological reconstruction, but if they were allocated according to some statistical rule, then storing the positions is typically *not* what is required. The underlying model of interest is a combination of the dendritic morphology with a statistical distribution of spines. A procedural description of the model might include a fragment of code for generating spine positions perhaps as a function of segment diameter — effectively a mini-program for sampling a probability density function. This is also *not* what is required, because it might be rather hard to deduce the density function from the procedural description, and moreover, code for sampling from implausible or ad-hoc distributions can look very similar to code for sampling from a standard distribution which imposes the fewest unfounded assumptions. A declarative description, on the other hand, would would simply state what the spine density function is and leave the process of sampling from the distribution up to the software that runs the model. The density function is made explicit and the extent to which it departs from minimal assumptions is immediately apparent from the number of statements required to define it.

There is also widespread agreement that object-based descriptions, where parameters are grouped within conceptual units which can then be used as a whole, are an obvious choice. Opinions vary, however, about where some of the boundaries should be drawn between objects, and most importantly about the global structure of a biological model. One obvious possibility is to use an object tree, where the objects are described where they are needed. So, for example, a model with two branched cells might have two nodes at the first level, one for each cell. Then each node would have a number of children, representing the different segments of the cells. Each segment could have a number of children representing the ion channels on that segment. The difficulty here is that the same ion channel model proba-

bly occurs many times, so the parameters have to be copied into every case where it is needed. Where then are the definitive parameters if the channel models is to be changed? Is it legitimate to change them on one segment but not on the others? What happens when the model is saved — do you get multiple descriptions of the same thing?

Various solutions to this problem are currently in use, mostly based on storing the parameters in a single place and making reference to them when needed. The approach taken here, which is also the one adopted for NeuroML, is that almost all objects should be stored in top-level sets, and that where one object "contains" objects of another type, it should in fact just contain a set of very simple objects (unique to itself) each of which makes a reference to the corresponding descriptive object. Thus, in the above example, there would be top-level sets of cells and of ion channels. Each cell object might contain a set of channel-reference objects. And each channel-reference object would point to an ion channel model in the top-level set. Channel-reference objects might contain other information, such as the density of the ion channel in the context of that particular cell, but would not contain any channel properties which are also required elsewhere. This principle of minimal redundancy, where an object should be stored in a top-level list if it is ever likely to be used in more than one place, eliminates problems of changing a model in one place but not in others, and allows graphical user interface tools to operate effectively on the models. In practice, it is also an excellent way to decide how models should be dissected into objects and references.

One feature discussed for various ML's including NeuroML and SBML(Hucka et al., 2001) (Systems Biology Markup Language) is the ability to use expressions in a model to define how a particular subcomponent is to be used in a given context. For example, the channel model might contain (as in the NeuroML draft(Goddard et al., 2001b)) a parameter, G_{max}, which is its maximal conductance *per square micron*. But this depends on the context of the channel. Each time the channel is used, there is then a need for some statement in the cell model to set G_{max} for the channel in that particular context, perhaps as a function of the radius and length of a dendritic segment. Catacomb2 contains no mechanism of this sort, and instead takes the apparent need for an expression in this context as evidence that the boundaries have been mis-drawn and that G_{max} should not, after all, be part of the channel model. So far, all cases where such requirements arise have been settled this way with careful consideration of what parameters should or should not be part of a particular object. Often the apparent need for expressions and functions results from a desire for conciseness which can instead be settled by introducing more objects and making a genuinely declarative statement about what the model is. For example, in the above case the only conductance parameter present in a Catacomb channel is the single channel conductance. A cell model has a set of channel reference ob-

jects each of which contains the name of the channel to be used, and the number density (channels per square micron) at which it is to be used in the cell. That is, there is a whole new layer of objects essentially just to hold references and densities. It should be stressed that the structures used to represent models do not impose any constraints on the internal representation of those models in a particular system. Nevertheless, because the requirements of a graphical user interface and of a model description system are so similar, Catacomb2 uses the standard structures for its user interface. These structures are not, however, well suited to numerical calculations, so there is a completely separate representation for the model implementation (section 4.3).

The model description is the only part of a model which the system can save to a file. There is deliberately no journaling facility, and no facility for saving the internal state of a calculation. Anything that needs to be reinstated when the model is reloaded must be made an explicit part of the model description. This includes, for example, statements that specify such things as which file should be used to read in weights of the synapses in a particular population. Models are therefore forced to be purely declarative, with all the potential advantages of robustness, visibility and portability that it entails. This also makes it natural to store models as XML (*www.w3c.org/XML*) which is now the default in preference to an earlier file format which used C-like structure definitions to store models.

Note that in this context, "models" are static descriptions of the properties of a system. They make no mention of the state information that will eventually be associated with instances of models when their behavior is computed, except occasionally to specify initial values where there is no other unambiguous way of assigning them. As far as Catacomb is concerned, state information remains implicit in the model, and only appears within numerical implementations. This is a compromise between completeness of description and implementability. As mentioned in the introduction, other systems(Cornelis and De Schutter (2003); *www.cellml.org*), are attempting to expose more of this information. The approach presented here can also be seen as employing the layering principles mentioned above: numerical implementation and state information is handled by programmers and implemented in source code; model structure and parameter values are handled by model builders and implemented in model descriptions. Although it would be conceptually elegant and theoretically powerful to provide complete system descriptions at the model level, this option has been sacrificed in the present system in favor of ease of implementation, on the assumption that very few users would exploit such a possibility.

3.1 Data structures for model description

The "objects" into which a model description is broken down will here be termed "frameworks" to avoid confusion with objects used in software implementations. A framework defines a structure —field names and types— but does not set any field values. Models can be made from a framework by adding information on the parameter values. Technically, a framework is no more than a set of parameter definitions where each definition specifies the name and the type of a parameter. Most of the allowed types come from a subset of those available in many programming languages comprising the primitive types boolean, int, double, String, and one-dimensional arrays of these primitive types. There are also two complex types, here termed `Set` and `Reference` following the NeuroML convention(Goddard et al., 2001b).

The present framework definition system is rather vague about two important model description issues: the units in which values are expressed; and the additional information about a parameter which is often required in order to build an effective user interface. The latter may include, for example, whether a logarithmic or linear scale should be used for floating point values and what the default range should be. Currently, the units are fixed by the framework, and the rest of the information is provided as optional hints to the user interface. This removes any danger of unit incompatibility, since the user has no choice about the units.

Besides the primitive data types and arrays, models are built exclusively from the two higher-level data structures — `Set` and `Reference`. A `Set` can contain any number of models but they must all be of the same type. When a `Set` is constructed, it is told the base framework to which its contents belong. A `Reference` simply holds a string referring to a model by its framework (hard-coded) and name (user-defined). It is the responsibility of the system to find the model being referred to when necessary by a closest-first tree search until it finds a set containing models with the right framework, and then by looking up the model name within that set. This is how the minimal-redundancy principle discussed above is implemented in practice. Any model component that is likely to be reused can be parameterized just once and stored in a `Set` somewhere. Wherever it is needed it can then be referred to simply with a `Reference` of the appropriate type mentioning the object by name.

The extensive use of references results in relatively shallow tree structures for models, with most objects living in top-level sets, or as the children of elements in top-level sets, and referring to other top-level components by name. This also means that almost any component which can meaningfully be reused is directly available without any duplication of the parameters. As an example of this structure, the representation of a simple aqueous solution is shown in figure 2. A solution contains a variety of compounds, each at a different concentration, and each compound is composed of charged species.

Port	Use	Notes
`Spike`	run-time	event propagation
`Vector`	run-time	data fetching
`Solution`	run-time	solution distribution
`Attachment`	compile-time	physical co-location
`Redirection`	compile-time	reference transfer
`Object`	run-time	general purpose object transfer requiring agreement between sender and receiver
`Stream`	run-time	continuous data streams

Table 1: Principal connection types between objects

Compounds and species are likely to be reused in many different contexts, so they should live in top-level sets. Each compound must then contain a set of private objects indicating the species it uses. Similarly each solution contains a set of "SolutionElement" components which combine a reference to a compound with local information on its concentration.

3.2 Hierarchical assemblies

Besides the primary frameworks which contain parameters for setting properties of particular types of model, there is one general purpose framework which covers almost all the rest of the model description problem — the `Assembly` framework. An assembly holds a set of other models, each of them an instance of a primary framework or of another assembly. Typically the elements of assemblies will make references among themselves encoding the structure of the model as a graph. In this picture there is no distinction made between edges and vertices since all elements have the potential to show both vertex-like and edge-like properties. Moreover, there is also no restriction that elements with edge-like properties should have only two ends, so, for example a catalyzed reaction component may connect three or more distinct pools — the reactants, the products and the catalyst.

Connections between models are defined by attaching ports on one component to those on another. The input and output ports are defined by the framework in terms of the data, events, or properties they mediate. An output port of a given type can only be connected to an input port of the same type. Ports are always accessible to other components in the same framework and, optionally, to components one layer up in the hierarchy. The main port types and their uses are shown in table 1.

Almost all run-time interactions are handled by the spike and vector ports. The former transmit discrete events, and the latter provide read-on-demand access of a vector of double values from one component to another. The attachment and redirection ports are the two main referencing mechanisms between objects within an assembly. Attachment ports provide a

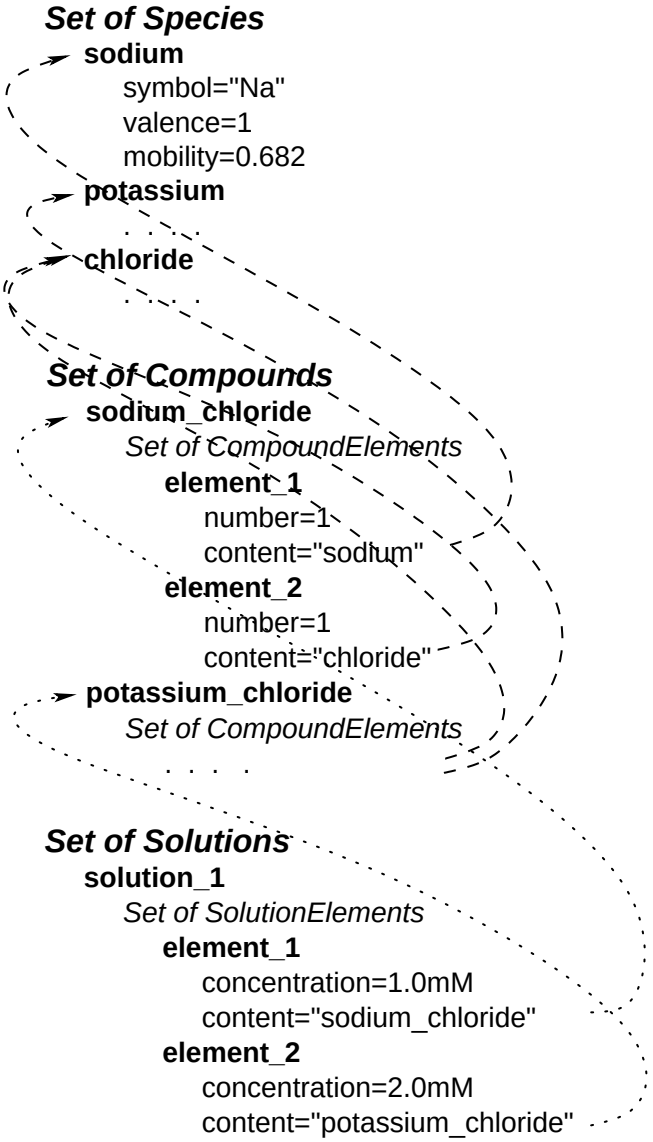

Figure 2: Set and reference structure for describing aqueous solutions. Slanting text indicates sets, bold normal text model components, and plain text fields and their values. Parts of the description have been omitted as indicated by rows of dots. Each specie is defined only once, in a top-level set of species. Likewise compounds and solutions are kept in top level sets. Each compound needs to refer to the species it contains, but with each reference it also needs to specify the number of occurrences. Therefore each compound has a local set of `CompoundElement` objects which are not visible elsewhere. Each element contains a number and a reference to the corresponding specie (dashed lines). Similarly, solutions contain a local set of objects to combine the concentration of a compound with a reference to its content (dotted lines).

means to specify that distinct objects are physically connected together (as in the components of an animat for example). Redirection ports are used in a variety of contexts where objects are represented by distinct frameworks but where one is somehow located inside another, as in a channel being inserted in a membrane, or a rat being placed in a maze. Finally, the `Object` and `Stream` components provide flexible data transfer between cooperating components such as reading files or transporting a connectivity matrix from the projection pattern that generated it to a display component.

4 Catacomb2 system architecture

The main components of Catacomb2 are shown in figure 3, grouped into sections for the user interface, model description package and a stand-alone numerical implementation package called Toucan, which is currently used for all the calculations. For each primary framework (iso-potential compartment, synapse population, projection pattern, etc.), there is a single class in the model description tree which defines the parameters of a model and handles its interaction with other items in the model description if necessary. In most cases the system generates a default user interface for these objects, but for some of them there is also a dedicated component in the user interface branch that provides more convenient or intuitive access to the parameters.

4.1 Internal representation of models

The internal representation of models is an exact parallel to the structures described in sections 3.1 and 3.2. For each framework, there is a corresponding class definition with fields for the parameters and for any set or reference items defined for the framework. References are stored externally by the type and name of the model element they refer to. Dereferencing (finding a model component given its name), is performed by a local closest-first search, looking first among the children of an item's parents, then the descendents of its grandparents and so on. This allows models of the same type and name to exist at different places in the model hierarchy without any ambiguity as to which one is meant by a particular reference. Although it is obviously inadvisable to give distinct models the same name, one consequence of constructing complex declarative models through a user interface is that a great many objects are created implicitly by the system with default names. Since these objects only ever occur within the context in which they are created, it is unnecessary to give each one a globally unique name or a unique ID. The local dereferencing rules prevent any ambiguity.

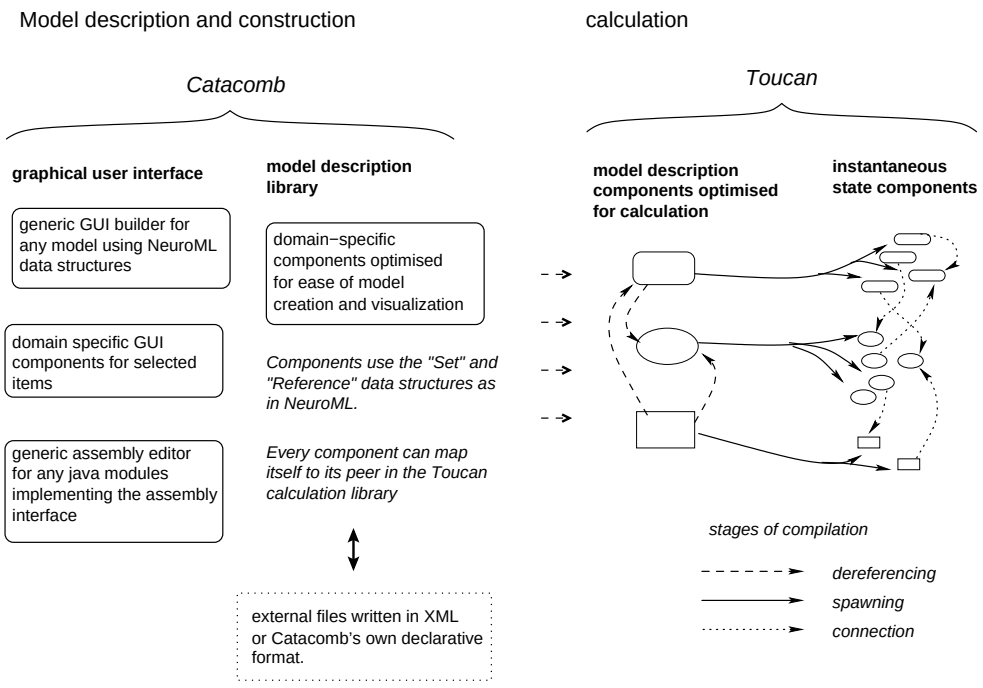

Figure 3: Catacomb2 system architecture. It is divided into two distinct parts: Catacomb itself which is concerned with the description, construction and visualization of models of biological systems; and Toucan, a numerical calculation package which is designed for memory-efficient calculation of model behavior. Toucan is a free-standing package which can be used entirely independently of the model description tools. Catacomb comprises model description components, a generic user interface system which can construct any model consistent with the data structures employed, and as large set of domain-specific user interface components for the exploration of certain types of models. The only part of the whole system which is persistent (can be saved to files and restored) is the description of the active model.

4.2 User interface

A primary objective has been to ensure that all models that the system is capable of computing should be constructible with the user interface alone, and, conversely, that the system should be capable of computing the behavior of any model which it allows to be constructed. This, rather than the more tenuous promises of inter-compatibility through XML and related technologies, has been the main motivation for using the highly restricted data structures and purely declarative model structure described above.

The user interface incorporates a dynamic interface builder which can examine the framework used by any model and produce a default set of interface components to display and change parameter values. This covers all single-valued fields. For reference fields, all possible referents within the current model are located and presented to the user in menus. These default interfaces are augmented in some cases with framework-specific interface components which provide a more intuitive or convenient way to interact with certain types of models. For example, the assemblies mentioned above are simply sets of components with references between them, but the user interface presents this as a graph where nodes can be added and connections made between them by dragging icons with the mouse.

4.3 Numerical implementation

The behavior of models is computed with a numerical modeling package called Toucan that is extensively referenced from Catacomb but which makes no backward reference to anything on the model description or user interface side. It can therefore be used on its own from programs, scripts, or from other modeling tools. Just as Catacomb contains model description frameworks optimized for efficient and convenient model construction, Toucan contains corresponding model description frameworks optimized for efficient computation. The first step when a model is run therefore involves mapping the model description into the corresponding Toucan components. These components are themselves only a model description: there is then a de-referencing stage where each component finds the Toucan versions of the other items it needs, and finally a "compilation" stage in which each Toucan description component spawns and connects up one or more state components which actually contain the state variables and numerical code of the model.

The process of building an executable system from the model description is illustrated schematically in figure 3 and with a more concrete example in figure 4. It is only in the compilation phase that the model is expanded from the minimally redundant version using references wherever possible, to the complete state space of the system. Even at this stage, the supplied parameters themselves are never duplicated: a model description component

in Catacomb which is referred to by other components maps into a corresponding structure in Toucan, still with multiple incoming references. After compilation there is only one Toucan representation of the parameters, but it delivers a new instance of the state variables for each incoming reference. These spawned state objects keep a reference back to the parameter object to compute their evolution but otherwise make reference only to other state objects to which they are connected in the compilation process. This structure is a way to minimize unnecessary memory use since the state variables frequently occupy very much less space than the parameters. For example, a four-state ion channel may easily have twenty or thirty parameters, but to represent its state only takes a single integer (which state it is in) when modeled stochastically, or a four element vector in the ensemble limit.

Toucan uses a combination of event-driven and fixed-timestep calculations, with each delaying component managing its own event queue. At each step in the calculation, the top-level object instructs all its children to advance by one timestep. They do the same to their children and so on down to the elementary components. Components fetch information from each other as necessary to update their state and any events which are generated are propagated and acted on immediately. This may typically involve being queued if there is a delay involved, changing state variables of another component, or, in some cases, setting off a long chain of event-driven updates.

This approach to numerics is quite different from that adopted in Neuron(Hines and Carnevale, 2001) which casts the whole model into a system of algebraic differential equations to be solved by a sophisticated differential equation package. Such packages have the advantage of being very reliable, able to achieve high accuracy, and providing efficiency improvements such as adaptive timestepping. One disadvantage is that there is frequently domain specific knowledge about the model which gets lost in the mapping to algebraic differential equations, and therefore cannot be used to improve efficiency. This occurs, for example, where the differential equation package must re-deduce the sparse matrix structure (Hines, private communication) of a branching cell even though it was known, by construction, to conform to the Hines numbering system(Hines, 1984). Perhaps the main reason for the different approach, however, is that the current system is easier to implement: the export of Catacomb models to a Neuron-like solution package, or even to Neuron itself would be useful in many contexts.

5 Software components

Most of the components available in Catacomb2 (version 2.034) are shown in tables 2 through 7. For all except the connectors in table 2, the small image shows how the component is represented in the user interface, and

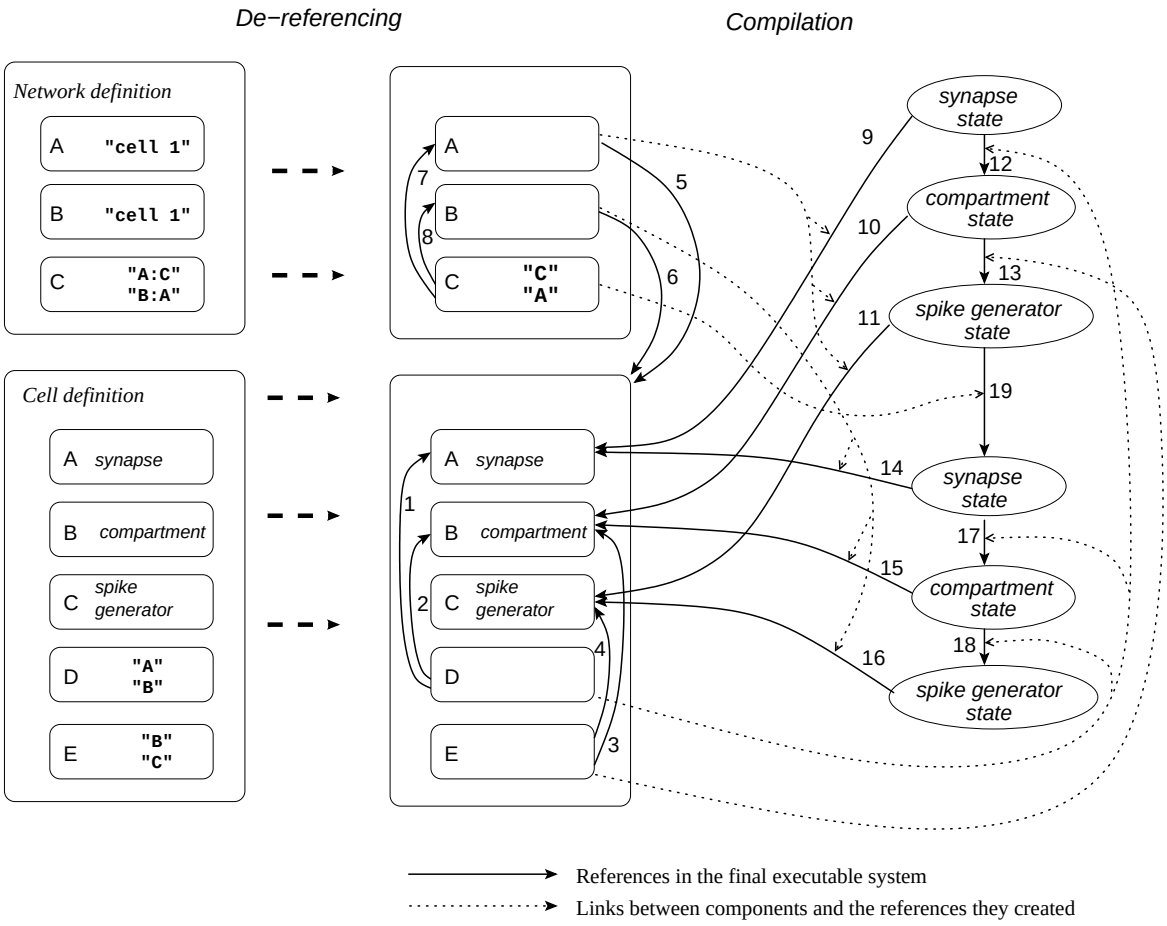

Figure 4: Stages in the preparation of an executable Toucan model instance. Solid arrows show direct references in the final data structure; dotted ones indicate which component is responsible for creating each reference. Numbers show the order in which references are created. There is a network of two cells of the same type (A and B at the top) with a single connection between them (C). The cell model has three components with two connections. First the references by name are looked up, giving the solid arrows in the middle section. Then the network creates an executable instance. The first cell reference calls the cell description to create a state instance. Each of the first three components in the cell description spawns a state instance; the last two make connections between them. This process is repeated when the second cell reference in the network calls for an executable instance. Finally the third component of the network, the connection between the two cells, finds and connects the appropriate subcomponents of the two cell instances.

Icon	Name	Function
red, curved	`spikeCable`	indexed event propagation
blue, curved	`vectorCable`	real-valued vectors (read on demand)
green, straight	`redirectionArrow`	insertion or redirection, such as putting channels in a membrane or a rat in a maze
yellow, curved	`cord`	physical or logical co-location
black, pipe	`objectPipe`	run-time movement of complex structures
black, bifurcating	`spikeProjection`	variety of projection patterns
brown, curved	`conductanceProjection`	continuous one-way connections

Table 2: Connectors and network projection patterns

can be used to read the diagrams presented in section nine and elsewhere. The filled squares around the edges represent the ports on the component and are color-coded according to the type of information which they accept or provide. These correspond to markers on the connectors themselves and the user interface will only allow markers of matching modalities to be connected (spike senders to spike receivers etc). Components are grouped into tables according to their primary use which can be building cells, generating and recording signals, specifying logical operations, constructing animats or building networks. The connectors can be used between components of any groups. The names of components are shown as they appear in the user interface. Each name is a concatenation of words where the first is in lower case, but subsequent words are capitalized. This is in line with the most common naming convention for Java in which class names are concatenations of capitalized words, and instances use the same name but with the first letter changed to lower case.

In addition to the items shown in the tables, for most sets the user interface supplies an empty box called a `...CaptureBox` (where the prefix could be `neuron`, `device`, `network`, or `animat`). These are not persistent items in a model description, but serves to construct assemblies of the corresponding type which can then be used as components in other models. The box is dragged around the components to be grouped and then captures them within an assembly. This can be done to part of a larger model: all the connections are rerouted through the assembly as required so the model behavior remains unchanged. Although there are several different flavors of capture box, they all work the same way. Indeed, the difference is purely conventional. Internally, the treatment of an assembly is independent of its type or name except that assemblies captured by the `neuronCapturebox` are

Icon	Name	Function
	`integratorCompartment`	resettable leaky integrator; accepts membrane conductances; exports potential
	`thresholdSpikeGenerator`	reads a floating value and generates events when it exceeds a threshold
	`spikeResponseFunction`	applies a conductance with a biexponential profile in response to events
	`cumulativeResponseFunction`	as above, but not reset after events
	`channelPopulation`	kinetic scheme channel models for insertion into a membrane
	`synapsePopulation`	creates and inserts synapses as events arrive on different afferents
	`conductanceClamp`	converts a vector signal to a conductance or reversal potential applied to a membrane

Table 3: Sub-cellular components

put in a set of neurons, whereas those captured by the `deviceCaptureBox` are put in a set of devices. This becomes significant when they are reused: a component expecting a neuron will only be offered items out of the set of neurons; one expecting a device will only be offered items form the set of devices. As yet, the only components which do make such references are the population sockets in table seven and discussed at the end of this section.

The components for making cell models are shown in table 3. As remarked in section 2, there is no single-cell component here: all single-cell models are instances of hierarchical assemblies. The core of any cell model is an `integratorCompartment` which represents a closed iso-potential area of membrane. It has an internal potential and only accepts inputs in the form of conductances and reversal potentials. The `thresholdSpikeGenerator` reads a continuous value and emits discrete events whenever the value crosses a specified threshold. All the other cell components are conductance providers which can be inserted into a membrane and provide conductance or driving potential changes in response to the arrival of discrete events or to the evolution of their own internal state.

Table four shows general purpose lab components for generating and recording signals and for directing the progress of a calculation. The components in this set are distinguished from other groups because they are unlikely to be used within compound assemblies and because most of them

Icon	Name	Function
	`signalGenerator`	generate a variety of signals including noise and predefined waveforms
	`spikeGenerator`	generate regular or Poisson events on multiple channels
	`spikeButton`	send events into a running calculation
	`vectorSwitch`	re-route data in a running model
	`vectorRecorder`	periodically read and record vectors for display
	`currentClamp`	read a membrane potential and adjust its own internal potential to impose a given current
	`spikePositionRecorder`	record and display positions at which spikes occur
	`spikeRecorder`	read source vector whenever spikes arrive; variety of display options

Table 4: Lab components

require some form of access to the internal state of the model after it has been compiled. That is, they are part of the model description, but they are also used to provide access points to the running model. Thus, for example, the `vectorRecorder` specifies a recording interval as part of the model description, but also receives and displays the recorded data as the model is running. Likewise, the `spikeButton` and `vectorSwitch` allow events to be sent into the compiled model as it is running. This contravenes the purely declarative modeling principle, which can be rescued by, for example, driving the switch programmatically from a pre-defined event sequence.

At present the logic components form the largest set, so only a selection of the most common ones is presented here (table 5). They cover a wide variety of data processing and signal processing functions, all acting on discrete events or vector-valued data. Components are intended to implement relatively simple functionality but in a few cases they cover complex algorithms such as the feature discretizer (see section 9 for examples of its use) or path linearizer. Such components represent a relatively large chunk of a final model, and can be seen as a compromise between explicit description of a model in simple units, and rapidly getting a model to work when it is

Icon	Name	Function
	`concatenator`	join spikes or vectors into a single stream
	`spikeSplitter`	divide spikes with different indices into distinct channels
	`counter`	deliver one output event for every `n` input events
	`constantVector`	export a fixed vector
	`jythonScript`	interface to logic implemented in Jython
	`featureDiscretizer`	read a vector and allocate new IDs when it is sufficiently different from previous values
	`eventSequence`	deliver a user-defined sequence of events
	`directionCalculator`	evaluate an arctangent function for the vector between two points
	`vectorRenormalizer`	change the range of a vector
	`delayBuffer`	hold events for a specified time before passing them on

Table 5: Logic components

not otherwise clear how to achieve the desired function.

Table six shows the components available for modeling animats and virtual environments. There is no sharp distinction between the two — an automated maze in which food supplies are replenished in response to lever presses is very close to an animat which presses levers to receive food. The components include a variety of sensors and effectors which, unlike all other components, communicate without any explicit connection in the model description. Instead, they rely on a concept of physical space which is provided by incorporating a scale bar item in a maze or attaching a spatial location object to the cluster of sensors and effectors which constitutes an animat. Internally, these two components register all the items they are associated with as having the possibility of long range interactions. At the compilation stage, real connections are established according to the properties of the components (sound sensors to sound sources etc).

Finally, table seven shows components which currently belong in sets of their own. In particular, it includes the `neuronSocket` which covers the whole field of populations of cells. The term `socket` indicates that it does

Icon	Name	Function
	`dispenser`	interact with the `ingester` by position, delivering events when it is emptied and refilling when it receives events
	`lever`	generates events when pressed
	`lightSource`	illuminate physical environment
	`soundSource`	generate (virtual) sound in response to events
	`camera`	provide visual display of physical objects
	`insertionSite`	target of a redirection or insertion in a physical environment
	`trajectorySite`	point in a predefined trajectory
	`soundSensor`	senses sounds
	`leverPresser`	senses and activates pressable objects
	`ingester`	senses and activates dispensers
	`whiskers`	proximity sensors to physical objects
	`spatialLocation`	physical position of an animat
	`insertionSwitch`	redirect insertions (move objects) in a running model
	`scaleBar`	set true size of physical objects
	`wall`	configurable boundary for constructing mazes

Table 6: Animats and virtual environments

Icon	Name	Function
	`neuronSocket`	population of one or more neurons
	`synapseRecorder`	recorder and display of synaptic weights
	`javaModuleSocket`	access to external resources supplied as java class files
	`eventSounder`	interface to midi or sampled sound synthesizer for sonifying events

Table 7: Network construction and miscellaneous components

not, of itself, specify any connections: these are all derived from the particular component it refers to. It does, however specify the layout of the population, as a grid or a single row. No distinction is made between items which represent a single cell, and those for a whole population. Internally they are both regarded as populations. All the ports within a neuron assembly which have been marked as externally visible also appear as ports on the population and are accessible for inputs from spike or vector connectors as well as the more specific spike and conductance projection components.

6 Model execution and meta-modeling

The discussion so far has centered on representing the static properties of a biological system, and the ability to compute its evolution over a short interval (seconds or minutes). Most scientific applications also require ways to study other properties of a model, such as its correspondence to existing data or the sensitivity of particular behaviors with respect to uncertain parameters. Frequently this information is acquired by tweaking and re-running a model and the whole process is never formally laid down. It involves developing a feel for the model construction process and constitutes a sizable and inaccessible body of knowledge. As such, it is a considerable barrier to new users of many modeling tools. As far as possible, therefore, tools should allow the user not only to compose the biological model description but also to compose, in an equally declarative form, the description of the data and processes which gave rise to the model, how it was tested, and how well it can be expected to perform. This "meta-modeling" should be described within the same context as the model itself, with the distinction that the subject is now a model, not a biological system. In the case that the meta-model represents a procedure for generating models from experimental data, it should ideally be sufficiently detailed that the model can be recreated from the source data alone without any of the procedural component normally present in complex modeling tasks.

6.1 Sensitivity Analysis

One of the most accessible forms of meta-modeling is to assess how sensitive the results of a model are to uncertainties in the parameters. In Catacomb2 this is accomplished with circuitry like that in figure 5. The `pararemterExposer` gives access to parameters to be varied, in this case the transition rates of an ion channel model. The `differenceCalculator` evaluates how far apart the behaviors of two models are in some predetermined space. The `sensitivityAnalyzer` first runs the model once with the `fixedStepRunner` to construct the result set to which subsequent runs will be compared. It then changes the selected parameters over specified ranges, re-running the model each time. The output of the `differenceCalculator` is recorded for each run, giving a simple view of how the measured behavior varies with respect to a single parameter.

This example illustrates many of the concepts that are needed for more subtle forms of meta-modeling. As elsewhere, the objective is to eliminate procedural definitions of the algorithm. Although it is described above as a sequence of operations, which matches the way in which it is calculated, the order of operations is unimportant. The nth point in the results is a function of the derived properties of the model at two different parameter values, independent of any of the other points in the results. The model description therefore makes no mention of this order, opening up the possibility, for example, that all points could be calculated in parallel. This possibility of processing model definitions in new and unanticipated ways is one of the key advantages of eliminating spurious procedural information and using only declarative structures.

6.2 Parameter optimization

Software libraries for parameter optimization are often large and sophisticated (see, e.g., Portlib(Fox et al., 1978) or the survey of methods for channel density optimization by Vanier and Bower (Vanier and Bower, 1999)). It is not the intention here to represent the algorithmic contents of these declaratively, only the way in which they are applied. The main aim is to allow the transfer of an optimization procedure between users in the same way that models can be exchanged, without requiring any extra knowledge about the optimization should be performed.

The description of an optimization should therefore include the names and initial values of parameters which are optimized, the error measures used to test convergence, the name and source of the algorithm used and any flags or parameters required by the particular implementation of the algorithm. Most of this is possible with the same components as are used for the sensitivity analysis, with the exception of specifying a choice of optimization algorithm. Presently only the conjugate gradient method(Press

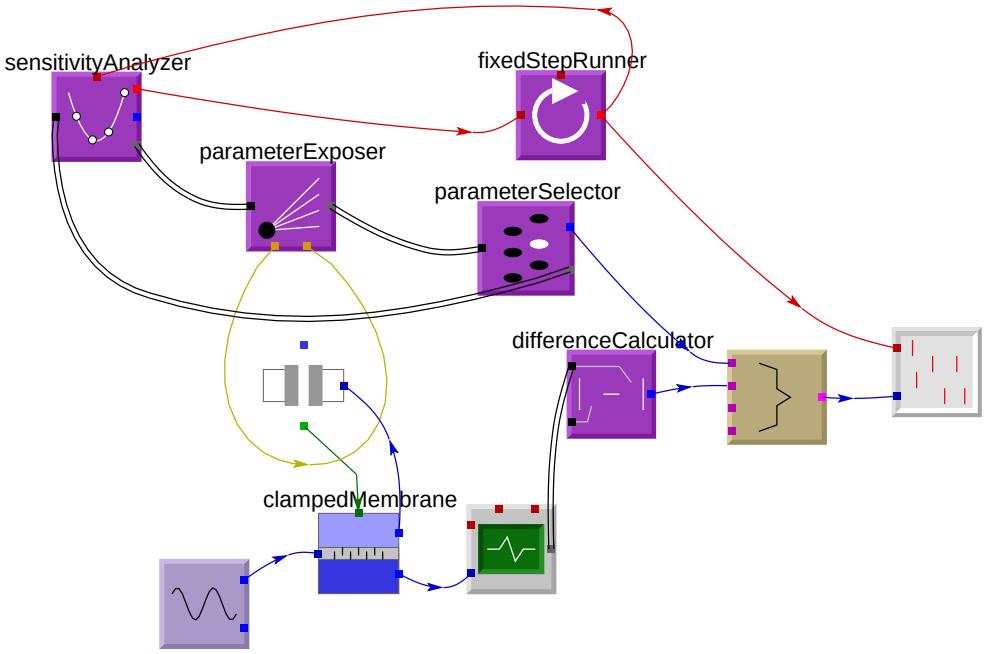

Figure 5: Meta-model for computing parameter sensitivity. The model under study is a kinetic scheme ion channel in a voltage-clamped membrane. The membrane potential is driven through a predefined profile with the signal generator at the bottom left, and the resulting current profile forms the argument of the difference calculator. The four components at the top define the structure of the analysis. The `parameterExposer` can be directed at any part of a model, and makes any parameters it finds accessible to other components. The `parameterSelector` picks one of the parameters as an argument for the analysis. This is then varied by the `sensitivityAnalyszer` which reruns the model for each value in a selected range. The results are read from the `differenceCalculator` for display by the unit at the far right.

et al., 1993) is available within Catacomb2 itself through the `cGOptimizer` component which uses an independent NeuroML compatible optimization library. Other algorithms can be implemented via the `optimizerScript` component, which works in very much the same way as the Jython script extension mechanism, but with hooks and call-backs appropriate to general optimization problems.

6.3 Robust model development and database access

A criticism of many cell and network models is that they have been tweaked by the model builder in order to show particular behaviors, and have thereby lost much of their predictive power. The modeling approach advocated in section one, and employed in the examples presented here, sidesteps this issue by focusing on design, in the way that engineers design circuits, rather than pretending to do bottom-up prediction of integrative properties. That is, the models can be shamelessly tweaked, because there is an independent test: do they perform the task for which they have been designed? Sensitivity analysis can then be used to check how much tweaking has been required — does the model still perform successfully when parameters are changed by 1%, 5% 10% or more? Nevertheless, it it is clearly of interest to incorporate empirical data where possible, and this remains a major challenge to systematic and robust model creation. Sensitivity analysis and parameter optimization are part of the solution, but a great deal remains to be done before a procedure can be laid down, the source data provided, and the software left to come up with the most plausible models. For the output to be useful, it should also include parameter confidence estimates and error tolerances. At present, this is not possible within Catacomb2, but the modular design does allow models and numerical implementation components to be used from external scripting languages such as MATLAB or from within independent robust modeling systems.

7 Publication, dissemination and archiving

Unlike empirical or analytical studies, the results of modeling work are very ill-suited to conventional text-only publishing methods. Complete model descriptions are often large, with many parameter values that are necessary within the context of the model, but that are not intrinsically interesting in the way experimental measurements of real biological quantities are. But the greatest difference is that whereas in an experimental study, anyone can readily get hold of the type of tissue claimed to give rise to the observed data, (even though the experiments may be very hard to repeat), in a modeling study, the raw material —the model itself— is rarely made available, except in a highly condensed form from which it would often take weeks or months to re-create.

This situation has been allowed to persist because of the recognition, first, that most users would not know what to do with it even if they had access to the model and, second, that the authors would probably be willing to grant such access if requested. However, one goal of any new modeling tool must be to make model sharing as straightforward as possible, both for the benefit of users, and because publishers are unlikely to carry on tolerating the current situation indefinitely. The transition from highly personal model implementations which are rarely, if ever, used by others, to widely and routinely accessible models is likely to be a gradual process involving sociological changes, and facilitated by incremental technological shifts. The sociological issues are clearly not insurmountable, since they have been effectively overcome in many areas, including public-domain software engineering(Raymond and Young, 2001) where, for example, the source code of NeuroML or Catacomb itself is normally freely available on the web within days of being written.

The first step towards easily shared models is the use of purely declarative structures instead of scripts. This makes it easy for readers to see what is in a model, and easy for machines to search and catalog them. Declarative model descriptions are also meaningful in the absence of any implementation, unlike scripts which depend on a particular interpreter. Other features likely to help the dissemination of models include single-file storage, documentation and annotation schemes, mechanisms for providing tutorials and, in some cases, making models directly accessible through a web browser (cf. *www.virtualcell.com*).

7.1 Self-contained model descriptions

One immediate disincentives to using another group's models is the need to install or update a whole range of ancillary libraries and software components before the model can be run. This problem has considerably eased with the development of cross-platform standards such as Java, and the relaxation of constraints on disk and memory usage. Although Catacomb2 is able to use model components from external sources, by default, whenever a model or subcomponent of a model is saved, all references are followed and everything necessary to reconstruct the model is put in a single file. The resulting model description is therefore completely self-contained. Anyone with that file who has installed the software should be able to run the model.

This policy does have a number of drawbacks, in particular, the duplication of model subcomponents. For example, many cell models may use exactly the same model for particular ion channels, but every file will contain its own copy of the channel description. Loading multiple cell models will also then load multiple models of the same ion channel. Because of the internal storage conventions (that ion channels should all go in a top level list and have unique names) this problem is easily spotted, and, by comparing

serializations (turning the model description into a string of characters), the system can work out whether two models are in fact the same, and throw out redundant copies. In practice, this policy works fairly well, despite seeming rather delicate and vulnerable to serialization conventions. Nevertheless, it seems likely that in future some system involving the assignment of unique model identifiers which change with every modification as in SBML(Hucka et al., 2001) or the Modeler's Workspace(Forss et al., 1999), will be required.

7.2 Internal and external documentation

Early versions of Catacomb(Cannon, 2001a) experimented with the idea of internal documentation for all model components. That is, one of the parameters of every object was a text document in which the creator of the model could provide whatever information they saw fit. This text would then be an inextricable part of the model. The immediate advantage is that anyone who has the model also has the documentation. The disadvantage is that the information provided this way is unstructured. Adding further fields such as "author", "date", "keywords" to every object would be very wasteful since only a small fraction of model components ever need documenting separately.

This option has now been replaced by a more structured external documentation mechanism based on the Axiope (*www.axiope.org*) non-curated distributed database project(Cannon et al., 2002). This allows complete models, or subcomponents such as individual ion channel models, to be documented according to external or user-generated templates. The information can be exported as both a collection of web pages and as XML files, constituting a self-contained website. This can be kept locally as a data-management system, or exposed on the web as part of a distributed model database. In the latter case the software can request one or more Axiope servers to visit and catalog the site. They then provide collective indexes and search services where the provided models appear among similar items from other participating sites. This approach to model documentation and publication is still very much in its infancy but is hoped to overcome many of the hurdles involved in submitting models to centralized databases. With the Axiope scheme, it is made very clear that authors retain complete physical and intellectual control over their work, while still making it accessible through collective access points.

7.3 Interactive tutorials

On-line help is provided as a set of local web pages which can be viewed from any standard web browser, or from a simplified built-in HTML browser. The latter has the advantage that it also understands Catacomb-specific links built into the web pages which can be used to issue commands to

the interpreter. These may typically be used to load example model files, open display windows for particular subcomponents, or perform actions such as running a model. The present implementation of such links is to use a pseudo-URL which includes a command name and optional arguments. Currently commands include "show", "run", "load", "press" and "set". What is required for subsequent arguments depends on the action being applied, but in most cases there is a single argument which is the fully qualified name of a model component.

8 Extensibility and interaction with other modeling systems

Software capable of computing the behavior of models of biological systems, is necessarily complex, and there is a continual drive to be able to accommodate larger and more complex models. It is therefore essential to find mechanisms for widespread collaborative work both in building model descriptions, and in developing software implementations. This problem is shared with many parts of the software industry and solutions have been under development for many years. The situation in neuroscience differs from many commercial applications, however, because of the complete lack of hierarchical organization, and the consequent need for structures which correctly apportion academic credit and intellectual property rights to all participants.

Perhaps the closest parallel to academic software may be found in the free software community(Raymond and Young, 2001) which has developed working structures based largely on conventions. The history of free software would suggest that widely accepted solutions are indeed likely to emerge, given time, but that there may be substantial duplication of effort in the process. In the meantime, Catacomb2 provides a number of mechanisms for users to extend its capabilities according to their needs, ranging from the use of its built in model definition structure, to cutting parts out for use in quite separate software packages.

8.1 Component grouping

A simple, yet versatile, way of making new components that show functionality not found in the standard set of frameworks is provided by the component grouping mechanism. As discussed in section three, primitive components can be connected together and encapsulated as a single item for future use. Since the primitive components can come from any of a wide variety of logical and biological components, as well as other grouped components, it is possible to implement models this way that bear no relation to the problem domains of the constituents. A very common case of extension

by component grouping, is the construction of single-cell models, which are nowhere represented in the elementary components, as discussed in section 2.

8.2 Scripting

The `jythonScript` component mentioned in table 5 provides an interface from a model to arbitrary logic implemented as a script written in Jython (*www.jython.org*), the Java flavor of Python (*www.python.org*). Any number of such scripts can be incorporated in a model. They can communicate with other components through vectors or discrete events. This approach to model extension is provided for flexibility and prototyping but is deprecated as part of any permanent model description because it breaks the convention about purely declarative model descriptions. The internal working of a script is generally opaque to archiving systems or search engines, and difficult for other users to understand. Functionality that is initially implemented in scripts should be migrated to new model frameworks if it proves to be widely used.

8.3 Writing new frameworks

The dynamic nature of Java class loading and instantiation allows Catacomb to be very liberal about what constitutes a model description framework. Indeed, any public class with a default constructor can be used. Any fields that are declared public and are within the set used in the software (section 3.1) will be made accessible through the user interface. Furthermore, if models use Catacomb's own set and reference objects, or those declared in the NeuroML development kit, then the full functionality of the model description framework will be available on the imported class. Adding a framework for the model description does not involve adding any numerical code for implementing a model. Models that use external class definitions can still be created, modified stored and retrieved. In general, adding numerical code so that a model could also be run would involve writing corresponding classes in the model implementation package. This type of extension, however, is very specific to one software package and therefore wastes much of the generality afforded by the technology. Instead of requiring package-specific extensions, Catacomb2 also supports a system-neutral extension mechanism at the code level that is being developed in the wider context of NeuroML as discussed in the next section.

8.4 Interfaces for runtime interaction

One of the most attractive ways to enable distinct software packages to work together when neither is strictly dependent on the other, is to establish a small area of common ground in some neutral space between them through

which all communications should pass. Each package must then know about the common ground, but can be insulated from any changes in the other package. In software terms, this common ground should be part of the global namespace which is not associated with either package. Much software written in Java now uses globally unique names dependent on ownership of particular domains. For example, the full names of classes developed by Sun Microsystems often begin *com.sun.* as in *com.sun.j3d.loaders.Loader*. By convention, this name is unique to Sun because they own the domain *sun.com*.

One natural piece of common ground for interaction between modeling packages is the *neuroml.org* domain which is already extensively used by various packages because of the facilities it provides for working with XML files. However, before sufficient agreement is reached as to exactly what such middle ground should contain, Catacomb uses an alternative neutral area under *compneuro.org* which is co-hosted with *neuroinf.org*. Amongst other things, this site holds documentation and interfaces which can be used to build Catacomb-compatible components without needing to read or use any of the source code. This protects developers from changes that may take place to the software itself, and guarantees that their work is genuinely system independent. It also allows Catacomb developers to modify and re-factor the internal architecture at will, without the risk of breaking dependent code because they only need to ensure that the interface definitions remain unchanged. At present, these interfaces cover everything needed for compatibility with the event send/receive and vector read/provide mechanisms. That is, provided objects from an external package implement the right interfaces from the *org.compneuro* package then it is possible to instantiate them through the user interface, connect them to other components with the spike and vector connections, and run the resulting model.

9 Case study: modeling spatial navigation

Much of the development effort behind Catacomb to date has been directed at building increasingly effective models of spatial navigation and food seeking behavior in order to explore the role of theta rhythm in hippocampal function. The development and structure of these models has been described elsewhere(Hasselmo et al., 2002a,b, 2000): here the focus is on how they have been implemented and refined using Catacomb2. First a brief overview is given of the problems under study and the hypotheses to be explored. Then the structure of the model is examined with reference to two key design issues: how to separate encoding and retrieval within the network so they do not interfere; and how to make the virtual rat move towards the food. The full model, as it appears on first loading in Catacomb is shown in figure 6: only a few aspects will be considered here.

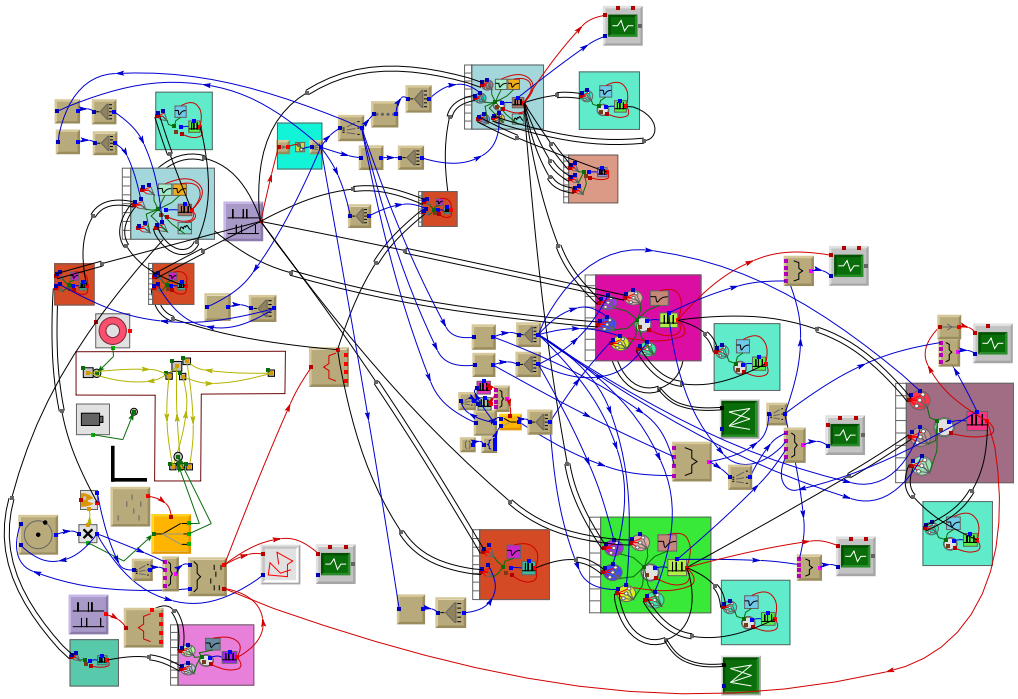

Figure 6: T-maze, virtual rat and network as it appears in Catacomb2. The symbols correspond to the items in tables 3—7. Larger boxes with a ladder down the left hand side are populations of cells, and double lines represent 1-1 or all-all projection patterns between populations.

9.1 A mechanism for environment learning

Behavioral data suggest a role for parahippocampal regions in memory guided behavior(Hagan et al., 1992) and the patterns of activity in particular regions have been extensively recorded during spatial navigation tasks(Yound et al., 1997; Suzuki et al., 1997; Frank et al., 2000). Cellular physiological properties have also been isolated which could be important for memory function(Klink and Alonso, 1997; Fransen et al., 2002). The model considered here is based on the idea of place fields in which patterns of neural activity are related to specific locations in the environment(O'Keefe and Dostrovsky, 1997; McNaughton et al., 1983; Muller et al., 1987) A number of models have addressed both the formation of place fields(Sharp, 1991; Kali and Dayan, 2000) and the conversion of place information to turning direction(Sharp et al., 1996; Burgess and O'Keefe, 1996; Redish and Touretzky, 1998). Instead of this, the current example focuses on the learning and use of a representation of the environment. The creation of place fields is of course necessary, but it is handled here by a single algorithmic component.

The idea to be implemented is that as the rat moves around its environment place fields are created which map an area in the environment to spiking activity in a group of cells. When the rat moves between place fields it learns an adjacency association between the corresponding groups of cells. Eventually, when it reaches a food reward, it also learns an association between the presence of food and the place field in which the food was found. Now, when the rat is placed in any part of the explored environment, a mechanism is be required for it to find the food. This involves a diffusion-like process where the desire for food excites the pattern of the place field corresponding to the location of the food. The learned adjacency relations are then used to propagate the activity into patterns for place fields one step removed from the food. The process is continued until the diffusing signal reaches a place field which is adjacent to the rat's current place field. The rat then moves into this place field, and so on until it gets to the food. In effect, there is a signal diffusing back through its internal representation of the environment and it heads to the point where this signal meets a short range signal moving out from its current location.

Just as with the design of software, although this scenario may seem plausible, it is next to impossible to find its flaws or weaknesses without actually implementing it. For example, one problem with the above which may not be immediately obvious is that the association learning mechanism is still active while recall is taking place, causing the strengths of learned associations to change in the absence of new information, and rapidly corrupting the internal representation of the environment. But this soon becomes apparent in the process of building a model to perform the task. The first step is to decide which features should be implemented with biologically plausible mechanisms, and which ones should be done with higher level components.

Because the main interest is in the learning and use of representations of the environment, motor control and the formation of place fields will be handled by logical components as described in the next subsection. Populations of spiking cells will be used for the hippocampal regions as described in sections 9.2 to 9.5.

9.2 A framework for studying interaction between network models and a simulated environment

The components used to represent the environment and to make contact between the virtual rat and the hippocampal network are shown in figure 7. The meanings of the icons can be read with the aid of tables 2 to 7. See the figure caption for a detailed description of their internal connections. There are two areas of interaction between this part of the model and the rest: definition of the experimental protocol; and closing the sensory to motor loop through the network model. In this case, the experimental protocol is very simple, defined by a sequence of events controlling the `insertionSwitch` which moves the animat around. More complex protocols might use the ports on the dispenser and other devices to encode, for example food delivery in response to lever presses. The access points that make contact with the network are the ports on the ingester and the feature discretizer. The ingester sends periodic events when the animat is within range of food, and can be instructed to eat by sending events to its input port. The feature discretizer acts as a single-component solution to the problem of creating place fields, which is not the focus of the present study. It keeps a list of known feature vectors, and continually reads the position of the rat. When the rat is far enough from any previous features, it creates a new one and adds it to the list. The index of the current feature is the main input to the rest of the model. The feature discretizer also works in reverse, taking a feature index provided by the hippocampal network, and converting it back to a feature vector. This is combined with the current position of the rat in the direction calculator in order to produce a command signal for the motor system.

9.3 Buffering sensory input

The algorithm outlined above requires information about the environment, essentially adjacency relations among place fields, to be coded in the synaptic weights of projections between patterns corresponding to different fields. For convenience, the present model uses single element patterns, so this involves strengthening a single selected recurrent connection within a fully connected network of place cells. The source of the adjacency information is the environment itself — the rat moving from one place field to the next provides the information that they are adjacent. But the rat moves between

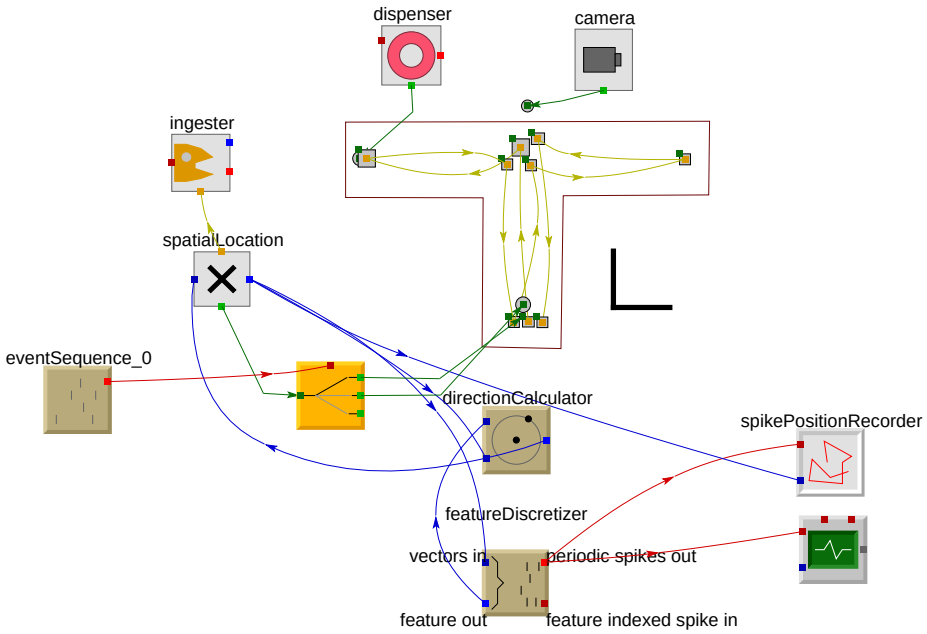

Figure 7: Physical and logical components implementing a virtual rat and its interaction with a maze. The animat itself is composed of the two components to the upper left which confer on it the property of having a physical position and size relative to the walls of the T-maze ("spatial location"), and the ability to eat from the dispenser ("ingester"). The straight arrows from the spatial location component indicates where it is placed in the maze. In this case, the arrow goes through a switch, allowing the animat to be moved between predefined tracks and free running mode according to the event sequence object on the left. The position output goes to a feature discretizer component which creates place fields as required when the rat moves around. The output of the feature discretizer forms the input to the biologically motivated parts of the model.

fields relatively slowly compared to the pre-to-post synaptic spike intervals of well under 100ms over which synaptic potentiation is experimentally observed to occur(Markram et al., 1997; Bi and Poo, 1998). There is therefore a need for some sort of buffering to provide persistent activity in one place field which overlaps with activity in the next field that the rat enters.

There is extensive physiological data suggesting that persistent action potential firing can occur in certain cells of the entorhinal cortex in response to activation of muscarinic receptors which results in prolonged calcium sensitive cation currents(Alonso and Klink, 1993; Klink and Alonso, 1997). This has also been implemented in detailed biophysical models(Hasselmo et al., 2000; Fransen et al., 2002). In the current model, persistent firing is achieved with a slow after-depolarization, as in part B or the examples in figure 1. Once a cell in the buffer population fires, it initiates a slow after-depolarization (ADP) which, in the absence of other input, is sufficient to bring it back to threshold in about 50 ms. Connections between cells, and through an inhibitory network allow for this persistent firing to be suppressed in the presence of other activity.

9.4 Using theta oscillations to separate encoding and retrieval

The next step in implementing the outline design, is to introduce the synapses in which the adjacency relations are to be encoded. The buffering network provides temporally overlapping activity patterns so it is sufficient to use a one-one projection to transfer these to a population of cells with recurrent connections which implement a long term potentiation rule. The parameters of synapse population components can be set to implement a wide range of spike timing dependent plasticity laws. The possible features are shown in table 8. First, the spiking output of the cell must be connected to the backward propagating spike port on the synapse population. This is necessary because the normal insertion connection between a synapse population and a membrane works in only one direction: the synapses affect the membrane conductance, but the membrane has no effect on the synapses. Once a synapse is receiving both afferent and back-propagating spike information, it is sufficient to specify the weight modification profile, normally by linear interpolation for the pre-to-post or post-to-pre time difference in a set of points.

Using recurrent connections with an LTP window of 40ms, the network readily learns the adjacency relations as increased synaptic weights, and the properties of the buffering population prevent any learning of second-neighbor relations. However, problems arise when the model is extended to make use of the weights, because the LTP mechanism makes no distinction between firing which results from supplied information and firing which results from internal recall. The encoded information risks being swamped by

spurious connections learned during recall, which will typically involve more network activity within the LTP window than during encoding. In software terms, some sort of clocking or gating mechanism seems to be necessary to switch between the two modes. Possible biologically-based solutions can be found in a number of experimental studies which have tested the relationship between theta oscillations (4to 12 Hz) in the hippocampus and the induction of long term potentiation(Pavlides et al., 1998; Holscher et al., 1997; Orr et al., 2001; Wyble et al., 2000). These experiments all show the strongest long-term potentiation at the peak of the theta cycle, which corresponds to the weakest synaptic transmission. This is exactly the function required here: when the information is internal, propagated by strong synaptic transmission, there should be no potentiation, but when the synaptic transmission is weak, indicating that spiking activity is driven externally, then the synapses should potentiate. Although it would be possible to implement this mechanism with biophysical units, Catacomb2 also allows direct modulation of plasticity and transmission at synapses as described in table 8. Using this mechanism it is sufficient to attach a signal generator with the right waveform directly to the synaptic transmission port, and for the same signal to be sent to the plasticity modulation port after passing it through a delay buffer component which induces a half-cycle delay. This allows the network to operate in two distinct modes, performing one encoding frame and one recall frame on each theta cycle. This illustrates another feature of the modeling philosophy described in section 2: a possible mechanism is implemented rapidly with algorithmic components to see if it fulfills the desired function. Only now that it has proved to be useful is it worth looking further for biological correlates and refining the model to incorporate biophysical mechanisms such as calcium dependent gating of transmission and modulation.

9.5 Goal-directed behavior and action selection

The second design issue to be considered is how information about the goal and the current location can be used to generate motor signals for the virtual rat. As discussed in section 9.1, the feature discretizer component works in both directions, so it is sufficient for the network to send spikes of the right index (coming from the right cell in a population with place fields) into the feature discretizer, which will reconstruct the locations in the environment. The direction calculator then computes a direction for the virtual rat, which advances at a default speed in the absence of any other signals from the network.

Although the discussion of encoding and retrieval focuses on a single recurrent network, the algorithm outlined in the introduction requires several populations, each with cells corresponding to place fields in the maze. In the current model region CA3 encodes an episodic memory of paths taken

Parameters	
`E_rev (mV)`	reversal potential of applied conductance (fixed)
`G_0 (nS)`	initial conductance
`G_max (nS)`	maximal conductance
`G_ModRule`	profile of weight change in terms of pre-post timing
`riseTime (ms)`	rise time of conductance change
`fallTime (ms)`	decay time of conductance change
`saturates`	if set, the synapse behaves as though it had a limited supply of transmitter which recovers at rate `fallTime`
`saturationFactor`	fraction of available transmitter used for each event
`voltageDependent`	if set, the conductance depends on membrane potential
`Vhalf`	membrane potential for half-maximal conductance
`z`	equivalent gating charge
Ports	
afferent spike	activates a synapse according to the spike index. Synapses are only created the first time they are needed.
membrane	attaches the population to a section of membrane
plasticity modulation	if the synapse has a plasticity law, the supplied value multiplies the applied weight change
transmission modulation	scales the post-synaptic conductance whenever the synapse is activated
back-prop spike	used by plasticity laws to determine pre-post spike timing

Table 8: Summary of synapse population properties. Synapse populations are among the most complex single components in Catacomb, and mix biophysical properties with logical properties, such as direct plasticity modulation to capture a wide range of possible functions.

during exploration, learning adjacency relations between place fields in the direction of the rat's motion. During the navigation task, activation of a CA3 cell corresponding to current location then recalls known neighboring place cells through the strengthened recurrent connections. The activity of recalled neighbors propagates to the CA1 population, where the afferent input elicits a persistent sub-threshold depolarizing response.

In the ECIII population, afferent input is received from a second oscillatory buffer (hypothesized to involve pre-frontal cortex) that maintains the activity of a place cell corresponding to the location at which the food reward was discovered. This afferent input causes recall along strengthened recurrent connections that encode associations between place fields in the reverse of the direction of motion during learning. Activity therefore spreads backwards through the ECIII representation of the environment via fast synaptic responses. Just as with CA3, the cells in ECIII projects onto CA1 with synaptic weights which induce sub-threshold depolarizations.

Both signals therefore combine in CA1: neither is strong enough on its own to activate any CA1 cells, but when a cell receives input from both ECIII and CA3, then it reaches threshold and fires an action potential. The first such spike on a cycle constitutes the desired next location signal which is sent back to the virtual rat. It is important that the spread of activity is greater in ECIII than in CA3, because the next desired location selected by the network should to be adjacent to the current location, not somewhere further afield or the virtual rat would try to head for a remote location and would be stopped by the wall. This is just one example of how using the simulated environment helps motivate and structure solutions to the network design. The broader spread of activity in ECIII than in CA3 is consistent with experimental data from recordings of the entorhinal cortex(Barnes et al., 1990; Quirk et al., 1992; Frank et al., 2000) that show much larger place fields in ECIII than in CA3.

A variety of views of the final model are shown, for illustrative purposes only, in figure 8. These are all captured directly from the user interface at the point where the rat is deciding to turn left, towards the food instead of right. The most recent full model is available for download and examination along with the software at *askja.bu.edu*.

9.6 Further model development

Although discussion of the model has been based entirely on the T-maze task, the modular construction allows the same virtual rat to be inserted in other mazes and tested on other tasks. The maze building components have been designed to allow representation of a wide variety of experimental configurations, with interacting levers, lights sounds and food rewards. One direction of development is therefore to build increasingly robust virtual rat models which use the same network to perform a variety of tasks.

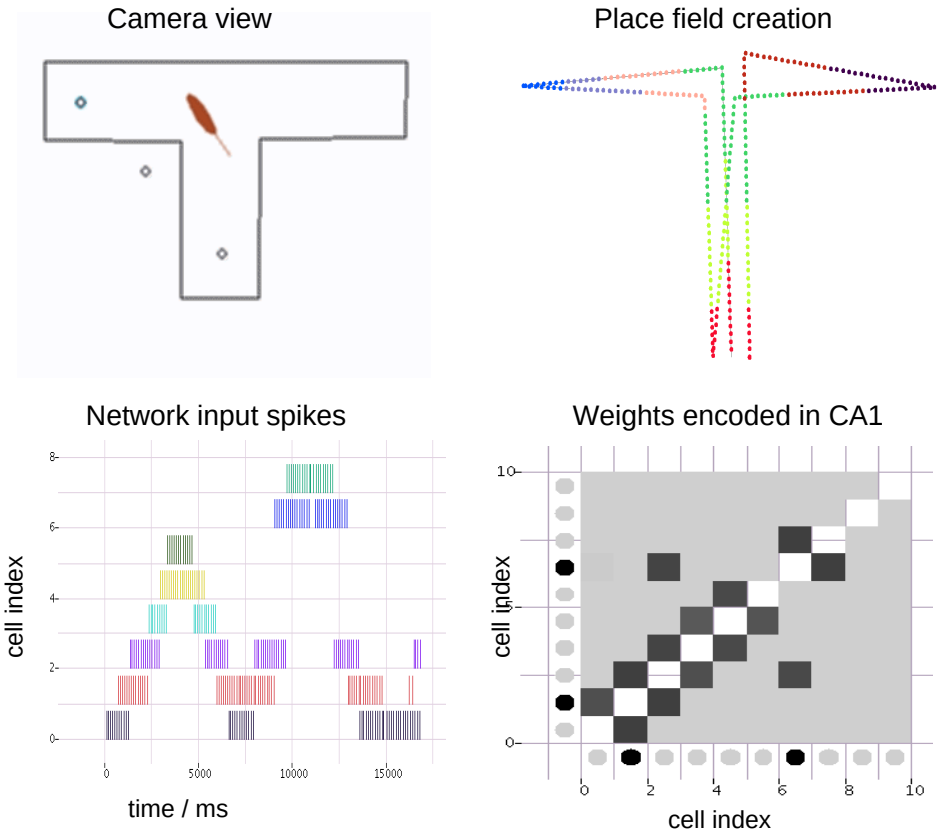

Figure 8: Illustration of visualization features during the T-maze task. All images are captured directly from the software as the model is running at the point when the rat has just chosen to turn left instead of right. The camera view shows the position of the rat in the maze as the model runs. The upper right view shows the points visited by the rat during environment learning and the place fields that were created (shades of gray) by the feature discretizer. The spiking activity of the buffer population of place cells is shown in the lower left, with the synaptic weights of recurrent connections in the CA1 population in the lower right. Darker shading indicates stronger weights: strong connections only exist between place cells for adjacent locations.

An essential component of this work is the generation of explicit predictions about spike timing in particular cell populations during navigation tasks. Comparison of these predictions with experimental results drives the next step of iterative refinement. Another direction for the model is to replace some of the algorithmic parts, such as the synaptic modulation, with more realistic biophysical components that allow direct comparison with electrophysiological or imaging data. At the same time, however, an equally fruitful direction may be to replace some of the features currently implemented by populations of cells with higher level components that capture the desired function but can be computed more efficiently. This keeps the running time down (the present model takes about a minute to learn the environment and then find the food reward on a fast PC) and allows attention to be focused on the areas of the model that are intended to be directly comparable to experimental data.

10 Discussion and future directions

Catacomb2 is a highly modular software system for modeling neuronal processes over a wide range of scales from ion channel kinetics to animal behavior. It has graphical components for setting up purely declarative model descriptions and an independent set of numerical components for evaluating and studying model behavior. The emphasis throughout is on modeling by design rather than modeling by mimicking biology as closely as possible. Features are added to the model only where it is found that particular behaviors are needed in order to achieve functional goals. This contrasts with the kitchen-sink approach where extra detail is added to a model simply because the real system is known to have such detail, irrespective of whether it is sufficiently well constrained to improve the predictive performance, or only adds more noise. In order for the design approach to work, it is essential to have a well defined task for the model to perform. It is also necessary that the model should succeed, in a loose sense, at the task in an early stage of model development in order to provide realistic feedback to internal components so that they can be iteratively improved to achieve a closer match to the system under study. This is achieved by allowing models to contain an eclectic mix of components capturing both biologically realistic and purely algorithmic behaviors.

One criticism of the design-based approach is that it provides no guarantees of arriving at the same solution to a particular problem as has been adopted by nature, either globally, in the overall structure of the model, or locally in the properties of individual components. Indeed, the use of components working at different levels of description may exacerbate the problem of the solution settling into an artificial local minimum created by the unbiological properties higher level components. As with any optimiza-

tion process, there is a trade-off to be made between the smoothness of the error function and the time it takes to evaluate the function and pick a new point in parameter space. If the error function has a guarantee unique minimum, then it is worth investing heavily in a slow but sure iterative procedure. If the function is irregular with many local minima, then it had better be quick and easy to examine points and generate a new guess because a lot of such guesses will be required to get to the right answer. In the present context, evaluating the error function corresponds to the two part question: does the model perform the task, and does it do it the same way as a real system? Picking a new point in parameter space corresponds to generating a new model. The design of Catacomb2 is focused on the second of the two optimization scenarios: the optimization surface is sure to be complex, so it should be quick and easy to generate and test models. The single most important point is simply that the error function should exist. That is, there should be a way to examine and test a model independent of the empirical data on which it is based. Moreover, this test should be readily accessible to any interested researcher with minimal effort (section 7). Essentially, exhibiting a model that does something interesting by the wrong mechanism so it can be shot down can be a much more fruitful step in research than providing a collection of correct mechanisms that do nothing very much.

The case study presented in section 9 highlights a number of directions in which future development is required. Some of these depend mainly upon exploiting recent progress in the software industry to make new applications possible. Others require further research work in the application of computational methods to biological problems.

10.1 Technology driven development

The growth of application libraries and progress in software engineering technology, make many technical problems much more accessible today than they have been in the past. Creating complex, multi-author software systems in a distributed academic environment requires mechanisms whereby each author can work in their own area, with minimal dependence on others and yet where their resulting software is useful outside the originating lab. The development of tools and interface standards within the NeuroML framework promises to make this much easier than has hitherto been the case. Even before such tools are mature, projects such as Catacomb2 can be both providers, and users of such modules. For example, Catacomb2 already relies on the XML parser provided by the NeuroML project and it provides visualization facilities for a variety of NeuroML models. These interactions currently work on a somewhat ad-hoc basis, but form one of the many cases to be considered in the development of NeuroML standards. In the same way, the Neosim(Goddard et al., 2001a) discrete event simulator is

being incorporated as an optional layer between the model description and execution stages. It will add the ability to run network models on parallel machines and workstation clusters, and the parallelization should be almost transparent to the user. Neosim handles all issues related to the configuration of a particular machine, and the model description is completely independent of how it is to be run.

Another area of substantial recent investment is what is being known as "e-science" (*www.escience-grid.org.uk*), which encompasses "grid computing" (*www.gridforum.org*) and the "semantic web" (*www.semanticweb.org*). These are technologies for turning the web from a browsable data storage medium into a distributed computing and knowledge management platform. One core focus is on facilitating collaborative work through shared databases, ontologies, and computing resources. The modularization of modeling systems envisaged by the NeuroML project, and the Axiope (*www.axiope.org*) model publication and sharing system clearly fall within this framework, along with many rather more nebulous concepts. The utility of these technologies to the neuroscience community should become clear as they take on more concrete forms.

10.2 Models and modeling strategies

One of the goals in the development of Catacomb2, as outlined in section 2.2, has been to structure models so that the parameter space is rich in biologically plausible models, and excludes implausible models as far as possible. The provision of large sets of standard components makes it much easier to construct models based on these than models using different formalizations. Thus, for example, even the simplest cell and synapse models are based on conductances and reversal potentials, instead of additive currents or potentials, because conductance-based models are both more realistic and intrinsically more stable. However, fragility and excessive sensitivity to certain parameters remains a major weakness of many models. There are at least two ways of tackling this problem: increasing the certainty about the properties of the systems on which a model is based by using database and ontology resources in model construction; and changing the parameterization so as to describe slow processes such as neuronal development and self-regulation instead of the instantaneous state of a system.

One example of how database access could work is present in Catacomb2 as a graphical browser for the CoCoMac(Stephan et al., 2000) database of connectivity in the Macaque brain. This is able to query the database across the web for connectivity data between regions in selected brain maps and presents the results superimposed on the maps with color coded connection information. At present the information is extracted from the database according to user generated queries and presented through the graphical interface. But it is a relatively small step from here to allowing queries

to be generated automatically by the software. One could ask, for example, whether the projection patterns built into a model are backed up by anatomical data. A further step would see the modeling process begin with database queries in order to set up the connectivity structure of the model. This structure would then be fleshed out by providing single-cell models for populations in different regions (perhaps coming from some other database), and by refining the details of the connectivity patterns. Following a recurrent theme of this paper, the model construction process would then be broken down form being one long single-user procedure into many smaller declarative steps in which existing resource are drawn together and combined in a novel way to construct the next layer.

But perhaps the biggest single problem of these and other models is their fragility and the sensitivity of their behavior to uncertain parameters as discussed in section 2.2. From the perspective of applied mathematics, when the numerical calculation of the behavior of a physical model is very difficult to implement or requires very high accuracy to be reproducible, it often means that the methods or equations are inappropriate, not that the system is genuinely delicate. This is even more likely in biology where the real systems are known to be robust to changes in temperature or chemical composition. Models which require parameters to be expressed to more than a couple of significant figures are therefore intrinsically implausible(Borg-Graham, 1999). Since this applies to at least some part of most models, there is scope for reformulating many models with more robust structures. One of the first aims of future Catacomb development is to move away from static descriptions of the instantaneous state of a model (such as actual ion channel densities for example) towards artificial or, where possible, more realistic parameterization of slow self-regulatory processes which govern these quantities. In general, systems are far less sensitive to the parameterization of regulatory mechanism than to their instantaneous state. In modeling biochemical cascades, for example it has even been argued that the only things that really matter are the presence or absence of reactions between particular species, not the details of their rates at all. Allowing the design of systems where the structure of what is possible, and perhaps order-of-magnitude parameter values are all that matters is therefore central to the development strategy.

Finally, the most realistic, and perhaps also the most reliable way of constructing working models of neural system, is to model the processes of growth and development which give rise to the real systems. The complete separation between the model description and computation leaves open the possibility of adding further layers of processing between them which would make changes in the executable system itself according to rules expressed in a model of growth and development. Although it is within the overall design, much further work is required in order to make such modeling readily accessible in Catacomb2.

11 *Acknowledgements

This work was supported by grants NSF IBN 9996177, NIH MH 60013 and NIH MH 61492 to Mike Hasselmo, a visiting postdoctoral fellowship to Robert Cannon from the FWO, Belgium and the United Kingdom Medical Research Council. Robert Cannon is particularly grateful to Hugo Cornelis for numerous valuable discussions and for his careful reading of the manuscript. The authors also thank the two reviewers for their their thoughtful commentary on the original version.

References

A Alonso and R Klink. Differential electroresponsiveness of stellate and pyramidal-like cells of medial entorhinal cortex layer II. *J. Neurophysiol.*, 70:128–143, 1993.

C A Barnes, B L McNaughton, S Mizumori, B W Leonard, and L H Lin. Comparison of spatial and temporal characteristics of neuronal activity in sequential stages of hippocampal processing. *Prog. Brain Res.*, 83: 287–300, 1990.

D Beeman, J M Bower, E De Schutter, E N Efthimiadis, N Goddard, and J Leigh. The GENESIS simulator-based neuronal database. In S H Koslow and M F Huerta, editors, *Neuroinformatics: An Overview of the Human Brain Project*, chapter 4. Lawrence Erlbaum Associates, Mahwah, NJ, 1997.

G Q Bi and M M Poo. Synaptic modifiction in cultured hippocampal neurons: dependence on spike timing, synaptic strength and postsynaptic cell type. *J. Neurosci.*, 18(24):10464–10472, 1998.

L Borg-Graham. Interpretations of data and mechanisms for hippocampal pyramidal cell models. In E. Jones, P. Ulinski, and A. Peters, editors, *Cerebral Cortex Vol. 13 - Cortical Models*, pages 19–138. Plenum Publishing Corporation, 1999.

L Borg-Graham. The surf-hippo neuron simulation system, v3.0. *http://www.cnrs-gif.fr/iaf/iaf9/surf-hippo.html*, 2001.

J M Bower and D Beeman. *The Book of GENESIS*. Teleos Publishing, Los Angeles, 1994.

N Burgess and J O'Keefe. Neuronal computing underlying the firing of place cells and their role in navigation. *Hippocampus*, 6(6):749–762, 1996.

R C Cannon. Cd-rom. In E De Schutter, editor, *Computational Neuroscience - Realistic Modelling for Experimentalists*. CRC Press, Boca-Raton, 2001a.

R C Cannon. Eggleton '71 revisited. In *ASP Conf. Ser. 229: Evolution of Binary and Multiple Star Systems*, pages 15+, 2001b.

R C Cannon, F W Howell, N Goddard, and E De Schutter. Non-curated distributed databases for experimental data and models in neuroscience. *Network*, in press, 2002.

R C Cannon, D A Turner, G Papyali, and H V Wheal. An on-line archive of reconstructed hippocampal neurons. *Journal of Neuroscience Methods*, 84(1-2):49–54, 1998.

H Cornelis and E De Schutter. NeuroSpaces : New approaches in neuronal modeling software. *Neurocomputing*, in press, 2003.

P P Eggleton. The evolution of low mass stars. *Monthly Notices of the Royal Astronomicl Society*, 151:351, 1971.

D J Faulkner. The evolution of helium shell-burning stars. *Monthly Notices of the Royal Astronomicl Society*, 140:223, 1968.

J Forss, D Beeman, J M Bower, and R Eichler-West. The Modeler's Workspace: a distributed digital library for neuroscience. *Future Generation Computer Systems*, 16:111–121, 1999.

P A Fox, A D Hall, and N L Schryer. The PORT mathematical subroutine library. *ACM Trans. Math. Software*, 4:104–126, 1978.

L M Frank, E N Brown, and M Wilson. Trajectory encoding in the hippocampus and entorhinal cortex. *Neuron*, 27(1):168–178, 2000.

E Fransen, A A Alonso, and M E Hasselmo. Simulation of the role of the muscarinic-activated calcium-sensitive non-specific cataion current I(NCM) in entorhinal neuronal activity during delayed matching tasks. *J. Neurosci.*, 22(3):1081–1097, 2002.

J D Funge. *AI for Computer Games and Animation: A Cognitive Modeling Approach*. J D Peters, 1999.

E Gamma, R Horn, R Johnson, and J Vlissides. *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

N Goddard, G Hood, F Howell, M Hines, and E De Schutter. NEOSIM: Portable large-scale plug and play modelling. *Neurocomputing*, 38:1657–1661, 2001a.

N H Goddard, M Hucka, F Howell, H Cornelis, K Shankar, and D Beeman. Towards NeuroML: model description methods for collaborative modelling in neuroscience. *Philos Trans R Soc Lond B Biol Sci*, 29(352):1209–1228, 2001b.

J J Hagan, E E Verheijck, M H Spigt, and G S Ruigt. Behavioral and electrophysiological studies of entorhinal cortex lesions in the rat. *Physiol. Behav.*, 51(2):155–266, 1992.

M E Hasselmo, C Bodelon, and B P Wyble. A proposed function for hippocampal theta rhythmn: Separate phases of encoding and retrieval enhance reversal of prior learning. *Neural Computation*, 14(4):792–812, 2002a.

M E Hasselmo, E Fransen, C T Dickson, and A A Alonso. Computational modeling of entorhinal cortex. *Annals NY Acad Sci*, 911:418–446, 2000.

M E Hasselmo, B P Wyble, and R C Cannon. From spike frequency to free recall: How neural circuits perform encoding and retrieval. In E Wilding A Parler, T J Busey, editor, *The Cognitive Neuroscience of Memory: Encoding and Retrieval*. Psychology Press, 2002b.

M Hines. Efficient computation of branched nerve equations. *Int. J. Bio-Med. Comput.*, 15:69–76, 1984.

M L Hines and N T Carnevale. NEURON: a tool for neuroscientists. *The Neuroscientist*, 7:123–135, 2001.

C Holscher, R Anwyl, and M J Rowan. Stimulation on the positive phase of hippocampal theta rhythm induces long-term potentiation that can be depotentiated by stimulation on the negative phase in area CA1 in vivo. *J. Neurosci.*, 17(16):6470–6477, 1997.

M Hucka, A Finney ad H Sauro, and H Bolouri. Systems Biology Markup Language (SBML) Level 1: Structures and Facilities for Basic Model Definitions. *web: http://www.cds.caltech.edu/erato/sbml/*, 2001.

S Kali and P Dayan. The involvement of recurrent connections in area CA3 in establishing the properties of place fields: a model. *J. Neurosci.*, 20 (19):7463–7477, 2000.

R Klink and A Alonso. Muscarinic modulation of oscillatory and repetitive firing properties of entorhinal cortex layer II neurons. *J. Neurphysiol*, 77: 1813–1828, 1997.

R Kötter, P Nielse, J Dyhrfjeld-Johnsen, F T Sommer, and G Northoff. ultilevel neuron and network modeling in computational neuroanatomy. In G Ascoli, editor, *Computational Neuroanatomy: Principles and Methods*. Humana Press, 2002.

J C Lattanzio. The asymptotic giant branch evolution of 1.0-3.0 solar mass stars as a function of mass and composition. *Astrophysical Journal*, 311: 708–730, 1986.

J C Lattanzio, C A Frost, R C Cannon, and P R Wood. Hot bottom burning nucleosynthesis in 6 $M_{\odot}$ stellar models. *Nuclear Physics A*, 621 (1-2):C435–C438, 1997.

H Markram, J Lubke, M Frotscher, and B Sakmann. Regulation of synaptic efficacy by coincidence of postsynaptic APs and EPSPs. *Science*, 275 (5297):213–215, 1997.

B L McNaughton, C A Barnes, and J O'Keefe. The contributions of position, direction and velocity to single unit activity in the hippocampus o freely-moving rats. *Exp. Brain Res.*, 52(1):41–49, 1983.

R U Muller, J L Kubie, and J B Ranck Jr. Spatial firing patterns of hippocampal complex-spike cells in a fixed environment. *J. Neurosci.*, 7(7): 1935–1950, 1987.

J O'Keefe and J Dostrovsky. The hippocampus as a spatial map. preliminary evidence from unit activity in the freely-moving rat. *Brain. Res.*, 34(1): 171–175, 1997.

G Orr, G Rao, F P Houston, B L McNaughton, and C A Barnes. Hippocampal synaptic plasticity is modulated by theta rhythm in fascia dentata of adult and aged freely behaving rats. *Hippocampus*, 11(6):647–654, 2001.

C Pavlides, Y J Greenstein, M Grudman, and J Winson. Long-term potentiation in the dentate gyrus is induced preferentially on the positive phase of theta-rhythmn. *Brain Res.*, 439(2):383–387, 1998.

W H Press, S A Teukolsky, B P Flannery, and T Vetterling. *Numerical Recipes in C: the art of scientific computing*. Cambridge University Press, 1993.

G J Quirk, R U Muller, J L Kubie, and J B Ranck. The positional firing properties of medial entorhinal neurons: description and comparison with hippocampal place cells. *J. Neurosci.*, 12(5):1945–1963, 1992.

E S Raymond and B Young. *The Cathedral and the Bazaar : Musings on Linux and Open Source by an Accidental Revolutionary*. O'Reilley and Associates, 2001.

A D Redish and D S Touretzky. The role of hippocampus in solving the Morris water maze. *Neural Comp.*, 10:73–111, 1998.

E Schroedinger. *What is Life? And Other Scientific Essays*. Doubleday, Garden City, New York, 1956.

E P Sharp. Computer simulation of hippocampal place cells. *Psychobiology*, 19:103–115, 1991.

P E Sharp, H T Blair, and M Brown. Neural network modeling of the hippocampal formation spatial signals and their possible role in navigation: a modular approach. *Hippocamous*, 6(6):720–734, 1996.

K E Stephan, L Kamper, A Bozkurt, G A Burns, M P Young, and R Kotter. Advanced database methodology for the collation of connectivity data on the macaque brain (CoCoMac). *Philos Trans R Soc Lond B Biol Sci.*, 355 (1393):37–54, 2000.

W A Suzuki, E K Miller, and R Desimone. Object and place memory in the macaque entorhinal cortex. *J. Neurophysiol.*, 78:1062–1081, 1997.

M C Vanier and J M Bower. A comparative survey of automated parameter-search methods for compartmental neural models. *Computational Neuroscience*, 7(2):149–171, 1999.

M Wenger, F Ochsenbein, D Egret, P Dubois, F Bonnarel, S Borde, F Genova, G Jasniewicz, S Laloë, S Lesteven, and R Monier. The SIMBAD astronomical database: the CDS reference database for astronomical objects. *Astronomy and Atrophysics Supplement*, 143:9–22, 2000.

B P Wyble, C Linster, and M E Hasselmo. Size of CA1 evoked synaptic potentials is related to theta rhythm phase in rat hippocampus. *J. Neurophysiol.*, 83:2138–2144, 2000.

B J Yound, T Otto, G Fox, and H Eiechenbaum. Memory representation within the parahippocampal region. *J. Neurosci*, 17:5183–5195, 1997.