Universität Bielefeld

Technische Fakultät
Int. NRW Graduate School in
Bioinformatics and Genome Research

Integrative Simulation Framework for Modeling Dynamic Cellular Phenomena in 3D over Time

Dissertation
zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften
(Dr. rer. nat.)

vorgelegt an
der Technischen Fakultät
der Universität Bielefeld

von
Bjoern Edwin Oleson

November 2008

Acknowledgments

"Writing a book is an adventure. To begin with, it is a toy and an amusement; then it becomes a mistress, and then it becomes a master, and then a tyrant. The last phase is that just as you are about to be reconciled to your servitude, you kill the monster, and fling him out to the public."

Sir Winston Churchill (1949)

I would like to express my gratitude to all those who gave me the possibility to complete this thesis. I want to thank the International Graduate School in Bioinformatics and Genome Research for giving me permission to commence this thesis, to use the departmental infrastructure, and the generosity in funding my research scholarship.

I am deeply indebted to my supervisors Prof. Dr. Ralf Hofestädt as well as Dr. Klaus Prank. Their help, suggestions, and encouragement helped me in all the time of research for as well as writing of this thesis. My sincere gratitude also goes to Prof. Dr. Christoph Sensen from the University of Calgary, Canada, for his guidance and valuable working environment I had the opportunity to enjoy.

I have spent very special years studying for this degree. There are many friends whom I would like to thank. Dr. Leila Taher shared with me not only the same office, but also some scientific ideas and joys of daily life. Dr. Mark Möller is one of the best colleagues I can think of. He has an answer for almost everything and can listen to questions considerately! I would like to thank Dr. Dirk Evers, Doris Hengel, and Volker Tölle for being so patient and helpful all along.

Most of all, I would like to thank my parents, Ursula and Robert E. Oleson, as well as my wife, Elke, for all their love, encouragement, and support. Not to forget my cute little daughters, Jennifer and Natalie, who are a bright sunshine in my life. Without all their help and backing this degree would never have been possible.

"We shall not grow wiser before we learn that much that we have done was very foolish."

Friedrich August von Hayek (1944)

Contents

CHAPTER 1

Introduction

1.1 Motivation

The emerging field of systems biology allows for the application and combination of experimental, theoretical, and modeling techniques. A key goal of systems biology is to understand biological processes as whole systems instead of isolated parts. With such a systems-level analysis, the study of biological phenomena at the molecular, cellular, or behavioral levels becomes more feasible (Kitano, 2002b; Kell, 2004). Improvements in the measurement of molecular interactions and rates have revolutionized our insight into the cell and its dynamics. An enormous amount of information has been gained over the past decades adding to the already-known mechanisms. This leads to a vast set of parameters making it difficult to evaluate new hypotheses on intuition alone. Thus, applying modeling techniques and computer simulations help to confirm such hypotheses (Kitano, 2002a; Shapiro et al., 2002).

In recent years much progress in the development of cell-biological modeling and simulation tools has been achieved. A multitude of existing programs can already fulfill the major requirements of execution speed and result accuracy. However, often such programs are highly specialized in their methodology and the applications they were designed for. There is no system that allows for the use of more than one algorithm in a simulation at a time. Furthermore, the existing tools are still behind on the development of three-dimensional (3D) visualization techniques. In comparison to existing two-dimensional (2D) simulation applications, the 3D approaches are still at their early stages. The additional spatial information allows for a more direct referencing of the model to its real object. Methods applied to 3D share many characteristics with 2D techniques. Many existing 2D methods therefore can be either adopted or carried forward.

Along with mechanistic models it is often possible for system behavior to be formalized into solvable differential equations. On the basis of such equations the evolution of time is regarded as predictable as well as continuous and is therefore a very fast and precise method (Ghosh and Tomlin, 2004). Nevertheless, deterministic approaches to model and simulate the dynamics of intracellular regulatory processes have their limitations. For the reason of combinatoric increase in the number of equations and the number of contained players this process becomes highly impractical. If only small numbers of particles are involved, deterministic modeling approaches and simulation might fail. Stochastic approaches, using closer knowledge of the subcellular architecture, are more appropriate in these situations (Kiehl et al., 2004).

A significant obstacle with given modeling and simulation tools is that they seem to be able to handle only a small set of applications. They were developed for simulating special models and lack the ability to integrate extra techniques for solving other problems. In fact, often systems focus their attention on handling a special problem to all its details in minimal computational time (Pettinen et al., 2005). The currently used programs therefore vary noticeably in their applicability for specific types of modeling. Moreover the integration of such programs is highly demanding. They are closed systems and often do not offer component interfaces. In practice it could be beneficial to allow for such interfaces to combine various different tools to just one tool only. A compromise to the absence of direct interfaces is the possibility of exchanging information via standardized file formats. For this, many tools use the Systems Biology Markup Language (SBML) as a standard (Hucka et al., 2004). Often, by the use of such standardized formats, important information to the model cannot be handled correctly and might get lost. Especially with SBML, the geometrical information cannot be saved entirely and subsequently has to be reconstructed.

Applications for such a modeling and simulation tool that allows for both 3D visualization and concurrent algorithms are imperative. The applications lie in the areas of electrical excitable cells, circadian rhythm, cell cycle, cellular motility, membrane transporters, metabolic pathways, and signal transduction networks. In comparison to 2D methods, the use of a 3D geometry provides considerably more significant data. The possibility of simulating different compartments of a cell with diverse algorithms can reduce computational effort dramatically. Consequently, this allows for both improved modeling and more information output on the spatio-temporal behavior of a system. Thus it is a scientific challenge to integrate concurrently running algorithms in combination with 3D visualization.

1.2 Objectives

This work addresses the problem of an in silico biology system-level analysis on the base of cellular signaling networks. Grounded on biological observations, computational simulation models for the calcium(II) (Ca^{2+}) signaling pathway were developed in an attempt to understand the non-linear dynamics of the system.

To create this kind of an all-encompassing application, several existing simulation systems were evaluated for their strengths and weaknesses. Current state-of-the-art cell simulation applications are either limited to a 2D representation of a cell or do not use any cellular geometry at all. The only exceptions to this finding are the deterministic simulator "VirtualCell", the stochastic simulators "MCell" and "SmartCell", as well as this approach of a hybrid four-dimensional (4D) Cell Simulator (4DiCeS). Despite the great variety of software packages available for modeling, simulation, and analysis of data (Hucka et al., 2004) as described above, there was no application that features the complete and variable integration of different simulation methods in a 3D environment.

The key improvement, which 4DiCeS has over the other existing systems, is the ability not to be fixed on either deterministic or stochastic modeling and simulation approaches. A system of specialized interfaces therefore was designed to allow the bonding of interchangeable algorithmic modules of various types. The internal representation of the model is designed for the easy exchangeability of data. Furthermore, the integration of cell model file format standards is permitted by exchangeable plug-ins. The implemented system includes an Application Programming Interface (API) for writing individual plug-ins to utilize different simulation algorithms. This facilitates the implementation of tailored programs and specific algorithms that can be developed for data mining as well as visualization. The resulting 4DiCeS framework presented in this work describes a concept for the integration of heterogenous data into an easy-to-use software.

1.3 Structure

This thesis is primarily concerned with the simulation of cellular phenomena. An application is provided to utilize hybrid mathematical models and to simulate 4D spatial dynamics within a cell. The work reveals that a 3D cellular geometry holds extended information of importance, and is closest to reality in its model and simulation. Also, an application framework will be shown that allows for the integration of various particle reaction and diffusion algorithms. Such algorithms can then be applied to a model either in sole or even in concurrency.

The work is divided into three parts. The first part (Chapters 1, 2, and 3) provides this motivation and an introduction to the topic of cellular dynamics, and its modeling and simulation. Relevant current modeling and simulation application will be introduced. The second part consists of Chapter 4 and focuses on the formal description, design, and the methodology of the 4D cell simulator. The third part (Chapters 5 and 6) presents a comparison of related works with the results of the project, discusses this work, and gives a brief outlook on further improvements to the system.

A condensed summary of all chapters of this work is provided with the following overview:

Chapter 2 reviews biological concepts of systems biology as well as mathematical aspects of modeling, simulation, and analysis of cellular dynamics.

Chapter 3 describes a number of existing systems that represent state-of-the-art biochemical modeling and simulation applications.

Chapter 4 focuses on the formal description as a basis for the design of the 4DiCeS framework for systems biology.

Chapter 5 comprises the results that were obtained during development and testing 4DiCeS.

Chapter 6 evaluates and discusses the 4DiCeS platform including some ideas that will illustrate further development and directions.

CHAPTER 2

Dynamics in Systems Biology

During the past century much progress was achieved in the measurement of cellular processes, molecular interactions, as well as their kinetics. Thus, a revolution was initiated in the understanding of the dynamics within biological cells. The following Figure 2.1 of a pyramid composed of different molecules wants to give insight into the complexity of cellular organization.

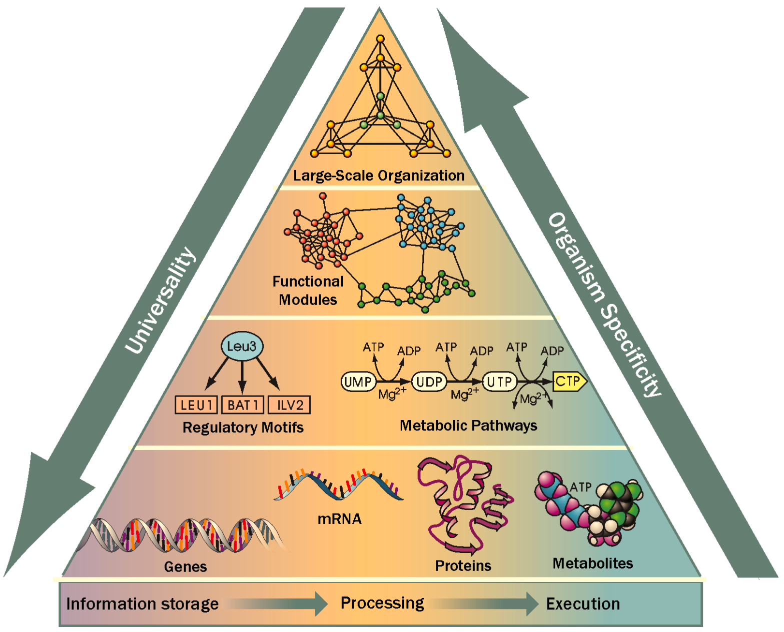


Figure 2.1: Life's complexity pyramid. The genomic information is both stored and translated into functional units such as proteins and metabolites. These units form operational molecules, consisting of regulatory systems or metabolic pathways. On top of these rather small units, large scale organizations implement the characteristic features of an organism. Adapted from Oltvai and Barabási (2002).

The fundamental units are arranged to either metabolic and signaling pathways or to motifs in genetic-regulatory networks. Motifs and pathways are linked to operational groups that are responsible for discrete cellular functions (Hartwell et al., 1999). Such groups are then nested hierarchically and characterize the large-scale organization of a cell (Ravasz et al., 2002).

While creating models of cellular dynamics, one has to understand complex properties. Such properties include the combination of regulatory mechanisms and interlocking transport in and among cells. Electrical activity, signal transduction, or other biochemical networks are examples of intricate behaviors taking place on the cellular level. In general such dynamic phenomena refer to arbitrary processes occurring and therefore changing over time. This forms the highly dynamic basis for living cells. To maintain the typical characteristics of a cell's life such as growth, movement, responsiveness, cell division, and intercellular communication, cells must continuously obtain energy from their direct neighborhood. Therefore cells need to act far from static thermal equilibrium as thermodynamically open systems. Hence, cells require a huge amount of energy to sustain the gradients of metabolites and ions in order to function properly (Fall and Keizer, 2002).

The following two Sections 2.1 and 2.2 give an overview on the scope of modeling and simulation of such dynamic systems within cells. By doing so, the need for simulation tools, such as the 4DiCeS software presented here, handling such models becomes apparent.

2.1 Modeling and Simulation

Theoretical methods joined with experimental measuring have offered comprehensive perception of dynamics for many years (Ortoleva et al., 2003). Computers have demonstrated to be a necessity in assisting the dissection of molecular processes. Yet, the bare amount of quantitative experimental cellular information allows for the cooperation of computer science and biology (Chong and Ray, 2002). The interaction of theory, experiment, and computation succeeds a conceptual formulation analog to successfully proven physical models (see Table 2.1).

Here all modeling is an abstraction of reality. The only exact model of any system is the system itself. So when a model of a system is designed, a choice must be made regarding the level of detail and feature types to be included into that model. To a large extent, this is prescribed by the characteristics of the examined system, the type of experimental data available, and the type of questions that are addressed to modeling (Bolouri and Davidson, 2002).

Step	Task	Description
I	Experimental Work	Figuring out the most plausible out of all possible molecular mechanisms as an initial step.
II	Schematic Description	Define a schematic description that characterizes the entire model from such selected mechanisms.
III	Mathematical Expressions	Translate the elementary steps of the mechanism into mathematical expressions.
IV	Differential Equations	Combine the changes in time described by such expressions into differential equations.
V	Analysis	Reveal the model's success for the biological system from the differential equations' study.

Table 2.1: Conceptual formulation of models. Here, the interdependency of experiment, theory, and computation describe the production of a conceptual formulation of models. All given steps depend highly on a close collaboration with experimentalists working at the same problem. Adapted from Fall and Keizer (2002).

The problems theorists encounter in biology are therefore very alike to that in physical sciences. At this level equations are analyzed, if possible simplified, solved, and, most importantly predictions can be made. These predictions are checked by further experiments. Moreover, such experiments may disclose discrepancies that in turn will require changes to the model (Alvarez-Vasquez et al., 2005). The procedure addressed here is an improving cycle of approximations where the theoretical model acts as a quantitative hypothesis (see Figure 2.2).

The history of simulation and in silico analysis of biological systems dates back to the earliest mechanical and analogue computers in 1940 (Chance, 2004). The recent progress in quantitative simulation and modeling are due to advances in modern information technology and was enhanced by the recent burst of molecular data. It becomes obvious that future progress in the understanding of biological functions will rely on the development and the use of computational methods (Arkin, 2001). Therefore, the following two Sections 2.1.1 and 2.1.2 give a closer look at both, the roles of biology as well as computation to the modeling of biological cellular systems.

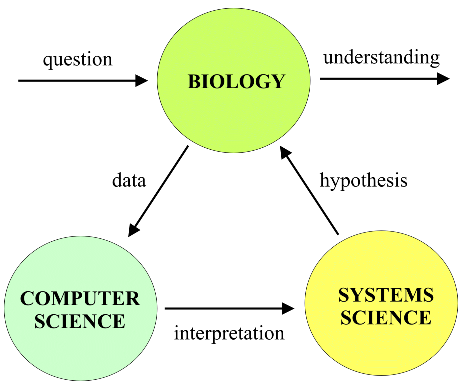


Figure 2.2: Systems biology triad. At the center are the complex biological phenomena. Interpretation of observations and data is supported by computational algorithms. The translation of interpretation to understanding is supported by systems science. In turn, systems science provides a framework for understanding. It indicates hypotheses to be tested and modified in an iterative cycle of experimentation. Adapted from Mesarovic et al. (2004).

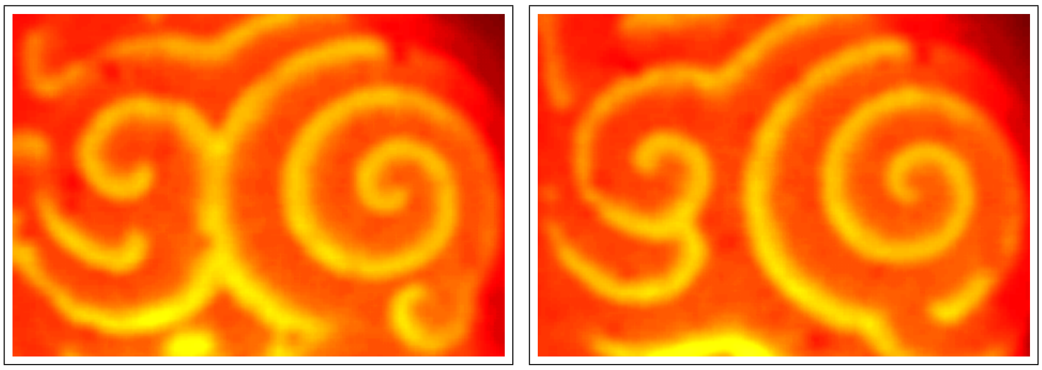
2.1.1 Biology as a Model Featurer

Cells show very complex and different behaviors. Single cells are able to contract, excrete, move, reproduce, send signals, or even respond to them. Furthermore, cells accomplish the energy handling required for such activities. In cooperation cells manage all of the various processes required to perpetuate life as is (Hofmeyr, 1986). However, everything that cells do can be represented in the form of basic natural laws. Although the rules of behavior are rather basic, cells comprise huge and complex networks of interacting substrates. An enormous amount of work was used disentangling only very few of such reaction schemes, and it is quite obvious that there are many more such interaction networks yet to be revealed (Keener and Sneyd, 2001). Table 2.2 gives a brief overview of dynamic behaviors happening on a cellular level. Accordingly, the need for powerful modeling and simulation tools, as 4DiCeS, arises very quickly from the study of such phenomena.

In the subsequent sections the dynamical behavior of Ca^{2+} signaling for its spatio-temporal patterns (see Figure 2.3) will be discussed in more detail. The modeling and simulation of such a phenomenon were used as examples for testing the 4DiCeS software. The role of computational techniques handling such models is going to be presented in the following subsection.

Phenomenon	Description
Circadian Rhythm	Regular changes in cellular processes and behavior that have a period of about one day.
Cell Cycle	The event of cell division where one cell proliferates into two descendants with full genome.
Cellular Motility	All cellular movement including the remodeling of cell membranes, cellular travel, and contraction.
Membrane Transporters	Transport catalysts at the membrane acting either as carriers, pumps, or channels for particles.
Electrical Excitable Cells	The display, acceptance, or propagation of electrical potentials along or within cells.
Excitable Oscillation	Coupled mechanism of membrane transporters and electrical activity resulting in oscillations.
Non-Excitable Oscillation	Self-contained spatio-temporal oscillations of particle concentrations within a cell.

Table 2.2: Phenomena of cellular dynamics are here depicted, which have trigger events in common. There exist theoretical models and simulations for each incident. Extracted from Fall et al. (2002).



(a) 0.5 *min* (b) 1 *min*

Figure 2.3: A spiral wave of Ca^{2+} ions detected from a dye with microinjection of IP_3 into an *Xenopus laevis* oocyte after 30 and 60 seconds. By courtesy of James D. Lechleiter, University of Texas, USA.

2.1.2 Computation as a Model Solver

Techniques applied to research problems in systems biology comprise very much of applied mathematics and computer science. A majority of features in computational modeling of cell biology play an important role. One of these features is the development of algorithms and techniques that give tools for numerical analysis (Takahashi et al., 2002). The computation of mathematical problems on computers is basically an estimation process. The efficiency and accuracy of these methods of estimation are the subjects of intensive study. The work on constructing models is also basically an approximation process. This is due to necessary simplifications that must be used to produce a helpful model. These simplifications must both be valid in terms of the physical process being studied and from a mathematical point of view (Eungdamrong and Iyengar, 2004).

Computer models allow for testing conditions that may be difficult to obtain or that have not been examined in the laboratory yet. Therefore, every solution of a mathematical model may offer a simulation of a potential or real experiment. Such simulations can help to estimate parameters, i.e. diffusion or kinetic constants, that are challenging to collect in experiments. Simulations can verify how pharmacological agents may affect a biological process (Mendes and Kell, 1998). Hypothesis about the role of individual mechanistic components can be checked by simulations. Accordingly, predictions made by such simulations then can be tested in the laboratory. It has to be noted that often the most important result of a simulation is a negative one. Therefore, a well-crafted model has to be redefined and tested again (Fall and Keizer, 2002).

Improvements in numerical analysis as well as computer hardware have made the solving of complex deterministic systems accurate and fast. However, models of biological processes almost always comprise nonlinear components in their control mechanisms. By using traditional mathematical methods, as coupled Ordinary Differential Equations (ODEs), often such problems can be solved exactly. But nonlinearities often create difficulties in getting any exact solution. Admittedly, good estimates of nonlinearities can be obtained by using computer implemented numerical methods. Often a major property in cellular mechanisms is spatial variation. Therefore, the analyzing and the solving of spatially explicit Partial Differential Equations (PDEs) is very important. Such PDEs can be more complex and less analytically tractable than ODEs.

Some models need to handle noise and the tracking of single particles instead of particle concentrations. Here stochastic methods such as Monte Carlo (MC) simulations (Metropolis and Ulam, 1949) come into play. Such discrete models facilitate the qualitative modeling and are based on various different computational approaches (Hofestädt and Meineke, 1995). Unlike the different deterministic methods, a stochastic approach does not approximate the model as a continuous macroscopic system. In contrast, it treats a model as a discrete and microscopic process. However, this increase in accuracy comes at high costs as each individual chemical entity has to be modeled as a stochastic process. Hence, stochastic simulations are computationally more demanding (Puchałka and Kierzek, 2004).

Method	Description
ODE	A relation that contains functions of one independent variable involving its derivatives.
DAE	Coupled ODEs with additional algebraic constrains (no derivatives).
PDE	Differential equations with more than one independent variable involving partial derivatives.
SDE	Differential equations including a random term that describes intrinsic noise.
MC	A set of discrete quantities and associated probabilities for interactions.
Boolean Network	A conversion of a model into a binary representation of only "true" and "false" states.
CA	Collection of different elements (cells) with distinct states on a grid of specified shape.
Bayesian Network	Acyclic directed probabilistic graph with random variables (nodes) and conditional independence assumptions (arcs).
Petri Net	Modeling for concurrent systems with a bi-partial directed graph. Generalization of the automata theory.

Table 2.3: Important modeling and simulation techniques. The given methods represent the main approaches by which modeling and simulation are handled in systems biology. Applications vary in their support for either a pure method's implementation or hybrid attempts mixing different techniques. Extracted from Hucka et al. (2004).

Various attempts have been made to construct and simulate biochemical behaviors. The majority of approaches depend on the use of the before mentioned deterministic and stochastic techniques (Hucka et al., 2004). Recently other well-established techniques have been applied including boolean networks (Kauffman, 1969), Cellular Automata (CA) (von Neumann, 1966), Bayesian networks (Pearl, 1988), and petri nets (Petri, 1962), to biological applications. Hybrid methods that combine the best features of all approaches exist as well (Lu et al., 2004). Table 2.3 summarizes some highly recognized basic techniques for modeling and simulation in systems biology.

To solve equations that result from a model is only one part of the work. The other side needs to comprehend the model's behavior. Mathematical methods were developed for the system analysis of models that characterize complex processes. Such methods disclose the dynamical behavior, properties, and structure of the system. This is very much as molecular biological, physiological, and anatomical techniques uncover the physical basis of the model (Hartwell et al., 1999). Hence, the analysis exposing complicated behaviors within a model may result in further study of these biological phenomena. Admittedly, decisive analysis of complex equations demands skill. This is due to the fact that there are many intricacies that can only be comprehended through intensive training (Mishra et al., 2003). The construction and alteration of simple models is within the reach of cell biologists. Therefore it is necessary for scientists to seek association with mathematicians and computer scientists for the effectual simplification of complex models (Sontag, 2004).

The next section will go into the detail of a well-studied Ca^{2+} model systems. This model was chosen to test the 4DiCeS software in its functionality and accuracy. The next chapter on the other hand will give insight into the design of the 4DiCeS system itself. After reading these sections, it will become obvious that the application should be capable of applying any of the before mentioned mathematical techniques. As for now stochastic and deterministic methods are implemented to be used by 4DiCeS (Oleson et al., 2006).

2.2 Cellular Calcium Models

Cellular Ca^{2+} has an overall very low concentration. At rest it is approximately $0.1\mu M$, and only about 1–$10\mu M$ at its peak. On the other hand, potassium (K^+) and sodium (Na^+) show millimolar concentrations. Cells require to keep cytoplasmic Ca^{2+} concentration ($[Ca^{2+}]_i$) at low levels due to the fact that Ca^{2+} can modify the enzymatic properties of binding proteins. Hence increases in the cellular Ca^{2+} level are locally defined and quick to circumvent the runaway activation of enzymatic cascades. Two basic mechanisms hold responsibility for this impoundment and buffering. The buffers are highly specialized Ca^{2+}-binding proteins that absorb 95–99% of the cytosol's Ca^{2+}. Ca^{2+} is impounded to either the sarcoplasmic reticulum (SR) in muscle cells or the endoplasmic reticulum (ER) in all other cell types. Proteins hydrolyze adenosine–3',5'–triphosphate (ATP) to transport Ca^{2+} against rampant concentration gradients. Such proteins are ATP hydrolases (ATPases) that are classified as SR/ER Ca^{2+}-transport ATPase (SERCA) pumps. On the other side, plasma membrane (PM) Ca^{2+}-ATPase (PMCA) pumps dispose Ca^{2+} of the cell. SR and ER membranes have ion channels, which are different from PMCAs, that transports Ca^{2+} back into the cytoplasm. Therefore, every cell has Ca^{2+} pumps for homeostasis as well as negative feedback. Some cells have developed ion channels such as the inositol–1,4,5–trisphosphate (IP_3) receptor (IP_3R), which is activated and inhibited by Ca^{2+}. The IP_3R is able to give both positive and negative feedback. Thus, brief channel openings may enable oscillations of free cytoplasmic Ca^{2+} that are utilized for signaling. Interestingly, Ca^{2+} oscillations were discovered in vitro after they were predicted by a model (Chay and Keizer, 1983).

2.2.1 Calcium Signaling

Ca^{2+} is the most common cellular signals' carrier. Due to its special adaptability as a ligand, it regulates very many important aspects of cellular activity. This goes from the creation of new life at fertilization to the radical incident of cellular apoptotic suicide. Signaling by Ca^{2+} shows a number of properties that make it unique among all other carriers of signals. An important example is its ability to function both as a first and as a second messenger (Carafoli, 2005). Then Ca^{2+} cannot be metabolized like other second-messenger molecules. Hence cells tightly regulate intracellular levels through numerous bindings as well as specialized extrusion proteins (Clapham, 1995).

Almost all eucaryotic cell types use both intracellular as well as extracellular resources of Ca^{2+}. The responsible regulating mechanisms for the influx of external Ca^{2+} are already well known to scientists (Carafoli, 2002). As an example, voltage-operated channels in neurons assist action potentials by triggering the release of neurotransmitters at synaptic junctions. Neurotransmitters can establish an influx of Ca^{2+} utilizing receptor-operated channels primarily localized postsynaptically. Despite such well-established influx pathways, there is not much known on the mechanism of the intracellular Ca^{2+} supply of neurons although IP_3Rs allocated all over the ER are accountable for the dispense of Ca^{2+} (Berridge, 2005).

The ER network in cells accounts for the dynamics of Ca^{2+} signaling by operating both as a sink as well as a source of Ca^{2+}. Such an internal storage of Ca^{2+} is possible to have a far reaching impact on Ca^{2+} signals in cells. Ca^{2+} can be localized within compartments in high levels or can spread across cells as widespread Ca^{2+} waves (Berridge, 1998).

The IP_3 signaling network (see Figure 2.4) is highly sophisticated within cell tissues. A manifold amount of diverse receptors excite the hydrolysis of phosphatidylinositol-4,5-bisphosphate (PIP_2) into IP_3 and sn–1,2–diacylglycerol (DAG). Both are well known second messengers. IP_3 then releases Ca^{2+} from the ER's IP_3Rs (Berridge, 1993).

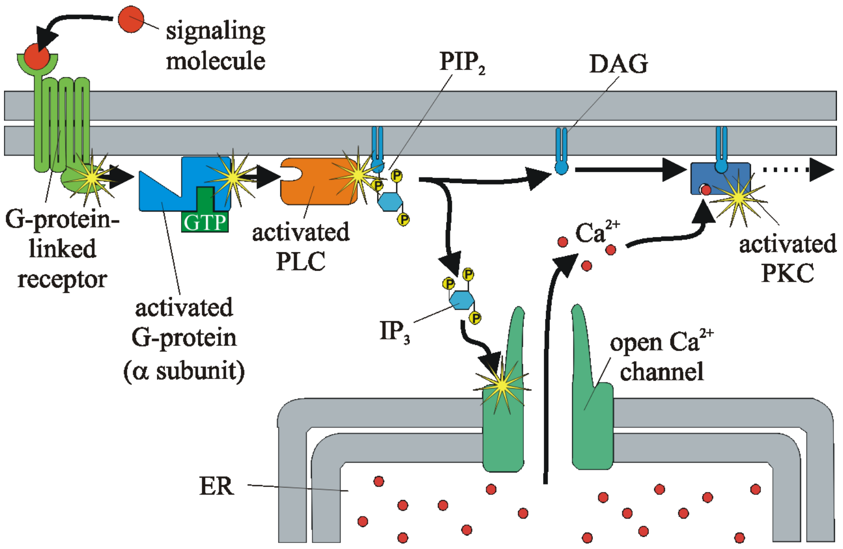


Figure 2.4: The Ca^{2+} signaling pathway. When a hormone or neurotransmitter ("first messenger") interacts with a receptor on the cell membrane, IP_3 is released within the cell, causing a calcium response. The ER plays a decisive role in calcium regulation next to the extracellular space as a storage for the cytotoxic Ca^{2+}. Adapted from Alberts et al. (2003).

By the fact that the intracellular Ca^{2+} release by IP_3Rs is sensitive to a diversity of different factors makes it very complex. IP_3Rs are most effectively activated when Ca^{2+} and IP_3 are both presented at the same time. This dual activation has noteworthy effects for signal transduction mechanisms. Foremost of these is that the IP_3 receptor may act as a cooccurrence detector due to the requirement for two separate messengers. Small levels of IP_3, unable to excite an immediate Ca^{2+} release, may enhance the IP_3R's Ca^{2+} responsiveness. Thereby they change the cytoplasm into a medium capable of producing regenerative Ca^{2+} waves. An increasing Ca^{2+} gradient within the ER can have a positive feedback by sensitizing the IP_3Rs' Ca^{2+} uptake. The increase of $[Ca^{2+}]_i$ represses IP_3Rs (Berridge, 2004).

2.2.2 Two Pool Model

The two pool model presented in this section is based on the works of Goldbeter et al. (1990); Berridge (1991). This model shows an oscillating behavior of Ca^{2+} concentrations very similar to phenomena discovered in living cells. If a diffusion term is applied to the system even 3D wave-fronts can be simulated.

The basis of the two pool model is that oscillations are set up through an interaction between two releasable pools of Ca^{2+}. Here it is assumed that the external signal triggers the synthesis of IP_3. The effect is simply a discharge of an intracellular pool of Ca^{2+} leading to a rise of cytosolic Ca^{2+}. A simple assumption is made that a constant influx of Ca^{2+} from the IP_3-sensitive pool occurs as long as the stimulus is present. This affects the probability of the occurrence of oscillations exclusively by the cycling of Ca^{2+} between the cytosol and the IP_3-insensitive pool. The IP_3-insensitive pool is therefore considered to remain constant as a result of a fast backfill by the influx of extracellular Ca^{2+} (Goldbeter et al., 1990). This autoregulatory mechanism is controlled by the content of Ca^{2+} in the IP_3-sensitive pool and by the uptake of cytosolic Ca^{2+} after a spike. The magnitude of the influx, $v_1\beta$, from this IP_3-sensitive pool is assumed to be proportional to the saturation function β of the IP_3R. The cooperative nature of this saturation function is expressed implicitly in β. The level of IP_3 is proposed to be caused by stimulation increases with the magnitude of the external signal. IP_3 thus controls the flow of Ca^{2+} into the cytosol. This again assists the IP_3-insensitive pool for releasing Ca^{2+} in oscillatory cycles (Berridge, 1991).

The two variables of this model are the concentration of free Ca^{2+} in the IP_3-insensitive pool (e.g., the ER or SR) and in the cytosol. These two variables are denoted by Z and Y, respectively. If assumed that buffering is linear with respect to the Ca^{2+} concentration, then the time evolution of the system is driven by the following two kinetic equations (Goldbeter et al., 1990):

$$\frac{dZ}{dt} = v_0 + v_1\beta - v_2 + v_3 + k_f Y - kZ\,, \tag{2.1}$$
$$\frac{dY}{dt} = v_2 - v_3 - k_f Y\,. \tag{2.2}$$

In the above equations, all rates and concentrations are defined with respect to the total cell volume. Here, v_0 and kZ relate to the influx and efflux of Ca^{2+} into and out of the cell. This occurs even in absence of external stimuli. These terms are assumed to be constant and linear for simplicity. The rate of ATP-driven pumping of Ca^{2+} from the cytosol into the IP$_3$-insensitive store is denoted with v_2. In contrast v_3 represents the rate of transport from this pool into the cytosol. The term $k_f Y$ refers to a leaky, non-activated transport from Y into Z. This process was found to stabilize the amplitude of Ca^{2+} transients at different levels of stimulation.

An increase in IP$_3$ is triggered the reception of an external signal. This in turn leads to a rise in the saturation function β and to a subsequent increase of cytosolic Ca^{2+}. The conditions in which this initial rise triggers Ca^{2+} oscillation can be determined by resorting to phase plane analysis. This is especially possible because the system of equations comprises only two variables. Here, it was indicated that the activation of v_3 by Z is most appropriate for inducing sustained oscillations upon external stimulation. This condition directly corresponds to an activation by cytosolic Ca^{2+} where Ca^{2+} is transported from the intracellular store into the cytosol. The two pool model predicts that at least in the absence of time delays, such a process does not satisfy the triggering of a sustained oscillatory response. When taking into account the cooperative nature of the pumping process, the Ca^{2+} release from the intracellular store, and the positive feedback performed by the latter transportation of cytosolic Ca^{2+}, the rates v_2 and v_3 in the equations 2.1 and 2.2 take the following form (Goldbeter et al., 1990):

$$v_2 = V_{M2}\frac{Z^n}{K_2^n + Z^n}\,, \tag{2.3}$$
$$v_3 = V_{M3}\frac{Y^m}{K_R^m + Y^m} \times \frac{Z^p}{K_A^p + Z^p}\,. \tag{2.4}$$

In these equations, V_{M2} and V_{M3} denote respectively the maximum rates of Ca^{2+} pumping into and the release from the intracellular store. These processes are described by Hill functions whose cooperativity coefficients are taken as n and m. Here, p denotes the degree of cooperativity of the activation process. K_2, K_R, as well as K_A are threshold constants for activation, pumping, and releasing.

The equations 2.1, 2.2, 2.3, and 2.4 admit a unique steady-state solution. Linear stability analysis of these equations indicated that the steady state is not always stable. In the absence of stimulation, a situation is considered in which the system is initially in a stable steady state characterized by a low cytosolic Ca^{2+} level close to 0.1μM. The system reacts to an increase in β up to 30%, due to a rise in IP_3 triggered externally. Here, an oscillation of cytosolic Ca^{2+} occurs. Such repeating spikes are accompanied by a sawtooth variation of the intracellular store's Ca^{2+} content. The period of the oscillations is of the order of $1s$, as in some experimental systems. Periods of $1min$ or more are readily obtained if the kinetic parameters are divided by a factor of 10–100 (Kraus and Wolf, 1992). A spatio-temporal extension of the two pool model allows for the modeling of intercellular Ca^{2+}-waves. To make this happen a diffusion term has to be added to the system. In the model it is assumed that the IP_3-sensitive Ca^{2+} pools are only located near the membrane and the IP_3-insensitive pools are spread all over the intracellular space.

The mathematical description of the deterministic methods account the system with a coupled set of nonlinear ODEs of first order. Kraus et al. (1992); Kraus and Wolf (1992) derived a stochastic model from this system, which is numerically traceable by means of a stochastic simulation. The stochastic method models the system through a master equation. With the first method the pools correspond to particle concentrations and the processes correlate with mathematical functions, which move into the pools via fluxes connected with concentrations. With differentiation of the master equation the pools contain a certain number of particles. Now the process describe transition probabilities between state transitions, which are connected by fluxes. The states are characterized by the overall occupation number of the pools. External entities comply in both cases with externally defined system parameters. While the deterministic method inspects the temporal trend of concentrations, the stochastic method describes changes in particle numbers of every particle species in contrast to that. Each reaction, where the number of particles changes, is simulated directly. The stochastic method therefore constitutes the microscopic view of the system, which is in diametric opposition to the deterministic - and thus macroscopic - depiction.

The stochastic model was applied to the simulator described in this work for final a testing and comparison purpose of state-of-the-art modeling and simulation tools in Section 5.2.2.3. The mathematical formulation of the stochastic two pool model is displayed in Table 2.4.

Reaction	Transition Probability	State Transition
$Ca^{2+}_{ex} \xrightarrow{k_1} X$	$k_1 Ca^{2+}_{ex} \equiv v_0$	$X \rightarrow X+1$ $Y = \text{const}$
$X \xrightarrow{k} Ca^{2+}ex$	kX	$X \rightarrow X-1$ $Y = \text{const}$
$Ca^{2+}_{ISCS} \xrightarrow{k_2\beta} X$	$k_2 Ca^{2+}_{ISCS}\beta \equiv v_1\beta$	$X \rightarrow X+1$ $Y = \text{const}$
$Y \xrightarrow{k_f} X$	$k_f Y$	$X \rightarrow X+1$ $Y \rightarrow Y-1$
$pX + mY \xrightarrow{v_3'} (m+p)X$	$v_3 = V_{M3}\frac{Y^m}{K_R^m+Y^m} \times \frac{Z^p}{K_A^p+Z^p}$	$X \rightarrow X+m$ $Y \rightarrow Y-m$
$nX \xrightarrow{v_2'} nY$	$v_2 = V_{M2}\frac{Z^n}{K_2^n+Z^n}$	$X \rightarrow X-n$ $Y \rightarrow Y+n$

Table 2.4: Stochastic two pool model. This table specifies the mathematical process of the stochastic two pool model. X and Y describe the number of Ca^{2+} ions in the cytosol or the IP_3-insensitive pool. There is the assumption that the number of Ca^{2+} ions in the extracellular space and within the IP_3-sensitive pools are kept constant. Adapted from Kraus et al. (1992).

Chapter 3 now gives a closer look on currently used simulation software tools. Their architecture and utilized algorithms will be continued and compared in Chapter 5 with a preceding formal description on 4DiCeS (see Chapter 4).

CHAPTER 3

Related Approaches and Tools

The relevance to model as well as to simulate biological systems was discussed in the previous chapter. The following chapters give an overview of the state-of-the-art in modeling and simulation in comparison to the new cell biology framework 4DiCeS. Therefore, some of the inherent problems in characterizing the different facets of biological function are stated. This includes a brief overview of how functional information is currently represented in databases. And also prevailing applications for modeling and simulation of biochemical networks are introduced.

3.1 Database and Information Retrieval

The tremendous but valuable information gathered together in recent years has to be organized and pooled in databases. In this respect databases are widely deployed to store the relationships of biochemical systems. Currently there exist over 1000 biological databases (Galperin, 2008) and about 45 databases supplying cellular signaling pathways at different levels of detail and complexity.

Public bio-molecular interaction databases are resources to basic building blocks of biological signaling pathways. Huge clusters of molecular interactions can be generated based only on this information. However, a molecular interaction cluster does not represent a signaling pathway per se. In effect, more information about each interaction, such as its outcome (e.g. activation as well as inhibition), is required for it to become a trustful component of a signaling pathway (Cary et al., 2005). Both public as well as private database initiatives have taken up the effort of creating biological pathway databases and providing computational biology tools for their analysis. Some of the databases focus on static (manually drawn) representations (Bhalla and Iyengar, 1999; Sivakumaran et al., 2003; Trost, 2002) whereas other systems support dynamic visualizations based on graph drawing algorithms (Fukuda and Takagi, 2001; Fukuda et al., 2004)). There is also a variety of databases specialized in molecular pathways with physical parameters as rate constants and concentrations (Igarashi and Kaminuma, 1997). In addition to the previously described pathway databases, there exist databases containing detailed information regarding characterized enzymatic reactions. Additional links to other databases provide useful information on involved enzymes and biochemical reactions (Gough and Ray, 2002; Gough, 2002).

Currently three simulation model repositories serve actively in the internet – namely the Cell Markup Language (CellML) (Lloyd et al., 2004), the JWS Online (Olivier and Snoep, 2004), and the BioModels (Novère et al., 2006) repository (see Table 3.1).

Designation	Web Site
BioModels	http://www.ebi.ac.uk/biomodels/
Cellerator	http://www.cellerator.info/nb.html
CellML Repository	http://www.cellml.org/models/
JWS Online	http://jjj.biochem.sun.ac.za/
xCellerator	http://www.xcellerator.info/examples/index.html

Table 3.1: Model repositories. The simulation model repositories of the two most prominent modeling languages SBML and CellML contain models on metabolic networks, cell cycle and cellular signaling. The (x)Cellerator sites provide example models as Mathematica (*.nb) files.

The SBML (Hucka et al., 2004) repository ceased work at the end of 2005. The E-Cell project (Takahashi et al., 2003) has plans for its own model repository, however, there is no concrete data available on the internet at present.

Further information on biological pathway databases can be retrieved from the Nucleic Acids Research database issues (Baxevanis, 2000, 2001, 2002, 2003; Galperin, 2004, 2005, 2006, 2007, 2008) and by the Pathway Resource List (PRL)[1] – a database that contains information on over 240 internet pathway resources. Most of these resources are databases themselves containing protein–protein interactions, metabolic reactions, or cellular signaling. The PRL provides resource links and is building up additional information such as the amount of data and the organism coverage within each pathway resource (Bader et al., 2006).

[1]Pathway Resource List: http://www.cbio.mskcc.org/prl/

3.2 Modeling and Simulation Software

The preceding section gave a brief introduction to an overwhelming amount of data repositories present for use in systems biology. The amount of simulation tools dealing with biochemical reaction and diffusion systems is not quite as huge, but is still plentiful. Therefore, this section deals with the description of only the most important applications of this category (see Table 3.2).

A simulation tool is defined as an application performing time series simulation of predefined mathematical models. In contrast a design or modeling tool is applied for building a model graphically. Often simulation tools bring an attached design tool along. If not then models can either be described by markup or scripting languages (Pettinen et al., 2005).

The simulation software tools can be categorized into either deterministic, stochastic, or hybrid (deterministic and stochastic) programs. Other categories apart from the algorithmic approaches are the modeling of either 2D or 3D geometry and the separation of programs into either stand-alone tools or frameworks.

One of the very first programs available for reaction simulations was the GEneral PAthway SImulator (Gepasi) (Mendes, 1993). It translates biochemical reaction equations into coupled ODEs which in turn are then solved numerically. Thus the Gepasi system represents a purely deterministic approach such as BIOCHemical Abstract Machine (BIOCHAM) (Calzone et al., 2006), Cellerator (Shapiro et al., 2003), E-Cell, the Python Simulator for Cellular Systems (PySCeS), and VirtualCell do as well (see Section 3.2.1). The deterministic simulators Genesis and Neuron were originally designed to model neurons and neuronal networks but have shown that cell signaling simulations are just as adequate (Bhalla, 2002). Xyce is actually a deterministic massively parallel simulator for electronic circuits that was used for solving biochemical problems (Schiek and May, 2003). Simulators such as the Stochastic Simulator (StochSim), the MC Simulator of Cellular Microphysiology (MCell) (see Section 3.2.2), and Mesoscopic Reaction Diffusion (simulator) (MesoRD) (Hattne et al., 2005) implement stochastic algorithms only. Examples for hybrid simulators are the Bio-chemical Network (stochastic) Simulator (BioNetS) (Adalsteinsson et al., 2004), the PySCeS (Olivier et al., 2005), WebCell (Lee et al., 2006), and xCellerator.

Very recently, efforts have been made to mix various approaches in order to obtain either the combination of many tools in one software package (Rost and

Designation	Web Site
BIOCHAM*	http://contraintes.inria.fr/BIOCHAM/
BioNetS*	http://x.amath.unc.edu:16080/BioNetS/
Bio-SPICE*	http://biospice.sourceforge.net/
Cellerator	http://www.cellerator.info/
Copasi*	http://www.copasi.org/
E-Cell*	http://www.e-cell.org/
GENESIS	http://www.genesis-sim.org/GENESIS/
Gepasi*	http://www.gepasi.org/
MCell*	http://www.mcell.cnl.salk.edu/
MesoRD	http://mesord.sourceforge.net/
NEURON	http://www.neuron.yale.edu/neuron/
PySCeS	http://pysces.sourceforge.net/
SBW*	http://sbw.sourceforge.net/
SmartCell*	http://smartcell.embl.de/
StochSim*	http://www.ebi.ac.uk/ lenov/stochsim.html
VirtualCell*	http://www.nrcam.uchc.edu/
WebCell	http://webcell.kaist.ac.kr/
xCellerator*	http://www.xcellerator.info/
XmdS	http://www.xmds.org/
Xyce	http://www.cs.sandia.gov/xyce/

Table 3.2: Simulation and modeling environments. This table presents a subset of software tools available today for cellular modeling and simulation. Shown here are 20 out of more than 80 applications. The simulators BioNetS, Copasi, MCell, SmartCell, and StochSim are stochastic applications. The SBW and the Bio-SPICE are actually simulation frameworks rather than programs. The remaining applications within this table are based on deterministic modeling and simulation. A special position take XmdS and BioNetS as they are code generators. The designations superscripted with an asterisk ('$*$') are going to be discussed in more detail in the following sections.

Kummer, 2004), e.g. the Complex Pathway Simulator (Copasi) (see Section 3.2.2) or a tool offering access to many different software packages, e.g. the Systems Biology Workbench (SBW) (see Section 3.2.3). A very special position takes the eXtensible multi-dimensional Simulator (XmdS) and BioNetS as they actually are C++ code generators (Collecutt and Drummond, 2001). If their code is compiled then the resulting programs are simulation applications by their own again. Further information on existing modeling and simulation applications can be retrieved by the SBML Software Guide[2] – a matrix that contains information on software providing support for SBML.

The following three sections give a closer look to ten well reputed applications and two frameworks. The last Section 3.2.4 then defines comparison criteria for the comparison of the ten programs. The 4DiCeS approach, which will be presented in the subsequent Chapter 4, is going to be brought into context with Section 5.3.

3.2.1 Deterministic

This section presents five well known and used deterministic simulation applications. Although Gepasi ceased further development, it is still in use to date and has played a major role in the development of all the other tools discussed here. There are plans to completely replace Gepasi by the newer Copasi (see Section 3.2.2.2). Excepting Gepasi all other described simulators are still under development and have a user community of their own.

3.2.1.1 BIOCHAM

BIOCHAM is a programming environment for modeling biochemical systems, making simulations, and querying the model in temporal logic. It provides a rule-based language for modeling biochemical systems, a simulation engine, and a query language based on temporal logic, Computational Tree Logic (CTL), or Linear Temporal Logic (LTL). A machine learning system is provided for correcting and completing models either by changing rules with respect to a CTL specification or by estimating parameters of an LTL specification. An interface to the symbolic model checker (NuSMV) is provided also. BIOCHAM was initiated by the Constraint Programming group of The National Institute for Research in Computer Science and Control (INRIA) at Rocquencourt, France.

[2] SBML Software Guide: http://sbml.org/SBML_Software_Guide/

3.2.1.2 E-Cell

The E-Cell project is based on international research aiming to model and reconstruct biological phenomena in silico and to develop necessary theoretical supports, technologies and software platforms to allow precise whole cell simulation (Tomita et al., 1997). The E-Cell Model Language (EML), a subset of the eXtensible Markup Language (XML), is used for describing the models. The SBML support was also included to enable a wide cross-platform model exchange (Tomita et al., 1999). The E-Cell project is managed by the Institute for Advanced Biosciences, Laboratory for Bioinformatics, Fujisawa and the Mitsui Knowledge Industry, Bioscience Division, in Tokyo, Japan.

3.2.1.3 General Pathway Simulator (Gepasi)

The GEneral PAthway SImulator is one of the first software packages for modeling biochemical systems. Gepasi simulates the biochemical reaction kinetics, provides a number of tools to fit models to existing data, optimizes the functions of the models, and performs a metabolic control analysis and a linear stability analysis (Mendes, 1993). The application simplifies the task of model-building by assisting the user in automatically translating given reactions into matrices and differential equations transparently (Mendes, 1997). This is combined with a set of numerical algorithms that ensure fast and accurate results (Mendes and Kell, 1998). It was developed at the Virginia Bioinformatics Institute, USA, as a pure deterministic modeling and simulation environment.

3.2.1.4 VirtualCell

The National Resource for Cell Analysis and Modeling at the University of Conneticut Health Center, in Conneticut, USA, created a remote user simulation and modeling application. A general purpose differential equation solver is used to translate the initial biological description into a set of differential equations (Loew and Schaff, 2001). The generated results are stored on a remote server and can be reviewed and exported into various formats. The compartments represent 3D volumetric regions, while the membranes represent 2D surfaces separating the compartments (Schaff et al., 1997; Schaff and Loew, 1999). The geometry may be captured by various imaging modalities, such as wide field, confocal, or electron microscopy (Slepchenko et al., 2002).

3.2.1.5 xCellerator

The xCellerator is the successor to the Cellerator package, which was designed as an interface to Wolfram Mathematica for facilitating biological modeling by automated equation generation. It provides tools for generating, translating, and numerically solving a potentially unlimited number of biochemical interactions (Shapiro et al., 2003). xCellerator solves the complete set of equations predicted by the law of mass action. The package also contains a number of transcriptional regulation models that are not Michaelis-Menten equations. xCellerator may write its results in SBML. The Cellerator package was formerly developed at the National Aeronautics and Space Administration's (NASA) Jet Propulsion Laboratory, California, USA, and is now privately continued.

3.2.2 Stochastic

The simulation applications shown in this section are stochastic approaches (Kibby, 1969). They either implement proprietary algorithms of their own (MCell and StochSim) or make use of simulation algorithms from literature (BioNetS, Copasi, and SmartCell). BioNetS and Copasi are also utilizing numerical solvers in combination to their stochastic algorithms.

3.2.2.1 Biochemical Network Stochastic Simulator (BioNetS)

BioNetS was developed at the University of North Carolina at Chapel Hill, USA. It was designed to simulate biochemical network models in a hybrid, stochastic and deterministic manner. The type of used discrete or continuous random variable for each chemical species in the network can be specified individually. The package was implemented to efficiently scale with any network size to allow the study of large systems. BioNetS is available as a stand alone package but runs also as a Bio–Simulation Program for Intra- and Inter-Cell Evaluation (Bio-SPICE) (see Section 3.2.3.1) agent. The output of the software is portable C/C++ code that may be compiled and run on any system with the appropriate compiler (Adalsteinsson et al., 2004).

3.2.2.2 Complex Pathway Simulator (Copasi)

The Copasi project is based on Gepasi and a program for the automatic parsing and STochastic simulation of ODEs (STODE)[3] (van Gend and Kummer, 2001). Copasi incorporates a model generator, user-friendly visualization platforms, optimization routines, methods from non-linear dynamics, and different simulation techniques. Copasi is supervised by Pedro Mendes (Bioinformatics Institute, Virginia, USA) along with Ursula Kummer of the European Media Laboratory, Heidelberg, Germany. This application is planned to enable the simulation of complex metabolic processes in cells without having to master complex mathematical or computer skills (Rost and Kummer, 2004).

3.2.2.3 MC Simulator of Cellular Microphysiology (MCell)

MCell is a modeling application for simulations of cellular signaling in complex 3D subcellular micro-environments. Optimized MC algorithms are used to track the stochastic behavior of discrete molecules in space and time. These particles diffuse and interact with other heterogeneously distributed molecules within the 3D geometry (Bartol Jr. et al., 1996). All simulation components are defined by using a specific programming language called Model Description Language (MDL) (DeSchutter and Cannon, 2000). The project was initiated by the Computational Neurobiology Laboratory at the Salk Institute for Biological Studies, in California, and by the Pittsburgh Supercomputing Center's working group for Biomedical Applications, Pennsylvania, USA.

3.2.2.4 SmartCell

SmartCell is a general tool for modeling and simulation of reaction and diffusion pathways within cells. It supports diffusion and localization by using a mesoscopic stochastic reaction model. The SmartCell package should handle various cell geometries, allows the localization of species, supports desoxyribonucleic acid (DNA) transcription and translation, membrane diffusion and multi-step reactions, as well as cellular growth. Moreover, different temporal and spatial constraints can be applied to the model (Ander et al., 2004). It is expected to provide a suitable model description format (Nasi, 2004). SmartCell is a project of the European Molecular Biology Laboratory, Heidelberg, Germany.

[3]STODE: http://atlas.villa-bosch.de/bcb/software/Carel/

3.2.2.5 Stochastic Simulator (StochSim)

StochSim was written by Carl Firth as part of his PhD work at the University of Cambridge (Morton-Firth, 1998). It was developed as part of a study of bacterial chemotaxis as a more realistic way to represent the stochastic features of this signaling pathway. It is able to handle large numbers of individual reactions encountered (Morton-Firth and Bray, 1998; Morton-Firth et al., 1999). The application consists of a platform-independent core simulation engine. The program encapsulates the algorithm described above as well as a separate graphical user interface. StochSim represents individual molecules or molecular complexes as individual software objects (Novère and Shimizu, 2001).

3.2.3 Frameworks

This section introduces frameworks rather than applications in comparison to the two previous sections. Such frameworks are developed to accomplish very particular problems. They provide entire workbenches of interfaces that allow for the integration of many tools and methods to interact with each other.

3.2.3.1 Bio–Simulation Program for Intra- and Inter-Cell Evaluation (Bio-SPICE)

The Bio-SPICE toolkit was developed to model and simulate cellular dynamic networks. Contributed modules are organized in the Bio-SPICE dashboard, which is a graphical user environment (Sauro et al., 2003). It permits data sources, models, simulation engines, and output displays provided by different investigators, and running on different machines, to work together across a distributed, heterogeneous network (Garvey et al., 2003). Among several other features, the environment enables users to create a graphical workflow by configuring and connecting available Bio-SPICE components (Kumar and Feidler, 2003a,b). The project was initiated by the Defense Advanced Research Projects Agency Information Processing Technology Office, Virginia, USA.

3.2.3.2 System Biology Workbench (SBW)

The SBW enables different tools to interact with each other. The framework supports tools written in different programming languages, which may run on different platforms and physical machines (Sauro et al., 2003). The aim is to facilitate collaboration among developers of systems biology software. Developers should find it easier to build an SBW interface than to recreate functionality. They can then concentrate on developing best-of-breed solutions in the areas where they have special expertise (Hucka et al., 2002). Both SBW and SBML are being developed in collaboration with several groups developing simulation packages as described in the last two sections.

3.2.4 Comparison

While the described simulation tools thus have their benefits, none have so far addressed all the currently emerging research problems. The efforts in the field of cellular simulation can be roughly categorized as stand-alone modeling and simulation tools or extendable frameworks. The first can then be divided again into either more or less strict deterministic or stochastic methods. Even though the stand-alone tools often provide software interfaces, frameworks broaden this ability to include new technologies to the system.

Specific interests of research groups often have great influence on the development of simulation applications. The usability of a tool is highly affected by user requirements (Schwehm, 2001). Here the chosen operating system and the selection of a Graphical User Interface (GUI) versus a scripting or batch mechanism are key features. Programs currently used vary noticeably in their applicability for specific types of modeling (Pettinen et al., 2005). The general usability indicated by the user's learning curve and application-provided model designers have here a great impact. Well featured textual or graphical utilities often aid the modeling process significantly. The number and quality of the utilizable algorithms, supported model-exchange formats, and additional features round up such user requirements dramatically. Next to this, it is of major importance to have a tool with reliable and precise results at optimal performance. An extension mechanism for the integration of new features and functionality is then crucial. The support of spacial modeling information is necessary for close to reality simulations. And last but not least it is important to segregate parts of the model and handle such parts differently.

The following Table 3.3 will therefore define 12 comparison criteria extracted form the preceding paragraphs. These criteria are applied to the previously introduced related applications in a three-state manner. The three states range from unsatisfactory to sufficient. This will allow for an easier qualification of the differences among the tools. Table 3.4 is then going to summarize all comparison findings from that definition.

Criteria	Description
Accuracy	Simulation results are reliable and precise.
Concurrency	Various algorithms can be handled in concurrency.
Designer	Modeling is supported by textual or graphical utilities.
Exchange	Models are exchangeable among other applications.
Extendability	The possibility to add new functionality to the system.
Features	Additional components of scientific interest.
GUI	The availability of a GUI for modeling and simulation.
Methodology	Algorithms are deterministic, stochastic, or hybrid.
Performance	The application's optimization for efficiency.
Scripting	The existence of an automating scripting mechanism.
Spaciality	The support for any sort of spacial information (2D or 3D).
Usability	Users' scientific needs and smoothness of learning curve.

Table 3.3: Comparison criteria. The defined comparison criteria were extracted from an objective analysis of state-of-the-art simulation applications and common software quality assurance considerations. They are further on used in a three-state manner (unsatisfactory, average, and sufficient). Table 3.4 will display the three states as circles from empty to filled.

As can be seen there is not one application making up for all defined comparison criteria. The concurrency feature is left out, because it is not supported by any of the comparison candidates. When weighting the three-state [unsatisfactory (○), average (◐ and ◑), and sufficient (●)] with zero (0), a half (0.5), and one (1) then the given applications range from four to eight criteria points. A new application as of 4DiCeS (see Section 5.3.3 for a comparison) should at least have an equal or even higher level to the best to keep up with or even outperform the state-of-the-art.

The following Chapter 4 will now formally describe the 4DiCeS approach.

	BIOCHAM	BioNetS	Copasi	E-Cell	Gepasi	MCell	SmartCell	StochSim	VirtualCell	xCellerator
Accuracy	◐	●	●	●	●	●	◐	●	●	●
Designer	○	◐	◐	◑	◐	○	◑	◐	●	◐
Exchange	◐	◐	●	●	◐	○	◐	◐	●	◐
Extendability	○	◐	○	◐	○	○	○	○	○	◐
Features	○	◐	●	●	●	◐	○	○	●	◐
GUI	●	●	●	●	●	○	●	●	●	◐
Methodology	○	●	●	○	○	◐	●	◐	○	●
Performance	◐	●	●	◐	●	○	◐	○	◐	◐
Scripting	●	○	◐	●	○	●	○	●	○	◐
Spaciality	○	○	◐	◐	○	●	●	◐	●	○
Usability	◐	●	◐	●	●	○	●	◐	◐	◐

Table 3.4: Comparison of simulators. Filled (●) circles are generally superior to their half-full (◐) counterparts. Empty (○) circles indicate the need for improvement or a total absence. In methodology hybrid approaches are considered most sufficient. Designers are either defined as textual (◐), graphical (◑), both (●), or none (○) at all. Concurrency was omitted, due to missing support by any of the comparison candidates.

CHAPTER 4

Definitions and Implementation

Based on the criteria described in the previous chapter with respect to already existing solutions, a general concept has been designed for the implementation of a 4D Cell Simulator (4DiCeS). The system is developed as a common and extensible framework flexible enough and well suited to serve as a platform in systems biology. The four dimensions describe here the utilization of the three space coordinates and a time axis. Cellular phenomena can therefore be simulated in a 3D geometric environment over time. The current project logo is presented in Figure 4.1. It consists of four rolling dice that represent the ability to model and simulate stochastically in 4D space.

Figure 4.1: The 4DiCeS logo. The four rolling dice stand for modeling and simulating by the use of 4D stochastic methods. Also, the dice could be compared to the cubes of the 3D geometry used with 4DiCeS.

The aim of this project is to provide a modeling and simulation application to the user that is most adaptive in its functionality. The main features can be roughly summarized as seven main design characteristics (see Table 4.1).

The first four features of Table 4.1 have a great impact on the modeling and simulation core itself. First of all 4DiCeS provides a 3D simulation geometry that makes use of either imported or user-defined cellular compartments (Oleson et al., 2003). Thus modeling and simulation will be based on such a geometry that in turn can then be easily reduced to a 2D or one-dimensional (1D)

Feature	Description	Section
Geometry	Enabling the import and the utilization of 3D cellular boundaries over time as a basis for simulation.	4.2.1
Solvers	Facilitate diverse types of algorithms including both deterministic and stochastic methodologies.	4.2.3
Algorithms	Allowing for both reaction and diffusion algorithms within the same simulation model and geometry.	4.2.3
Concurrency	Offer the possibility of running different algorithms in concurrency in the same simulation environment.	4.2
Import and Export	Provide a platform of different import as well as export filters to well-known model storage formats.	4.2.2
User Interfaces	Permit the variable access to the system through different user interfaces as of GUIs and batch processing.	4.2.4
Extensions	Make the entire system easily extendable by other programming as well as scripting languages.	4.2

Table 4.1: 4DiCeS feature list. These seven features describe the main goal of the design and implementation of 4DiCeS. An eighth feature is the definition of a strong and well-defined software interface system (see Section 4.2). This then should give the application users a modeling and simulation utility that can be easily adopted or even extended to their needs.

model if needed (Möller et al., 2002). Possible sources for importing geometrical data could be either cellular image stacks such as from Confocal Laser Scanning Microcopy (CLSM) and Multi-Photon Fluorescence Microscopy (MPFM), or topological information such as from Scanning Electron Microscopy (SEM) and Atomic Force Microscopy (AFM). Such data has to be adapted to allow for entirely sealed compartments on which algorithms can work on within simulations. This already implies other features as well. Both reaction and diffusion algorithms should be applicable to the system (Möller et al., 2003). Diffusion must work for either freely diffusible particles or for membrane-bound particles. This means that it is either possible to apply a free 3D, or a lateral 2D membrane diffusion to simulated particles (Oleson et al., 2002). Additionally definable boundary conditions are essential to reaction and diffusion by other means as well. It must be possible to have different reaction-diffusion systems in unequal compartments of the same simulation environment. There is also the need for particles to undergo transmembrane diffusion under definable conditions. As another feature all reaction and diffusion algorithms can be of diverse types including either deterministic or stochastic methods. In doing so it should then be possible to concurrently run different algorithms on varying compartments. Single compartments can be handled deterministically whilst others can use stochastic methods in the same simulation environment (Oleson et al., 2006).

The remaining three requirements of Table 4.1 give more attention to the usability of the application as is. In this case it is important to provide diverse import and export filters of well-established storage formats such as CellML and SBML (see Section 4.2.2). This should enable the exchange of existing models with the 4DiCeS system and then back to model repositories again. Thus simulations should run on either GUI-based graphical frameworks or as Command-Line Interface (CLI)-based (i.e. textual) batch jobs. This allows for both high-throughput simulation applications and experimental model evaluation simulations (Oleson et al., 2004). The whole architecture must also enable other programming or scripting languages to effortlessly extend the 4DiCeS system. All these requirements depend on a well-defined plug-in mechanism that can be easily maintained and used by the system's core. This again is beneficial for third party programmers because they may also use the same plug-in concept to integrate their own ideas readily (Oleson et al., 2006).

It has to be noted that within this work the word 'interface' is mainly used for connection sockets between different parts of the application. In the case of human-machine interfaces the term 'user interface' is going to be applied.

The following sections will now subsequently describe the 4DiCeS application in a top-down manner. In connection with the general overview (see Section 4.1) the implemented system will be eventually broken down into its functional parts. These functional units are then discussed at more detail later on. Anyway, each level of detail is going to comprise an overview, its dynamics, and dependencies (see Section 4.2); therefore every subsection will define the environment of the level of detail, its components, the possible states, and the error-case handling, if applicable. This chapter finally closes with remarks to the deployment, the applicability, and the availability of the 4DiCeS system to the public. This includes specific programming design decisions made for applied programming languages, third party APIs, and further technologies. Noteworthy technicalities to concrete implementations and a general complexity consideration are then also referenced to Section 4.3.

The formal description of the 4DiCeS application presented in this chapter makes extensive use of the Unified Modeling Language (UML). The UML diagrams and types used in this work are therefore described in Appendix C in further detail.

4.1 General Overview

This section will examine the general structure of the 4DiCeS framework. Both UML use case studies (see Appendix C.1) and UML static program structures (see Appendix C.2) provide an introduction of the overall architecture and features of the system. In doing so it is thereafter easier to comprehend with the system's dynamics.

4.1.1 Survey of Integral Parts

The primary goal for 4DiCeS is to provide a modeling and simulation environment capable of interpreting a given data model and running simulation iterations therewith. For this purpose the application framework was divided into functional pieces. Whereas the central part of the program is the 4DiCeS kernel, which manages incoming as well as outgoing events and monitors the stability of the system and all its units.

To allow for reusability, portability, and thus effectiveness of the 4DiCeS application, a consistent system of kernel interfaces (see Section 4.2) was established. These kernel interfaces offer a centralized gateway to and from simulation algorithms, model data, and the actual user interface. The general concept is displayed in Figure 4.2 for clarification, which displays a static UML class diagram (see Appendix C.2).

4.1.1.1 Application Kernel

The 4DiCeS kernel provides functions for creating a simulation model independent of a specific application. Through the user interface, either the user or another program takes access to these functions. Input and output devices form a module, which is linked by the module interface with the kernel. The

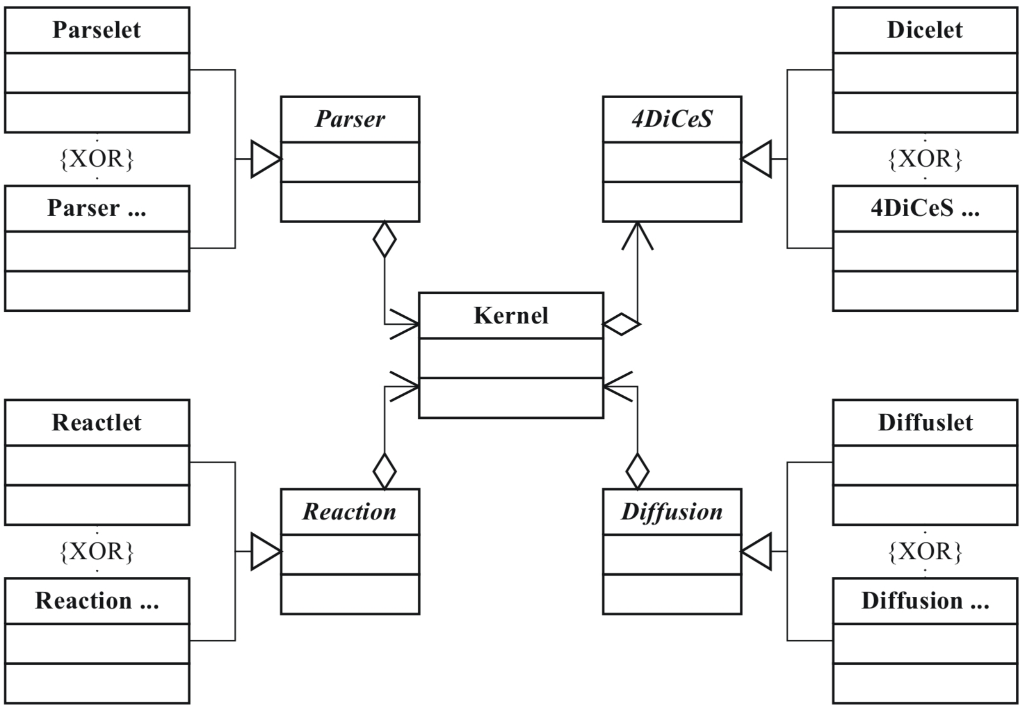


Figure 4.2: Established interface system. The kernel, as the central part of the simulator, is surrounded by interfaces for simulation algorithms, model data input and the user interface integration. The extending modules to all interfaces are programming language independent. First implementations include the use of Java modules as can be seen from the Java typical names ending in '-let'.

kernel maps its functions at the user interface to the various capabilities of the different modules. Then the realization of the device independent from the requirement of single modules is accomplished by individual module drivers. The kernel interfaces are defined independent of special programming languages. The utilization of the underlying kernel therefore requires an adaptor interface instance for particular programming or scripting languages. This can be easily illustrated by the separation of the kernel into layers (compare to Figure 4.3). Application-dependent layers may then build upon any individual programming language layer.

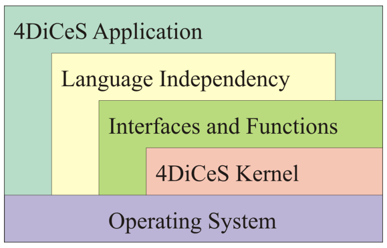


Figure 4.3: Kernel layer model. Very similar to an operating system layered model, the 4DiCeS kernel is separated into layers of different disposability. This structure allows for connecting various programming and scripting languages to the functions and interfaces provided by the kernel.

4.1.1.2 Data Model

All utilized data structures of 4DiCeS are integrated as the 4DiCeS application data model. The data model itself is not used for a simulation model representation directly but to a much bigger extent for testing and simulation purposes. This scheme follows the overall aim to understand, visualize, and experiment with the underlying workflow of the application. The data model does not necessarily include implicit simulation model information. Such can rather be extracted out of the application's data model if the components from Table 4.2 are given.

4.1.2 Function Classes

To allow for any state-changing functionality within the kernel functions are required that address the design elements, attributes, modules, data structures as well as formats, and the human-machine interaction. All error conditions are processed by the kernel with its own exception handling and logging mechanism. The description of kernel functions includes the function identifiers, names, numbers, data-types, and the meaning of every parameter. Additionally the functions' effect(s) and possible error conditions are contained also. An effect of a specific kernel function could be either to change the underlying model, to return values to the user, to change the kernel's state, or a mixture of all three.

The state of the kernel is given by values from a set of state lists that are associated to the kernel itself, to modules, and to segments. The data structure of the state lists with its data types, its intent, and the preallocation of its

Component	Description
Basic Data	Basic data structures holding the information about the model itself, used algorithms, and the model geometry.
Dependencies	All fundamental links among issued data structures as well as data dependencies within closed data structures.
Identifiers	Applied names as well as other criteria for the identification of data structures and their direct dependencies.
Precepts	General rule sets for the assembly as well as storage of structured data, its dependencies, and its identifiers.

Table 4.2: Extraction of simulation model information. Since the application's data model does not necessarily include the simulation model itself there must be a way to extract such information. If the basic data is given with its dependencies, identifiers, and data extraction rules the simulation data can be extracted explicitly from the application's data model.

contained values are defined by kernel specifications. To avoid the inclusion of the full set of functions provided by the kernel, all functions are organized into levels of duty according to their increasing requirements. They are built as matrices of three levels for the input, the output, as well as the segmentation.

A brief overview of the kernel functions' classes is presented in the following. In subsequent sections, significant functions and interfaces will be considered in greater detail. The realizations of concrete modules will then be described in Section 4.3.

Design Elements: Design elements are the elementary components from what a simulation model is constituted of. In 4DiCeS there are particles, reactions, kinetics, and compartments. Particularly for 3D modeling, the design elements voxel and grid matrix are of special importance. A voxel is an indivisible and therefore distinct volume element of the overall grid matrix. The grid matrix is the simulation space. For addressing platform or language dependent (i.e. non-standardized) functions, the "general modeling primitive" is defined.

Design Attributes: A design element can be characterized by its geometrical form but also by its appearance. This is described by a set of design attributes. For voxels these can be position, size, volume, and their particle incorporation.

The grid matrix offers attributes that depict the grid type, number of voxels, and compartment details. Then the compartments are segmentations of the grid matrix providing details on the voxels owned and boundary conditions.

Modules: A significant component of the 4DiCeS architecture is the concept of interface modules. These allow for language-independent programming of simulation applications and plug-ins. A module is an abstraction of interfaces to the 4DiCeS kernel implementation. Also modules can be processed in sequence or in parallel. A design element instance may vary in display and in execution according to the special abilities of a connected module. The application is able to request features of a connected module through an underlying event-handling mechanism.

Model Structure: Interactive applications also require the capability for manipulating parts of the given model structure. There must exist opportunities to handle all design elements as separate entities when constructing a model. Therefore, a model can be composed of sub-models, "segments". Segments may be individually manipulated, deleted, and processed. 4DiCeS manages segments from within its segment repository.

Human-Machine Interaction: An application can request input values from a user through a module. 4DiCeS is currently aware of five classes that can be connected dynamically to the system (see Table 4.3).

Interface	Description
Geometry	Set of models of geometric simulation space segmentations.
Activity	Applied particles and their activities with the simulation.
Reaction	List of used reaction algorithms during simulation.
Diffusion	Used diffusion algorithms within the simulation spaces.
Interaction	Interfaces to either users or other connected applications.

Table 4.3: Module classes. The input description for both geometry as well as activity, the algorithm description for both reactions as well as diffusion, and the programmatic interaction interfaces provide the flexible base for designing models in 4DiCeS. They are designed to be accessed independently of programming language, and to run concurrently when appropriate.

The flexible control of these module classes by the kernel in combination with the segmentation provide the base of designing models without a loss of programming language dependency. The concurrency of algorithms gives then the basis for the segmentation of models and their simulation in parallel.

Model Format: 4DiCeS contains an interface to the model, which allows for both the long-term storage as well as the reload of models. The model file can either be used to persist models, for their transmission, or for a resume of a previously interrupted simulation session. Models are stored including their segmentation structure.

Event-Handling: All values in the tables, which describe the system's state, can be retrieved by calling functions. There exist both mechanisms for actively calling for information (synchronous events) and obtaining data from a model automatically (asynchronous events) when a certain trigger condition is reached.

Error-Handling: If an error occurs during the execution of a 4DiCeS function, then the kernel is set to a special error-handling state. Depending on the severity of the error, an error-handling routine of the used module can be invoked, or the kernel can try to solve the fault by itself. In either case an error-log file will be produced for further maintenance purposes.

4.1.3 Use-Case Descriptions

In the following section a use-case scenario is presented for the general use of the application. The input as well as the output, the simulation handling, and the algorithm management will be emphasized. All these components will then be described in more detail later on.

4.1.3.1 User Perspective

A general approach to 4DiCeS is the description of its main features by a use-case scenario that shows the user's perspective of the system. Figure 4.4 presents a general overview UML use-case diagram of the simulator, while Table 4.4 provides further details to the specific use-case.

Here the UML use-case actor is a potential user of the simulation program. This user has a direct interface to three essential use-cases to the system: the simulation engine, the input data, and the output data. In the center of the diagram

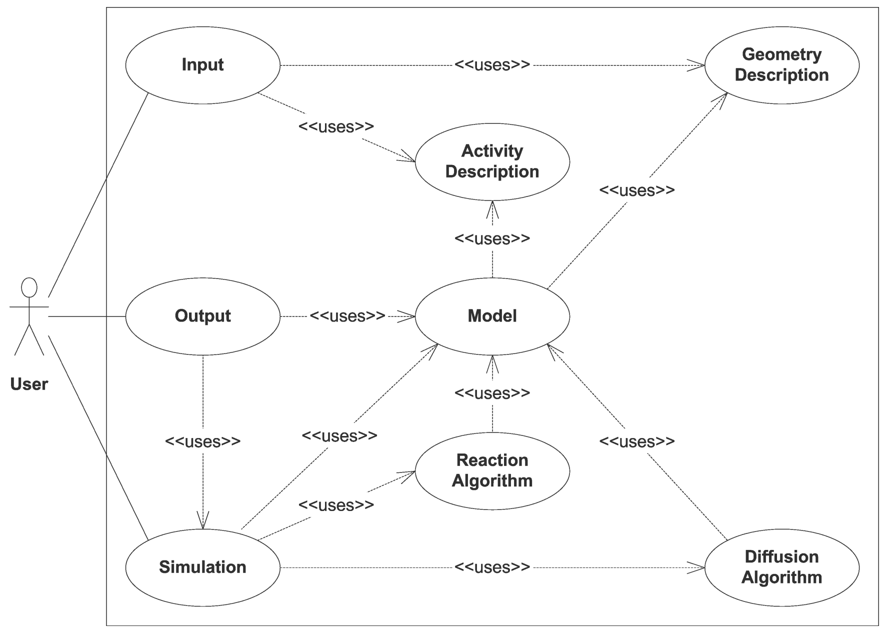


Figure 4.4: Use-case: User perspective.

Actors:	• User • User Interfaces
Objects:	• Input descriptions (geometry and activity) • Algorithms (reaction and diffusion) • Model • Simulation • Output
Requires:	• Initialization of model by input descriptions • Initialization of simulation with used algorithms • Simulation iterations for output
Assumptions:	• Input is syntactically and semantically correct. • Used algorithms terminate. • Simulation need not terminate. • Order of requirements is preserved.
Error Cases:	• Malformed input description(s). • Internally processing algorithm(s). • Order of requirements not meet.

Table 4.4: Use-case: User perspective. Both this table and Figure 4.4 show the user's perspective of the 4DiCeS system. The figure displays a UML diagram, while this table provides further information about the requirements, assumptions, and possible error-cases on the use-case.

there is the application data model, which is fed with information from both the descriptions of the geometries as well as the activities that were uploaded by the actor of the system. A geometry description is a 3D mesh of compartments subdividing the simulation space into distinct vessels for both reaction and diffusion events. These events in turn are defined by the activity description. The upper part of the use-case diagram represents the model description and definition part of the application. The lower part on the contrary shows use-cases utilizing these model boundaries for further simulation. Both reaction as well as diffusion algorithms perform using the given rules and produce output data in turn.

The user is required to feed the simulator with model definitions and algorithms before a simulation can be started. Then the production of output depends on at least one successful iteration of the simulation. Another point of possible conflict is the dependencies between model description and simulation description. The definition files must either be general enough to cope with diverse types of algorithms or preferred algorithms must also be defined. The precise location of every algorithm used has to be defined before the simulation is attempted, as is stipulated with the criteria that algorithms may run concurrently.

4.1.4 Program Dynamics

This section provides a more in-depth examination of the overall program's dynamics by highlighting the previously introduced user's perspective through a UML sequence diagram (see Figure 4.5 and Appendix C.3).

From the user's perspective, the application data model first must be constructed and initialized. The construction (see Figure 4.6) consists of the loading of data files required in the subsequent simulation process. The data that is added to the model is optional but highly dependant on the simulation's purpose. Here it is defined which model segments are used as modules for the description of the geometry and the activity.

The initialization (see Figure 4.7) of the model then links the previously loaded data together. This makes sure that every segment of the model knows which data has to be applied and how it is going to be used. After the linking of data has terminated, the simulation engine is instantiated and further connected to the data model. This accounts for possible errors that could occur, if the data was either incompletely or improperly linked together. Therefore, the initialization is also a data-verification and data-linkage verification mechanism.

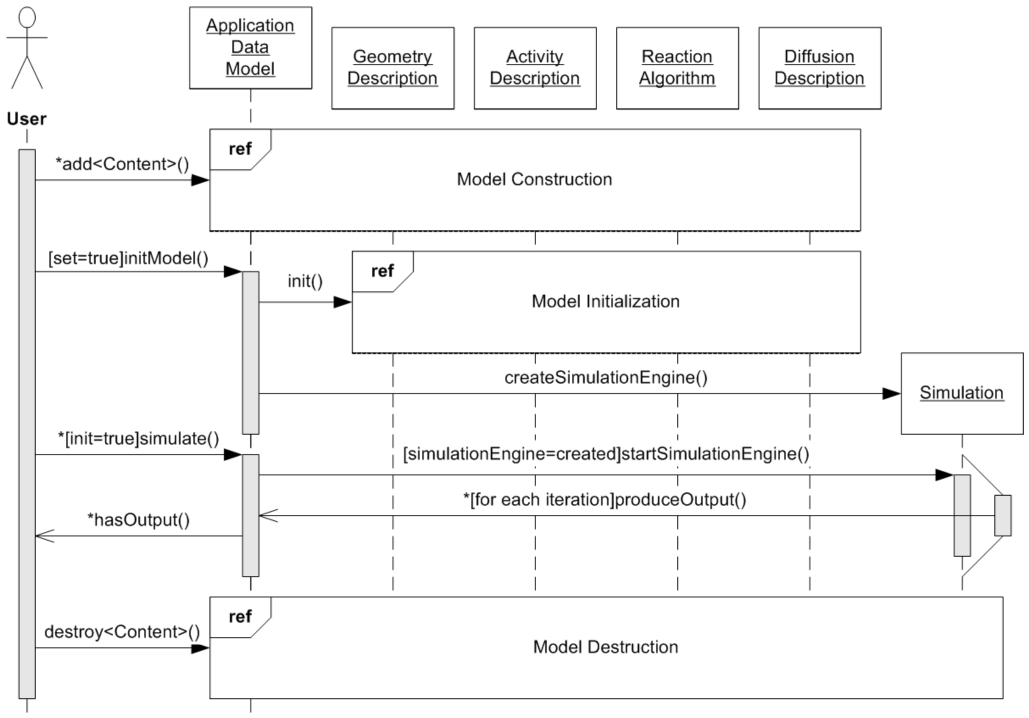


Figure 4.5: Sequence diagram: User perspective. The given figure is a UML sequence diagram representation of the user's perspective use-case as described in the previous Section (4.1.3.1). Here, the function calls for the creation of data objects and the triggering of a simulation are outlined. It can be seen that in addition to ordinary function calls with a synchronous return of approval, there is an additional incidence of an asynchronous return event: This repetitive event call states the availability of simulation output data that can be collected by the user. The model construction (Figure 4.6), initialization (Figure 4.7), and destruction (Figure 4.8) are out-sourced for better clarity and readability.

Hence a simulation can only occur if the initial data is complete and properly linked.

If the model was loaded and found acceptable by the initialization routines, the actual simulation can begin. By doing so the simulation engine is invoked to iterate over the given data through the use of the given algorithms in defined time steps (see Figure 4.5). After each iteration output data is produced which can be further used as a base for the next iteration cycle and is returned to the

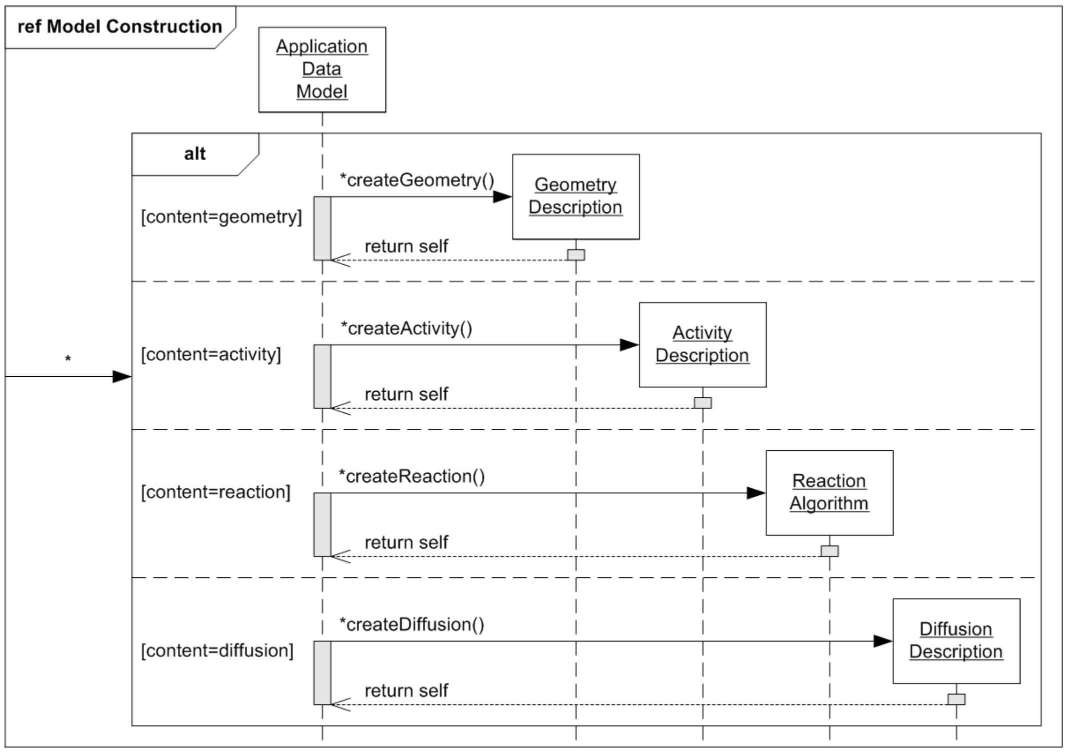


Figure 4.6: Sequence diagram reference: Model construction. The construction of the application's data model consists of the addition of data segments as of the geometry description, the activity description, reaction, as well as diffusion algorithms. The adding of such segments is optional and highly depends on the purpose of the simulation. At this stage the data is loaded but not further connected to each other. This connection process will be executed during data initialization (see Figure 4.7).

user for subsequent analysis. To ensure that the running simulation does not block any user interactions, it is set up as a separate thread. Hence the user is informed of newly created output data via asynchronous calls from that thread. The output data itself is continuously saved to the connected permanent storage of the systems hardware.

Next either the user is able to stop the simulation manually or simply waits until the previously defined number of iterations have been processed and the simulation engine auto-terminates. With either suspension mode, the user is able to continue simulation since the output data can be reused as new input

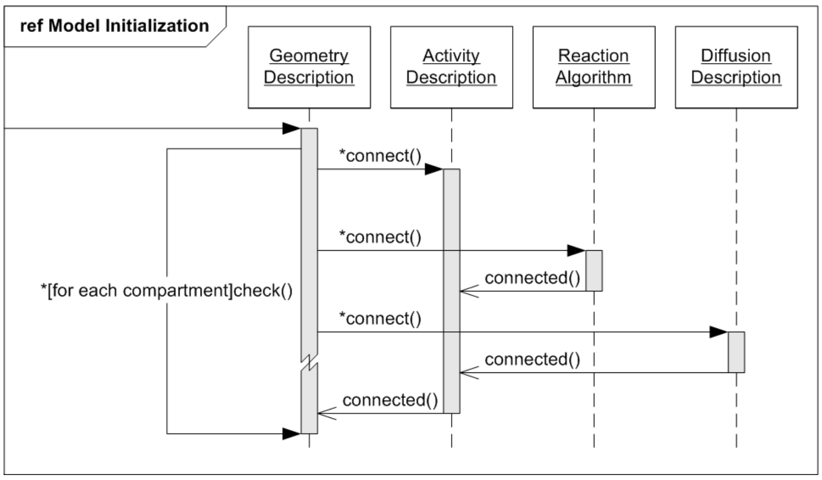


Figure 4.7: Sequence diagram reference: Model initialization. After the application model data was loaded and set to the program's internal data sets it has to be connected to each other. This is done iteratively by searching the geometry descriptions for compartments that give further clues on what other definitions have to be linked.

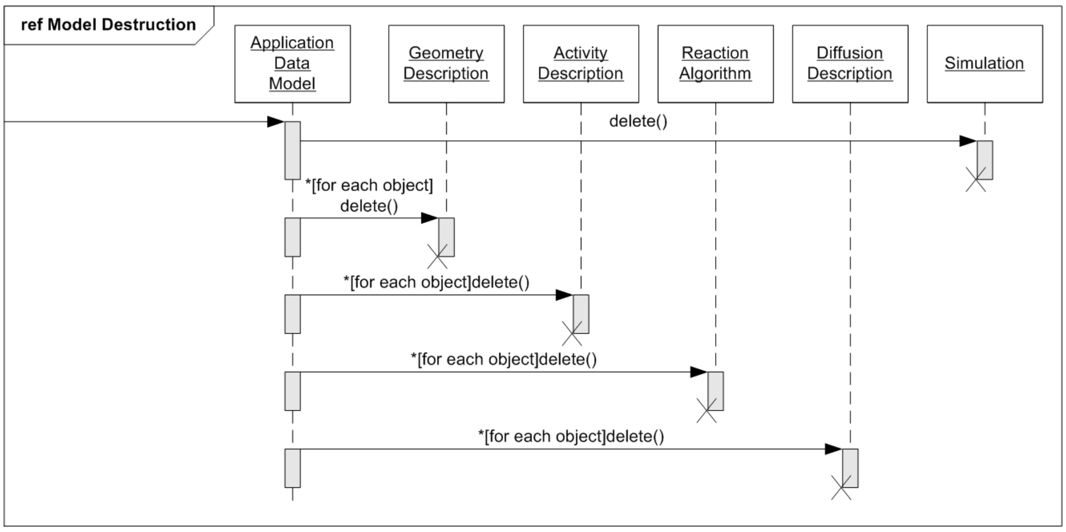


Figure 4.8: Sequence diagram reference: Model destruction. If the currently loaded model is no longer needed, it has to be destroyed before other (new) models can be loaded. The process of destruction is highly dependant on the chronology of what object has to be deleted first. It is of major importance to remove the simulation engine instance first and then destroy the input description and the algorithms in this order. Doing so ensures that the data is removed from its linkage dependencies safely without creating broken connections that may trigger undesirable memory leaks.

data. Here no reinitialization is necessary because all data remains complete and linked until model destruction.

The destruction of data (see Figure 4.8) has to be then performed to avoid possible memory leakage or the overcrowding of physical memory. The destruction is an ordered process of chronological data model segment instance removal. First the simulation engine is destroyed since the destruction will finally invalidate the given data thereby making the simulation engine obsolete. At this point the data is removed from top to bottom by first deleting the input descriptions and then also the loaded algorithms from memory. It should be noted that no data is finally lost at this point due to the automatic saving of output data by the simulation engine. If the user decides to remove such stored files as well, then the data is definitely lost and can only be reproduced by a repeated simulation.

4.2 Considerations and Definitions

As the general concept of the 4DiCeS framework was displayed already, this section is concerned with the design and definition of the interfaces. Details regarding how 4DiCeS offers its implemented modularity as combined with cross platform compatibility will then be further described in Section 4.3.

The design of the framework specifies that a plug-in module must be chosen and connected to the kernel for the system to run successfully. All interfaces are defined in a modular way in order to provide greater exchangeability of algorithms and to facilitate hassle-free software updates.

Currently there are four different interfaces to the 4DiCeS kernel (see Table 4.3). They allow for connections of modules that handle diffusion and reaction algorithms, the parsing of description files, and describe the connection to user interfaces. The dynamics of the interfaces facilitate the integration of new methods and algorithms not yet implemented. Additionally it is thus more easily possible to exchange old modules for new versions. The potential for extending the interfaces to other programming languages as well as other applications is significant. The plug-ins could also integrate sockets to allow for network distributed computing or the remote diagnostics of running processes.

The following subsections characterize the current implementations for modeling of 3D geometry (Section 4.2.1), for handling activity descriptions (Section

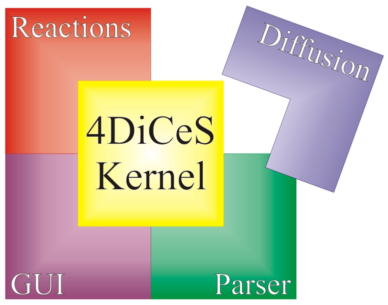


Figure 4.9: 4DiCeS modules. The kernel of 4DiCeS comprises four interfaces to modules that have to be engaged before a simulation is able to run. These ports allow for the integration of various reactions as well as diffusion algorithm modules, model (activity and geometry) parsers, and for user interactions.

4.2.2), and for utilizing algorithms (Section 4.2.3). Thereafter a subsection describing the user interfaces will consider the momentary connections between 4DiCeS and the user (see Section 4.2.4). The underlying plug-in design details are then discussed in Section 4.3.

4.2.1 Geometry Model

Before considering the actual geometry implementation of 4DiCeS the following section briefly introduces fundamental 3D geometry characterization descriptions. Actually, there exist several techniques to present 3D geometries. But common representation schemas for modeling of 3D objects can be divided into three main groups (see Figure 4.10): 'wire-frame models', 'surface models', and 'body models' (Encarnação et al., 1997b).

Wire-Frame Models: Among the 3D model types described above the wire-frame model requires the least information for representing 3D bodies. At large, its structure elements are limited to outline elements as of 'straight edges', 'circular arcs', or 'spline curves'. These elements are not related to each other within this model type. Hence, an assignment of such elements to surfaces is not defined. Therefore, wire-frame models can be considered simple in that data on the surface and interiors are lost in the sense that the wire-frame model no longer describes that information. Hence they do not qualify for a representation of body models due to their incomplete and ambiguous presentation. In many cases algorithms on wire-frame models demand for an intensive interac-

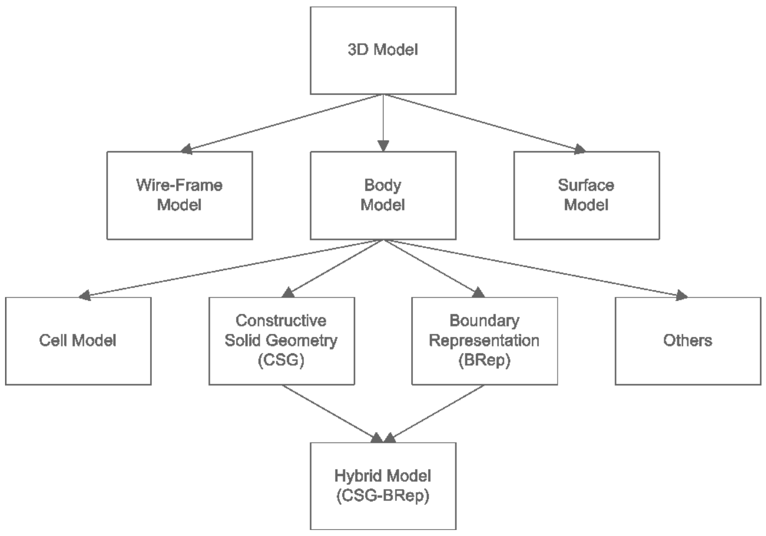


Figure 4.10: Representation schemas for 3D models. Both the wire-frame and the surface models allow for only an visually-interactive user interpretation. They are not complete and distinct in their representation of 3D objects. On the contrary body models provide a comprehensive description of a 3D object, and can be interpreted algorithmically. There are many representations for body models. The most commonly used models are the cell model, CSG, BRep, and hybrids between the CSG and BRep representations. Adapted from Encarnação et al. (1997b).

tion with the user, who then has to specify corresponding surfaces interactively.

Surface Models: Surface models comprise the creation of 3D objects where the facing and its attributes as for bend, torsion, and smoothness are of major importance. Therefore the main content of information lies in single face descriptions. Again there is no correspondence between such faces. In particular relationships to neighbors are not included to its data representation. Such a surface modeler is not qualified for presenting proper body models. A classification of one point in space according to an object cannot be dealt with. That is why it is hard to decide if a point lies inside or outside of a displayed 3D object. To come to such a decision, normal vectors have to be applied facing to, and therefore defining, the outside. Surface models without such orientated faces are neither distinct nor complete. Likewise surface models are better than wire-frame because more data is included. This data is however not a comprehensive representation of the information it is meant to embody.

Body Models: The term body models stands for many representation schemas (see Figure 4.10). Body models form a complete description of 3D objects and can be interpreted automatically by a program. Due to complete storage of the body's 3D representation geometry, the question can be addressed algorithmically. As another positive feature the consistence of objects can be ensured with algorithmic manipulation, as the result of an operation (e.g. the combination of two objects) gives a valid object again. The following geometry considerations will deal with body models only.

As for the current implementation of 4DiCeS, body models were applied as the geometrical description of a simulation space. Therefore the following Section 4.2.1.1 will go further into illustrating the general simulation space definition in 4DiCeS. The proximate Section 4.2.1.2 will then provide details on the reference implementation of this body model, and how it is handled by the 4DiCeS kernel. The activity description (see Section 4.2.2) then formally describes the interaction of both the currently supported geometry data structure and activity models in detail.

4.2.1.1 Simulation Space

As can be seen from Figure 4.10, there are several different approaches of modeling a 3D geometry. As the most commonly used representatives of such techniques are BRep, the CSG, and the cell model, they are going to be briefly described. A specialization of the cell model was used for the underlying simulation space geometry of 4DiCeS and is going to be laid open in detail with the upcoming section.

Boundary Representation: A body model can be described explicitly by its surface and an associated topological orientation. Because of this topological orientation, every point is well-defined with regard to whether it lies inside or outside of a plain. Boundary Representations (BReps) use this fact, and describe 3D objects by their surface. Polygon-oriented data-structures as edge-oriented, node-oriented, or winged-edge data-structures can be applied for the storage of BReps.

Constructive Solid Geometry: With Constructive Solid Geometry (CSG), 3D bodies are described by trees of Boolean operators and primitives. These are also called CSG-trees. Each primitive can then be defined by either BReps or half-space models. The display of CSG-objects as trees is closely linked to the construction of the objects. The nodes of the tree represent regulatory Boolean

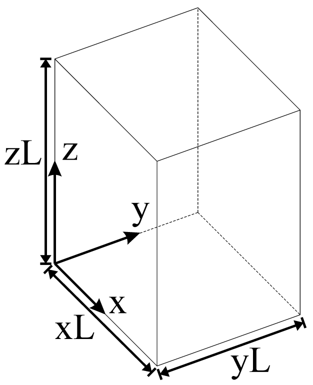


Figure 4.11: Model primitives. As for any body model primitives, the dimension parameters (here a cube with: xL, yL, and zL) as well as the position of the point of origin (x, y, and z) define a primitive unambiguously. Adapted from Encarnação et al. (1997b).

set operations or transformations in space. The leaves refer to the primitives. A data-structure would make use of the tree-structure and the particular primitive representation used.

Cell Model: In a cell model, a 3D body is divided into a set of non-overlapping neighbor cells. Such cells have various forms, sizes, positions, and orientations. Therefore, the division of cells depends on a distinct number of cell types and simple operations for their assembly. The single cells can be arbitrary objects that are topologically equivalent to a sphere in 3D space. A 3D body is therefore composed of a set of half-disjunct cells. Data-structures highly depend on the cell type used. Often tree-structures come into use as for quad-trees and octrees.

A base body primitive is well-defined by fixing the dimension parameters of a 3D primitive (see Figure 4.11). In a cell model, the base bodies can have various shapes: cylinders, cones, spheres, toroids, or cuboids. Dimension parameters are e.g. three-side lengths for cuboids or one length and a diameter for a cylinder. Each object has a point of origin to a local coordinate system. This local coordinate system cannot be altered, since it is *à priori* fixed for every base body primitive.

In 4DiCeS special constraints have to be applied to a cell model for it to be suitable for representing a simulation space (see Table 4.5). First of all it is of major importance that the complete geometrical space of simulation is described in full. This means that there must exist an uninterrupted representation of both a global reaction and diffusion area. This area may be segmented into closed

Requirement	Description
Uninterrupted	The global geometry definition must enclose the entire simulation space without any gaps and exceptions.
Nesting	The simulation space can be segmented into compartments that can be partitioned further on again.
No Overlap	Compartments must not overlap with themselves or other compartments within the simulation space.
Data Slots	There exist defined slots for the linkage of the activity description and the used algorithms of the data model.

Table 4.5: Requirements to the simulation space. In addition to the definition rules of a used body model there exist four further specifications for a geometry representation in 4DiCeS.

compartments that may be partitioned again. Such a nested structure of 3D region descriptions ensures that a model can be simulated. Another constraint is that compartments must not overlap. Furthermore all defined structures must provide slots to link in other parts of the application data model. A simulation space is complete if all given constraints are obeyed. During the creation and the initialization, the application data model relies on this additional information.

Actually, all body models could be applied to 4DiCeS if they obey the given requirements. But as a first reference implementation to such a model, a specialized form of the cell model was used that is specified in the subsequent section.

4.2.1.2 Geometrical Data Structure

The cell model described earlier (see Section 4.2.1.1) was used for a reference implementation of a geometry model in 4DiCeS. A cuboid was used to define the cell type. Thus the 3D simulation space is divided into a 3D grid of equally-sized sub-volumes. In the literature such a structure is also referred to as a grid of voxels, which is an aggregation of the terms “volume” and “pixel” (Encarnação et al., 1997a). Often voxels are also called Volume Elements (VEs), in accordance to a grided volume of distinct elements. A generalized scheme

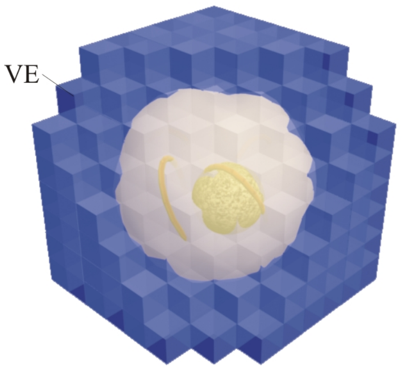


Figure 4.12: Simulation voxel space. A biological cell is subdivided into discrete cubes, called VEs. These VEs can harbor multiple particle species and may have a membrane as their surface area. To allow for various geometries, the VEs are designed to be freely scalable in size.

of such a diced model can be seen in Figure 4.12 that shows a schematized biological cell following this 3D geometry description.

The VEs used here are able to form all sorts of complex compartments. This disposition of distinct but regular cubes has enormous benefits to further modeling and simulation. The size of the VEs can be specified during initialization that helps to keep computational time low. Also such a geometry can be easily persisted in a simple 2D data-structure.

The VE defines the smallest geometrical structure within the simulation space. It can also specify boundary conditions that can then be used as membranes for particle diffusion. Every side of the cuboid voxel can have a membrane area. Thus a VE can be considered to be an open system if no membranes are set. Conversely the system is closed and reflecting if a VE is fully covered by membrane. The total coverage of a VE needs six membrane faces. Hence two neighboring VEs with adjacent membranes are analogous to a cellular lipid bi-layered membrane. The membranes function either as barriers for definable particle species or as binding areas for transmembranous proteins (see Figure 4.13). Opposite marginal VEs on the grid, if not divided by membranes, allow the movement of particles from one side of the grid to the other (i.e. exiting the far right and reentering to the left of the grid again).

Every cube is able to hold particle concentrations for each particle species applied to the model. Unbound particles are allowed to diffuse freely in all

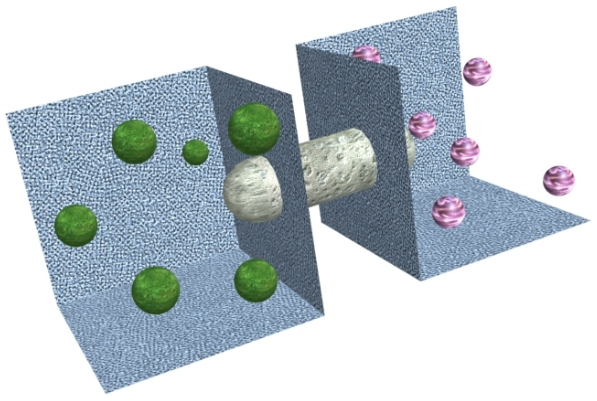


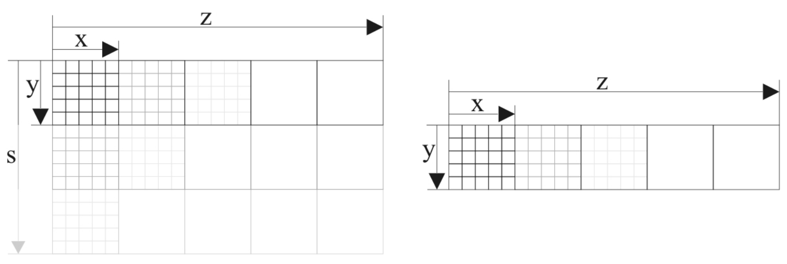
Figure 4.13: Boundary conditions. Every VE may be covered by membranes on every one of its six faces. Two neighboring VEs and membranes make a lipid bi-layer. Therefore it is possible to have both free and membrane-bound particles. The permeability of particles through the membranes may be individually adjusted.

directions. Membrane bound particles can only diffuse laterally within the membrane (see Figure 4.13). A voxel is composed of six direct faces, 12 edges and eight corners. While each face has one direct neighbor, the edges have three and the corners eleven. The diffusion from one VE to another is handled such that the probability of leaving a VE is higher with the face neighbors than with edge neighbors. The corners could be neglected since statistically corner diffusion is constantly small and therefore often dispensable.

The allocation of particles is uniformly distributed within a VE at initialization. This means that it is not possible to further specify particle positions within a voxel. However the number of particles per species and the membrane coverage can be set by the user.

To store and manage the 3D voxel geometry, a common 2D graphics format was applied. The Tagged Image File Format (TIFF)[1] was chosen due to its ability to handle different color spaces, to utilize powerful as well as lossless data compression algorithms, and to allow for multi-page-TIFFs. TIFF is designed to work well in many viewing applications, such as the Internet, so it is fully streamable with a progressive display option. TIFF is robust providing both full file integrity checking, as well as simple detection of common transmission errors (Miano, 1999). TIFF can store gamma and chromaticity data for improved color

[1]TIFF Specification: http://partners.adobe.com/public/developer/en/tiff/TIFF6.pdf



(a) TIFF of Particles (b) TIFF of Membranes

Figure 4.14: Placement of relevant geometrical information. (a) The number of particles per voxel is represented as a pixel in an image. (b) Represents the membrane-setting for all cubes of the grid. Here, membranes are binary-encoded as can be seen at Table 4.6. (a)+(b) All the x–y slices (the z stack) are spread in the picture's x direction. (a) Thereby, every single particle species can be represented in its 3D geometry in the TIFF's y direction. Each new step in time produces a new particle TIFF page.

matching on heterogeneous platforms. Last but not least, TIFF does not require licensing and can therefore be freely used (Murray and Ryper, 1994).

The challenge of converting the 3D data into a 2D picture format was resolved by displaying a 3D stack of VEs in two dimensions (see Figures 4.14 and 4.15). The grid of VEs has its three spatial dimensions (x, y, and z). Every VE can be directly addressed via these unique space coordinates in 3D. It is adequate to speak of x–y-layers in the grid's z axis. These layers are organized sequentially in one row of a TIFF image. The VE information of the number of particles is stored as a 64bit integer pixel. Every modeled particle species is given by its own row of x–y-layers (see Figure 4.14(a)). The result is a static 3D image of the VEs' particles of the grid. Every VE-grid-snapshot in time is stored as a separate TIFF-page (see Figure 4.15). The reusability of the output is enhanced, because a simulation can always be restarted with any of the previously produced TIFF as input data again.

Similar as to the particle storage in TIFF images, the structural information of membranes is also captured in a TIFF. Again, the x–y-layers in the grid's z axis are saved. In contrast to the before mentioned, only the six faces are stored as a sequence of 'set' and 'unset' bits (see Figure 4.14(b) and Table 4.6).

The TIFF-file can also store additional textual information. This capability is used to provide auxiliary data on the geometry of the 3D grid representa-

left	right	top	down	front	back
2^0	2^1	2^2	2^3	2^4	2^5

Table 4.6: Membrane binary codes. Stored to a TIFF-page, membranes are binary-encoded. Each of the six cube faces has a unique identification bit. Combined together into a pixel value, this encoding represent the coverage of a VE by membrane. A fully covered voxel with membrane would therefore have the value 0x3F. A plain voxel without any membranes is represented by a value of zero (0x00).

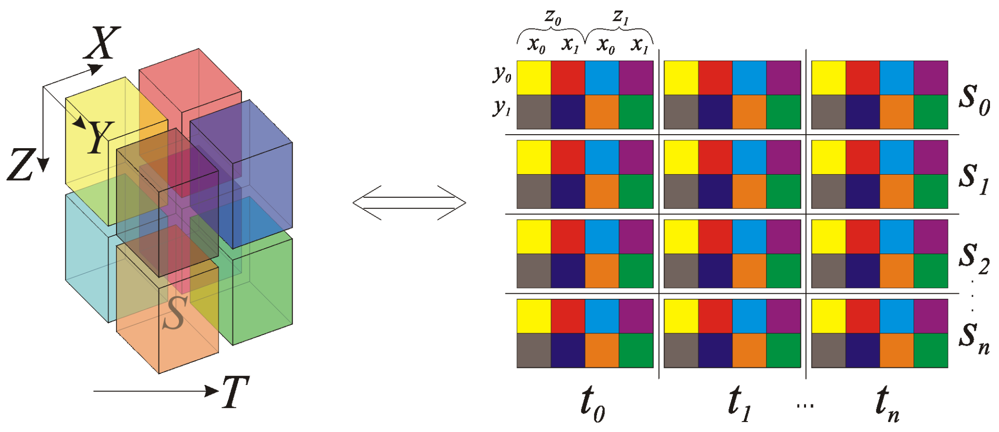


Figure 4.15: Transformation of 3D into 2D data. The picture shows the conversion of VEs into two 2D TIFF-pages. Here, the underlying information of all particles and the membrane geometry is stored as pixel values of an image file. The different colors yellow $\{x_0, y_0, z_0\}$, red $\{x_1, y_0, z_0\}$, brown $\{x_0, y_1, z_0\}$, blue $\{x_1, y_1, z_0\}$, cyan $\{x_0, y_0, z_1\}$, purple $\{x_1, y_0, z_1\}$, orange $\{x_0, y_1, z_1\}$, and green $\{x_1, y_1, z_1\}$ denote specific VEs at their corresponding positions in X, Y, and Z respectively. Each voxel V includes a constant set of particle species S. After every iteration cycle t of time T the same structure is stored as a separate page to the TIFF image.

tion. The underlying activity description, geometrical details, links to applied algorithms, Globally Unique IDentifiers (GUIDs), and version numbers can be stored as "tags" to the TIFF image files. The grid representation is stored as three integer values that mark the length of each grid axis (x, y, and z). The underlying cell model is tagged as the file name of the modeling file relative to the TIFF directory position.

TIFF files can be loaded with a great variety of existing image processing applications. Furthermore the implementation of an analysis tool for this kind of file format is not difficult. Several image processing libraries in various programming languages offer accordant TIFF support[2]. The compression rate of TIFF images allows for huge amounts of time series pictures to be created with relatively low storage requirements. As compared with flat text files of the same information, the TIFF used here achieves a compression rate of approximately 1/200. This partially addresses the commonly encountered problem of storing large amounts of data.

4.2.2 Activity Description

As seen in the previous section the representation of 4DiCeS geometry was achieved by analyzing as well as utilizing existing 3D modeling methodologies. The computational modeling of activity descriptions requires a similar approach. Again model description files have to be utilized for an adequate persistence mechanism to the underlying system. To be useful as a formal characterization of biological systems understanding, mathematical models are put into a format that may be transferred successfully between different software tools (Bower and Bolouri, 2004). Such a format then addresses a number of problems facing simulation and modeling (three of such major problems are stated in Table 4.7).

One problem occurs when users have to handle multiple resources from various software tools during a project. Different tools have different capabilities and strengths. Working with different tools today mainly requires the re-encoding of models in each tool that is a very error-prone and time-consuming process. Another problem arises with the need for obtaining the model definitions in electronic form. This especially can be found with journal publications where authors use different modeling environments and model representation languages. Such definitions are often not straightforward to examine, compare, test, or reuse. A researcher typically must manually transcribe the model into his software tool's appropriate format. Last but not least, a problem exists if a simulation tool exceeds support. Models developed on such systems may become obsolete and therefore are often abandoned. The development of new

[2]Free graphics libraries: http://www.thefreecountry.com/sourcecode/graphics.shtml

Requirement	Description
Multiple Resources	Often various resources from different tools or databases have to be collected and stored away as one model file.
Reproducibility	There is a great need for allowing other people to view and reuse models – especially when they were published.
File Format Support	The stop of support for software tools often also ends the assistance on non-standardized modeling formats.

Table 4.7: Problems facing simulation and modeling. This table states three problems facing simulation and modeling.

tools will only intensify these problems unless the issue of standard formats or clear and automatic translation between them is addressed (Hucka et al., 2004).

The development of standard modeling languages are being conducted in effort to address such problems. Standardized formats can then be used to communicate and exchange models between different software tools. By supporting such general formats as input and output formats, different software systems can all operate on the identical representation of a model. This provides a common starting point for simulations as well as analysis and eliminates the possibility of translation errors.

Model-data storage is currently accomplished either with proprietary data formats or by derivatives of markup languages such as XML (see Table 4.8. Recently an attempt was made to use UML as a biological model description language (Webb and White, 2005). Most of the biological model description formats are freely available standards. While numerous language definition draft documents are still copyrighted, a non-commercial use is usually accepted and even often designated.

The popularity of XML in the area of computational biology has grown in the past decade. XML provides the capability of representing data in a comprehensive and standardized data structure. The structure of XML documents defined using a XML Document Type Definition (DTD) however is limited to representing data in a hierarchical tree fashion. Such an approach imposes severe limitations on both the structure, as well as the ability to validate a document. The utilization of the World Wide Web (WWW) Consortium XML "schema" approach to document definition thus overcomes many of such limitations.

Designation	Web Site
AnatML	http://www.physiome.org.nz/anatml/pages/
BioPAX	http://www.biopax.org/
CellML and ModelML	http://www.CellML.org/
DSML	http://www.hippron.com/hippron/
EML	http://www.e-cell.org/
FieldML	http://www.physiome.org.nz/fieldml/pages/
Jarnac	http://www.cds.caltech.edu/ hsauro/Jarnac.htm
MDL	http://www.mcell.psc.edu/
SBML	http://www.sbml.org/
SciLab	http://www.scilab.org/
XPP	http://www.math.pitt.edu/ bard/xpp/xpp.html

Table 4.8: Input description file standards. The description file formats displayed in this table are either cellular modeling languages or their specific helper-languages. They can be roughly described as proprietary and markup languages.

Many data formats have been proposed for representing models in the life-sciences including the BIOpolymer Markup Language (BIOML)[3] (Fenyo, 1999), the Chemical Markup Language (CML)[4], the EML ((Sakurada et al., 2002)), the MicroArray Gene Expression Markup Language (MAGE-ML)[5] (Spellman et al., 2002), the Proteomics Experiment Markup Language (PEML)[6] (Taylor et al., 2003), the Protein Markup Language (ProML)[7] (Hanisch et al., 2002), and the Proteomics Standards Initiative's Molecular Interaction (PSI-MI)[8] lan-

[3]BIOML: http://xml.coverpages.org/bioml.html
[4]CML: http://www.XML-CML.org/
[5]MAGE-ML: http://www.mged.org/MAGE/
[6]PEML: http://www.ccbm.jhu.edu/
[7]ProML: http://www.scai.fraunhofer.de/
[8]PSI-MI: http://psidev.sourceforge.net/

guage (Hermjakob et al., 2004). There are also a few general-purpose modeling definition formats that can be found for biological model documentations. Two very popular examples are the Petri-Net Markup Language (PNML)[9] and the Resource Description Framework (RDF)[10].

There are only two XML-based formats that are designed to unambiguously specify, store, and exchange biological models in a form that is both computer and human readable – CellML (Crampin et al., 2004), and SBML (Hucka et al., 2004). The Dynamic Signaling Maps Language (DSML) that could be used in the same way is not XML-based but a proprietary flat-file language.

Both SBML and CellML support the Mathematical Markup Language (MathML) (Ausbrooks et al., 2003). MathML is an XML application for describing mathematical notation and capturing both its structure and content. The goal of MathML is to enable mathematics to be served, received, and processed, just as the HyperText Markup Language (HTML) has enabled this functionality for text on the WWW.

4DiCeS has currently reference implementations for two model description formats. The first is a proprietary description language designed during the development of 4DiCeS and can be considered the systems natively supported activity description format. This flat-file format called 4DiCeS Model Description (FMD) is further discussed in Section 4.2.2.1. The second file format supported at present is SBML (see Section 4.2.2.2). The activity considerations will be closed by a closer look at the formal integration of these two description file formats into the 4DiCeS framework (see Section 4.2.2.3).

4.2.2.1 4DiCeS Model Description (FMD) File Format

During early development of the 4DiCeS system, a custom-made modeling file format was used together with the kernel-parsing interface. This format allows for the design of models consisting of both data for the reaction as well as diffusion processes. The format is a flat-file that may integrate TIFF geometry files as discussed in Section 4.2.1.2. In the following section, the four-tuple grammar $G = (T, N, P, S)$ of this flat-file modeling language is defined using the Backus–Naur Form (BNF) (see Appendix B):

[9]PNML: http://www.informatik.hu-berlin.de/top/pnml/
[10]RDF: http://www.w3.org/RDF/

$$T = \{ \; 0, \ldots 9, a, \ldots z, A, \ldots Z, +, >, |, @, *, \sim, -, ., \_ \; \} \tag{4.1}$$

$$\begin{aligned} N = \{ \; & \langle outerproduct\rangle, \langle innerproduct\rangle, \langle educt\rangle, \langle particle\rangle, \\ & \langle species\rangle, \langle volumeelement\rangle, \langle reaction\rangle, \langle rate\rangle, \\ & \langle product\rangle, \langle letter\rangle, \langle digit\rangle, \langle amount\rangle, \langle sign\rangle, \\ & \langle name\rangle, \langle membrane\rangle \; \} \end{aligned} \tag{4.2}$$

$$S = \langle volumeelement\rangle, \tag{4.3}$$

where T denotes the set of allowed terminal symbols. N specifies the set of all possible non-terminal symbols, and S defines the starting symbol as further defined in 4.10. The production system P is further specified by:

$$\begin{aligned} \langle letter\rangle &::= 'a'..'z'|'A'..'Z' && (4.4) \\ \langle digit\rangle &::= '0'..'9' && (4.5) \\ \langle percent\rangle &::= ['0']'.'\{\langle digit\rangle\}|'1' && (4.6) \\ \langle sign\rangle &::= '*'|'\sim'|'-'|'.' && (4.7) \\ \langle name\rangle &::= ['\_']\langle letter\rangle\{\langle letter\rangle|'\_'| \\ & \quad\; \langle digit\rangle|\langle sign\rangle\} && (4.8) \\ & && (4.9) \end{aligned}$$

for base primitives, and:

$$\begin{aligned} \langle volumeelement\rangle &::= \langle reaction\rangle\{',' \langle reaction\rangle\} && (4.10) \\ \langle reaction\rangle &::= \langle rate\rangle\langle educt\rangle\langle product\rangle && (4.11) \\ \langle rate\rangle &::= '('\langle percent\rangle')' && (4.12) \\ \langle educt\rangle &::= \langle species\rangle\{'+'\langle species\rangle\} && (4.13) \\ \langle species\rangle &::= \langle amount\rangle\langle particle\rangle && (4.14) \\ \langle particle\rangle &::= \langle name\rangle[\langle membrane\rangle] && (4.15) \\ \langle amount\rangle &::= \{\langle digit\rangle\} && (4.16) \\ \langle membrane\rangle &::= '@'\langle name\rangle && (4.17) \end{aligned}$$

$$\begin{aligned}\langle \mathit{product}\rangle &::= \langle \mathit{innerproduct}\rangle[\langle \mathit{outerproduct}\rangle]| \\ &\quad\;\; \langle \mathit{outerproduct}\rangle[\langle \mathit{innerproduct}\rangle] \qquad (4.18)\\ \langle \mathit{innerproduct}\rangle &::= {}' >' \langle \mathit{educt}\rangle \qquad (4.19)\\ \langle \mathit{outerproduct}\rangle &::= {}'|'\langle \mathit{educt}\rangle \qquad (4.20)\end{aligned}$$

where a reaction (4.11) is defined by a set of educts (4.13), products (4.18), and a reaction rate (4.12). A product can further be divided into outer (4.20) and inner products (4.19) that define a reaction output to be either in the inside or the outside of the current reaction compartment. Both educts and products are specified by particle species (4.14) and their momentary concentration (4.16). Particle species can be tagged as being membrane-bound by concatenating a membrane (4.17), by the use of an '@' symbol, to them.

This description specification is of course by far not complete but applicable for many usage scenarios of 4DiCeS in its present state. FMD was especially useful for testing as well as verification purposes during implementation and can be considered to be the native description file format of activities in 4DiCeS.

4.2.2.2 Systems Biology Markup Language (SBML)

The Caltech ERATO Kitano systems biology project has developed the Systems Biology Markup Language (SBML) for the representation and modeling of information components in cellular systems. SBML serves as an attempt to specify a common, model-based description language for systems biology simulation software (Hucka et al., 2004). At the moment the world-wide SBML community counts over 80 software tools supporting this language in one or the other form (SBML Software Guide[11]). The SBML representation language is organized around five categories of information: model, compartment, geometry, species, and reaction. The intent is rather to cover the range of data structures needed by the collection of all of the simulators examined so far (Finney and Hucka, 2003). A major drawback is that SBML does not currently have the means to represent Partial Differential Equation (PDE)-level models or diffusion terms (Hucka et al., 2003). Cellular geometry was also rather neglected. To surround such deficiencies, especially the lack of support for geometry definitions, great efforts have to be made on part of 4DiCeS (see the following section 4.2.2.3). How other application are dealing with such is further shown in Section 5.4.

[11]SBML Software Guide: http://sbml.org/SBML_Software_Guide/

In the following the import to as well as the export from 4DiCeS of the previously presented two file formats FMD and SBML is going to be presented in more detail.

4.2.2.3 Import and Export

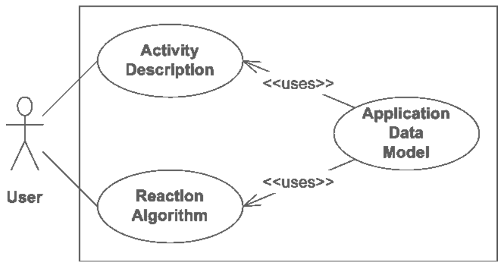
As for applying a model to the 4DiCeS suite, there are three specific user scenarios (see Figures 4.16). Depending on what type of data is to be included to the application data model, there is one use-case for loading an activity description without any further geometry definitions, then a use-case with rudimentary geometry to allow for concurrent algorithms, and lastly a use-case describing the full interaction with both activity and geometry description in 3D.

Plain Activity Model: If a user simply applies a given activity model file to 4DiCeS, then there is no further need for any geometry information. The only thing that has to be specified in addition to the activity data is what kind of reaction algorithm should be applied to the simulation process. Within such a scenario, there is no algorithmic concurrency allowed, because of missing segmentations that would be defined by a geometry description. The use-case can be seen in Figure 4.16(a). Such a simulation can be referred to as a "single segment simulation".

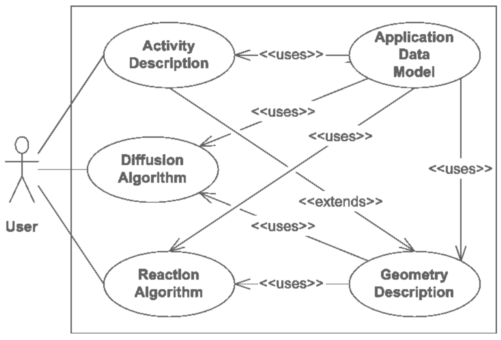
Rudimentary Geometry: If compartments with concurrently running algorithms are required, but no further 3D geometry is needed, then a limited form of the geometry description must be applied in addition to the activity description. Only defined compartments are mapped to segments of the geometry to allow for setting individual algorithms for each compartment. The use-case is visually described by Figure 4.16(b). This type of simulation is called here a "2D multiple segments simulation".

Full Input Description: When a trackable 3D geometry is required in addition to the activity and concurrency of algorithms, then the user has to input two complete description files of both geometry and activity. The use-case for this type of model can be found in Figure 4.16(c). This model type leads to a full "3D multiple segments simulation".

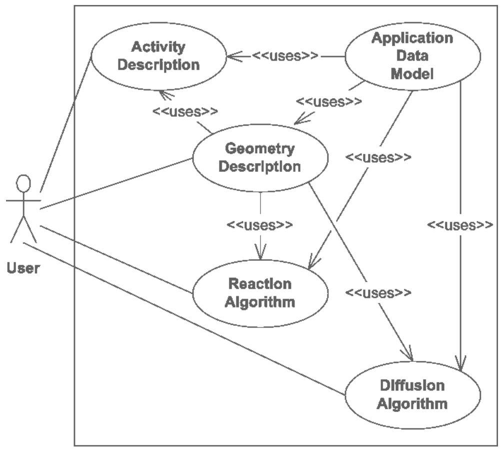
All three use-cases have an activity description and the application data model in common. While use-case Figure 4.16(a) requires only a separate setting of the applied reaction algorithm to establish the simulation engine, the other two use-cases (Figures 4.16(b) and 4.16(c)) require both a geometry description *and* a set of diffusion algorithms. With Figure 4.16(b) the application data



(a) Single Segment Model



(b) 2D Multiple Segments Model



(c) 3D Multiple Segments Model

Figure 4.16: Use-cases: Modeling. Depending on what type of data is applied to 4DiCeS, there are three different use-case scenarios. Figure (a) shows the input of an activity description without any further geometry definitions. Figure (b) displays a use-case with limited geometry to allow for concurrent algorithms, and Figure (c) describes the full interaction with both activity and geometry description in 3D.

model will attempt to set the needed connections itself, since no further geometry is needed. With Figure 4.16(c) the user must explicitly set the geometry description so that a simulation engine can be established.

A UML sequence diagram (see Figure 4.17) displays the behavioral specification in more detail. The conditions are highlighted that produce an alternate behavior of the application data model during initialization. The three conditions are summarized in Table 4.9. It should be noted that if any of the three alternatives fails to connect to the remaining data sets, then the simulation engine

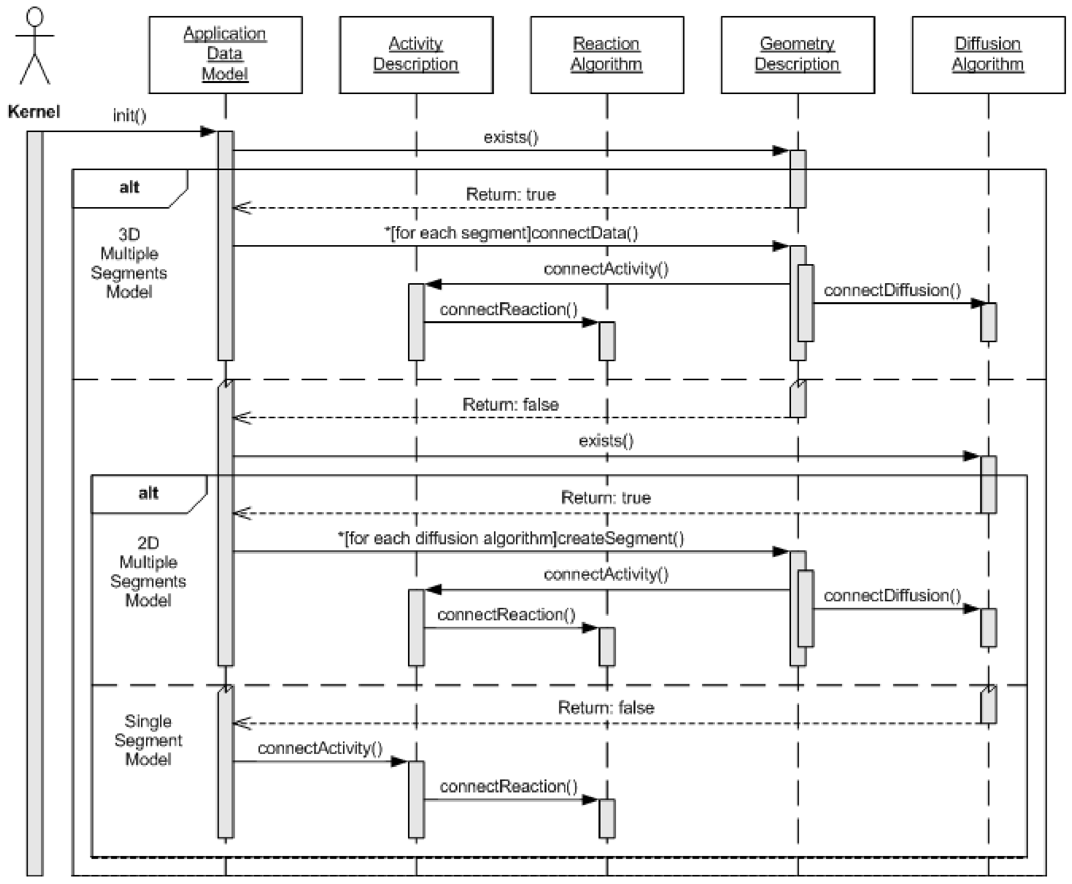


Figure 4.17: Sequence diagram: Model alternatives. This figure displays the conditions that lead to the three alternative model instances. On initialization, the kernel checks for the existence of a geometry description. If and only if such a description file exists the application data model instructs the geometry description to connect to the remaining data set. Otherwise, the kernel will further check for an existing set of diffusion algorithms. If and only if that set exists a simple geometry description is created, which again connects to the remaining data set. Only if both test are negative the kernel instructs the activity description to connect to the corresponding reaction algorithm. In either case, if and only if a final connection to the remaining data sets is successful, then a simulation engine is instantiated.

will not be produced and a simulation of the model will not be feasible. The

Condition	Resulting Use-Case	Description
$\Phi \neq \emptyset$	4.16(c)	3D Multiple Segments Model
$\Phi = \emptyset \ \wedge \ \Theta \neq \emptyset$	4.16(b)	2D Multiple Segments Model
$\Phi = \emptyset \ \wedge \ \Theta = \emptyset$	4.16(a)	Single Segment Model

Table 4.9: Model alternative conditions. This table summarizes the conditions that results in the three alternative use-case scenarios. Here Φ is a set of segments and Θ defines a set of diffusion algorithms. These sets can either be empty or filled with their corresponding elements. The case $\Phi \neq \emptyset \wedge \Theta = \emptyset$ results into an error event, the initialization process is interrupted, and the application data model remains in its uninitialized state.

application data model thus stays in its uninitialized state until the missing data is added and an additional initialization call is invoked.

As already mentioned in the preceding section the SBML is missing an advanced functionally in supporting geometry. Therefore this model description format is handled somewhat different. The single segment model as well as the 2D multiple segments model, there initialization routines, and the actual simulation work very similar as described before. But in contrast to the 3D-FMD handling the 3D multiple segments model is currently not facilitated. Also concurrent algorithms cannot be inserted by SBML into 4DiCeS, since this file-format is missing such a concept completely. Anyhow, at least with the later on described GUI implementation for 4DiCeS, the user is able to connect different algorithms to the system manually. This change of course cannot be exported to SBML again, which occasionally results in a loss of persistency of the data model.

Noteworthily, the application data model offers the entire bandwidth of "getter" and "setter" functions. This is therefore the place where the loading, saving, and validation of models occur. Data sets, parameters, boundary information, and the simulation space can all be manipulated through this one object instance. The application data model actually offers the most direct and powerful set of methods to the kernel. With this the security of the 4DiCeS system becomes a concern, since the system is not always able to safely process erroneous model data through the parsing interface. Fall-back mechanisms within the system will therefore run a controlled model rollback if syntactical errors are detected during parsing or during initialization.

The now following section will deal with the algorithmic back-end of the 4DiCeS framework. The modeling part of the system's description will be hence left and turned towards the depiction of simulation mechanisms.

4.2.3 Algorithm Handling

Both ports to the diffusion and reaction modules are actually designed as interfaces within interfaces. The reason for this architecture is the algorithms' handles that are needed to perform further tasks. There are algorithms that focus only on single compartments, whilst others need the entire simulation space to work on. Thus there is a low-level interface allowing for full access to the simulation space and its particles. In case of a compartment algorithm, there is a second high-level interface that handles the traversing of the nested segments. Therefore, it is possible to integrate various different algorithms to the 4DiCeS kernel without code modification. The high-level interface then allows for simulating a model with different algorithms in concurrency.

The following Section 4.2.3.1 defines both the high and the low-level interfaces. Two sections will thereafter focus on the connection of diffusion (Section 4.2.3.2) and reaction (Section 4.2.3.3) algorithms to the 4DiCeS kernel in greater detail.

4.2.3.1 Algorithm Interface Design

The interface-within-interface design introduced earlier is displayed in Figure 4.18 by using a UML component diagram (see Appendix C.4). It can be seen here that there is a base interface providing a set of low-level methods to the application data model. This set of low-level commands actually maps the application data model with all its data sets very closely. However the directness of control is a trade-off against convenience functions that the high-level interface then provides. As can be seen, the high-level interface connects to the low-level interface using its direct functionality. Even though not all methods are passed on from the low-level to the high-level command set, the high-level functions are enriched by methods that allow for the traversal of given model data geometries.

The interfaces are connected through inheritance, where a general low-level interface description feeds all other application data model interfaces (see Figure

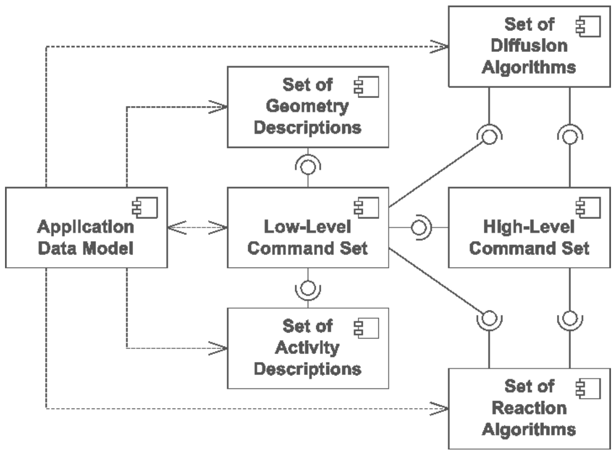

Set of Diffusion Algorithms
Set of Geometry Descriptions
Application Data Model
Low-Level Command Set
High-Level Command Set
Set of Activity Descriptions
Set of Reaction Algorithms

Figure 4.18: Component diagram: Interface within interface. The given figure shows an UML component diagram on the interface architecture for the 4DiCeS application data model. While both the geometry as well as the activity description sets are linked directly to the low-level interface, the algorithm sets for either reactions or diffusion also have the choice to connect to a high-level command layer. Here, the high-level interface is linked with the low-level commands. It provides functions for the automatic traversing of the input descriptions. Therefore, the higher-level interfaces allow for the user to work on segments and thus enable the possibility for algorithms to run concurrently.

4.19). Only the algorithm interfaces present a high-level interface to their low-level counterparts. As will be later seen, the user interface description will also be based upon this interface inheritance connection and doing so ensures that no data redundancies occur.

A realization of either reaction or diffusion algorithm modules can be implemented in a straight forward fashion. First a module must inform the kernel about its presence and nature. Next the user is able to access global information regarding the algorithm, particularly the algorithm's name, a brief description, a list of references, as well as the names of both the algorithm inventor, and the author of the concrete implementation. A very important piece of information that is also directly used by the application is the type of algorithm (i.e. if it is stochastic or deterministic). Depending on this information the application data model can ensure that parameters (e.g. rate constants) are automatically transformed into the underlying type of algorithm. If this first information ini-

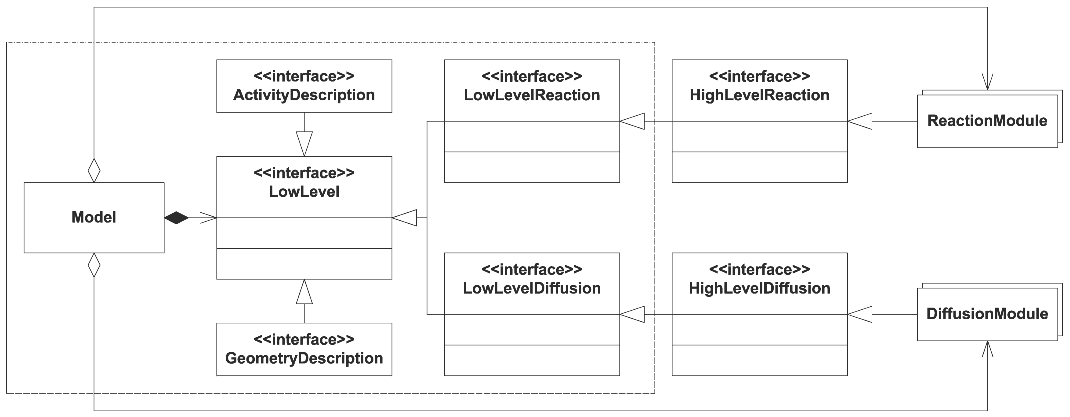


Figure 4.19: Static diagram: Interface inheritance. As can be seen in this static UML diagram, the interfaces are linked to each other through a chain of inheritance. On the low-level side (dash-doted rectangle), all interfaces (including the interfaces for the input descriptions) are specializations to a global low-level interface. Only the reaction and the diffusion algorithm interfaces provide a high-level interface. These are in turn specializations of their low-level counterparts. The application data model is linked through a composition to the general low-level interface, but has aggregation linkage to concrete realizations of algorithms.

tialization has finished, an algorithm iteration cycle can be induced by calling the methods '*react()*' or '*diffuse()*' respectively. As for the current implementation of 4DiCeS, each iteration cycle will activate both present reaction and diffusion algorithms for every segment. This may lead to the problem that alternatively switching between the two algorithm forms is not desirable. In some cases this will also result in faulty output. A solution to this scenario is a coupled algorithm allowing for reaction and diffusion of particles at the same time. This, of course, could easily be done by connecting both interfaces to the same algorithm that is capable of processing both simultaneously.

The next two sections discuss applicable algorithms to the interfaces. While the concrete implementations of such algorithms will then be introduced in Section 4.3 and discussed in greater detail within Appendix A.

4.2.3.2 Diffusion

The 4DiCeS diffusion interfaces are able to handle various different algorithms. To achieve such a behavior, the interface depends on the separation into high and low-level interfaces as described earlier.

Method	Type	Scale	Primitive	Time	Reference
BD	stoch	micro	particle	solution	(Elcock, 2003; Ping et al., 2004)
Gillespie	stoch	meso	full	events	(Gillespie, 1976)
GFRD	stoch	micro	particle	events	(van Zon and ten Wolde, 2004)
Lattice Gas CA	stoch	micro	segment	steps	(Berry, 2002)
MD	det	micro	particle	solution	(Friedel and Shea, 2004; Baynes and Trout, 2004)
ODE	det	macro	full	solution	(Broderick et al., 2005)
PDE	det	macro	grid	solution	(Schaff et al., 1997)
Smoldyn	stoch	micro	particle	steps	(Andrews and Bray, 2004)
Weimar CA	stoch	meso	segment	steps	(Weimar, 1997)
Spacial Gillespie	stoch	meso	segment	events	(Kang et al., 2002; Stundzia and Lumsden, 1996)

Table 4.10: Diffusion methods in biology. This table lists methods that are used for simulating spatial movement. The type column determines if a simulation is of stochastic or deterministic nature. The scale column distinguishes microscopic, macroscopic, as well as mesoscopic schemes. The primitive column differentiates between data primitives that are used in the simulation. The time column depicts between discrete events, discrete time-steps, and the results of a numeric solution. The non-spatial Gillespie Methods and ODEs were included for comparison. Adapted from Takahashi et al. (2005).

In the case of random-walk simulations, where every particle is treated separately, the global grid handle or the global particle handle is sufficient. The segments' interface would however not be very useful in such a scenario. Rate-walk diffusions, where only numbers of particles or concentrations are shifted from one segment to another, can take advantage of this. Other deterministic methods could make use of the interface of segments also. With every iteration calculated results are passed back to the kernel through the interfaces, which must then take care of the reallocation of particles to their compartments.

The input and the output of the number of particles per species or their respective concentrations then highly depend on the interface used. For a global handling the number of particles (or concentrations) are returned for every explicitly-specified segment and particle species. In case of the segments' interface, the compartments' data is pushed to the algorithms and will be traversed by the simulation space itself. With global particle access, all particles of every particle species are delivered and retrieved by the interface at once. The reintegration of particles into the simulation space and its nested segments is handled internally.

Diffusion is normally referred to as Brownian motion (Brown, 1828) of either uncharged or charged particles in solution. The first quantitative description of such a process was *Fick's Law of Diffusion* (Fick, 1855). Later, an explanation for Fick's law was discovered (Einstein, 1905; von Smoluchowski, 1906). Today, there are several methods to model diffusion with computational assistance (see Table 4.10). Prominent examples are deterministic approaches as for ODE, PDE, as well as Molecular Dynamics (MD). Next to them a considerable number of stochastic methods exist such as the Brownian Dynamics (BD), Gillespie derivatives, some CA variations, and others. A stochastic random-walk simulation was chosen as a reference implementation for the 4DiCeS application. A brief description of the algorithm can be found within Appendix A.2.

4.2.3.3 Reactions

Similarly to the integration of diffusion algorithms reaction interfaces are able to handle various different algorithms in 4DiCeS. The interface definition again depends on the splitting of command sets into high and low-level interfaces functions. But in contrast to diffusion, the reaction interface must provide several specific methods that allow for the connection of reaction algorithm modules. Given reaction equations from an activity description can always be used, independent of the underlying algorithm.

The reaction equations contain information about the reaction educts, the resulting products, and a variable that specify the kinetics of a reaction. The products occur as a result of a reaction within a segment, or they can be passed through membranes. Admittedly only elementary chemical reactions with educts up to and including the second order are allowed with 4DiCeS, because collisions of three or more molecules at an instant are considered to be fairly improbable (Gillespie, 1976, 1977). Therefore the system needs to be modeled by using only three types of elementary reactions: first order, second order, and the homodimer formation (Gillespie, 1996). Hence only reaction equations that match the following three patterns will be allowed to pass through the reaction interface:

$$A \longrightarrow \dots \tag{4.21}$$

$$A + B \longrightarrow \dots \tag{4.22}$$

$$2A \longrightarrow \dots \tag{4.23}$$

The reaction equation (4.21) is of first order, which includes isomeric reactions (e.g. $A \longrightarrow B$) where a single educt is converted into a single product, and degradations (e.g. $A \longrightarrow B + C$) where the single educt is split into multiple products. The second order equation (4.22) is then assigned to reactions involving two different educts (e.g. $A + B \longrightarrow C$, $A + B \longrightarrow C + 3D$, ...). The homodimer formation reaction (4.23) is finally a special form of the second order equation with two educts of the same type (e.g. $2A \longrightarrow B$, $2A \longrightarrow B + 2C$, ...). The reason why this last kind of reaction cannot be treated in the same way as a second order reaction is that there exist different numbers of possible educt combinations. For a system of N_A particles of A and N_B particles of B the number of possible distinct educts within B is $N_A \times N_B$ whereas the number of possible distinct encounters among A is $N_A(N_A - 1)/2$. In 4DiCeS third and higher order reactions can be reasonably estimated by the combination of multiple reactions of the supported three types.

From a mechanistic perspective the actual reactants (reaction partners or educts) are piped to the interface by three matrices: one matrix for either of the educts M_E, the internal products $M_{P_{in}}$, or the external products $M_{P_{out}}$. The rows of each matrix specify the reaction equations and the columns define the reaction particle species. As an example the following five reactions can be considered:

(a) Equations	(b) M_E					(c) $M_{P_{in}}$					(d) $M_{P_{out}}$				
Reactions	A	P	X	Y	Z	A	P	X	Y	Z	A	P	X	Y	Z
(4.24)	1	0	0	1	0	0	1	1	0	0	0	0	0	0	0
(4.25)	0	0	1	1	0	0	2	0	0	0	0	0	0	0	0
(4.26)	1	0	1	0	0	0	0	2	0	2	0	0	0	0	0
(4.27)	0	0	2	0	0	1	1	0	0	0	0	0	0	0	0
(4.28)	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0

Table 4.11: Internal representation of reaction equations. In 4DiCeS, reaction equations are represented as matrices. There are three matrices for each element of a reaction equation: one for the educts M_E (b); one for the inner products $M_{P_{in}}$ (c); and one for the outer products $M_{P_{out}}$ (d). The rows specify the different equations (a) and the columns (b)+(c)+(d) indicate the used particle species in the activity description. The cells of the matrices may have numbers from zero to two. Each number indicates the corresponding number of particles reacting in an equation.

$$A + Y \longrightarrow X + P \tag{4.24}$$
$$X + Y \longrightarrow 2P \tag{4.25}$$
$$A + X \longrightarrow 2X + 2Z \tag{4.26}$$
$$2X \longrightarrow A + P \tag{4.27}$$
$$Z \longrightarrow fY \tag{4.28}$$

These five reactions (4.24 to 4.28) were devised by (Belousov, 1958) and are called the Belousov–Zhabotinsky Reaction (BZR) today. They generate sustained temporal oscillations in the concentrations of intermediates X, Y, and Z. In an alphabetical ordering of reactants, the three matrices would look like those in Table 4.11. There are no external products in the reaction equations (4.24) to (4.28). Therefore $M_{P_{out}}$ remains as a zero matrix.

The input and output of the number of particles per species (or their respective concentrations) in the current compartment work in much of the same way as with the reaction channel matrices. A matrix of amounts or concentrations is passed to the reaction algorithm were every row represents a particle species. The result is returned as two matrices – one matrix for the resulting internal $M_{R_{in}}$ and another matrix for external changes $M_{R_{out}}$.

Method	Type	Scale	Primitive	Time	Reference
Binomial τ-leap	stoch	meso	full	events	(Cao et al., 2004a; Chatterjee et al., 2005)
COAST	hybrid	meso	full	events	(Möller, 2006)
E-Cell method	hybrid	meso	full	steps	(Takahashi et al., 2003)
Gillespie	stoch	meso	full	steps	(Gillespie, 1976, 1977, 2006; Gibson and Bruck, 2000)
Maximal Time-Step	hybrid	meso	full	events	(Puchałka and Kierzek, 2004)
Multi-Scale	hybrid	meso	full	events	(Burrage et al., 2004; Kiehl et al., 2004)
ODE	det	macro	full	solution	(Chickarmane et al., 2005)
Partition System	hybrid	meso	full	events	(Haseltine and Rawlings, 2002)
PDE	det	macro	grid	solution	(Lebiedz and Maurer, 2004; Mayawala et al., 2006)
τ-leap	stoch	meso	full	events	(Gillespie, 2001; Rathinam et al., 2003; Cao et al., 2004b)

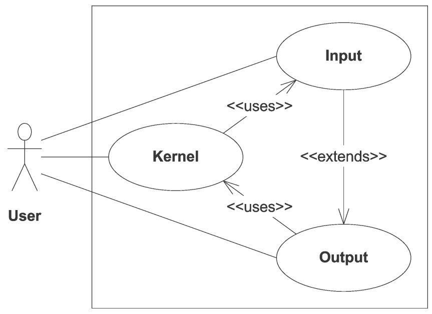
Table 4.12: Reaction methods in biology. The columns represent almost the same information as those in Table 4.10. With column "Type" there is a new term "hybrid" introduced, which states algorithms that use both stochastic and deterministic methodologies. It generally can be seen that there are in addition to deterministic approaches various derivatives of the Gillespie and the τ-leaping method.

Similar to diffusion there exist various different methods for computational reaction processing. Table 4.12 provides an overview of important candidates in this category. Deterministic approaches for ODEs as well as PDEs are again commonly utilized. Also stochastic methods such as the Gillespie derivatives and some τ-leaping variations exist. Attempts were made to combine deterministic and stochastic reaction methods to form "hybrid" approaches. Therefore one deterministic and several stochastic methods were chosen as reference implementations for the 4DiCeS application. These are Gillespie's "First Reaction Method", Gillespie's "Direct Method", as well as Gibson and Bruck's "Next Reaction Method" working on particles and the Runge–Kutta 4^th order method using concentrations. A brief description of these two algorithms can be found in Appendix A.1. In addition to the two reference implementations, there is an implementation of the hybrid COntrollable Approximative STochastic (reaction-algorithm) (COAST) (Wagner et al., 2006; Möller, 2006) by kind courtesy of Dr. Mark Möller, Bielefeld University, Germany.

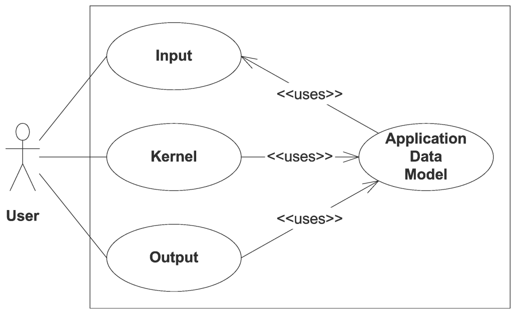
4.2.4 User Interfaces

The previous sections described interfaces that can be used to insert application data model relevant modules. This section now provides more detail regarding the human-machine interdependency with the 4DiCeS system. The system requires user interaction since various functional and exchangeable modules for diffusion reaction and input description parsing have to be defined manually sometimes. There are two usage scenarios for user interfaces – the Command-Line Interface (CLI) and a Graphical User Interface (GUI). Both use-case scenarios are displayed in Figure 4.20.

The CLI (see Figure 4.20(a)) allows for initiating simulation processes from a shell-based environment. Alternatively, the application can be opened and further controlled by a GUI (see Figure 4.20(b)). While a GUI has the advantage of being fully-manipulatable even during simulation, a CLI allows for the one-time input of command-line options to run the program. Each simulation has to be newly initiated by CLI parameters that can only be interrupted but not manipulated any further by the user. Such CLI simulations can thus be used for multiple high-throughput simulations if embedded into batch-jobs. This implies that one or more application data models exist already and that the user is only interested in the simulation output. With a GUI the user is in a position to manage both the data and the simulation simultaneously. Figures 4.21 and 4.22 highlight this difference through the use of two UML sequence



(a) UML use-case: CLI



(b) UML use-case: GUI

Figure 4.20: Use-cases: User interfaces. The two UML use-case diagrams display the two possible ways the 4DiCeS application can be communicate with. The shell-based CLI (a) is the simpler one of both scenarios. Here the user activates a simulation via parameters on the command line and receives the result at termination. Beyond that the GUI-based approach (b) can additionally manipulate the data the simulation is working on.

diagrams for a CLI and a GUI respectively.

Both sequence diagrams have the application data model instantiation and deletion in common. Their difference lies in the additional interaction points for the user. For the CLI the only additional manipulation the user is able to do besides simulation initialization is canceling the simulation manually before it auto-terminates. The GUI provides the user with much more flexibility as well as the complete control over the data set and simulation methods.

As can be seen with the now following Section 4.3, there exist reference implementations for both CLI and GUI interactions. A mixture of both scenarios should be possible as well but was not investigated any further.

4.3 Application Details

Previous sections of this chapter defined the formal structure of the 4DiCeS system. This section provides details on how these formal definitions were realized and what the user can expect from it. More significant than other implementation decisions, there was the goal of having good performance and extendability

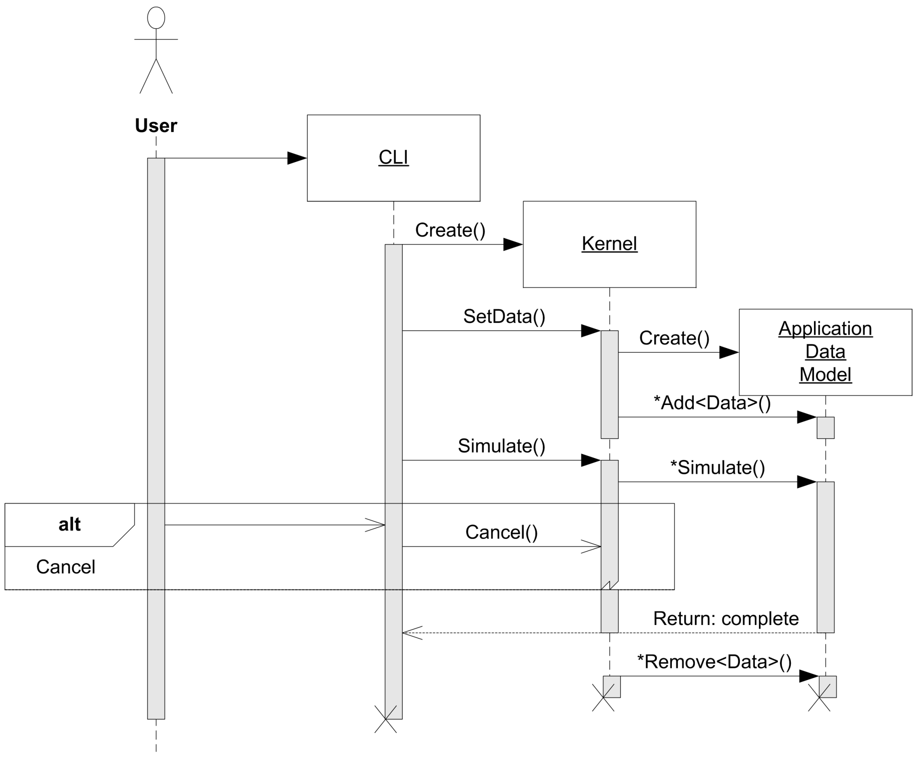


Figure 4.21: Sequence diagram: CLI. As the simpler of the two usage scenarios, this figure presents an UML sequence diagram for a CLI. Here it can be seen that the lifetime of the CLI spans the complete model instantiation and simulation process. The user is only able to initialize or abort a simulation. If simulation output is produced, the user can collect it by the time the CLI will terminate.

for the 4DiCeS application. Both topics were addressed by choosing a native programming language for the kernel implementation. C/C++ came to use as a programming language supporting object-oriented features (C++) and allowing for very performing compiles (C). A drawback to this decision is that the kernel has to be recompiled for every operating system that the program is used on. This was resolved by using only standard elements of C/C++ and by continuously cross-compiling the source code on at least one Windows and one UNIX platform during development.

Dynamically extending a program in runtime with new subclasses on different platforms is one of the main design aspects of 4DiCeS. On UNIX systems this makes use of the Executable and Linking Format (ELF) and Shared Objects

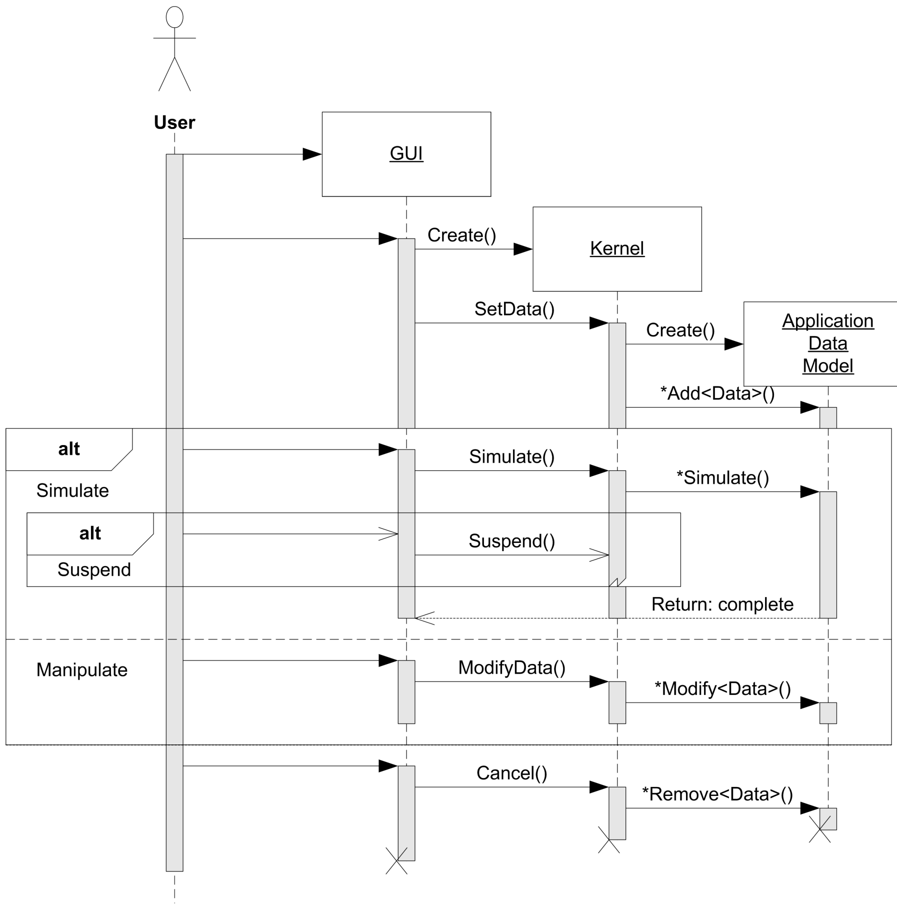


Figure 4.22: Sequence diagram: GUI. This figure shows the GUI usage as an UML sequence diagram. In contrast to the CLI, the GUI allows the user to perform alternative interactions with the 4DiCeS kernel. After an application data model has been set, the user has the choice between running a simulation on the model, manipulate it further, or load a completely different data set. Of course a simulation can also be interrupted by the user. If the user has completed his task, he can instruct the application to terminate. Object instantiation and deletion work for both scenarios equally well.

(SOs). It can also take advantage of Windows regular Dynamic Link Librarys (DLLs). Having two incompatible strategies on two different architectures leads to cross-platform-incompatible code.

To better resolve this problem of cross-platform compatibility, Alex Holkner

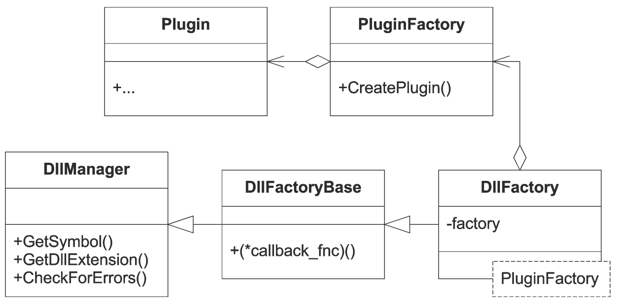


Figure 4.23: Static diagram: DynLoad. The DynLoad is designed as a dynamic link factory method that lets a class defer its instantiation to subclasses. This class checks the current operating system and loads the required file format accordingly. Hence, self-controlled DynLoad is able to open the necessary file and dynamically link it to the loading system framework.

(Holkner, 2002) therefore introduced a simple API, called "DynLoad", which allows developers to write dynamically-linkable code once and for all platforms. A new compilation of code is however required for each destination platform. Holkner's reference code consists of a dynamic link factory method (see Figure 4.23). This method defines an interface for creating an object but allows subclasses to decide which class to instantiate. Here it determines the file format by checking the current operating system. It can hence automatically incorporate the necessary file and dynamically link it to 4DiCeS.

The interfaces were implemented to allow for an easy connection to other programming languages. Special attention was paid to languages that do not need a recompilation if transferred to another platform. As a first prototype an interface connection to Java is provided. It compiles and runs by a Java Virtual Machine (JVM) implementation, which is natively ported to most current operation systems. Python as another programming language can be used through the Java interface.

This open interface structure should allow for the integration of numerous implementations by other programmers in their favorite programming language. Updates to the modules can easily be integrated to the system for the same reason. This modularity also allows for much more since the system can be directly adjusted to suit the needs of a user. The user interface itself is therefore designed to allow various administrative and analytical methods on the 4DiCeS framework.

The following Section 4.3.1 is concerned with the current realizations of user interfaces to the 4DiCeS application. Section (Section 4.3.2) then states what a user can expect from the current state of the application and Section 4.3.3 thereafter considers the overall computational complexity the 4DiCeS system might experience in regards to storage and performance. The last section of this chapter (Section 4.3.4) gives details on where the 4DiCeS program and accompanying documentation can be obtained from.

4.3.1 User Interfaces

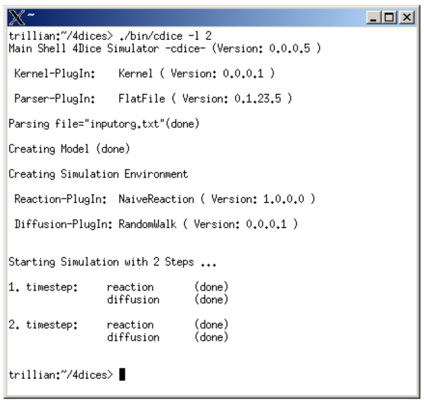
There are two user interface implementations for 4DiCeS at present. Figure 4.24 demonstrates these realizations as well as one CAVE Automatic Virtual Environment (CAVE) visualization (Figure 4.24(b)) of 4DiCeS output data. Figure 4.24(a) shows a CLI execution. As can be seen a full run of initialization and simulation iterations was performed. Figure 4.24(c) then shows a GUI implemented to allow for convenient state-of-the-art user interaction with the software.

The CLI can be used as a command-line shell approach to the 4DiCeS project. All necessary settings can be adjusted by giving it lines of textual commands either from keyboard input or from a script. The 4DiCeS CLI can perform operations in a batch processing mode without user interaction. If there is thus the need to repeat certain simulations more often, a "script" could hold all the necessary information. The operation can thereafter be conducted without any further effort in analysis. In contrast GUI users have to restart their exploration every time manually.

As for the GUI, the user is provided with a tool to view and verify the model and its geometrical structure. The GUI was designed as a Tabbed Document Interface (TDI) that allows for multiple documents to be contained within a single window (see Figure 4.24(c)), using tabs to navigate between them. The child documents can be freely resized, positioned, and undocked from as well as docked within the parent window. The single child documents contain different graphical or textual interfaces for either manipulating the model, triggering the simulation environment, or the visualization of simulation output.

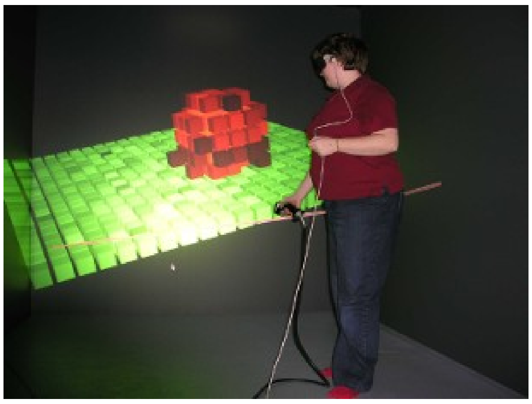
The GUI described here was developed by making intensive use of the Qt library by Qt Software (formerly Trolltech)[12]. It should be easily possible to implement an

[12]Qt Software: http://www.qtsoftware.com/

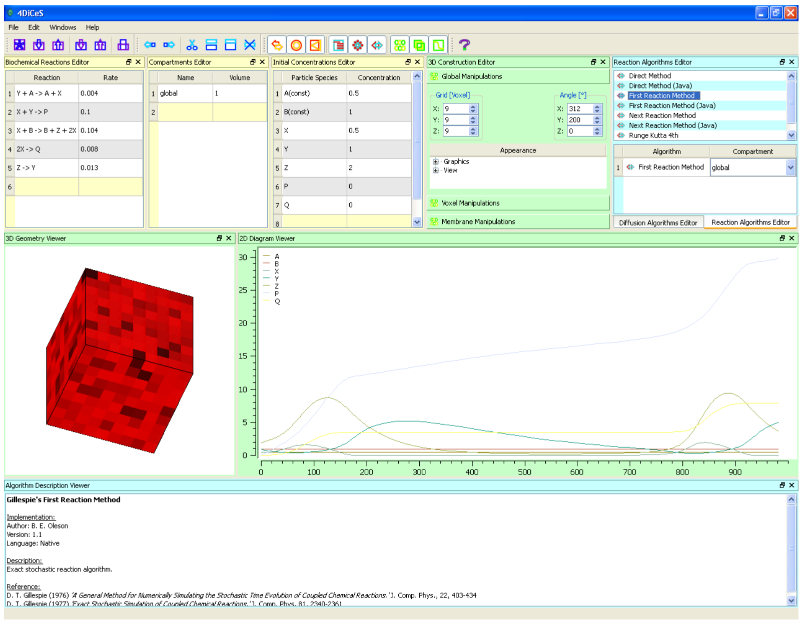


```
trillian:~/4dices> ./bin/cdice -l 2
Main Shell 4Dice Simulator -cdice- (Version: 0.0.0.5 )

 Kernel-PlugIn:    Kernel ( Version: 0.0.0.1 )

 Parser-PlugIn:    FlatFile ( Version: 0.1.23.5 )

Parsing file="inputorg.txt"(done)

Creating Model (done)

Creating Simulation Environment

 Reaction-PlugIn:  NaiveReaction ( Version: 1.0.0.0 )

 Diffusion-PlugIn: RandomWalk ( Version: 0.0.0.1 )


Starting Simulation with 2 Steps ...

1. timestep:       reaction        (done)
                   diffusion       (done)

2. timestep:       reaction        (done)
                   diffusion       (done)


trillian:~/4dices> ▮
```

(a) CLI Realization



(b) CAVE Visualization



Biochemical Reactions Editor

	Reaction	Rate
1	Y + A -> A + X	0.004
2	X + Y -> P	0.1
3	X + B -> B + Z + 2X	0.104
4	2X -> Q	0.008
5	Z -> Y	0.013
6		

Compartments Editor

	Name	Volume
1	global	1
2		

Initial Concentrations Editor

	Particle Species	Concentration
1	A(const)	0.5
2	B(const)	1
3	X	0.5
4	Y	1
5	Z	2
6	P	0
7	Q	0
8		

3D Construction Editor

Global Manipulations

Grid [Voxel] X: 9 Y: 9 Z: 9

Angle [°] X: 312 Y: 200 Z: 0

Appearance

Graphics

View

Voxel Manipulations

Membrane Manipulations

Reaction Algorithms Editor

Direct Method

Direct Method (Java)

First Reaction Method

First Reaction Method (Java)

Next Reaction Method

Next Reaction Method (Java)

Runge Kutta 4th

	Algorithm	Compartment
1	First Reaction Method	global

Diffusion Algorithms Editor | Reaction Algorithms Editor

Algorithm Description Viewer

Gillespie's First Reaction Method

Implementation:
Author: B. E. Oleson
Version: 1.1
Language: Native

Description:
Exact stochastic reaction algorithm.

Reference:
D. T. Gillespie (1976) *'A General Method for Numerically Simulating the Stochastic Time Evolution of Coupled Chemical Reactions.'* J. Comp. Phys., 22, 403-434
D. T. Gillespie (1977) *'Exact Stochastic Simulation of Coupled Chemical Reactions.'* J. Comp. Phys. 81, 2340-2361

(c) GUI Realization

Figure 4.24: The user interface for 4DiCeS. Controlling a program on the command line (a) is not very convenient, but has the advantage of being able to use massive computing. This list is then automatically processed and runs without additional user input until completion. A GUI (c) is needed if there is a user who wants to see direct run-time results and needs to make changes to the system as it runs. Here the front panel of the current realization of the 4DiCeS-GUI is shown. The GUI allows 4DiCeS users to choose their desired application data model, and manipulate it during program run-time. Lastly a picture (b) was taken of Julie Stromer (at the University of Calgary, Canada) handling a 4DiCeS visualization inside of the 4D virtual environment (CAVE).

equivalent or even more powerful GUI with another library and the currently supported set of programming languages. As a feasibility study, the 4DiCeS user interface mechanism was connected to a 4D virtual environment (see Figure 4.24(b)) (CAVE) (DeFanti et al., 1993). This implementation used Java and its 3D extension library Java3D.

4.3.2 Applicability

Although the 4DiCeS kernel is the central part of this work, the application itself is mainly accessed through user interfaces. Therefore this section specifies the human-machine interaction with 4DiCeS, its possibilities, and, of course, its limitations in greater detail.

As already stated in the previous sections two distinct reference implementations of user interfaces exist at present. The first, the CLI, serves as an automation utility for massive throughput simulations. Modeling is not a concern of this user interface, since no further interaction can be accomplished after the application was started in this way. On the contrary the GUI reference implementation mainly concentrates on visualizing the design of a model, geometry details, and the output of a simulation.

The model design functionality ranges from importing existing models to freely defining new model scenarios off the scratch. Here various child windows of the parent TDI window support the user in editing reaction and diffusion parameters as well as simulation space settings. There currently exist textual editors for setting reactions, compartments, as well as initial parameters. All dialogs together form a tool to model reaction-diffusion systems and specify further dependencies as of connected algorithms and the geometry information.

Since both the 4DiCeS kernel and the application data model provide support for 3D simulations, the GUI's visualization output can also be displayed in 3D. This task comes with some difficulties and limitations. As everything is simulated at once, it is difficult for the user to trace all the occurring events in a compact 3D image on a 2D display. Differentiating also between different particle species is limited to a small set of different particle species at a time due to the internal computer coloring scheme.

At present the underlying reference geometry description definition (cell model) is visualized as a 3D grid of cubes covered by a combination of the three fundamental colors obeying the red, green, and blue (RGB) color scheme. Each color represents one particle species, and the shading of this color indicates the particle concentration in that specific voxel. Since there are 256 different shades for each color on a computer, this number limits proper differentiation of different particle concentrations. For the user to be able to get the most out of such a data display, 4DiCeS provides features to manipulate the proportionality scale of colors to represent logarithmic changes in concentration. The total change of concentration is displayed by a 2D plot next to the 3D visualization.

As can be seen from attempts made in the following Chapter 5 in testing the system itself (see Section 5.2) and the comparison study with related tools (see Section 5.3) 4DiCeS is a well suited tool for the modeling and simulating of biochemical reaction-diffusion systems. Furthermore it is feasible for the analysis of highly dynamic models as of the oscillating BZR (see Section 5.2.2.2) and the two pool calcium oscillation model (see Section 5.2.2.3). The now following section will consider the overall computational complexity the 4DiCeS system might experience with such experiments in regards to storage and performance.

4.3.3 Complexity Considerations

This section is concerned with the prediction of resources that the 4DiCeS application requires in total. The influence of the algorithms (see Appendix A) is actually very minimal, since they all grow asymptotically slower than n^2 and the traversal of VEs in 3D during one simulation iteration already yields an $O(n^3)$ upper bound on the overall worst-case running time.

But the discrete Lévy process should then be considered separately. As already stated earlier this algorithm does not depend on the grid of VEs in the simulation space, but only traverses over all particles during one iteration. Anyhow, by the time the result output has to be stored to the kernel then again all VEs have to be traversed. Algorithms that have to perform some sort of initialization routines sum up this overhead during a simulation over more than one VE over time.

Therefore it can be said that the overall worst-case performance of 4DiCeS converts in $O(t \times r \times p \times x \times y \times z)$, were t represents time, r denotes the number of reactions, p is the total number of particle species, and the product of x, y, as well as z describes the total number of VEs in 3D. The storage worst-case complexity is growing in a very similar asymptotically expansion, because the same six-tuple has to be considered when observing the overall memory consumption.

4.3.4 Availability

The 4DiCeS project, along with all its sources and relevant documentation is hosted by SourceForge[13]. The project's domain is http://www.4DiCeS.de/. The domain address is automatically forwarded to the project's web-content at SourceForge (http://four-dices.sourceforge.net/).

For compiling the 4DiCeS sources, additional programs, libraries, as well as extensions are required as can be seen in Table 4.13. A detailed description for various C/C++ Integrated Development Environments (IDEs) is included in the 4DiCeS documentation. The current version of 4DiCeS has been tested and developed under Linux and Windows. As long as the tools and packages described above are in place, 4DiCeS should install and run smoothly.

Tool	Description	Link
Boost	C++ extensions library.	http://www.boost.org/
CxxTest	C++ unit-testing.	http://cxxtest.sourceforge.net/
Java	Programming language.	http://java.sun.com/
libtiff	C TIFF library.	http://remotesensing.org/libtiff/
Perl	Scripting language.	http://www.activeperl.org/
Python	Scripting language. ● Jython for Java:	 http://www.jython.org/
Qt	C++ GUI library.	http://www.qt.no/

Table 4.13: Required third-party tools. Perl and CxxTest are only for compiling and running 4DiCeS unit tests. Qt is required for compiling and running the 4DiCeS-GUI. The use of Java depends on the need for this specific programming language on plug-ins within 4DiCeS.

The source-code as well as the compiled program(s) are licensed under the GNU Lesser General Public License (LGPL)[14] and are therefore free for non-commercial use. Commercial usage can only be granted after consultation with the current source-code administrator.

[13]SourceForge: http://sourceforge.net/
[14]LGPL: http://www.gnu.org/licenses/lgpl.html

CHAPTER 5

Applications, Results, and Analysis

In this work the 4DiCeS framework was developed for modeling and simulation of cellular dynamic phenomena in 3D. As described in the previous chapter, the implementation of this project includes a modular design of the 4DiCeS kernel and four modular interfaces. Various reference implementations of algorithms add up to these interfaces (see Table 5.1).

Module	Implementations
User Interface	Command-Line Interface (CLI). Graphical User Interface (GUI).
Parser	4DiCeS Flat-File. 4DiCeS TIFF Geometry-File. Systems Biology Markup Language (SBML).
Diffusion	Discrete Lévy Process.
Reaction	Direct Method. First Reaction Method. Next Reaction Method. Runge–Kutta 4th Order ODE Solver.

Table 5.1: Interface reference implementations. The 4DiCeS framework comprises a variety of different reference implementations for the given four interfaces. Additionally to these modules an implementation of the COntrollable Approximative STochastic (reaction-algorithm) (COAST) is also available by kind curtesy of Dr. Mark Möller.

With these first modules it was possible to test and verify the framework in its behavior and reliability. Of special interest was the impact of diffusion to well known reaction algorithms. The propagation of wave-like structures of oscillating model systems was expected but not detected. Although it was possible to generate massively random 3D grids out of uniformly distributed particles there were no concrete pattern formations perceived. A discussion of this finding and all other results will be handled in Chapter 6.

The following sections now want to give insight to further outcomes in regards to application development (Section 5.1), applied models for simulation (Section 5.2), as well as ten reference programs in comparison to 4DiCeS (Section 5.3). The last Section 5.4 summarizes the findings from the comparison of 4DiCeS with the other ten tools.

5.1 Application Development

As previously stated the main characteristic of the simulation tool is its central simulation kernel and the interfaces to the reaction as well as diffusion modules, the model input parsing environment, and the visualization (see Table 5.2). Front-end user modules support both the CLI processing and a GUI-based version.

This section examines the resulting plug-in structure of the module interfaces. The possible connectivity to other programming languages then follows directly out of this architectural definition.

5.1.1 Plug-Ins

Every interface presented here (see Tables 5.1 and 5.2) has a special module extending the interface to expand to other programming languages (see Figure 5.1). These modules make use of the underlying 4DiCeS kernel API. The API's functionality again is passed on to the respective language module plug-ins.

A plug-in generally consists of a module instantiation function and an object destruction facility to enable a safe shout-down behavior of the kernel. These two maintenance methods provide a simple interface to all plug-in interfaces. The instantiated modules are then specialized forms of either interface class. These classes bring along the connection to the underlying model description, kernel functionality, and offer distinct entry points to execute their particular operation.

In the case of the algorithmic modules the entry points are functions named *diffuse* and *react*. These functions either work with entire segmentations (compartments) or on single voxels. The kernel as well as the model binding are rather limited to methods for retrieving information. The modification of data

Interface	Description
Reaction	This interface handles the reaction equations and kinetics. After a successful reaction, the resulting particle quantities or concentrations are passed to the application data model for further processing.
Diffusion	The shared diffusion library holds the complete grid of VEs with all relevant structural information and content. After a successful diffusion, the entire simulation space must be updated for further processing.
Input/Output	This set of interfaces is responsible for the simulation model input and output capabilities. The modules are able to automatically recognize the data formats and securely integrate their information into the 4DiCeS data model.
Visualization	This interface is designed to cope with all sorts of different visualization techniques (CLI and GUI). It must ensure that all possible data is made available through the interface.

Table 5.2: 4DiCeS interface definitions. This table briefly describes the kernel interfaces. The reaction and diffusion processes are all handled within the simulation space. The simulation interfaces (reaction and diffusion) are computationally more demanding than the input parsing modules. There are normally only a few loading and saving events during a simulation. Hence there is no need for highly optimized computational efforts on this topic. The visualization interface can be of both types if CLI and GUI applications are compared.

is restricted to the extension that an algorithm must not alter the structure of a model. It should only modify its values.

In contrast the parsing and visualization interfaces have full control over a model. The main difference between these two is the connectivity to the kernel. While a parser plug-in is controlled by the kernel via its entry points, *parse* and *save*, the visualization has full control over the kernel. The kernel actually acts as a plug-in to the user interface rather than the user interface to the kernel. This structure allows for the easy integration of simulation results into other existing tools and frameworks for further analysis.

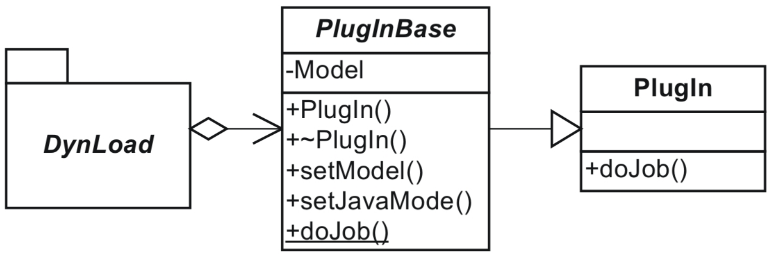


Figure 5.1: Static diagram: A general plug-in interface. The 4DiCeS API provides users with the choice of which algorithm to use, and programmers with the ability to modify given modules or to implement new ones. The API provides additional programming language plug-ins for each interface.

5.1.2 Programming Languages

C/C++, Sun Microsystems Java, and Python were selected as the project's programming languages. All are commonly used in computational biology and have their respective benefits. C/C++ was used for the system's kernel and the interface implementation. The other programming languages were chosen as extensions to guarantee an easy API to the 4DiCeS framework. Thus the widespread use of either language will allow many researchers to use 4DiCeS as a workbench for their implementation of algorithms and tools.

All libraries used were carefully chosen to allow for an easy migration of the 4DiCeS framework between operation systems. The current target platforms are the Microsoft Windows operating systems (95 and upwards) as well as current POSIX derivatives including Linux. It should be easy to port the program to other operation systems by a simple recompilation of the given C/C++ sources.

5.2 Modeling and Simulation

As described in Section 4.2.1, the geometrical representation of the 4DiCeS data model consists of a grid of VEs. As each VE can be covered by membrane, 3D compartmental substructures can be included to the grid. Hence it is possible to define particle barriers in the 3D simulation space. This also means that different kinds of boundary conditions can be modeled in the system. Before the two models from Section 5.3 were applied a few test simulations examined the overall functionality of the system.

The first subsection illustrates a simple diffusion application in 4DiCeS to test the system's 2D and 3D capabilities. Then the second subsection demonstrates diffusion-reaction systems from simple chemical reactions to finally a non-linear system such as the BZR. All grids are divided into $9,261$ VEs ($21 \times 21 \times 21$-grid) with an edge length of 100 nm if not stated otherwise. The analyzed time-points are enclosed with the figures showing the simulation results.

5.2.1 Simple Diffusion Simulation

The testing of the diffusion functionality of 4DiCeS was deemed early on as being of great value. The geometry of models and the diffusion within the modeled geometry had to be verified in case studies involving 2D as well as 3D space. It was therefore necessary to design test models of only the diffusion components at first. Two of these models will be presented in the following subsections. The membrane-bound diffusion model in lateral (2D) and the 3D displacement model are illustrated first. Thereafter a more complex membranous system will be introduced, that tests if boundary conditions are securely handled by 4DiCeS.

5.2.1.1 Lateral and 3D Diffusion

The simplest diffusion test described here consists of a cubic grid of VEs. A continuous x–y-layer of membrane is set near to the bottom of the grid defining two separate compartments within the grid. Four of the six simulation space boundaries are not limited by membranes. Their boundary conditions at these four borders allow for the direct migration of particles from one side of the simulation space to the opposite side. The only exceptions to this are the top and the floor layer. Membranes have to prevent particles from diffusing out of the top VEs and into the bottom layer, which defines the other compartment. Thus the geometrical model is a closed system with a membranous barrier to the bottom of the grid. Two diffusible particle species are placed in this model. One is assumed to translocate laterally over the membrane, whereas the other is permitted to move freely in 3D. The membrane is set to hinder the freely diffusing particles from switching from one compartment to the other.

As can be seen from Figures 5.2 (a) to (c), this simple application of diffusion works as expected. Both the lateral and the 3D diffusion mechanisms show anticipated translocation. The dividing membrane stops the further diffusion

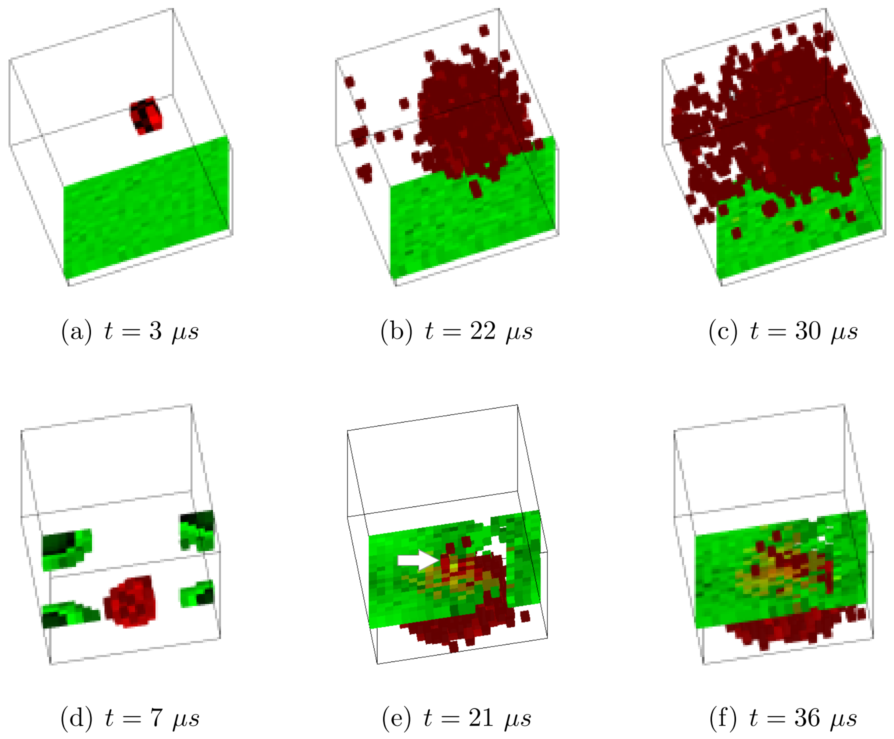


Figure 5.2: Diffusion application. The Figures (a) to (c) illustrate the simple diffusion of particle species on a lateral (2D) membrane and in free 3D space. With Figures (d) to (f) the grid is divided into two compartments again, but with an opened tube in the membrane's middle. For both applications, the green particles are bound to the membrane of the model and can only diffuse along this barrier. Conversely, the red particle species is allowed to diffuse freely in 3D space with the only limitation of not being able to cross a membrane. In the second model the red particles can pass the tube's top opening (see arrow in Figure (e))

of the 3D diffusing particle species to the other side of the boundary. Also the laterally diffusing particles stay to their membrane bound condition as expected.

5.2.1.2 Complex Membrane System

Here the diffusion is tested on a more complex membranous system. The overall model is still a closed system. Particles are still able to leave four sides of the grid at one side and reenter it on the opposite side. The top and bottom layer are blocked for diffusion. Instead of the previous plain membrane layer near the floor of the grid, a membranous tube is placed on top of the structure. The tube is uneven and opens to the top of the grid. The simulation of diffusion (see Figures 5.2 (d) to (f)) begins with freely diffusible particles at the bottom right corner of the grid. Next to the 3D diffusion, the 2D diffusion is started. The particles diffuse along the tube to its top opening. At the end of the simulation, the 3D particles were able to enter the opposite compartment through the tube's top opening.

5.2.2 Applying Diffusion-Reaction Systems

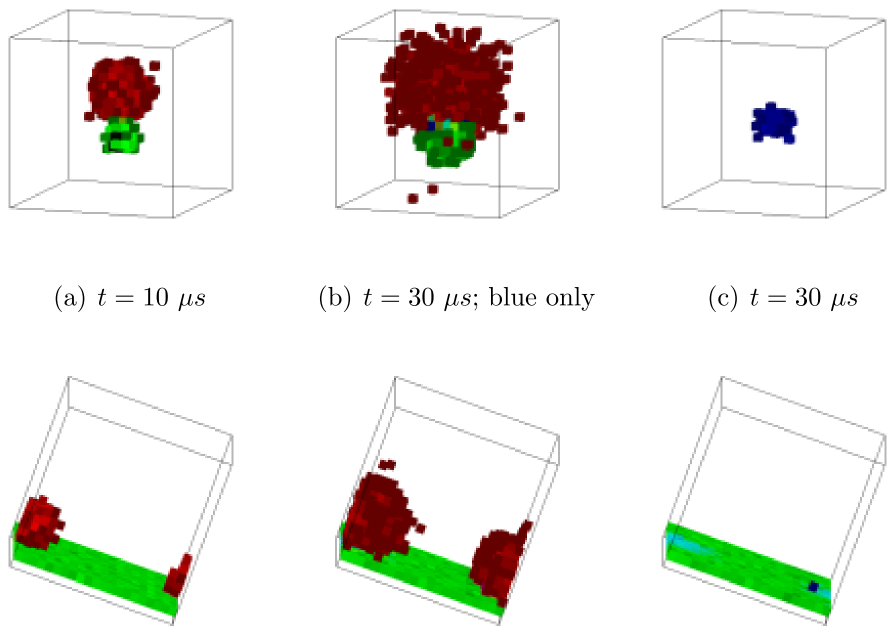
Besides diffusion the reaction capability of the 4DiCeS framework was also tested. Since reaction systems ignoring diffusion would only evaluate the reaction algorithm by its own and not the 4DiCeS framework completely, two diffusion-reaction systems were applied for testing. At first a simple chemical reaction was verified in a membrane-free grid as well as under boundary conditions. Then a BZR was used to heavily test oscillating chemical systems.

5.2.2.1 Common Saturation Reactions

The test of a simple chemical reaction system of the form

$$A \overset{B}{\rightleftharpoons} C \tag{5.1}$$

was handled using two different geometrical models (see Figure 5.3). The first model is completely free of any membranous boundaries, whilst the other contains a grid structure equal to that described in Section 5.2.1.1. The reaction consists of an educt A, a product C, and a catalyzing enzyme B. In the first simulation experiment (see Figures 5.3 (a) to (c)), all particle species (A, B, and C) are allowed to diffuse freely in 3D space. In the second case (see Figures 5.3 (d) to (f)), the enzyme species B is limited to the dividing membrane.



(a) $t = 10\ \mu s$ (b) $t = 30\ \mu s$; blue only (c) $t = 30\ \mu s$

(d) $t = 10\ \mu s$ (e) $t = 30\ \mu s$ (f) $t = 30\ \mu s$; red hidden

Figure 5.3: A simple chemical reaction. The reaction system of the form $A + B \rightleftharpoons B + C$ is tested with the 4DiCeS framework. Figures (a)–(c) illustrate the reaction without any membranous barriers and (d)–(f) show the same reaction system with a membrane layer. The colors green and red stand for the reactants in this system. In (d)–(f) the green particles (enzymes) are membrane bound and must diffuse laterally along the membrane. The Figures (c) and (f) show the reaction product C as a blue particle species. Here, in (c) both educts A as well as B and in (f) only A were hidden away to have a better display of the product.

As can be seen, the reactions start with particles of both educts present in the hosting VEs. The simulation was halted shortly after the first products appeared. As expected the educts A continuously react in the presence of enzyme B to the product C. The membrane-divided model needed more time for first reactions to occur. This was expected since the educts A had to move the entire distance to the anchored enzymes B.

5.2.2.2 Sustained Oscillating Reaction

The BZR (see Equations 4.24 to 4.28 at Section 4.2.3.3) is probably the most widespread oscillating reaction system both theoretical and experimental. On the theoretical side, the Field–Körös–Noyes (FKN) model system (Field and Noyes, 1974a,b) quantitatively mimics the actual BZRs (Field et al., 1972). The Brusselator (Glansdorff and Prigogine, 1971; Nicolis and Prigogine, 1971) as well as the Oregonator (Noyes, 1976a,b) are then variations of the FKN model system (Ipsen et al., 1997).

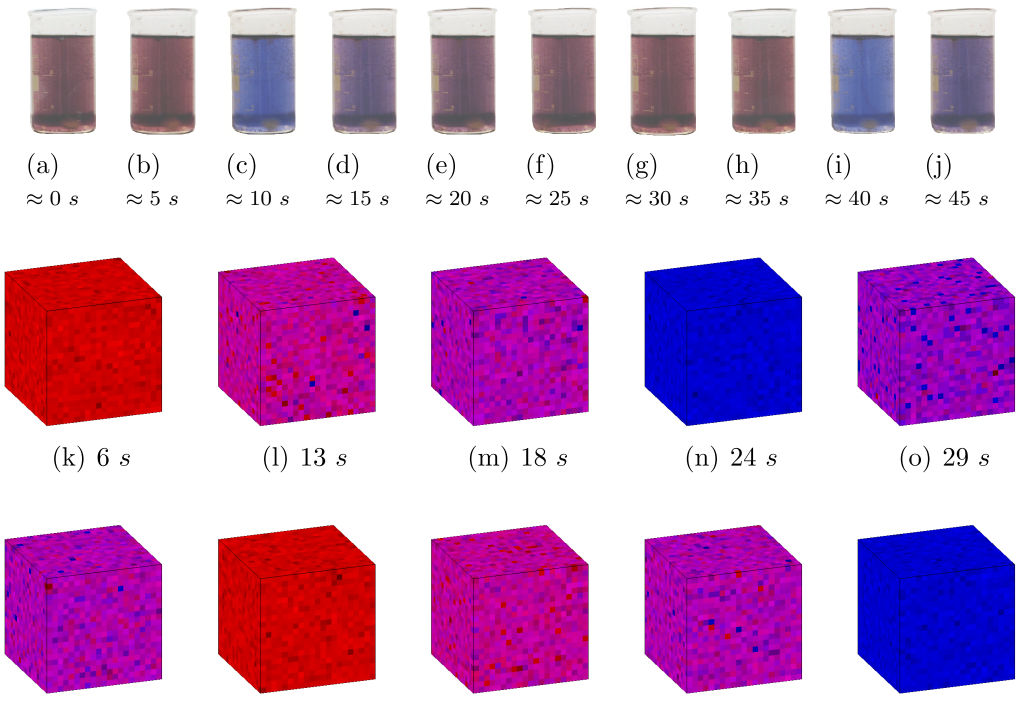
Boris Belousov stated that in a mix of potassium bromate, cerium sulfate, propanedioic acid and citric acid in dilute sulfuric acid, the ratio of concentration of the cerium(IV) (Ce^{4+}) and cerium(III) (Ce^{3+}) ions in combination with bromate(V) (Br^-) oscillated.

Basically the reactions can be divided into two parts. The concentration $[Br^-]$ determines which part is dominant at any time. While there is a high $[Br^-]$ the first part is dominant. During this stage Br^- is consumed and the cerium is mainly in the Ce^{3+} state. As $[Br^-]$ decreases further it passes through a critical value and then drops quickly to a low level. At this stage the second part takes over. The Ce^{3+} changes to Ce^{4+}. Ce^{4+} reacts to produce Br^- again while it reverts to the Ce^{3+} state. Now $[Br^-]$ increases. By the time $[Br^-]$ is sufficiently high the first part gets dominant again (Murray, 2002a,b). The whole sequence is continually repeated and hence produces colored oscillations.

The test environment in the 4DiCeS framework considers no internal membrane barriers. Only the grid side walls are covered with membrane to change the model to inflexible boundary conditions. The particle species are initially distributed equally over the grid of VEs. As expected it can be seen (see Figure 5.4) that an oscillation of the reactants' concentrations takes place with a cycle duration of nearly 30 s in the hosting VEs. The simulation shows a short latency of about 8 s at the very beginning. The oscillation process was halted after one full cycle ($t = 55\ s$). There are no distinguishable wavefronts, but that is due to the rather big VEs compared to the rather small grid.

5.2.2.3 Two Pool Oscillatory Calcium Model

The two pool model (see Section 2.2.2) describing oscillating Ca^{2+} changes within a cellular model was also applied to 4DiCeS. Figure 5.5 shows the output of four simulation iterations with slightly varying parameters. This difference in



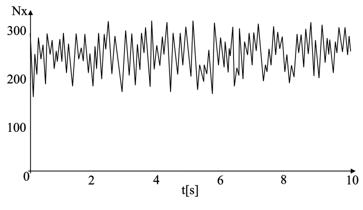
(a) ≈ 0 *s* (b) ≈ 5 *s* (c) ≈ 10 *s* (d) ≈ 15 *s* (e) ≈ 20 *s* (f) ≈ 25 *s* (g) ≈ 30 *s* (h) ≈ 35 *s* (i) ≈ 40 *s* (j) ≈ 45 *s*

(k) 6 *s* (l) 13 *s* (m) 18 *s* (n) 24 *s* (o) 29 *s*

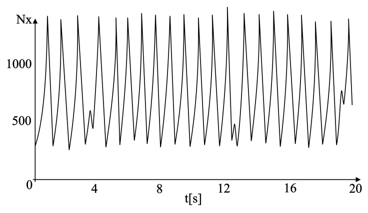
(p) 33 *s* (q) 38 *s* (r) 44 *s* (s) 49 *s* (t) 54 *s*

Figure 5.4: An oscillating reaction. Here, the BZR system was tested with the 4DiCeS framework. The images (a)–(j) illustrate a stirred 60 *ml*-beaker with BZ reactants. Adapted from Winfree (1987, 2001). The images (k)–(t) show the same BZR from a 4DiCeS simulation. The times-step for the simulation was set to 1 *s* and the edge length of the VEs was 5 *mm*.

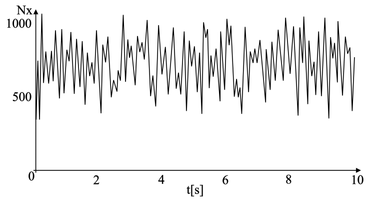
output can only be seen, due to the fact that the stochastic model was applied. The deterministic version of the model does not show this behavior Kraus et al. (1992). All parameters used for the simulation are stated within the caption of Figure 5.5. The application of a diffusion term and the simulation of the pool model in 3D did not show any wave-patterns. Reasons for this behavior could be the coarse granularity of the simulation space ($21 \times 21 \times 21$) or a bad selection of either the time-steps or the edge length of the VE. Although, varying these geometry parameters did not produce any better results. The overall particle number dynamics output was very similar to Kraus and Wolf (1992).



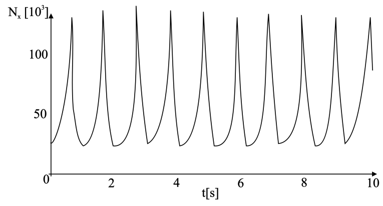
(a) $\beta = 0.20$



(b) $\beta = 0.40$



(c) $\beta = 0.80$



(d) $\beta = 0.40$

Figure 5.5: Stochastic Simulation of Two Pool Model. The two pool model was stochastically simulated with different strong stimulations of the receptors expressed through β. The diagrams show the number of Ca^{2+} ions in the cytosol (N_x) as a function of time t. The parameters for (a) to (c) are (see Table 2.4): $n = m = 2$, $p = 4$, $v_0 = 1000s^{-1}$, $v_1 = 7300s^{-1}$, $k = 10s^{-1}$, $k_f = 1s^{-1}$, $K_2 = 1000$, $V_{M2} = 32500s^{-1}$, $k_R = 2000$, $K_A = 900$, and $V_{M3} = 2.5 \times 10^5 s^{-1}$. The parameters of (d) are chosen that way that they are by a factor of 1000 greater than (a) to (c). The specific βs can be found in the corresponding caption of every sub-figure. The times-step for the simulation was set to 0.01 s and the edge length of the VEs was 0.095 μm. The model description and all its reaction parameters were extracted from Kraus and Wolf (1992).

5.3 Comparing Tools

This section continues the comparison of the ten modeling and simulation tools from Section 3.2.4. In addition this study brings all presented tools in context with the 4DiCeS application, which was introduced in Chapter 4. The first Section 5.3.1 will reconsider the criteria important for comparing all the applications. Then the following Section 5.3.2 will compare the applications to each other in more detail; while, Section 5.3.3 will deal with 4DiCeS.

5.3.1 Comparison Criteria

Next to the 12 comparison criteria defined in Section 3.2.4 it was crucial to see how the applications handle equal test scenarios. Therefore the final tests for every tool were two oscillating test models.

As a straight forward model, the previously described BZR (see Section 5.2.2.2) was applied either in the form of differential equations with concentrations or by reaction equations with particle numbers. This first test case requires no compartmentalization of the model in any way. The second test is based on the two pool model (as described in Section 2.2.2). Here the simulation space is subdivided into two distinct compartments: the cytoplasm, and the ER.

Both models should show oscillation as a result similar to the one found in literature. Depending on the ability of the application to provide spatial simulations, it will further on be attended to apply both test cases to 3D. Again wave-structures (as found in literature) should be observable.

5.3.2 Comparing Related Works

For this comparison it was important to consider the very different strategies of the applications, rather than bringing almost equal tools into contrast. The choice of the given applications is therefore a mixture from deterministic, stochastic, and hybrid approaches.

Some tools are already rather dated, some allow for 3D simulations, and some have proprietary or standardized file format extensions (see Tables 5.3, 5.4, and 5.5). The ten simulators were tested against the test models as defined in the last section. Here the implementations of the test cases were not necessarily straight forward to all compared programs.

A feasible implementation of the model was however possible with almost all ten applications. The results of these simulations are found to be nearly convergent for most of the tools. Variations observed may be due to different methods for solving the equations or application data model incompatibilities. In the following the results are presented in further detail.

BIOCHAM: Installing BIOCHAM was accomplished with a simple button click. The only prerequisite is for the computer to have a JVM. The application tested was version 2.5, which can be used under the terms of the GNU General Public License (GPL). BIOCHAM allows for simulations with either a boolean

CA algorithm, the Runge–Kutta 4^{th} order solver, or the Rosenbrock's 3^{rd} order solver. With this pure deterministic simulator, the modeling and the results can only be displayed in one dimension. Models can either be defined in BIOCHAM's own description language (BM) or by SBML files. The simulation results are displayed in a common 2D plot and can be viewed as a table. Explicit export functionality for the results is however missing. Both testing models could be applied to the application by setting up the reaction rules in BM. Importing a model through valid SBML files did not work well. Both the boolean and the numeric solvers gave the same predicted results, and it can be said to be accurate and very fast ($< 1sec$).

BioNetS: The download of BioNetS is split into several packages. For the application to run, all these packages have to be installed. In particular for version 2.0 of BioNetS, a JVM and GNUStep, a cross-platform, object-oriented framework for desktop application development must be present. Since the application produces C++ code from given models, there is always a compilation iteration preceding any simulation attempt. Models can then be simulated with either a Langevin equation solver or by the use of the Next Reaction Method (Gibson and Bruck, 2000). The description of models is only possible with BioNetS's proprietary XML file format BNET, but a modeling GUI takes away any need to cope with that directly. The setup of the two testing models was difficult for another reason. Both solvers can handle only very limited particle amounts that is why the models' particle numbers had to be condensed a 1000-fold. Doing so resulted in the predicted results for both solver classes. Simulation times ranged from below one second for the Langevin solver and several minutes for the next reaction method. The application can be used under the terms of the Berkeley Software Distribution (BSD) license.

Copasi: This simulation tool was tested in the beta version 4.0.19 and was easy to install. The tool's dependency on the rather expensive GUI library Qt is only of importance if the user wants to recompile the application from source code. This is possible since the source code is provided under the Copasi Non-Commercial License (CN-CL) and is therefore free for academic usage. Import and export file formats are in CPS, GPS, and SBML. As for simulation methods, the programm ranges from a pure ODE solver, to stochastic algorithms, to hybrid forms. The modeling of reactions leaves nothing to be desired. The tool is able to perform steady-state, time course, as well as parameter estimation simulations. In the case of the two applied test models, Copasi provided expected results in the time-range of seconds to several minutes depending on the algorithm used. The results are displayed in plots and tables but cannot be exported as such directly.

Designation	Methods	Version	License	Dependencies
BIOCHAM	CA, RB_3, & RK_4	2.5	GPL	Java
BioNetS	LE & NR	2.0	BSD	Java & GNUStep†
Copasi	NR, $LS^{(h)}$, & ${RK_4}^h$	$4.0.19^b$	CN-CL	Qt*
E-Cell	DA & NR	3.1.105	GPL	Python* & GTK*
Gepasi	LS	3.30	Freeware	-
MCell	MC	2.50	NRBSC-SL	Cygwin† (DReAMM)
SmartCell	NR	2.5	ASL	Java& Java3D
StochSim	SA	1.6	LGPL	Perl & Tcl/Tk
VirtualCell	AM_5, FE, LS, & $RK_{2,4,5}$	4.3^b	Freeware	Java
xCellerator	Mathematica & NR	0.27	LGPL	Mathematica

Table 5.3: Tested simulation tools. The table lists all simulation tools tested in this chapter. Here the specific features for simulation are applied methods and algorithms (Column 2), the version number of the tested software (Column 3), the license agreement depending on the version number (Column 4), and further software dependencies to the application are included. The methods of Column 2 are further described in Table 5.4 with the exception of the xCellerator software, which utilizes the equation solvers provided by Mathematica. A superscripted uncapitalized ‘h’ determines an hybridized form of the given method. If there are both variants (hybridized and plain), then the uncapitalized ‘h’ is parenthesized. A subscripted number gives further information about the order of consistence an ODE solver is working on. A superscripted ‘b’ with version numbers in column 3 denotes a beta-version. The dependencies in column 5 contain special footnote-marks that indicate that the additional package is either provided as a separate download (†), or is already included ($*$) in the installation routine of the software package. All other dependencies are prerequisites to either installation or running of the program and must be obtained and installed separately. Further information and links are provided by Section 3.2.

Mnemonic	Designation
AM	Adams-Moulton ODE Solver
CA	Boolean CA
DA	DAE Solver
FE	Forward Euler Method
LE	Langevin Equation Solver
LS	LSODA Solver for Stiff and Non-Stiff Systems
MC	MC Algorithm
NR	Next Reaction Method
RB	Rosenbrock ODE Solver
RK	Runge–Kutta ODE Solver
SA	StochSim Algorithm
τL	τ-Leap Method

Table 5.4: Contained algorithms. This table provides mnemonics for the methods and algorithms applied to the different software packages described in Table 5.3. Further information on such algorithms can be obtained from Section 4.2.3.3.

E-Cell: The E-Cell programm comes in version 3.1.105 under the terms of the GPL and can be easily installed. The prerequisite packages of the Python scripting language and the GTK-GUI library are automatically installed if not present. The modeling is aided graphically. Model description languages are the E-Cell's own EM/EML formats and SBML. For simulation the user can choose between Differential-Algebraic Equation (DAE) solvers and an implementation of the next reaction method. The applied test models had to be remodeled with the help of E-Cell's model editor but ran as expected afterwards. Simulation time ranged from a few seconds to several minutes depending on the used simulation method. The output can be visualized both by plots as well as tables. The export of the data is thus not explicitly possible. It should be noted that the application could not be restarted after the computer was turned off and on again after the first installation.

Gepasi: As the oldest of all the tested applications, Gepasi is also the smallest in size and the fastest when it comes to comparison of ODE solvers. As of the

Designation	File Formats
BIOCHAM	BC* & SBML
BioNetS	BNET* & SBML
Copasi	CPS*, GPS & SBML
E-Cell	EM/EML* & SBML
Gepasi	GPS* & SBML
MCell	MDL*
SmartCell	XML* + SBML
StochSim	SBML
VirtualCell	SBML (CellML, Matlab, and VCML)
xCellerator	NB* & SBML

Table 5.5: Supported file extensions. All tested applications have their own proprietary file formats (denoted by an '$*$'). Most of the packages support the import and export of SBML files. In the case of Copasi, the precursor file format from Gepasi is still supported. Further on, the VitualCell allows for model exports to the CellML, Mathlab, and the value chain markup language (VCML). This last file format is primarily used as a business collaboration standard and not for modeling in particular.

last released version 3.30, which is provided as freeware, the installation was easy. The only simulation methodology provided by Gepasi is the LSODA solver for stiff and non-stiff systems. A straight forward GUI and the import as well as export formats for its own model description file GPS and the support for earlier versions of SBML round out its appearance. The tested models both gave good results, which could be displayed in a 2D plot. The plot is produced by the required GNUPlot package, which comes with Gepasi already. As stated before the simulations were very fast and ranged for both test cases within a second of total execution time.

MCell: As a purely stochastic application with its own reaction and diffusion algorithm implementation, MCell stands apart from all other simulation programs considered here. It is a command-based tool in version 2.50. Running under the terms of the National Resource for Biomedical SuperComputing Software License (NRBSC-SL), MCell is free for academic usage and can be run in combination with a visualization utility named DReAMM. The dependencies to other software packages are limited to Cygwin for MCell alone and OpenDX as well as for UNIX X-Server in the case of DReAMM. The download page provides a single Cygwin-DLL that makes installation very simple. Since MCell only supports its own proprietary file format MDL and gives no further GUI tool at hand, the modeling of the test cases was very complex. Obtaining each of the simulation results took almost a day of computational time and results were far from what was expected. Both test cases showed no oscillations even after several modifications of the model.

SmartCell: Next to MCell SmartCell was not able to provide results on the simulation test cases as expected. The reaction equations, which can only be set up by SmartCell's graphical model editor, seem to always lose connection between reactants and their multipliers. Also SmartCell only supports SBML as a helper description that requires a SmartCell's XML reaction placement description in addition. Without this file SmartCell will neither let the user load nor run a single SBML model. SmartCell has many current versions ranging between version 2.5 and version 3.0. The current beta version of SmartCell 3.0 was tested and did not produce any output for the applied models. Consequently the promising 3D features, which make use of the required Java3D, could not be tested to their full extent. SmartCell is available under the terms of the Academic Software License (ASL) and utilizes the stochastic next reaction method for both reaction and diffusion process with version 2.5.

StochSim: Next to Gepasi StochSim is the oldest stochastic simulation tool of these considered here. It comes with its own algorithm for simulating every two possible reaction partners in discrete time steps. The installation of StochSim is similar to BioNetS with regards to unpacking and putting together of different files. If this is done and the required scripting language Perl and the GUI library Tcl/Tk are installed beforehand, the application is ready to run. In version 1.6 StochSim runs under the terms of the LGPL and can therefore be used freely. As for a model description format, it supports SBML. The test case models applied to the system showed the expected behavior although under high computational costs. The overall simulation time for both models reached nearly an hour for every simulation turn. This is even more surprising, since StochSim gives no support for 3D modeling. The results can be either directly viewed on the GUI or further analyzed by third-party table-calculation utilities.

VirtualCell: As the most powerful application of all discussed with respect to modeling editors and 3D geometry, the VirtualCell is also very different to the others. A server-based approach was chosen that handles the model storage and provides the necessary computational power. Then the Java-based client helps with the design and the setup of the model. The tested beta version 4.3 of the client can be used freely, but a user has to log on to the server every time he wants to use the application. This presupposes an internet connection for program use. VirtualCell provides various ODE solvers, a very sophisticated model editing environment, and 3D functionality. The test case could be easily applied via the SBML import and gave good results in 1D. Although the models both oscillated in 3D, the particles did not diffuse in space as predicted. This of course can only be a fault in the model design. Another feature of the VirtualCell is its ability to export models and output results directly to predefined formats including CellML, Mathlab, and the value chain markup language (VCML). This last file format can be used for the output data.

xCellerator: The xCellerator is very special because it is an 'add-on' to the Wolfram Mathematica package. Here the great benefit is the fact that all solvers and further features of input/output or visualization provided by Mathematica can directly be utilized. A major drawback is the fact that Mathematica is only available commercially. The description of models is done by Mathematica files that obey the additional palette rules of xCellerator. All resulting output for both simulations was as expected, and the simulation time sets standards. Both test case models needed less than a second to be solved. The tested program was of version 0.27 and can be used under the terms of the LGPL (provided that a valid license for Mathematica is obtained).

5.3.3 In Comparison to 4DiCeS

As with all the other ten simulation applications, 4DiCeS was critically checked for all the given test criteria (see Section 3.2.4). It has to be said that 4DiCeS is not able to compete with other applications in single disciplines. Other simulation applications were implemented under the precondition to solve one or a few very specific model scenarios.

In contrast to this, 4DiCeS was implemented as an open tool to solve multiple scenarios correctly but not fully optimized. Therefore 4DiCeS is a trade-off between features as well as functionality on the one hand, and on the other hand optimized computational demands. The feature list is not fully comparable. 4DiCeS provides some functionality that none of the other applications can compete with. Such features are the simulation with algorithmic concurrency and the open interface design. Contrarily 4DiCeS is missing certain functionality other tools have such as a graphical model editor and a data output interface.

Compared by the test criteria and the test models specified in Section 5.3.1, 4DiCeS has initial implementations for both a GUI and CLI. It was shown that the user interface is capable of applying other user interaction scenarios combined with other programming languages through the implementation of a CAVE front-end.

Then the question of the ease of use is a hard task for the programmer of exactly this application. Of course the intention was to have a clearly arranged user interface with as much assistance to the user as possible. In fact first user reactions to the system will show if further any improvement is necessary. For now it is reasonable to say that 4DiCeS clearly ranges among the tools that were indicated to be for experienced users. Admittedly MCell is the only tool that was defined to be hard to handle since it lacks a GUI.

4DiCeS was able to simulate both test cases correctly, even in 3D. However first user reactions are needed in order to make further statements. This holds true for two important reasons. Firstly the test models were chosen by the comparator. This is the complete opposite of a "double-blind study" known from pharmaceutical research. Secondly, many mistakes happen unobserved by application architects. Here only bug testing by people, other than the actual programmer, can unearth possible inconsistencies and problems. Nonetheless 4DiCeS ranged in times for both test cases of a 1D simulation within two seconds to five minutes and within a complete day for the 3D counterparts.

5.4 Related Work

Based on the comparison (see Section 5.3), the following findings can be summarized for the ten compared programs. The usability of an application is somewhat critical from the user's perspective. For example Gepasi and BIOCHAM are easy to use, whereas a program such as xCellerator requires further programming experience with Mathematica.

Additionally the lack of standards and interfaces between tools becomes apparent. For example the support for external file formats, such as SBML, will become more important in the future as the amount of available data increases - "tower of Babel" problem. An example is SmartCell, which uses SBML in combination with its own XML file format in such a way that an import of plain SBML files to the application is no longer possible. Also it was often not feasible to save a model by one application as an SBML file and reload it with another. This was the case for BIOCHAM and Copasi in both directions.

Then a sufficient documentation and transparency of implementation details are fundamental principles of scientific practice. Here e.g. SmartCell is lacking further information regarding the algorithms it applies. Simulation tools also differ in their ability to utilize external triggers.

Another challenging area is the computational parameter estimation for models. Presently, *à priori* information of the parameters may be unavailable and the parameter values are adjusted either with some semi-automatic methods or by manually varying them. The two tools Gepasi and Copasi, provide methods for computational parameter estimation already.

CHAPTER 6

Conclusions

A well structured and extensible platform for systems biology was developed by combining a set of functional components. The already implemented modules for the 4DiCeS application allow for simulation of preliminary applications of biochemical models.

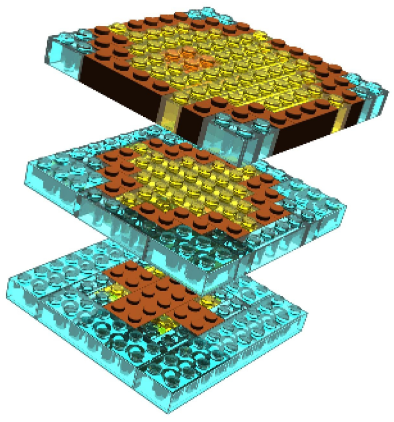


Figure 6.1: Lego drawing of a generalized nucleus. An illustration of the modeling of the nucleus by Lego blocks. The blocks can be compared to the 4DiCeS VEs that form a complex structure of cellular compartments. In contrast to Lego-blocks however, the VE can be adjusted in size. Thus, the granularity of the system can be adapted to the scientific requirements at hand.

Systems biology poses new challenges for visualization as the data types often contain 4D information. The representation of such data is currently unresolved. This work provides an approach to easily handle multi-dimensional data. The presented format could become a standard for 3D modeling environments. In respect thereof this approach works similar to the construction of an object from a box of Lego[1] blocks, in which Lego blocks are exchanged against VEs (see Figure 6.1). Also the general system's design can be seen such that each part of the 4DiCeS application is exchangeable but will not work alone.

[1]The Lego Group: http://www.lego.com/

6.1 Design Decisions

With 4DiCeS a modular approach was chosen to allow for the integration of diverse algorithms and file formats. With 4DiCeS models can be composed on different scales and complexity. It allows for the integration of numerous different algorithms to perform on either single particles, particle concentrations, or distinct VEs. More importantly it is also possible to mix different algorithmic approaches within one simulation space.

In this work the 4DiCeS framework was developed for modeling and simulation of signal transduction networks in 3D. The implementation of this project includes a modular design of the 4DiCeS kernel and various interfaces as described below:

- 4DiCeS-Kernel:
  The kernel of the 4DiCeS framework is the core of the project. It is responsible for securely interfacing with other modules. Furthermore it provides a platform for whole cell modeling as a result of its underlying data structure. Here 3D geometry, reactions, and diffusion are supported.
- User Interface:
  The user interface facilitates the adaption of tailored user-interaction solutions to the 4DiCeS system. Both GUI as well as CLI approaches are supported.
- Model Parsing Interface:
  The input parsing interfaces provide the capability to extend the 4DiCeS framework to several modeling input description languages. Thus there is the potential to include models from various data repositories.
- Reaction Interface:
  The reaction interface allows for the integration of reaction algorithms. These plug-ins provide the actual modular simulation functionality to the 4DiCeS system beside the diffusion interface.
- Diffusion Interface:
  The diffusion interface assists the combination of both known and new diffusion techniques. It is essential for simulations in 3D to have the ability of translocating particles.

Each interface has its own API, including other programming language extensions facilitating the implementation and integration of native code as well as other language plug-ins. The complete APIs offer a purpose-specific control over the 4DiCeS platform.

4DiCeS aims to serve as a universal simulation framework, which can integrate any set of different simulation algorithms, including differential-equation-based models, diffusion-reaction, stochastic algorithms (Gillespie, 1976, 1977, 2001; Gibson and Bruck, 2000) as well as many from CA (Wurthner et al., 2000) to GMA/S-Systems (Hernández-Bermejo et al., 2000).

The tool is designed to conduct efficient cellular signaling simulations on a cell model consisting of segments with their specialization on different simulation tasks.

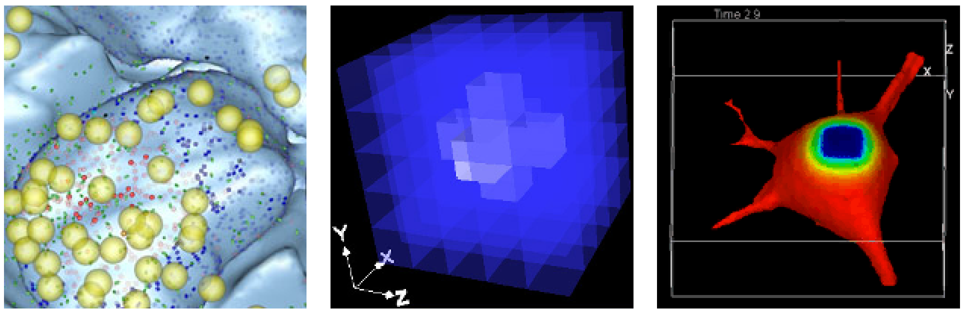
Successful attempts were also made to introduce the visualization environment to a CAVE (see Section 4.2.4) for virtual reality. The user interface from the 4DiCeS package was used for displaying the simulation results onto the four stereo-enabled screens of a CAVE.

Finally the usefulness and utility of this framework was shown for two sample diffusion and reaction applications in comparison to other related tools.

6.2 Related Attempts

In comparison to other existing systems, the 4DiCeS workbench has its benefits in its modularity and extendability. Other simulation tools are either limited by mathematical constraints or do not support 3D structures (see Section 5.3.2). The 4DiCeS framework has no limits in this respect. It furthermore provides the potential to include any type of diffusion and/or reaction algorithm. The integration of existing simulation tools is also possible via the API definitions as well.

Only a few of the existing cell simulators conceptualize the division of a cell into its representative compartments, organelles. Most of the simulation tools allowing for such an approach do not offer a "real" 3D analogous representation of exactly such segments. They do however make use of encircling parts of the model to define confined compartments. The only cell simulation tools that currently support 3D geometry (see Figure 6.2), besides the one developed here, are MCell (see Section 3.2.2.3), SmartCell (see Section 3.2.2.4), and the VirtualCell (see Section 3.2.1.4).



(a) MCell (b) SmartCell (c) VirtualCell

Figure 6.2: 3D visualizations of cell simulators. (a) shows the communication between neurons in the brain occurs at synapses with MCell (Coggan et al., 2005), (b) displays a very simple model of a cell with its nucleus created by SmartCell, and (c) presents the 3D surface visualization viewer of the VirtualCell project showing a neuronal cell filled with calcium indicator collected by confocal microscopy at University of Connecticut Health Center Farmington, USA, by Ion Moraru.

Here VirtualCell uses a purely deterministic approach for 2D as well as 3D modeling and simulation. Within MCell every compartment is modeled using 3D vector schemes (Casanova et al., 2004). This modeling approach comes with 3D representations that approximate reality but at the costs of being computationally high demanding. Visualizing every particle and its trajectory over time increases complexity and resource requirements dramatically. Lastly SmartCell utilizes a very similar approach as 4DiCeS does. The reaction space is a grid of VEs that can define separate compartments.

With this approach the computational time does not necessarily depend on the number of particles and their trajectories but on the number of defined VEs. If either the VEs are scaled to the size of particles or the internal computation of VEs is handled by calculating trajectories, then the simulation might run as slow as MCell.

The grid layout offers another benefit. It is possible to run different reaction and diffusion algorithms on the model. MCell is limited to the calculation of trajectories of single particles and VirtualCell can only handle differential equations. With SmartCell and 4DiCeS, arbitrary algorithms can be chosen and executed on the same model all the time. This holds true for SmartCell in theory only since it is lacking a well-defined programming interface for plugable algorithms. In contrast 4DiCeS offers these advantageous interfaces and even goes a step further in providing a framework where not only algorithms can be plugged to the system, but the modeling, visualization, and analysis engines may be also exchanged. Although 4DiCeS is delivered with a preliminary set of algorithms and tools, the programming languages' APIs feature a good extendability to future needs. Copasi (see Section 3.2.2.2) as the only competitor offers a similar extendability, but then lacks the 3D support.

	Accuracy	Concurrency	Designer	Exchange	Extendability	Features	GUI	Methodology	Performance	Scripting	Spaciality	Usability	Score
4DiCeS	●	●	◐	◐	●	◐	●	●	◐	●	●	◐	9.5
Copasi	●	○	◐	●	○	●	●	●	●	◐	◐	◐	8.0
E-Cell	●	○	◑	●	◐	●	●	○	◐	●	◐	●	8.0
BioNetS	●	○	◐	◐	◐	◐	●	●	●	○	○	●	7.0
VirtualCell	●	○	●	●	○	●	●	○	◐	○	●	◐	7.0
Gepasi	●	○	◐	◐	○	●	●	○	●	○	○	●	6.0
SmartCell	◐	○	◑	◐	○	○	●	●	◐	○	●	●	6.0
xCellerator	●	○	◐	◐	◐	◐	◐	●	◐	◐	○	◐	6.0
StochSim	●	○	◐	◐	○	○	●	◐	○	●	◐	◐	5.5
BIOCHAM	◐	○	○	◐	○	○	●	○	◐	●	○	◐	4.0
MCell	●	○	○	○	○	◐	○	◐	○	●	●	○	4.0

Table 6.1: 4DiCeS comparison. Equal to Table 3.4 filled (●) circles are generally superior to their half-full (◐) counterparts. Empty (○) circles indicate the need for improvement or a total absence. In methodology hybrid approaches are considered most sufficient. Designers are either defined as textual (◐), graphical (◑), both (●), or none (○) at all. Additionally, the score was calculated and printed to this table also. Each score is the sum of circle weights of the respective application. The weights are defined with empty = 0, half = 0.5, and full = 1 circles. All comparison criteria were assessed equally. The applications are then sorted by their individual score in a top-down order. If two or more application have an equal score then they are sorted in alphabetical order.

Table 6.1 provides an overview of how 4DiCeS is rated against the 12 comparison criteria from Section 3.4. If the same weighting scheme with zero (0), a half (0.5), and one (1) is applied then 4DiCeS exceeds all other related application with a comparison value of 9.5. The two best tools (Copasi and E-Cell) only reached a value of eight.

In summary it can be said that 4DiCeS might not outperform single other simulation applications in their specialized disciplines but rather provides a tool that incudes the ability of allowing all the different models run on one single simulation framework only. The modular design and the well-defined interfaces make it easy for software developers to extend 4DiCeS to their needs. The use of cross-platform libraries makes the application executable on almost all recent operation systems. Also the 4DiCeS system is not limited to client applications alone but may be used as a server-based system as well.

6.3 Challenges and Accomplishment

In recent years whole cell simulation has been declared as a new scientific goal (Tomita, 2001). The E-Cell project (see Section 3.2.1.2) with its deterministic approach is only one such example. Although still not within reach, many approaches exist that use deterministic as well as stochastic modeling methods for simulating at least biochemical processes within a cell.

Stochastic simulators consume more computational power than deterministic simulators and are thus generally used for smaller models. Endy et al. (2000) simulated a bacteriophage by stochastic models. The stochastic simulation algorithms by Gillespie (1976, 1977) were further optimized (Gibson, 2000) such that "whole cell simulation" goal comes within reach. To allow for an appreciation of how long it might take for current programs and computers to simulate such a challenge, the following sections consider the complexity of whole cell models and their calculation time.

6.3.1 Model Complexity

For the simulation of whole cell models, the complexity of such an approach should be first estimated. For example the base limit for the particle numbers involved can be obtained from the genome of an arbitrary organism. As for

Escherichia coli (an intestinal bacteria) the total number is currently counted near 4,500 genes (Karp et al., 2000, 2002). The cytoplasm of *E. coli* includes according to an estimation (Goodsell, 1997) around 22,500 proteins, 15,000 ribosomes, over 170,000 ribonucleic acid (RNA) molecules, 15,000,000 small organic molecules as nucleotides, amino acids, sugar, and others, as well as 25,000,000 ions. Then, about 70% of the bacteria's cell volume consist of water.

During one full cell cycle of *E. coli* (~ 50 *sec*), the number of biochemical reactions executed was estimated (Endy and Brent, 2001) to be somewhere between 10^{14} and 10^{16}. The complexity of other cell types is about a factor thousand smaller for the smallest known cell types (*mycoplasms*) and about a factor of thousand larger for typical plant and animal cells (Schwehm, 2001). Also the relation of compartments to the cellular volume has to be taken into account if one considers higher organized cells (see Table 6.2).

Designation	Volume [%]	Number
Cytosol	54	1
Mitochondria	22	1,700
Endoplasmic Reticulum	12	1
Nucleus	6	1
Golgi Apparatus	3	1
Peroxisomes	1	400
Lysosomes	1	300
Endosomes	1	200

Table 6.2: Membranous compartments. The table shows the relative volumes occupied by the major membranous compartments (organelles) in a liver cell (Hepatocyte). An organelle is a specialized subunit within a cell that has a specific function, and is separately enclosed within its own lipid membrane. The first column holds the names of the intracellular compartments. The second and third columns display the percentage of due to the total cell volume and the approximate number of compartments per cell. Adapted from Alberts et al. (2003).

6.3.2 Performance Evaluation

Analyzing the given numbers, the question arises how far scientific research is away from realizing whole cell simulation. A very optimistic estimation implies an appropriate whole cell model with 10^{14} biochemical reactions for a single cell cycle of *E. coli* (Endy and Brent, 2001). With a mean of $250,000$ processed reactions per second, (Schwehm, 2001) a stochastic whole cell simulation of only one *E. coli* cell cycle would take 4 ± 10^8 *sec* or nearly 13 *years* for a single computer processing unit.

Thus taking into account a factor of thousand for typical eukaryotic cells the simulation of whole cells is even more complex. Here either the modularization of the problem domain or much faster algorithms as well as heuristics are required. Combined heterogenous simulation methods, as done with 4DiCeS, is going to complement such approaches. Unfortunately future enhancements in computer power or the utilization of highly distributed computing will again feature a linear speedup. However considering that it once will be possible to simulate such a process within a few hours, the discrepancy to the actual 50 *sec* of a cell cycle is still far to great.

6.4 Outlook

Further extension as well as further development of this software could be directed towards incorporating additional specialized interfaces for (partly) automated pathway construction, a geometrical input tool for real cellular geometries from microscopy, model and simulation analysis, verification tools, and an own repository of models in addition to the collected kinetic data on molecules, reaction, diffusion, and pathways.

These interfaces should be implemented in a dynamically-linked manner as used throughout the existing components of the 4DiCeS framework. Module updates or supplements can be easily incorporated. The following paragraphs briefly illustrate some possible future extensions.

Distributed Computing: The simulation kernel could distribute its work onto heterogenous computer-networks. On a very low level, it is already possible to distribute the simulation process by using a screensaver-based approach of distributed computing – called Models@Home[2] (Krieger and Vriend, 2002).

[2]Models@Home: http://www.cmbi.kun.nl/models/

Models@Home is building a network of idle computers while allowing each of them to work on small pieces of a scientifically demanding project. The world's largest distributed computing project, which is the Search for ExtraTerrestrial Intelligence (SETI)[3], works on a very similar but much larger scale.

Model Construction Interface: A model construction interface could facilitate the construction of new or the extension of existing models for 4DiCeS users. Plug-ins could connect the simulation environment with common pathway and molecular databases (see Section 3.1) to automatically retrieve information needed for the current model being developed. The construction kernel allows for the manual design of such models as well.

Geometry Input Interfaces: To allow for close-to-reality 3D cell structures, files from different microscopy techniques such as CLSM imaging are planned to be importable. Additionally it would be possible to take time-dependant pictures of living cells in 4D. This can then be used for verification in modeling and simulation. As a preliminary approach, it could be designed to have a few different input modules for various CLSM picture stack formats from leading microscopy companies such as Leica[4], Nikon[5], Olympus[6], TILL Photonics[7], and Zeiss[8].

Analysis Interface: Analysis is normally done through third party products such as spreadsheet analysis programs and graph drawing utilities. A future implementation for an analysis interface would not provide a full replacement package for these powerful programs with a possibility to analyze the current output of a given simulation in real time. Further massive analysis of the data has to be done externally but could then use such an interface for automated data transportation.

Programming Language Bindings: Further programming language bindings could enlarge the usability and acceptance of 4DiCeS. Microsoft's .NET framework for example could easily open this tool to all .NET languages as for C♯, VisualBasic, and Managed C++. The framework is also ported by the third-party Mono project[9] and can therefore be used on various other operation systems as well.

[3]Seti@Home: http://setiathome.berkeley.edu/
[4]Leica Microsystems: http://www.leica-microsystems.com/
[5]Nikon: http://www.nikon.com/
[6]Olympus: http://www.olympus.com/
[7]TILL Photonics: http://www.till-photonics.com/ (now part of Agilent Technologies)
[8]Carl Zeiss: http://www.zeiss.de/
[9]Mono Project: http://www.mono-project.com/

4DiCeS Repository: The database storage could provide future 4DiCeS users with data from past simulations and models. This is meant to keep the model construction time small and the reusability high. The simulation process delivers different kinds of data, and the simulator can always reuse a previous output as a new input.

APPENDIX A

Algorithms in Detail

This chapter gives a brief overview of the algorithms that were applied as first module implementations to 4DiCeS. The next two sections explain stochastic approaches regarding reaction and diffusion processes and an ODE-solver used for reaction equations.

A.1 Reaction Algorithms

Continuous biochemical rate equations do not accurately predict cellular reactions since they rely on bulk reactions that require the interactions of millions of particles. They are typically modeled as a set of coupled ordinary differential equations. In contrast dynamic MC algorithms allow for a discrete and stochastic simulation of a system with few reactants, because each reaction is explicitly simulated.

It has to be noted that the reaction constants for dynamic MC algorithms are not the traditional macroscopic or deterministic rate constants. Rather they are mesoscopic rate constants, which are only related to the macroscopic rate constants. Macroscopic rate constants depend on concentrations of particles, while mesoscopic reaction constants are based on numbers of particles.

For 4DiCeS a numeric ODE-solver based on the Runge–Kutta 4^{th} order algorithm and three dynamic MC methods were implemented.

A.1.1 Runge–Kutta Method

This explicit method numerically integrates ODEs by using trial steps at the midpoint of an interval to cancel out lower-order error terms. Let an initial value problem be specified as follows

$$\frac{dy_i}{dt} = f_i(t, y_1, .., y_n) \tag{A.1}$$

with $t \in [a, b]$ and the initial values $y_i(a) = y_{a_i}$, $1 \leq i \leq n$. Then, the fourth-order formula for this problem is given by

$$y_{n+1} = y_n + h \times K \tag{A.2}$$

where

$$K = \frac{k_1 + 2k_2 + 2k_3 + k_4}{6} \tag{A.3}$$

with

$$\begin{aligned}
k_1 &= f(t_n, y_n)\,, && \text{(A.4)}\\
k_2 &= f(t_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1)\,, && \text{(A.5)}\\
k_3 &= f(t_n + \frac{1}{2}h, y_n + \frac{1}{2}k_2)\,, \text{ and} && \text{(A.6)}\\
k_4 &= f(t_n + h, y_n + k_3) && \text{(A.7)}
\end{aligned}$$

(Runge, 1895; Kutta, 1901). The next value y_{n+1} is determined by the present y_n, an estimated slope K, and the product of the size of the interval h. K is a weighted average of slopes. Here k_1 is the slope at the start of h. k_2 is the slope at the midpoint of the interval, using slope k_1 to define the value of y at the point $tn + \frac{h}{2}$ using Euler's method. Again k_3 is the slope at the midpoint however using the slope k_2 to determine the y-value. k_4 is the slope at the end of h, with its y-value provided using k_3. A greater weight is given to the two slopes at the midpoint.

The total accumulated error is on the order of h^4, while the error per step has order h^5. The Runge–Kutta Method is reasonably robust as well as simple and can be generally used for numerical solution of differential equations if combined with a flexible step-size method. For a given set of N differential equations and M intermediate steps this algorithm takes time proportional to $N \times M$ to solve the equations for t_{n+1}(Press et al., 1992).

A.1.2 Dynamic MC Algorithms

Dynamic MC algorithms can be used for modeling the dynamic behaviors of particles by comparing the rates of individual steps by random numbers. Such algorithms generate statistically correct trajectories of stochastic equations. The 'Direct Method' and the 'First-Reaction Method', both developed and published by Gillespie (1976, 1977), are algorithms to simulate chemical or biochemical reaction systems accurately as well as efficiently. These two algorithms are computationally demanding. Therefore many adaptations and modifications exist. These include the 'Next-Reaction Method' (Gibson and Bruck, 2000), τ-leaping, as well as hybrid techniques where abundant reactants are modeled with deterministic behavior.

Both Gillespie algorithms as well as the Next-Reaction Method by Gibson and Bruck were implemented in 4DiCeS. In the following a well-stirred system of N chemical species $S_1, .., S_N$ is undergoing M chemical reactions $R_1, .., R_M$. The current state of the system is specified by the vector $\mu = (\mu_1, .., \mu_N)$, where μ_i is the current number of S_i molecules in the system. Each reaction channel R_j is characterized by its propensity function $a_j(\mu)$ and its state-change vector $\upsilon_j = (\upsilon_{1j}, .., \upsilon_{Nj})$. Here $a_j(\mu)d\tau$ gives the probability that the system will experience an R_j reaction in the next infinitesimal time $d\tau$, and υ_{ij} is the change in the number of S_i molecules caused by one R_j reaction.

A.1.2.1 Direct Method

Gillespie's Direct Method calculates explicitly which reaction occurs next and when it appears. This is achieved by specifying the probability density $P(\mu, \tau)$ that the next reaction is μ that occurs at time τ. The algorithm is direct in the sense that it generates μ and τ instantaneously. It can be shown that

$$P(\mu, \tau)\, d\tau = a_\mu\, e^{\left(-\tau \sum_j a_j\right)}\, d\tau \tag{A.8}$$

Integrating $P(\mu, \tau)$, for all τ, $0 \leq \tau \leq \infty$ results in

$$Pr(\mu) = \int P(\mu, \tau) dt = \frac{a_\mu}{\sum_j a_j}. \tag{A.9}$$

Then, summing $P(\mu, \tau)$ over all μ gives

$$P(\mu, \tau)\, d\tau = \left(\sum_j a_j\right) e^{\left(-\tau \sum_j a_j\right)}\, d\tau. \tag{A.10}$$

These two distributions lead to Gillespie's direct algorithm:

1. Initialization: Set initial numbers of particles and $t_0 = 0$.
2. Calculate the propensity functions a_i for all i.
3. Choose μ according to the distribution in Eq.A.9.
4. Choose τ by an exponential with parameter $\sum_j a_j$ as in Eq.A.10.
5. Change the number of particles according to μ. Set $t_{n+1} = t_n + \tau$.
6. Go to Step 2 unless the simulation time exceeded or a reactant's number is zero.

This algorithm uses two random numbers per iteration, takes time proportional to M to update the a_is, and takes time proportional to M to calculate $\sum_j a_j$ as well as to generate a random number according to the distribution in Eq.A.9 (Gillespie, 1976).

A.1.2.2 First-Reaction Method

Gillespie's First-Reaction Method generates for each reaction a putative time τ_μ at which the reaction μ occurs. Thereafter, the reaction μ^* with the smallest ${\tau_\mu}^*$ is chosen and executed. Formally, the algorithm is as follows:

1. Initialization: Set initial numbers of particles and $t_0 = 0$.
2. Calculate the propensity functions a_i for all i.
3. For each i, generate a putative time τ by an exponential distribution with parameter a_i.
4. Let μ be the reaction whose putative time τ_μ is least.
5. Let τ be τ_μ.
6. Change the number of particles according to μ. Set $t_{n+1} = t_n + \tau$.
7. Go to Step 2 unless the simulation time exceeded or a reactant's number is zero.

For a given set of M reactions this algorithm uses I random numbers per iteration, takes time proportional to I to update the a_is, and takes time proportional to I to identify the smallest τ_μ (Gillespie, 1977).

A.1.2.3 Next-Reaction Method

As a modification of Gillespie's two methods the Next-Reaction Method is based on:

- Store τ_i, not just a_i.
- Recalculate a_i only if it changes.
- Re-use τ_is where appropriate.
- Switch from relative time between reactions to absolute time.
- Store a_is and τ_is so that updating will be very efficient.

Here a *dependency graph* is introduced to update the minimum number of a_is and an *indexed priority queue* stores all a_is and τ_i. These modifications lead to the algorithm:

1. Initialization:

   a Set initial numbers of particles and set $t_0 = 0$. Generate a dependency graph G.

   b Calculate the propensity functions a_i for all i.

   c For each i, generate a putative time τ according to an exponential distribution with parameter a_i.

   d Store the τ_i values in an indexed priority queue P.

2. Let μ be the reaction whose putative time τ_μ stored in P is least.
3. Let τ be τ_μ.
4. Change the number of particles according to μ. Set $t_{n+1} = t_n + \tau$.
5. For each edge (μ, α) in the dependency graph G,

   a Update a_α.

   b If $\alpha \neq \mu$, set $\tau_\alpha = (a_\alpha, old/a_\alpha, new)(\tau_\alpha - t) + t$.

   c If $\alpha = \mu$, generate a random number ρ according to an exponential distribution with parameter a_μ. Set $\tau_\alpha = \rho + t$.

   d Replace the old τ_α value in P with the new value.

6. Go to Step 2 unless the simulation time exceeded or a reactant's number is zero.

The total number of operations per iteration is at most $c_{2,3,4,5a,6} + c_{5b}(k-1) + c_{5c} + c_{5d}(k)(2\log M)$, where each c is a machine specific constant, k is the times of executing c_5a, and M is the number of reactions. This results in $O(\log M)$ (Gibson and Bruck, 2000).

A.2 Diffusion Algorithm

The mathematics of Brownian motion, as diffusion processes are also called, is often studied using simple models. The simplest is a random-walk where equal sized steps can be taken in any direction. For purposeful movement the start-to-end displacement increases at a rate proportional to time t leading to a Gaussian distribution of particles. A Gaussian random-walk is self-similar and such processes were originally discussed by the mathematician Paul Lévy (1948). Random-walks with self-similar dynamics and power-law scaling are therefore called discrete Lévy processes.

The approach used here for simulating the diffusion of particles either bound to membranes or freely distributing is a straight forward random-walk implementation.

A.2.1 Discrete Lévy Process

Let (X_n), $n \in \mathbb{N}_0$ be a stochastic process. Applies $|X_{n_1+h}-X_{n_1}| \sim |X_{n_2+h}-X_{n_2}|$ for all n_1, n_2, and $h \in \mathbb{N}_0$ then X is a process of stationary increase with the given representation

$$X_{n+1} = X_n + \sum_{i=1}^{t} Y_i. \tag{A.11}$$

With a mean quadratic shift of Ω using the diffusion coefficient $D = \frac{K_B T}{f}$ with the Bolzmann constant K_B, a coefficient of friction f and temperature T, it is possible to change the position $X_n = (x_{n1}, x_{n2}, x_{n3})$ of a particle over two random angles α and β in 3D. For a lateral diffusion in 2D with $X_n = (x_{n1}, x_{n2})$ only one angle α is required. For an ideally sphere shaped particle $f = 6\pi\eta R_S$ (Einstein-Stokes Equation) (Einstein, 1956), where η is the viscosity of water and R_S is the Stokes-radius, a 2D diffusion is described by

$$X_{n+1} = X_n + \Omega \begin{pmatrix} \cos(\alpha) \\ \sin(\alpha) \end{pmatrix} \tag{A.12}$$

where $\Omega = \sqrt{4Dn}$ and $-\pi \leq \alpha \leq \pi$. The 3D diffusion is then described by

$$X_{n+1} = X_n + \Omega \begin{pmatrix} \cos(\alpha)\cos(\beta) \\ \sin(\alpha)\cos(\beta) \\ \sin(\beta) \end{pmatrix} \tag{A.13}$$

where $\Omega = \sqrt{6Dn}$, $-\frac{\pi}{2} \leq \alpha \leq \frac{\pi}{2}$, and $-\pi \leq \beta \leq \pi$. The total number of operations n per iteration i is proportional to the numbers of particles p existing in the system.

APPENDIX B

Backus–Naur Form

The Backus–Naur Form (BNF) is a meta-syntax used to express context-free grammars in a formal way. It was named after two pioneers in computer science, John Warner Backus and Peter Naur (Knuth, 1964, 2004). BNF is widely used as a notation for the grammars of communication protocols, computer programming languages, and instruction sets. A BNF specification is a set of derivation rules written as

$$\langle S \rangle ::= \langle E \rangle \tag{B.1}$$

where $\langle S \rangle$ is a non-terminal symbol, and the $\langle E \rangle$ is an expression consisting of sequences of other symbols and/or a set of symbols. Sets are separated by a vertical bar ('|'), indicating a choice. Symbols that never appear on a left side are called terminals (Backus, 1959; Nauer, 1960). There are many extensions and variants of the BNF, where the Extended BNF (EBNF) is a very common derivative. Often such specifications include some of the additional syntax rules stated in Table B.1.

Rule	Description
Alternative	Choices in a production are concatenated by a |'.
Grouping	Simple parenthesis enclose groups of symbols.
Option	Options are enclosed in squared brackets.
Repetition	Repeats are characterized by a following '$*$'.
0+ Repeat	Zero or more time repeats are enclosed in curly brackets.
1+ Repeat	One or more time repeats are followed by a '+'.
Typeface	Terminals appear in bold and non-terminals in plain text.

Table B.1: BNF extensions. Today there exist many BNF grammar variants. This table describes the most commonly found extensions used by such extended derivatives (Wirth, 1977).

APPENDIX C

Unified Modeling Language

The Unified Modeling Language (UML) is a general-purpose modeling and specification language for object-oriented software engineering. It was designed to specify, construct, visualize and document software-intensive systems and includes a standardized graphical notation for the creation of common concepts like use-cases, components, classes, generalization, aggregation, and behaviors. UML is officially defined at the Object Management Group[1] by the UML metamodel.

It is noteworthy to differentiate between the UML model and the set of UML diagrams of a system. A diagram is only a partial graphical visualization of the model. Next to such diagrams the model may also contain textual documentation. In UML 2.0 there exist 13 types of diagrams (see Table C.1), which can be grouped into the three categories of behavior, interaction, and structure. UML does not restrict its element types to a certain diagram type. Actually every UML element may appear on almost all types of diagrams (Scott, 2004).

The following sections give details on the used UML diagram types in this work.

C.1 Use-Case Diagrams

An important part of the UML is the possibility of drawing use-case diagrams. Normally use-cases are utilized during the analysis phase of a project to identify the functional parts of a system. However use-cases can be used to demonstrate the actual systems functionality. Since a graphical UML use-case has only a very low information content it is often accompanied by its textual representation.

Generally UML use-case diagrams separate a system into the respective use-cases and actors. Actors represent the roles of users (i.e. human beings, other hardware, or software) of the system. The actors are external to the system described by the use-cases. They trigger the system and may receive output from it. To emphasize the externality of actors use-cases are framed into a box.

[1]Object Management Group: http://www.omg.org/

Category	Diagram	Description
Behavior	Activity	Depicts the operational work-flows of a system and their interaction with each other.
	Use-Case*	Gives a graphical notation to represent various use-cases of a system's behavior.
	State Machine	Describes a system as a finite state machine graph.
Interaction	Communication	Models interactions between objects (vertically) by sequenced messages (horizontally).
	Overview	Combines other interaction or behavioral diagrams to a global overview.
	Sequence*	Displays different but simultaneous processes and their exchanged messages.
	Timing	Shows a coordinate system with time on the abscissa and object states on the ordinate.
Structure	Class*	Displays a static view of classes of a system and their relationships with each other.
	Component*	Shows physical components (i.e. files) of a system and dependencies to each other.
	Composite	Presents the internal structure of one class and its collaboration dependencies.
	Deployment	Serves to model the utilized hardware by the system and its associations to the software.
	Object	Gives a temporal snapshot of a complete or partial view of the system's object structure.
	Package	Represents how a system is grouped into logical units and dependencies to each other.

Table C.1: UML diagram types. This table describes the 13 types of diagrams currently defined by the UML in brief. For a better structuring the diagrams are often categorized into the three types of behavioral, statical, and interaction deportment (Larman, 2001; Scott, 2004). The four diagram types used in this work are marked with an asterisk ('*').

In UML use-cases are represented by ovals and the actors are drawn as simple stick figures. The actors are connected with the use-cases by plain lines. Then the use-cases may have dashed relationship arrows of two kind. The first is a *uses* assignment, which can be compared to a function call or subroutine in programming languages. The other link-up is the *extends* assignment, which manipulates the original use-case to fit other purposes. Both relationships are placed on top of their arrows surrounded by doubled angle brackets (i.e. ≪uses≫ and ≪extends≫) (Larman, 2001; Scott, 2004).

C.2 Class Diagrams

UML class diagrams have the purpose to depict the classes within a model. In an object-oriented application classes have attributes, methods, and relationships to other classes. The fundamental element of a class diagram is an icon that depicts two of these components into compartments of one class-rectangle. The topmost compartment contains the class name, the middle comprises all attributes and the bottom includes a list of methods. Often the bottom two compartments are omitted or do only present attributes and methods that are meaningful for the current diagram. If a class is abstract its name will be indicated in an italic typeface.

The relationship to other classes is represented by different kinds of lines. UML distinguishes aggregations, associations, compositions, dependencies, inheritance, and interfaces. The inheritance relationship in UML is depicted by a peculiar triangular arrowhead with a plain line. The arrowhead points to its base class connecting it to its derived class(es). The composition relationship depicts a strong form of an aggregation and is also called a by-value-relation. It is represented by a black diamond at the class that composes another class. The other class is pointed to by an ordinary arrowhead, if the relationship is only navigable in one direction. The weak form of an aggregation is denoted by an open diamond. An other form of a containment shows no diamond at all. This plain arrow is called an association. Both the aggregation and the association are often referred to by-reference-relations. Sometimes the relationship between two classes is very weak. In this case they are only implemented as method arguments. Such relationships are represented by dashed arrows and are called dependencies. An UML way to display interfaces (abstract classes with no attributes at all) is the so-called “lollypop” notation. The interface has a line with an empty circle at its end. This circle is the point to which other classes draw their dependency arrows to (Larman, 2001; Scott, 2004).

C.3 Sequence Diagrams

A UML sequence diagram emphasizes the sequence of messages between objects. Rectangles represent objects. The names of the objects are underlined to distinguish them from classes. Additionally the object name is separated from the class name by a colon. If objects do not have a particular object name then the colon precedes the class name without an object name in front of it.

Below of each object is a dashed line known as the "lifeline". Such lines define time axes of the diagram. By convention time proceeds in a downward direction. A variation could though be to tip a sequence diagram on its side so that time proceeds to the right. The lifelines depict how long the objects that they are connected to are in existence. If lifelines extend from the very top of the diagram to the very bottom then this implies that the objects portrayed in the diagram are in existence before the start of time of the diagram and remain in existence beyond the end of the diagram.

The arrows between lifelines represent messages being sent between objects. Sequence numbers are permitted but not necessary. White arrow rectangles indicate that the arrow terminates on a called activation. They show the duration of the execution of a method in response to a message. The methods implicitly return to their caller at the end of the activation. Such returns can be displayed by an unlabeled arrow that extends from the bottom of the activation back to the lifeline of the calling object. In the case of asynchronous messages the end of an activation does not imply a return.

Narrow rectangles on top of lifelines that enclose groups of messages define an iteration. The looping condition for such an iteration is then shown at the bottom of the rectangle.

The creation of objects is denoted by a message arrow that terminates on an object's rectangle. Deletion is likewise denoted by a message arrow that terminates on a capital 'X' at the end of the object's lifeline.

Incomplete (half) arrowheads denote asynchronous messages. An asynchronous message is a message that the receiving object executed the method in a separated thread. That threat could be in existence prior to the sending of the asynchronous message and just waiting for something to do. This gives sequence diagrams the power to display concurrent multi-threaded interactions. Wherever concurrency is present, race conditions are possible. Race conditions occur when a single thread or object receives messages from two competing sources. If not handled properly, the participating objects can get quite confused.

C.4 Component Diagrams

Components are qualified in UML as reusable program code that implements well-defined duties. Therefore components are very similar to UML packages. They combine thematically related elements to a set. As a general rule, this topical relationship is also reflected in similar characteristics of elements, as for their attributes, interfaces, and methods. Whereas packages represent a view of the contents, components highlight the software-technical aspects of elements' commonality. Particularly the structure of the program is of major importance.

Components establish a unity with one or more interfaces to the outside. Component diagrams visualize this graphically. They describe dependencies between software components and their interfaces. Here a component is displayed as a box. Additionally, the box can be marked with the phrase 'component' surrounded by doubled angle brackets (i.e. ≪component≫) or by a component symbol. Then interfaces are connected through the "lollypop" notation. Component diagrams are also used to describe static dependencies between programs, as for example compiler-dependencies. Such dependencies between programs are illustrated by dashed arrows. The dependent components points at the independent component.

Components can contain additional elements, as for objects, other components, or nodes. For comprehension as regards content, components are often substantiated by other diagrams, as for class diagrams and use-case diagrams. For technical aspects of the implementation, deployment diagrams are utilized.

List of Abbreviations

1D	one-dimensional
2D	two-dimensional
3D	three-dimensional
4D	four-dimensional
4DiCeS	4D Cell Simulator
AFM	Atomic Force Microscopy
AnatML	Anatomical Markup Language
API	Application Programming Interface
ASL	Academic Software License
ATP	adenosine–3',5'–triphosphate
ATPase	ATP hydrolase
BD	Brownian Dynamics
BIOCHAM	BIOCHemical Abstract Machine
BIOML	BIOpolymer Markup Language
BioNetS	Bio-chemical Network (stochastic) Simulator
BioPAX	Biological PAthways eXchange
Bio-SPICE	Bio–Simulation Program for Intra- and Inter-Cell Evaluation
BNF	Backus–Naur Form
BRep	Boundary Representation
Br^-	bromate(V) ion
BSD	Berkeley Software Distribution
BZR	Belousov–Zhabotinsky Reaction
CA	Cellular Automata
Ca^{2+}	calcium(II) ion
$[Ca^{2+}]_i$	cytoplasmic Ca^{2+} concentration
CAVE	recursive acronym for CAVE Automatic Virtual Environment
Ce^{3+}	cerium(III) ion
Ce^{4+}	cerium(IV) ion
CellML	Cell Markup Language
CLI	Command-Line Interface
CLSM	Confocal Laser Scanning Microcopy
CML	Chemical Markup Language
CN-CL	Copasi Non-Commercial License
COAST	COntrollable Approximative STochastic (reaction-algorithm)
Copasi	Complex Pathway Simulator
CSG	Constructive Solid Geometry

CTL	Computational Tree Logic
DAE	Differential-Algebraic Equation
DAG	sn–1,2–diacylglycerol
DLL	Dynamic Link Library
DNA	desoxyribonucleic acid
DSM	Dynamic Signaling Maps
DSML	Dynamic Signaling Maps Language
DTD	Document Type Definition
DynLoad	DYNamic link cross-platform LOADer
EBNF	Extended BNF
ELF	Executable and Linking Format (UNIX)
EML	E-Cell Model (description) Language
ER	endoplasmic reticulum
FieldML	Field Markup Language
FKN	Field–Körös–Noyes model system
FMD	4DiCeS Model Description
GENESIS	GEneral NEural SImulation System
Gepasi	GEneral PAthway SImulator
GFRD	Green's Function Reaction Dynamics
GPL	GNU General Public License
GUI	Graphical User Interface
GUID	Globally Unique IDentifier
HTML	HyperText Markup Language
IDE	Integrated Development Environment
IP_3	inositol–1,4,5–trisphosphate
IP_3R	IP_3 receptor
JVM	Java Virtual Machine
K^+	potassium
LGPL	GNU Lesser General Public License
LTL	Linear Temporal Logic
MAGE-ML	MicroArray Gene Expression Markup Language
MathML	Mathematical Markup Language
MC	Monte Carlo (method)
MCell	MC Simulator of Cellular Microphysiology
MD	Molecular Dynamics
MDL	MCell's Model Description Language
MesoRD	Mesoscopic Reaction Diffusion (simulator)
ModelML	Modeling Markup Language
MPFM	Multi-Photon Fluorescence Microscopy

Na^+	sodium
NRBSC-SL	National Resource for Biomedical SuperComputing Software License
ODE	Ordinary Differential Equation
PDE	Partial Differential Equation
PEML	Proteomics Experiment Markup Language
PIP_2	phosphatidylinositol-4,5-bisphosphate
PM	plasma membrane
PMCA	PM Ca^{2+}-ATPase
PNML	Petri-Net Markup Language
PRL	Pathway Resource List
ProML	Protein Markup Language
PSI-MI	Proteomics Standards Initiative's Molecular Interaction
PySCeS	Python Simulator for Cellular Systems
RDF	Resource Description Framework
RGB	red, green, and blue color scheme
RNA	ribonucleic acid
SBML	Systems Biology Markup Language
SBW	Systems Biology Workbench
SDE	Stochastic Differential Equation
SEM	Scanning Electron Microscopy
SERCA	SR/ER Ca^{2+}-transport ATPase
SETI	Search for ExtraTerrestrial Intelligence
SO	Shared Object (UNIX)
SR	sarcoplasmic reticulum
STODE	STochastic simulation of ODEs
StochSim	Stochastic Simulator
TDI	Tabbed Document Interface
TIFF	Tagged Image File Format
UML	Unified Modeling Language
VE	Volume Element
WWW	World Wide Web
XmdS	eXtensible multi-dimensional Simulator
XML	eXtensible Markup Language
XPP	X-windows Phase-Plane

List of Figures

List of Tables

Bibliography

Adalsteinsson, D., McMillen, D., and Elston, T. C. (2004). Biochemical network stochastic simulator (BioNetS): Software for stochastic modeling of biochemical networks. *BMC Bioinformatics*, 5(1):24.

Alberts, B., Bray, D., Johnson, A., Lewis, J., Raff, M., Roberts, K., and Walter, P. (2003). *Essential cell biology: An introduction to the molecular biology of the cell.* Garland Publishing, 2nd edition.

Alvarez-Vasquez, F., Sims, K. J., Cowart, L. A., Okamoto, Y., Voit, E. O., and Hannun, Y. A. (2005). Simulation and validation of modelled sphingolipid metabolism in *Saccharomyces cerevisiae*. *Nature*, 433:425–430.

Ander, M., Beltrao, P., Ventura, B. D., Ferkinghoff-Borg, J., Foglierini, M., Kaplan, A., Lemerle, C., Tomás-Oliveira, I., and Serrano, L. (2004). SmartCell, a framework to simulate cellular processes that combines stochastic approximation with diffusion and localisation: Analysis of simple networks. *Syst. Biol.*, 1(1):129–138.

Andrews, S. S. and Bray, D. (2004). Stochastic simulation of chemical reactions with spatial resolution and single molecule detail. *Phys. Biol.*, 1(3–4):137–151.

Arkin, A. P. (2001). Synthetic cell biology. *Curr. Opin. Biotechnol.*, 12(6):638–644.

Ausbrooks, R., Buswell, S., Carlisle, D., Dalmas, S., Devitt, S., Diaz, A., Froumentin, M., Hunter, R., Ion, P., Kohlhase, M., Miner, R., Poppelier, N., Smith, B., Soiffer, N., Sutor, R., and Watt, S. (2003). Mathematical markup language (mathml) version 2.0 (second edition). Recommendation http://www.w3.org/TR/MathML2/, W3C.

Backus, J. W. (1959). The syntax and semantics of the proposed international algebraic language of the Zürich ACM-GAMM conference. In *Proc. Int. Conf. Inform. Proc. (ICIP)*, pages 125–132, Paris.

Bader, G. D., Cary, M. P., and Sander, C. (2006). Pathguide: a pathway resource list. *Nucl. Acids Res.*, 34(Database issue):D504–D506.

Bartol Jr., T. M., Stiles., J. R., Salpeter, M. M., Salpeter, E. E., and Sejnowski, T. J. (1996). MCELL: Generalized Monte Carlo computer simulation of synaptic transmission and chemical signaling. In *Soc. Neuro. Abs.*, volume 22, page 1742.

Baxevanis, A. D. (2000). The molecular biology database collection: An online compilation of relevant database resources. *Nucl. Acids Res.*, 28(1):1–7.

Baxevanis, A. D. (2001). The molecular biology database collection: An updated compilation of biological database resources. *Nucl. Acids Res.*, 29(1):1–10.

Baxevanis, A. D. (2002). The molecular biology database collection: 2002 update. *Nucl. Acids Res.*, 30(1):1–12.

Baxevanis, A. D. (2003). The molecular biology database collection: 2003 update. *Nucl. Acids Res.*, 31(1):1–12.

Baynes, B. M. and Trout, B. L. (2004). Rational design of solution additives for the prevention of protein aggregation. *Biophys J*, 87:1631–1639.

Belousov, B. P. (1958). A periodic reaction and its mechanism (org. Russian). *Sborn. Referat. Radiat. Med. (Moskow)*, page 145.

Berridge, M. (2004). Conformational coupling: A physiological calcium entry mechanism. *Sci. STKE*, 2004(243):PE33.

Berridge, M. J. (1991). Cytoplasmic calcium oscillations: A two pool model. *Cell Calcium*, 12(2–3):63–72.

Berridge, M. J. (1993). Inositol trisphosphate and calcium signalling. *Nature*, 361(6410):315–325.

Berridge, M. J. (1998). Neuronal calcium signaling. *Neuron*, 21(1):13–26.

Berridge, M. J. (2005). Unlocking the secrets of cell signaling. *Annu. Rev. Physiol.*, 67:1–21.

Berry, H. (2002). Monte Carlo simulations of enzyme reactions in two dimensions: Fractal kinetics and spatial segregation. *Biophys. J.*, 83(4):1891–1901.

Bhalla, U. S. (2002). Use of Kinetikit and GENESIS for modeling signaling pathways. *Methods Enzymol.*, 345:3–23.

Bhalla, U. S. and Iyengar, R. (1999). Emergent properties of networks of biological signaling pathways. *Science*, 283(5400):381–387.

Bolouri, H. and Davidson, E. H. (2002). Modeling transcriptional regulatory networks. *Bioessays*, 24(12):1118–1129.

Bower, J. M. and Bolouri, H. (2004). *Computational Modeling of Genetic and Biochemical Networks*. Computational Molecular Biology Series. Bradford Book, 1st (paper back) edition.

Broderick, G., Ru'aini, M., Chan, E., and Ellison, M. J. (2005). A life-like virtual cell membrane using discrete automata. *In Silico Biol.*, 5(2):163–178.

Brown, R. (1828). A brief account of microscopical observations made in the months on june, july, and august, 1827, on the particles contained in the pollen of plants; and on the general existence of active molecules in organic and inorganic bodies. *Phil. Mag.*, 4:161–173.

Burrage, K., Tian, T., and Burrage, P. (2004). A multi-scaled approach for simulating chemical reaction systems. *Prog. Biophys. Mol. Biol.*, 85(2–3):217–234.

Calzone, L., Fages, F., and Soliman, S. (2006). BIOCHAM: an environment for modeling biological systems and formalizing experimental knowledge. *Bioinformatics*, 22(14):1805–1807.

Cao, Y., Li, H., and Petzold, L. (2004a). Efficient formulation of the stochastic simulation algorithm for chemically reacting systems. *J. Chem. Phys.*, 121(9):4059–4067.

Cao, Y., Petzold, L., Rathinam, M., and Gillespie, D. (2004b). The numerical stability of leaping methods for stochastic simulation of chemically reacting systems. *J. Chem. Phys.*, 121(24):12169–12178.

Carafoli, E. (2002). Calcium signaling: A tale for all seasons. *Proc. Natl. Acad. Sci. USA*, 99(3):1115–1122.

Carafoli, E. (2005). Calcium – a universal carrier of biological signals. *FEBS J.*, 272(5):1073–1089.

Cary, M. P., Bader, G. D., and Sander, C. (2005). Pathway information for systems biology. *FEBS Lett.*, 579(8):1815–1820.

Casanova, H., Berman, F., Bartol, T., Gokcay, E., Sejnowski, T., Birnbaum, A., Dongarra, J., Miller, M., Ellisman, M., Faerman, M., Obertelli, G., Wolski, R., Pomerantz, S., and Stiles, J. (2004). The virtual instrument: Support for grid-enabled MCell simulations. *Intl. J. of High Perf. Comp. App.*, 18:3–17.

Chance, B. (2004). The stopped-flow method and chemical intermediates in enzyme reactions - A personal essay. *Photosynth. Res.*, 80(1–3):387–400.

Chatterjee, A., Mayawala, K., Edwards, J. S., and Vlachos, D. G. (2005). Time accelerated monte carlo simulations of biological networks using the binomial τ-leap method. *Bioinformatics*, 21(9):2136–2137.

Chay, T. R. and Keizer, J. (1983). Minimal model for membrane oscillations in the pancreatic beta-cell. *Biophys J.*, 42(2):181–190.

Chickarmane, V., Paladugu, S. R., Bergmann, F., and Sauro, H. M. (2005). Bifurcation discovery tool. *Bioinformatics*, 21(18):3688–3690.

Chong, L. and Ray, L. B. (2002). Whole-istic biology. *Science*, 295(5560):1661.

Churchill, W. (1949). Speaking at Britain's National Book Exhibition about his World War II memoirs.

Clapham, D. E. (1995). Calcium signaling. *Cell*, 80(2):259–268.

Coggan, J. S., Bartol, T. M., Esquenazi, E., Stiles, J. R., Lamont, S., Martone, M. E., Berg, D. K., Ellisman, M. H., and Sejnowski, T. J. (2005). Evidence for ectopic neurotransmission at a neuronal synapse. *Science*, 309(5733):446–451.

Collecutt, G. and Drummond, P. D. (2001). Xmds: eXtensible multi-dimensional Simulator. *Comp. Phys. Comm.*, 142(1–3):219–223.

Crampin, E. J., Halstead, M., Hunter, P. J., Nielsen, P., Noble, D., Smith, N., and Tawhai, M. (2004). Computational physiology and the physiome project. *Exp. Physiol.*, 89(1):1–26.

DeFanti, T. A., Sandin, D. J., and Cruz-Neira, C. (1993). A 'room' with a 'view'. *IEEE Spectrum*, 30(10):30–33.

DeSchutter, E. and Cannon, R. C. (2000). *Computational Neuroscience: Realistic Modeling for Experimentalists*, volume 5 of *Frontiers in Neuroscience.* CRC Press.

Einstein, A. (1905). Über die von der molekularkinetischen Theorie der Wärme geforderte Bewegung von in ruhenden Flüssigkeiten suspendierten Teilchen. *Ann. d. Phys.*, 17(4):549–560.

Einstein, A. (1956). *Investigations on the Theory of the Brownian Movement.* Dover Publications, Inc., 1st edition.

Elcock, A. H. (2003). Atomic-level observation of macromolecular crowding effects: Escape of a protein from the GroEL case. *Proc. Natl. Acad. Sci. USA*, 100:2340–2344.

Encarnação, J. L., Straßer, W., and Klein, R. (1997a). *Graphische Datenverarbeitung 1: Gerätetechnik, Programmierung und Anwendung graphischer Systeme.* Oldenbourg, 4th edition.

Encarnação, J. L., Straßer, W., and Klein, R. (1997b). *Graphische Datenverarbeitung 2: Modellierung komplexer Objekte und photorealistischer Bilderzeugung.* Oldenbourg, 4th edition.

Endy, D. and Brent, R. (2001). Modelling cellular behaviour. *Nature*, 409(6818):391–395.

Endy, D., You, L., Yin, J., and Molineux, I. J. (2000). Computation, prediction, and experimental tests of fitness for bacteriophage T7 mutants with permuted genomes. *Proc. Natl. Acad. Sci. USA*, 97(10):5375–5380.

Eungdamrong, N. J. and Iyengar, R. (2004). Modeling cell signaling networks. *Biol. Cell.*, 96(5):355–362.

Fall, C. P. and Keizer, J. E. (2002). *Computational Cell Biology*, chapter Dynamic Phenomena in Cells, pages 3–20. In Fall et al. (2002), 1st edition.

Fall, C. P., Marland, E. S., Wagner, J. M., and Tyson, J. J. (2002). *Computational Cell Biology.* Springer-Verlag New York Inc., 1st edition.

Fenyo, D. (1999). The biopolymer markup language. *Bioinformatics*, 15(4):339–340.

Fick, A. (1855). Über Diffusion. *Poggendorf's Ann. d. Phys.*, 94:59–86.

Field, R. J., Körös, E., and Noyes, R. M. (1972). Oscillations in chemical systems. II. Thourough analysis of temporal oscillations in the bromate-cerium-malonic acid system. *J. Am. Chem. Soc.*, 94(25):8649–8664.

Field, R. J. and Noyes, R. M. (1974a). Oscillations in chemical systems. IV. Limit cycle behavior in a model of a real chemical reaction. *J. Chem. Phys.*, 60(5):1877–1884.

Field, R. J. and Noyes, R. M. (1974b). Oscillations in chemical systems. V. Quantitative explanation of band migration in the belousov-zhabotinskii reaction. *J. Am. Chem. Soc.*, 96(7):2001–2006.

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

Friedel, M. and Shea, J.-E. (2004). Self-assembly of peptides into a β-barrel motif. *J. Chem. Phys.*, 120(12):5809–5823.

Fukuda, K. and Takagi, T. (2001). Knowledge representation of signal transduction pathways. *Bioinformatics*, 17(9):829–837.

Fukuda, K., Yamagata, Y., and Takagi, T. (2004). FREX: A query interface for biological processes with a hierarchical and recursive structures. *In Silico Biol.*, 4(1):63–79.

Galperin, M. Y. (2004). The molecular biology database collection: 2004 update. *Nucl. Acids Res.*, 32(Database issue):D3–D22.

Galperin, M. Y. (2005). The molecular biology database collection: 2005 update. *Nucl. Acids Res.*, 33(Database issue):D5–D24.

Galperin, M. Y. (2006). The molecular biology database collection: 2006 update. *Nucl. Acids Res.*, 34(Database issue):D3–D5.

Galperin, M. Y. (2007). The molecular biology database collection: 2007 update. *Nucl. Acids Res.*, 35(Database issue):D3–D4.

Galperin, M. Y. (2008). The molecular biology database collection: 2008 update. *Nucl. Acids Res.*, 36(Database issue):D2–D4.

Garvey, T. D., Lincoln, P., Pedersen, C. J., Martin, D., and Johnson, M. (2003). BioSPICE: Access to the most current computational tools for biologists. *OMICS*, 7(4):411–420.

Ghosh, R. and Tomlin, C. (2004). Symbolic reachable set computation of piecewise affine hybrid automata and its application to biological modelling: Delta-Notch protein signalling. *Syst. Biol.*, 1(1):170–183.

Gibson, M. A. (2000). *Computational Methods for Stochastic Biological Systems.* PhD thesis, California Inst. Technology, Pasadena, California.

Gibson, M. A. and Bruck, J. (2000). Efficient exact stochastic simulation of chemical systems with many species and many channels. *J. Phy. Chem. A*, 104(9):1876–1889.

Gillespie, D. T. (1976). A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *J. Comp. Phys.*, 22:403–434.

Gillespie, D. T. (1977). Exact stochastic simulation of coupled chemical reactions. *J. Comp. Phys.*, 81(25):2340–2361.

Gillespie, D. T. (1996). Exact numerical simulation of the Ornstein-Uhlenbeck process and its integral. *Phys. Rev. E*, 54(2):2084–2091.

Gillespie, D. T. (2001). Approximate accelerated stochastic simulation of chemically reacting systems. *J. Chem. Phys.*, 115(4):1716–1733.

Gillespie, D. T. (2006). Stochastic simulation of chemical kinetics. *Annu. Rev. Phys. Chem.*, page [Epub ahead of print].

Glansdorff, P. and Prigogine, I. (1971). *Thermodynamic Theory of Structure, Stability and Fluctuations.* Wiley, 1st edition.

Goldbeter, A., Dupont, G., and Berridge, M. J. (1990). Minimal model for signal induced ca^{2+} oscillations and for their frequency encoding through protein phosphorylation. *Proc. Natl. Acad. Sci. USA*, 87(4):1461–1465.

Goodsell, D. S. (1997). *The Machinery of Life.* Copernicus, reprint edition.

Gough, N. R. (2002). Science's signal transduction knowledge environment: The connections maps database. *Ann. N. Y. Acad. Sci.*, 971:585–587.

Gough, N. R. and Ray, L. B. (2002). Mapping cellular signaling. *Sci. STKE*, 2002(135):EG8.

Hanisch, D., Zimmer, R., and Lengauer, T. (2002). ProML–The protein markup language for specification of protein sequences, structures and families. *In Silico Biol.*, 2(3):313–324.

Hartwell, L. H., Hopfield, J. J., Leibler, S., and Murray, A. W. (1999). From molecular to modular cell biology. *Nature*, 402(6761 Suppl):C47–C52.

Haseltine, E. L. and Rawlings, J. B. (2002). Approximate simulation of coupled fast and slow reactions for stochastic chemical kinetics. *J. Chem. Phys.*, 117(15):6959–6969.

Hattne, J., Fange, D., and Elf, J. (2005). Stochastic reaction-diffusion simulation with MesoRD. *Bioinformatics*, 21(12):2923–2924.

Hermjakob, H., Montecchi-Palazzi, L., Bader, G., Wojcik, J., Salwinski, L., Ceol, A., Moore, S., Orchard, S., Sarkans, U., von Mering, C., Roechert, B., Poux, S., Jung, E., Mersch, H., Kersey, P., Lappe, M., Li, Y., Zeng, R., Rana, D., Nikolski, M., Husi, H., Brun, C., Shanker, K., Grant, S. G. N., Sander, C., Bork, P., Zhu, W., Pandey, A., Brazma, A., Jacq, B., Vidal, M., Sherman, D., Legrain, P., Cesareni, G., Xenarios, I., Eisenberg, D., Steipe, B., Hogue, C., and Apweiler, R. (2004). The HUPO PSI's molecular interaction format–A community standard for the representation of protein interaction data. *Nat. Biotechnol.*, 22(2):177–183.

Hernández-Bermejo, B., Fairén, V., and Sorribas, A. (2000). Power-law modeling based on least-squares criteria: Consequences for system analysis and simulation. *Math. Biosci.*, 167(2):87–107.

Hofestädt, R. and Meineke, F. (1995). Interactive modelling and simulation of biochemical networks. *Comput. Biol. Med.*, 25(3):321–334.

Hofmeyr, J. H. (1986). Steady-state modelling of metabolic pathways: A guide for the prospective simulator. *Comput. Appl. Biosci.*, 2(1):5–11.

Holkner, A. (2002). Dynamically loaded C++ plugins for all platforms. Internet. http://yallara.cs.rmit.edu.au/ aholkner/dynload/index.html.

Hucka, M., Finney, A., Bornstein, B. J., Keating, S. M., Shapiro, B. E., Matthews, J., Kovitz, B. L., Schilstra, M. J., Funahashi, A., Doyle, J. C., and Kitano, H. (2004). Evolving a lingua franca and associated software infrastructure for computational systems biology: The Systems Biology Markup Language (SBML) project. *Syst. Biol.*, 1(1):41–53.

Hucka, M., Finney, A., Sauro, H., and Bolouri, H. (2002). The ERATO systems biology workbench: Enabling interaction and exchange between software tools for computational biology. In et al., R. A., editor, *Proc. Pac. Biocomp. Symp.*, volume PBS 2002.

Hucka, M., Finney, A., Sauro, H. M., Bolouri, H., Doyle, J. C., Kitano, H., Arkin, A. P., Bornstein, B. J., Bray, D., Cornish-Bowden, A., Cuellar, A. A., Dronov, S., Gilles, E. D., Ginkel, M., Gor, V., Goryanin, I. I., Hedley, W. J., Hodgman, T. C., Hofmeyr, J.-H., Hunter, P. J., Juty, N. S., Kasberger, J. L., Kremling, A., Kummer, U., Novére, N. L., Loew, L. M., Lucio, D., Mendes, P., Minch, E., Mjolsness, E. D., Nakayama, Y., Nelson, M. R., Nielsen, P. F., Sakurada, T., Schaff, J. C., Shapiro, B. E., Shimizu, T. S., Spence, H. D., Stelling, J., Takahashi, K., Tomita, M., Wagner, J., and Wang, J. (2003). The systems biology markup language (SBML): A medium for representation and exchange of biochemical network models. *Bioinformatics*, 19(4):524–531.

Igarashi, T. and Kaminuma, T. (1997). Development of a cell signaling networks database. In *Pac. Symp. Biocomput.*, volume PSB 1997, pages 187–197.

Ipsen, M., Hynne, F., and Sørensen, P. G. (1997). Amplitude equations and chemical reaction-diffusion systems. *Int. J. Bifurcat. Chaos*, 7(7):1539–1554.

Kang, Q., Zhang, D., Chen, S., and He, X. (2002). Lattice boltzmann simulation of chemical dissolution in porous media. *Phys. Rev. E Stat. Nonlin. Soft. Matter Phys.*, 65(3 Pt 2B):036318.

Karp, P. D., Riley, M., Saier, M., Paulsen, I. T., Collado-Vides, J., Paley, S. M., Pellegrini-Toole, A., Bonavides, C., and Gama-Castro, S. (2002). The EcoCyc database. *Nucl. Acids Res.*, 30(1):56–58.

Karp, P. D., Riley, M., Saier, M., Paulsen, I. T., Paley, S. M., and Pellegrini-Toole, A. (2000). The EcoCyc and MetaCyc databases. *Nucl. Acids Res.*, 28(1):56–59.

Kauffman, S. A. (1969). Metabolic stability and epigenesis in randomly constructed genetic nets. *J. Theor. Biol.*, 22(3):437–367.

Keener, J. and Sneyd, J. (2001). *Mathematical Physiology.* Springer.

Kell, D. B. (2004). Metabolomics and systems biology: making sense of the soup. *Curr. Opin. Microbiol.*, 7(3):296–307.

Kibby, M. R. (1969). Stochastic method for the simulation of biochemical systems on a digital computer. *Nature*, 222(190):298–299.

Kiehl, T. R., Mattheyses, R. M., and Simmons, M. K. (2004). Hybrid simulation of cellular behavior. *Bioinformatics*, 20(3):316–322.

Kitano, H. (2002a). Computational systems biology. *Nature*, 420(6912):206–210.

Kitano, H. (2002b). Systems biology: A brief overview. *Science*, 295(5560):1662–1664.

Knuth, D. E. (1964). Backus Normal Form versus Backus Naur Form. *Commun. ACM*, 7(12):735–736.

Knuth, D. E. (2004). *Selected Papers on Computer Science*. Cambridge University Press.

Kraus, M., Lais, P., and Wolf, B. (1992). Structured biological modelling: A method for the analysis and simulation of biological systems applied to oscillatory intracellular calcium waves. *Biosystems*, 27(3):145–169.

Kraus, M. and Wolf, B. (1992). Modellbildung in der Biologie: Strukturierte analyse intrazellulrer Calcium-Oszillationen in elektrisch nicht erregbaren Zellen. *Naturwissenschaften*, 79(7):289–299.

Krieger, E. and Vriend, G. (2002). Models@Home: Distributed computing in bioinformatics using a screensaver based approach. *Bioinformatics*, 18(2):315–318.

Kumar, S. P. and Feidler, J. C. (2003a). BioSPICE, 2. *OMICS*, 7(4):335.

Kumar, S. P. and Feidler, J. C. (2003b). BioSPICE: A computational infrastructure for integrative biology. *OMICS*, 7(3):225.

Kutta, W. M. (1901). Beitrag zur näherungsweisen integration totaler differentialgleichungen. *Z. Math. Phys.*, 46:435–453.

Larman, C. (2001). *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process*. Prentice Hall, 2nd edition.

Lebiedz, D. and Maurer, H. (2004). External optimal control of self-organisation dynamics in a chemotaxis reaction diffusion system. *Syst. Biol.*, 1(2):222–229.

Lee, D.-Y., Yun, C., Cho, A., Hou, B. K., Park, S., and Lee, S. Y. (2006). WebCell: a web-based environment for kinetic modeling and dynamic simulation of cellular networks. *Bioinformatics*, 22(9):1150–1151.

Lévy, P. P. (1948). *Processus stochastiques et mouvement brownien.* Gauthier-Villars, Paris.

Lloyd, C., Halstead, M., and Nielsen, P. (2004). CellML: Its future, present and past. *Prog. Biophys. Mol. Biol.*, 85(2–3):433–450.

Loew, L. M. and Schaff, J. C. (2001). The Virtual Cell: A software environment for computational cell biology. *Trends Biotechnol.*, 19(10):401–406.

Lu, T., Volfson, D., Tsimring, L., and Hasty, J. (2004). Cellular growth and division in the Gillespie algorithm. *Syst. Biol.*, 1(1):121–128.

Mayawala, K., Vlachos, D. G., and Edwards, J. S. (2006). Spatial modeling of dimerization reaction dynamics in the plasma membrane: Monte Carlo vs. continuum differential equations. *Biophys. Chem.*, 121(3):194–208.

Mendes, P. (1993). Gepasi: a software package for modelling the dynamics, steady states and control of biochemical and other systems. *Comput. Appl. Biosci.*, 9(5):563–571.

Mendes, P. (1997). Biochemistry by numbers: Simulation of biochemical pathways with Gepasi 3. *Trends Biochem. Sci.*, 21(9):361–363.

Mendes, P. and Kell, D. B. (1998). Non-linear optimization of biochemical pathways: Applications to metabolic engineering and parameter estimation. *Bioinformatics*, 14(10):869–883.

Mesarovic, M., Sreenath, S., and Keene, J. (2004). Search for organising principles: Understanding in systems biology. *Syst. Biol.*, 1(1):19–27.

Metropolis, N. and Ulam, S. (1949). The Monte Carlo method. *J. Am. Stat. Assoc.*, 44(247):335–341.

Miano, J. (1999). *Compressed Image File Formats.* Addison-Wesley Professional, 1st edition.

Mishra, B., Daruwala, R.-S., Zhou, Y., Ugel, N., Policriti, A., Antoniotti, M., Paxia, S., Rejali, M., Rudra, A., Cherepinsky, V., Silver, N., Casey, W., Piazza, C., Simeoni, M., Barbano, P., Spivak, M., Feng, J., Gill, O., Venkatesh, M., Cheng, F., Sun, B., Ioniata, I., Anantharaman, T., Hubbard, E. J. A., Pnueli, A., Harel, D., Chandru, V., Hariharan, R., Wigler, M., Park, F., Lin, S.-C., Lazebnik, Y., Winkler, F., Cantor, C. R., Carbone, A., and Gromov,

M. (2003). A sense of life: Computational and experimental investigations with models of biochemical and evolutionary processes. *OMICS*, 7(3):253–268.

Möller, M. (2006). *A Hybrid Algorithm for the Simulation of Biochemical Reactions and Diffusion*. PhD thesis, Bielefeld University.

Möller, M., Oleson, B. E., and Prank, K. (2002). 4DiCeS: A workbench for the simulation of intracellular signalling. In *Europ. Conf. Comp. Biol.*, volume ECCB 2002, page 109.

Möller, M., Oleson, B. E., and Prank, K. (2003). 4DiCeS: From random walk to rate walk: Statistic modeling and simulation. In *Europ. Conf. Comp. Biol.*, volume ECCB 2003, page MSB13.

Morton-Firth, C. J. (1998). *Stochastic Simualtion of Cell Signalling Pathways*. PhD thesis, University of Cambridge.

Morton-Firth, C. J. and Bray, D. (1998). Predicting temporal fluctuations in an intracellular signalling pathway. *J. Theor. Biol.*, 192(1):117–128.

Morton-Firth, C. J., Shimizu, T. S., and Bray, D. (1999). A free-energy-based stochastic simulation of the Tar receptor complex. *J. Mol. Biol.*, 286(4):1059–1074.

Murray, J. D. (2002a). *Mathematical Biology I: An Introduction*. Springer-Verlag New York Inc., 3rd edition.

Murray, J. D. (2002b). *Mathematical Biology II: Spatial Models and Biomedical Applications*. Springer-Verlag New York Inc., 3rd edition.

Murray, J. D. and Ryper, W. V. (1994). *Encyclopedia of Graphics File Formats*. O'reilly & Associates Inc., 1st edition.

Nasi, S. (2004). From databases to modelling of functional pathways. *Comp. Funct. Genom.*, 5(2):179–183.

Nauer, P. (1960). Revised report on the algorithmic language ALGOL 60. *Commun. ACM*, 3(5):299–314.

Nicolis, G. and Prigogine, I. (1971). Fluctuations in nonequilibrium systems. *Proc. Natl. Acad. Sci. USA*, 68(9):2102–2107.

Novère, N. L., Bornstein, B., Broicher, A., Courtot, M., Donizelli, M., Dharuri, H., Li, L., Sauro, H., Schilstra, M., Shapiro, B., Snoep, J. L., and Hucka, M. (2006). BioModels database: A free, centralized database of curated, published, quantitative kinetic models of biochemical and cellular systems. *Nucleic Acids Res.*, 34(Database Issue):D689–D691.

Novère, N. L. and Shimizu, T. S. (2001). StochSim: Modelling of stochastic biomolecular processes. *Bioinformatics*, 17(6):575–576.

Noyes, R. M. (1976a). Oscillations in chemical systems. XII. Applicability to closed systems of models with two and three variables. *J. Chem. Phys.*, 64(4):1266–1269.

Noyes, R. M. (1976b). Oscillations in chemical systems. XIV. Corrected stoichiometry of the Oregonator. *J. Chem. Phys.*, 65(2):848–849.

Oleson, B. E., Moeller, M., Evers, D., and Prank, K. (2006). 4DiCeS: Four-dimensional cell simulator. In *Int. Conf. Mol. Sys. Biol.*, number T25, pages 44–46.

Oleson, B. E., Möller, M., and Prank, K. (2002). 4DiCeS: Simulating diffusion of signalling molecules within a cell. In *Europ. Conf. Comp. Biol.*, volume ECCB 2002, page 116.

Oleson, B. E., Möller, M., and Prank, K. (2003). 4DiCeS: Four-dimensional cell simulation and visualization. In *Europ. Conf. Comp. Biol.*, volume ECCB 2003, page MSB14.

Oleson, B. E., Möller, M., and Prank, K. (2004). 4DiCeS: Simulation, parallelisation and model construction. In *Europ. Conf. Comp. Biol.*, volume ECCB 2004, page L27.

Olivier, B. G., Rohwer, J. M., and Hofmeyr, J.-H. S. (2005). Modelling cellular systems with PySCeS. *Bioinformatics*, 21(4):560–561.

Olivier, B. G. and Snoep, J. L. (2004). Web-based kinetic modelling using JWS online. *Bioinformatics*, 20(13):2143–2144.

Oltvai, Z. N. and Barabási, A.-L. (2002). Systems biology. Life's complexity pyramid. *Science*, 298(5594):763–764.

Ortoleva, P., Berry, E., Brun, Y., Fan, J., Fontus, M., Hubbard, K., Jaqaman, K., Jarymowycz, L., Navid, A., Sayyed-Ahmad, A., Shreif, Z., Stanley, F., Tuncay, K., Weitzke, E., and Wu, L.-C. (2003). The Karyote physico-chemical genomic, proteomic, metabolic cell modeling system. *OMICS*, 7(3):269–283.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers.

Petri, C. A. (1962). *Kommunikation mit Automaten*. PhD thesis, Technische Hochschule Darmstadt.

Pettinen, A., Aho, T., Smolander, O.-P., Manninen, T., Saarinen, A., Taattola, K.-L., Yli-Harja, O., and Linne, M.-L. (2005). Simulation tools for biochemical networks: evaluation of performance and usability. *Bioinformatics*, 21(3):357–363.

Ping, G., Yuan, J.-M., Sun, Z., and Wei, Y. (2004). Studies of effects of macromolecular crowding and confinement on protein folding and protein stability. *J. Mol. Recog.*, 17(5):433–440.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (1992). *Numerical recipies in C: The art of scientific computing*, chapter Random Numbers, pages 274–328. Cambridge University Press, 2nd edition.

Puchałka, J. and Kierzek, A. M. (2004). Bridging the gap between stochastic and deterministic regimes in the kinetic simulations of the biochemical reaction networks. *Biophys. J.*, 86(3):1357–1372.

Rathinam, M., Petzold, L. R., Cao, Y., and Gillespie, D. T. (2003). Stiffness in stochastic chemically reacting systems: The implicit tau-leaping method. *J. Chem. Phys.*, 119(24):12784–12794.

Ravasz, E., Somera, A. L., Mongru, D. A., Oltvai, Z. N., and Barabási, A. L. (2002). Hierarchical organization of modularity in metabolic networks. *Science*, 297(5586):1551–1555.

Rost, U. and Kummer, U. (2004). Visualization of biochemical network simulations with SimWiz. *Syst. Biol.*, 1(1):184–189.

Runge, C. D. T. (1895). Über die numerische Auflösung von Differentialgleichungen. *Math. Ann.*, 46:167–178.

Sakurada, T., Takahashi, K., and Tomita, M. (2002). E-CELL3 modeling environment. In *Int. Conf. Syst. Biol.*, volume ICSB 2002.

Sauro, H. M., Hucka, M., Finney, A., Wellock, C., Bolouri, H., Doyle, J., and Kitano, H. (2003). Next generation simulation tools: The systems biology workbench and BioSPICE integration. *OMICS*, 7(4):355–372.

Schaff, J., Fink, C. C., Slepchenko, B., Carson, J. H., and Loew, L. M. (1997). A general computational framework for modeling cellular structure and function. *Biophys. J.*, 73(3):1135–1146.

Schaff, J. and Loew, L. M. (1999). The virtual cell. *Pac. Symp. Biocomput.*, pages 228–239.

Schiek, R. L. and May, E. E. (2003). Development of a massively-parallel, biological circuit simulator. *IEEE Comp. Soc. Bioinf. Conf.*, 2003:620–622.

Schwehm, M. (2001). Fast stochastic simulation of metabolic networks. In Wingender, E., Hofestädt, R., and Liebich, I., editors, *Proc. German Conf. Bioinformatics (GCB 2001)*, pages 223–226.

Scott, K. (2004). *Fast Track UML 2.0.* Apress, 1st edition.

Shapiro, B. E., Levchenko, A., Meyerowitz, E. M., Wold, B. J., and Mjolsness, E. D. (2003). Cellerator: Extending a computer algebra system to include biochemical arrows for signal transduction simulations. *Bioinformatics*, 19(5):677–678.

Shapiro, B. E., Levchenko, A., and Mjolsness, E. (2002). Automated models generation for signal transduction with applications to MAP-Kinase pathways. In Kitano, H., editor, *Found. Syst. Biol.*, volume FSB 2002, pages 145–162. MIT Press. (first published at ICSB'00).

Sivakumaran, S., Hariharaputran, S., Mishra, J., and Bhalla, U. S. (2003). The database of quantitative cellular signaling: management and analysis of chemical kinetic models of signaling networks. *Bioinformatics*, 19(3):408–415.

Slepchenko, B. M., Schaff, J. C., Carson, J. H., and Loew, L. M. (2002). Computational cell biology: Spatiotemporal simulation of cellular events. *Annu. Rev. Biophys. Biomol. Struct.*, 31:423–441.

Sontag, E. D. (2004). Some new directions in control theory inspired by systems biology. *Syst. Biol.*, 1(1):9–18.

Spellman, P. T., Miller, M., Stewart, J., Troup, C., Sarkans, U., Chervitz, S., Bernhart, D., Sherlock, G., Ball, C., Lepage, M., Swiatek, M., Marks, W. L., Goncalves, J., Markel, S., Iordan, D., Shojatalab, M., Pizarro, A., White, J., Hubley, R., Deutsch, E., Senger, M., Aronow, B. J., Robinson, A., Bassett, D., Jr., C. J. S., and Brazma, A. (2002). Design and implementation of microarray gene expression markup language (mage-ml). *Genome Biol.*, 3(9):0046.0041–0046.0049.

Stundzia, A. and Lumsden, C. (1996). Stochastic simulation of coupled reaction-diffusion processes. *J. Comp. Phys.*, 127(1):196–207.

Takahashi, K., Arjunan, S. N. V., and Tomita, M. (2005). Space in systems biology of signaling pathways - towards intracellular molecular crowding in silico. *FEBS Letters*, 579:1783–1788.

Takahashi, K., Ishikawa, N., Sadamoto, Y., Sasamoto, H., Ohta, S., Shiozawa, A., Miyoshi, F., Naito, Y., Nakayama, Y., and Tomita, M. (2003). E-Cell 2: Multi-platform E-Cell simulation system. *Bioinformatics*, 19(13):1727–1729.

Takahashi, K., Yugi, K., Hashimoto, K., Yamada, Y., Pickett, C. J., and Tomita, M. (2002). Computational challenges in cell simulation: A software engineering approach. *IEEE Intell. Syst.*, 17(5):64–71.

Taylor, C. F., Paton, N. W., Garwood, K. L., Kirby, P. D., Stead, D. A., Yin, Z., Deutsch, E. W., Selway, L., Walker, J., Riba-Garcia, I., Mohammed, S., Deery, M. J., Howard, J. A., Dunkley, T., Aebersold, R., Kell, D. B., Lilley, K. S., Roepstorff, P., III., J. R. Y., Brass, A., Brown, A. J., Cash, P., Gaskell, S. J., Hubbard, S. J., and Oliver, S. G. (2003). A systematic approach to modeling, capturing, and disseminating proteomics experimental data. *Nat. Biotechnol.*, 21(3):247–254.

Tomita, M. (2001). Whole-cell simulation: A grand challenge of the 21st century. *Trends Biotechnol.*, 19(6):205–210.

Tomita, M., Hashimoto, K., Takahashi, K., Shimizu, T. S., Matsuzaki, Y., Miyoshi, F., Saito, K., Tanida, S., Yugi, K., Venter, J. C., and Hutchison III, C. A. (1997). E-CELL: Software environment for whole cell simulation. In *Genome Inform. Ser. Workshop Genome Inform.*, volume 8, pages 147–155.

Tomita, M., Hashimoto, K., Takahashi, K., Shimizu, T. S., Matsuzaki, Y., Miyoshi, F., Saito, K., Tanida, S., Yugi, K., Venter, J. C., and Hutchison III, C. A. (1999). E-CELL: Software environment for whole-cell simulation. *Bioinformatics*, 15(1):72–84.

Trost, E. (2002). Development of a pathway–editor and a web-application for lipid–associated disorders. Master's thesis, Technische Universität Graz.

van Gend, C. and Kummer, U. (2001). STODE-automatic stochastic simulation of systems described by differential equations. In *Int. Conf. Syst. Biol.*, volume ICSB 2001, pages 326–332.

van Zon, J. S. and ten Wolde, P. R. (2004). Green's-function reaction dynamics: A particle-based approach for simulating biochemical networks in time and space. *J. Chem. Phys.*, 123(23):234910–234926.

von Hayek, F. A. (1944). *The Road to Serfdom*. B&T.

von Neumann, J. (1966). *Theory of Self–reproducing Automata*. University of Illinois Press.

von Smoluchowski, M. (1906). Zur kinetischen Theorie der Brownschen Molekularbewegung und der Suspensionen. *Ann. d. Phys.*, 21(4):756–780.

Wagner, H., Moeller, M., and Prank, K. (2006). Coast: Controllable approximative stochastic reaction-algorithm. *J. Chem. Phys.*, 125(17):174104–174114.

Webb, K. and White, T. (2005). UML as a cell and biochemistry modeling language. *Biosystems*, 80(3):282–302.

Weimar, J. R. (1997). Cellular automata for reaction-diffusion systems. *Parall. Comp.*, 23(11):1699–1715.

Winfree, A. T. (1987). *The timing of biological clocks*. Number 19. Scientific American Library, 1st edition.

Winfree, A. T. (2001). *The Geometry of Biological Time*. Springer, 2nd edition.

Wirth, N. (1977). What can we do about the unnecessary diversity of notation for syntactic definitions? *Commun. ACM*, 20(11):822–823.

Wurthner, J. U., Mukhopadhyay, A. K., and Peimann, C.-J. (2000). A cellular automaton model of cellular signal transduction. *Comput. Biol. Med.*, 30(1):1–21.