Model-Based Biology Using Soft Real-Time Systems

By JACOPIN Eliott

Ph.D. Thesis

In fulfillment of the requirements for a Doctor’s Degree

March 2024

Graduate School of Frontier Biosciences

Osaka University

Abstract

From the Human Genome Project onwards, biology research has happily undertaken the path of high-throughput methodologies with the promise that big data would help reverse engineering biological systems. I formulate quantitative and qualitative objections to this sole inductive practice of biology, and I defend that biologists should give more consideration to deduction. Model-Based Biology (MBB) is my take at such deductive approach.

To incite biologists to practice deduction, we need to lead them to care more about the shape of the solution space of biology rather than the specific solutions observable on Earth. Biologists need tools to easily iterate on versions of models, test multiple alternatives, check for their integrity, and share with colleagues. That is why, I developed tools focusing on interactivity, trial and error, and collaboration.

The two software I developed are based on Real-Time (RT) technologies originally successful in engineering and, more recently, in entertainment. Indeed, the interactivity of RT simulation systems, as well as their scalability to support even complex systems such as digital twins or the Metaverse, makes them highly relevant for the interactive exploration of small or large biological models.

The first software, *ECellEngine*, was designed to build, simulate, and analyze plausible biological systems via a node-based scripting interface similar to flow-based programing in a GUI editor, linking with the RT simulation engine. Hence, this software has two levels; the low level is an interface in C++ for real-time simulation of scientific models for seasoned programmers, and the high level is a node-based editor that non-programmers can use to intuitively encode models. The second software, *Kosmogora* + *ECellDive*, is leveraging the benefits of RT collaboration in a scientific metaverse for the iterative design of models of biological systems. *Kosmogora* is a server instance helping to centralize biological data and simulation requests for users in the virtual reality software *ECellDive*. This explores the modes of collaboration between biologists to also promote MBB thanks to RT collaboration.

ECellEngine is part of the few software in systems biology which try to integrate all steps of modelling in systems biology. This is the unity of space because there is only one tool. The edge of *ECellEngine* over the other software lies in the unity of time because there is no clear separation between model building, simulation, and analysis thanks to the real-time architecture. Users can modify anything about a model, at any time, and immediately watch the effects. *Kosmogora* + *ECellDive* are among the first VR software for systems biology that go beyond data visualization to include modeling as its main feature. In addition, *ECellDive* may not be as polished as other software visually, but it questions the place of the Metaverse in a research field which other software hardly do.

ACKNOWLEDGEMENTS

I would like to thank Prof. Taiji for allowing me to apply to the doctor course of the FBS under his guidance, as well as for the valuable comments during the report meetings.

Thank you also very much to Dr. Takahashi for accepting me within his laboratory and providing me with an incredible work environment! Thank you to Dr. Kaizu for supervising my work during the past 3 years and our numerous fascinating conversations. Thank you very much also to Dr. Sakamoto for his great help in implementing *Kosmogora*! It saved me a considerable amount of time to focus on *ECellDive*. Thank you very much to Julien Soysouvanh for his high-quality insight into software architecture, C++ development, and video games engines: *ECellEngine* would have been very (very) different without you.

I would also like to thank Ikegami Kaoru, assistant for Dr. Takahashi's laboratory, Ohyama Yumie, assistant for Prof. Taiji's laboratory, and all the staff of the FBS Educational Affairs section for your constant help. Your presence and invaluable support have greatly facilitated my life as an international student in Japan.

Which brings me to my final thanks: thank you Japan for welcoming me. It might sound strange, but I think it is far too easy to forget that international and intercultural exchanges are fragile and should be protected by all the parties involved in it.

CONTENTS

List of Abbreviations

API	Application Programming Interface
AGC	Apollo Guidance Computer
CPU	Control Processor Unit
DARTS	Design Approach for Real-Time Systems
FAIR	Findable, Accessible, Interoperable, Reusable
FBA	Flux Balance Analysis
FPS	Frames Per Second
GE	Game Engine
GL	Game Loop
GO	Game Object
GUI	Graphic User Interface
HGP	Human Genome Project
HIL	Hardware-In-the-Loop
HRT-HOOD	Hard Real-Time Hierarchical Object-Oriented Design
HTTP	HyperText Transfer Protocol
MARS	Maintainable Real-Time System
MBB	Model-Based Biology
ML	Machine Learning
NASA	National Aeronautics and Space Administration
ODE	Ordinary Differential Equation
OOD	Object-Oriented Design
OOP	Object-Oriented Programming
RBM	Rule-Based Modeling
RNG	Random Number Generator
RT	Real-Time
RUDIL	Real-time User's Design and Implementation Language
SE	Stochastic Event
SIL	Software-In-the-Loop
SS	Stochastic System
STAR	Self-Testing And Repairing
TARP	Testing And Repair Processor
TMO	Time-triggered Message-triggered Object
UI	User Interface
URL	Uniform Resource Locators
USD	Universal Scene Description
UX	User eXprience
VR	Virtual Reality
WC	Whole-Cell
WCET	Worst Case Estimation Time
XR	eXtended Reality

LIST OF FIGURES

LIST OF TABLES

I. Introduction

Induction is Dominating Biology

Induction is fueled by the big data

Modern biology research is guided by the advent of high-throughput methodologies to the point that the field produces more data than it can handle. Sydney Brenner expressed a similar opinion in his Nobel prize lecture (Brenner, 2002) as a reaction to the data generated by the Human Genome Project (HGP) (Genome Sequencing Workshop, 1986; Bitensky, 1986; Collins and Galas, 1993; Collins *et al.*, 1998; Lander *et al.*, 2001; Venter *et al.*, 2001):

"We are all conscious today that we are drowning in a sea of data and starving for knowledge."

At the time the HGP was launched, the committee estimated sequencing speeds around to be around 100,000 base pairs (bp) per year which converts to about 11 bp per hour. Since then, data collection in biology has never ceased to accelerate with current sequencing technologies[1] advertising rates of 3.7 billion bp per hour to even 184 billion bp per hour. The data acquisition protocols in proteomics or metabolomics via nuclear magnetic resonance or mass spectrometry (and the prior processing steps) is much more fragile than for genomics. That is why many studies (Derks *et al.*, 2023; Jeppesen and Powers, 2023) are dedicated to improving the protocols. (Hajjar *et al.*, 2023) have, while reviewing metabolomics studies in human and plant research, pointed out the lack of standardized protocols in the communities yet. Moreover, they found that "[…] methods generally enabled detecting less than 250 metabolites in human biofluids, except for lipidomics approaches where more than 500 lipid species can be profiled in large populations […]" which is one or two orders of magnitude lower than the theoretical diversity. About proteomics, opinion paper (Slavov, 2021) also reports a wide diversity of protocols to identify a couple thousand proteins at most. Moreover, if the scan of a sample takes only a few minutes, the cohorts may very well reach several thousand which, with the preparation of the samples upstream and analysis downstream inevitably limits the throughput in comparison to genomics. Nevertheless, we observe the same efforts to strive for higher throughput methods in proteomics, with a notable interest for "untargeted" studies that falls right into big data approaches and, hence, induction. Independently of extraction methods, we also observe consequent efforts to break free from human limitations by using automated robot platforms to perform biological assays without human input (King *et al.*, 2009; Coutant *et al.*, 2019; Brunnsåker *et al.*, 2023).

[1] Pacbio (https://www.pacb.com/) Revio is based on long reads of 15-20kb yielding 90Gb in about 24h, Onso is based on short reads of 200bp yielding 100Gb in about 32h. Oxford Nanopore Technology (https://nanoporetech.com/) and its flagship *PromethION* advertises a theoretical output of 13.3Tb in 72h. While Illumina (https://www.illumina.com/) and its system *NextSeq 550* indicates yielding 120Gb with runs of 30h and advertises support for transcriptomics. These statistics are provided by the companies and were not verified.

The seminal paper (King *et al.*, 2009) describes a prototype robotic platform designed to work without human intervention to automate the discovery of biological knowledge. This platform works as a closed loop between experiments and hypothesis generation, ultimately relying on the data produced[2]. This closed-loop architecture has then been further developed as illustrated by (Coutant *et al.*, 2019) whose goal is to accelerate the iteration process of the model design in systems biology. The introductive statement of the paper acknowledges the need to increase the throughput of research techniques if biologists want to continue doing systems biology. An approach which will inevitably lead to more data. The work from (Brunnsåker *et al.*, 2023) specifically targets the automatization of cell cultures of *Saccharomyces cerevisiae* with untargeted mass-spectrometry to smooth the bottleneck of data throughput in metabolomics studies.

Following these strategies, induction in biology has yielded notable results. This is all thanks to a new kind of observations that the double helix structure of the DNA was discovered. It is all thanks to observations that diseases are detected, understood, and cured. It is all thanks to observations that species' genomes can edited.

Despite these incredible feats, I advocate there are reasons to worry about pursuing the big data in biology in the long-term. The first group of problems are quantitative and related to the intractability of data and knowledge acquisitions. Here, I use "intractability" as defined in the field of computer science: there does not exist any efficient solution to acquire data and knowledge and one can only use a "brute force" approach. The second group of problems are qualitative and related to ethical and philosophical considerations.

Quantitative problem 1: data acquisition is intractable

The first quantitative problem is that biologists will never have enough time or resources to acquire and store the data describing the whole biology of the universe. This can easily be illustrated thanks to the UniProt entry P04637 of the protein p53. The available information indicates 23 possible post-translational modifications effectively representing a solution space of 2^{23} states to investigate. Taken alone, there is no way to know which states have a causal relationship with biological behaviors so they must all be experimentally tested. And as experimental tests must be repeated to increase their statistical value the number of samples is probably greater than 2^{25} (i.e., at least 4 samples for each test). In the incredibly good situation where the output of the experiment can be measured with a single binary result (0 or 1), we can store 2 experiments of 4 samples into one byte. Therefore, we need to acquire 2^{22} bytes of data which only amounts to 4 MB. This lower bound is very acceptable today but would have already been a real challenge in the 90s. A bit more realistically, the output of the experiment will likely be, at least, a 16-bits (2 bytes) numeric value for each sample so the required data. This corresponds to 512 MB. That is still acceptable by today's

[2] The experiments to test the hypothesis are deduced from a model so, strictly speaking, it is incorrect to say that there is only induction (or abduction).

storage standards but note that this is the amount of data required to totally explore a unique causal link for all states. Of course, the holistic approach of systems biology enforces the view that the protein is only a single piece of a much bigger system. And causal link between the protein and other pieces of the system should, in theory, also be tested. So, one sample do not generate one 16-bits value but an arbitrary number x. Which linearly scales the required data to 512 x MB. Considering a human cell typically has at least hundreds modalities of interest, we can expect to easily reach 51,200 MB. That's 50 GB to describe the whole interaction space of one human protein, in interaction with a hundred elements in one type of cell. Then, the next question is what about the remaining protein states in the cell. What about the other species? What about multi-cellular organisms? What about the rest of the universe? Even if we limited ourselves to the biology on Earth, it is unrealistic to acquire enough data to describe all these subjects. The human species will surely go extinct or will not look like the human species anymore before we can manage this amount of data.

Of course, the classic response to this issue I am raising is that my worries are unwarranted because biologists do not need everything. But if that is really true, why bother with big data to begin with? Indeed, as (Leonelli, 2014) points out "[…] having a lot of data is not the same as having all of them […]". Indeed, Leonelli found that a major claim for advocates of the big data[3] is that one can truly start relying on correlations thanks to the innate diversity of the collected data. But as soon as one starts removing or favorizing a segment of the data collection, this "self-correcting" property of the big data shatters. So, no one who advocates for big data in biology can say that we don't need all the data without contradicting himself. Leonelli also argues that, anyway, "The ways in which Big Data is assembled for further analysis clearly introduce numerous biases related to methods for data collection, storage, dissemination and visualization." This point is also brought forward in (Leonelli, 2019) where Leonelli argues that the tools themselves used to acquire data are biased toward what biologists think is relevant data in the first place. Thus, the act of creating an online database to aggregate a type of biological data rather than another is not without consequences. It elevates this database to the rank of de facto benchmark for this collection of data (e.g., PDB or UniProt), it is guiding the way mainstream biology think this data, and the scientific value it supposedly brings. Consequently, the belief that big data in biology can reach "self-correct[ness]" is a misconception because the big data in biology is full of biases. Unfortunately, despite this misconception, Leonelli obverses that databases have enabled enough breakthroughs that biologists are ignoring the weaknesses of their arguments in favour of the big data, and even promotes the acquisition of more data.

[3] In the paper, Leonelli analyses the claims of Mayer-Schönberger and Cukier in their book "Big Data"(Mayer-Schönberger and Cukier, 2014)

Quantitative problem 2: knowledge acquisition is intractable

The second quantitative problem is that, even if we somehow acquire all the data, there is nonetheless the issue of extracting meaningful information from it. Brenner's criticism of the big data for biology was also about this problem of knowledge acquisition. He feared that the amount of individual collected facts renders the extraction of core knowledge particularly hard. He was convinced that the holistic approach of systems biology to reverse engineer the fundamental properties of life would fail because of the intractable nature of knowledge extraction in biology. The knowledge extraction methodologies are always lagging behind the quantity of data. He argued that biology should develop a better theoretical scheme to frame data acquisition to a specific context and extract the essential biological knowledge encoded in the extracted data. Since 2002 and his Nobel prize lecture, he suggested that the correct unit to study biology was the cell, in opposition to the predominant view that genes were the correct unit. According to his claims, focusing on genes would make the task too complicated for a long time because we have yet too much to learn about them to fully grasp their implications. Cells, on the other hand, were easier to probe —this still holds true today— and stand at a perfect junction point between biomolecules and organisms. Brenner also argued that there would be a methodological unification of sorts where we would start seeing organisms as a network of interacting cells the same way we approach cells as a network of interacting molecules (Brenner, 2010). He called this approach *CELLMAP* and suggested that biological science should undergo a planet-wide project similar to the HGP to build this map. Funnily enough, he noted (Brenner, 2002) that it took 17 years from the start of the HGP to the publication of the genome; so we'd be finished with *CELLMAP* by 2020. None of that ever happened during Brenner's career, but we must today recognize the launch of the Human Cell Atlas project (Rozenblatt-Rosen *et al.*, 2017; Regev *et al.*, 2017, 2018) which shares the ambition although, maybe, too oriented toward cataloging the diversity of cells thanks to experiments rather than an effort to modeling and reductionism as Brenner appeared to defend. Around 2010, Brenner himself seemed to acknowledge biology had gone full-in the big data approach instead of anything close to what he suggested (Brenner, 2010, 2012). He continued to argue that reductionism and forward modeling was the only way.

What Brenner did not foresee is the rise of Machine Learning (ML) methods that can actually help us extract knowledge from this kind of gigantic data repositories. Consequently, system biologists have, as in other fields, started relying on ML methods to filter the noise out of the big data and extract biological knowledge from it. From the perspective of reductionists, this is an admission of weakness of holism. From the holist's perspective, this is just a methodological continuation and a golden opportunity to eventually see their approach prevail.

But there is still a fatal flow to ML methods. They are themselves dependent on data to perform correctly. Which means that using ML methods to solve *Quantitative Problem 2* on knowledge extraction from data, actually exacerbates *Quantitative Problem 1* on data acquisition because additional data must be extracted upstream to train ML methods. Hence, using ML methods puts even more pressure on biologists to acquire data.

Moreover, biologists themselves tend to forget the epistemic heterogeneity of their own field. This is illustrated by (Callebaut, 2012) where he points out the overlap and lacks of clear definitions of “bioinformatics”, “computational biology”, “systems biology”, and “synthetic biology”. All four of which are part of the movement of the big data in biology and are working with the same data objects but have different point of view. Similarly, “the lack of unary definitions for biological terms” is also acknowledged in (Leonelli, 2019) as operating as a multiplicative factor for the interpretation of data.

Qualitative problem 1: “always more” is morally questionable

The big data is yet another embodiment of our societies thinking “more is better”? As opportunistic as it may seem, and fully aware that analogies and comparison across history are dangerous, I point out that big data approaches bear many similarities with economical productivism. Productivism has supported the capitalistic economies of western countries since the industrial revolution, and the whole planet since, let’s say, the end of the 2nd World War (this is, of course, a very crude approximation). Furthermore, it is nowadays largely accepted that this way of life is chiefly responsible for Earth’s resources depletion and climate downfall. Despite this tragic consequences, human societies and social sub-groups default practices perpetuates the thinking that “more is better”. I am sincerely anxious to see that, once again, a human population (biologists) has reacted to the problem of “we have too much” (data) by relying on a technological solution that “needs even more” (machine learning). Training deep ML models on the latest GPUs takes terra watts of electricity (Strubell *et al.*, 2019; Bender *et al.*, 2021) and, ironically, newest learning algorithms often require even more data than we already had to properly train the previous generation. If that is not a self-reinforcing predatory pattern, I don’t know what is.

In my opinion, despite the spectacular results that tools like AlphaFold (Jumper *et al.*, 2021) brought in biology to predict protein structure, it is a serious ethical breach to let supervised ML become the new default methodology to extract biological knowledge from biological facts. I would very much like if every biologist (experimentalists and modelers alike) could think a bit more about the data sustainability of their experiments before they implement them. How much data is enough to answer a biological question? What is the sufficient minimal amount of information to solve my problem?

Answers to those questions obviously depend on a lot of factors, and their difficulty is also a reason why it is so tempting to embrace big data. Because it is much more convenient to ignore this problem, acquire as much data as possible to, only then, see what knowledge you can extract from it. But this is a devil in disguise: the less focused you behave, the more entropy you generate, and the more you indirectly participate to global warming. There are environmental consequences to scientific practices, and I do not want to ignore them.

That is why, it feels wrong to practice systems biology the same way my peers do. And this is where I agree with Brenner's incentive to develop a theoretical framework for biology which he reiterated in a short opinion paper (Brenner, 2012) at the occasion of Turing's centenary[4]. Given the legacy of Turing's work, I understand Brenner had very high standards as to what a theoretical framework of biology should be. And I think he had hoped his *CELLMAP* project would eventually lead to that. For sure, I would have sincerely liked to see that.

Qualitative problem 2: induction is biasing biology research toward mechanisms

The second qualitative problem is that, if we only increase our biological knowledge based on what we can observe, then we will never be able to cope with the biology we do not have access to. The corollary of this problem is that biologists focus more on how biological systems solve environmental problems on Earth, rather than what defines biotic systems.

It is my hypothesis that biology's legacy from naturalists has biased it toward studying biological systems that live only on Earth with little consideration for investigating life in general. This stance is a direct consequence of naturalists' practice to mainly dedicate themselves to the description and observation of biological systems rather than their analysis. On that note, (Callebaut, 2012; Leonelli, 2014) both argue that the work of naturalists or taxonomists from the 18th century greatly diminishes the claims that the big data in biology is a new and transformative scientific practice. Focusing on data acquisition has deprived biologists of a whole way of investigating life by restricting them to reverse engineering based on observations. Indeed, the more biological data biologists have, the more they focus on understanding the process exemplified by the data rather than the problem solved by the process. Thus, more often asking "how does this biological system solves that problem according to the data?" instead of "what is the problem?", even when the latter question is a more general way to reach the answer to the former.

In fact, the concept of homoplasy[5] between organisms in the field of phylogeny illustrates that the "what-question" is as much relevant as the "how-question". Indeed, if you understand the problem that must be solved ("what-question"), the solution found by a biological system ("how-question") is but one occurrence in a much larger solution space. Focusing on answering a set of "how-questions" will only yield a point cloud in the solution space, and not the overall shape. Interestingly, this can be linked back to the enthusiasm of system biologists for ML because the primary goal of a classifier in ML is to find the shape of a solution space from only observations making the point cloud. Therefore, machine learning

[4] Here is the quote of the first paragraph of the paper: "Biological research is in crisis, and in Alan Turing's work there is much to guide us. Technology gives us the tools to analyse organisms at all scales, but we are drowning in a sea of data and thirsting for some theoretical framework with which to understand it. Although many believe that 'more is better', history tells us that 'least is best'. We need theory and a firm grasp on the nature of the objects we study to predict the rest."

[5] Homoplasy is the convergence of unrelated biological systems (species) over time toward similar features to solve an environmental problem.

and the big data of biology are the technological solutions to exploring biology primarily through “how-questions” in the hope of eventually answering “what-questions”.

In my opinion, focusing on “how-questions” has led the biology community to rather ironic situations where the consensus was that life in deep seas or other extreme environments was impossible because it wouldn’t match the current observations. Nowadays, this kind of life is acknowledged and often described as “extremophiles”. Which is no less ironic because it perpetuates the belief that the life observed until then was the norm and precludes biologists from considering the existence of yet unknown life forms.

Although Brenner apparently decided to accept the big data of biology[6] (Brenner, 2010) and, therefore, the dominance of “how-questions” over “what-question”, I do not. I believe that there is room for an alternative to inductive biology.

Defining a Deductive Approach for Biology

Instead of induction, it is my thesis that a deductive approach would benefit biology tremendously.

Non-inductive biology already exists

Theoretical frameworks for biology have received a lot of interest from the side of philosophers (Woodger, 1937, 1952, 1962), mathematicians (Turing, 1952), computer scientists (Nagasaki *et al.*, 1999; Danos and Laneve, 2004; Blinov *et al.*, 2004), astrobiologists (Bartlett and Wong, 2020; Wong *et al.*, 2022), data scientists (Wong and Prabhu, 2023), artificial life researchers (Lehman and Stanley, 2015; Hernández-Orozco *et al.*, 2018; Gershenson, 2023), and probably others. But even if we decide to dismiss these peripheral frameworks for some reason, Darwin’s evolution theory (Darwin, 1859) or the work on genetics initiated by Mendel (Mendel, 1865) and later Fischer (Fisher, 1930), do not need any observations to build them.

Of course, it so happens that these theories seem verified by observations on Earth. But, for instance, in the case of Darwin’s evolution theory, a founding hypothesis along the line of: “*only biological systems matching the constraints of their ecosystems can survive*”, is enough. From this statement, any change in the environment implies failure of some biological systems and a long term “selection” of the more “fitted” ones takes place. I think anything related to evolution and genetics do not require biological observations to be formalized. Consider the following statements:

- Information can be encoded physically (as in, it takes physical space)
- Information can be stored physically

[6] Here is a citation: “*No use will be served by regretting the passing of the golden years of molecular genetics when much was accomplished by combining thought with a few well-chosen experiments in simple virus and bacterial systems; nor is it useful to decry the present approach of ‘low input, high throughput, no output’ biology which dominates the pages of our relentlessly competing scientific journals.*”

- Information can be shuffled

The phrasing I used for this set of hypotheses is indeed very influenced by human's young history of devices to store information (arguably only 150 years old if I put paper and clay tablets aside). On that note, it is interesting that Mendel's work on genetic inheritance (Mendel, 1865) was about a decade before Edison's tin foil phonograph (1877). It is a fascinating coincidence that, while Mendel describes biological information inheritance, Edison builds a device to store sound information. Had Mendel never worked on this subject, then Fisher (Fisher, 1930) would have discovered it anyway (since it is said that nor Darwin nor Fisher knew about Mendel's work).

If I push this fictional situation further, had Fisher never worked on this subject, then Turing machines are an example of a system that satisfies the hypothesis (and even more since a Turing machine is capable of encoding and storing, for the sake of computation which might result in shuffling). So, I hope these examples illustrate that one does not need biological observations to have hints about how biological systems work.

A definition for model-based biology

Model-based biology is **the definition of symbols which can be combined to encode and deduce biological knowledge.** Many would probably argue that "model-based" is nothing new in biology. I might agree with them to some extent because we can find plethora of studies (see for example (Chance *et al.*, 1960; Goldbeter *et al.*, 1990; Novak *et al.*, 2001; Nijhout *et al.*, 2004; Akman *et al.*, 2008; Sivakumar *et al.*, 2011; Dritschel *et al.*, 2018; Dudziuk *et al.*, 2019; Nikolov *et al.*, 2020; Novak and Tyson, 2022), and hundreds of others on the BioModels database (Malik-Sheriff *et al.*, 2020)) that work out a model of a phenomenon first and analyze its qualitative behavior; some pursue further to confront hypothesis with experiments, and may update the model accordingly. This approach is, in my understanding, what Brenner's meant with his "way forward" to biology (as opposed to reverse engineering)(Brenner, 2010) and it contains some level of deduction. This is also very much a classic strategy in physics with famous successful examples in astronomy, gravity, or fundamental particle models.

Name	Addition	Multiplication
Associativity	$(a+b)+c=a+(b+c)$	$(a\,b)\,c=a\,(b\,c)$
Commutativity	$a+b=b+a$	$a\,b=b\,a$
Distributivity	$a\,(b+c)=a\,b+a\,c$	$(a+b)\,c=a\,c+b\,c$
Identity	$a+0=a=0+a$	$a\,1=a=1\,a$
Inverses	$a+(-a)=0=(-a)+a$	$a\,a^{-1}=1=a^{-1}a\;if\;a\neq 0$

Table 1: Field Axioms. Reproduced from (Weisstein)

The difference with my model-based biology is that I do not limit it to the formal expression of a target phenomenon. In fact, I include the formal expression of the problem associated with the phenomenon using the same symbols used to encode the model of the phenomenon and its solutions. This is classically the case in mathematics where axioms form elementary symbols that can be used to encode questions about the symbols themselves (problems), definitions of new symbols (models), and the relationships between the symbols (solutions). For example, the field axioms (see Table 1) specify the basic behavior for addition and multiplication of scalars. These axioms are used to encode a structure of the same name (i.e., *Field*); The *identity axiom* has the interesting consequence that it requires *Fields* to have at least two elements (0 and 1).

Name	**Vector**	**Scalar**
Commutativity	$X + Y = Y + X$	
Associativity	$(X + Y) + Z = X + (Y + Z)$	$a\,(b\,X) = (a\,b)\,X$
Identity	$X + 0 = 0 = 0 + X$	$1\,X = X$
Additive inverse	$\forall X \in V, \exists(-X), X + (-X) = 0$	
Distributivity	$a\,(X + Y) = a\,X + a\,Y$	$(a + b)\,X = a\,X + b\,X$

Table 2: Vector Space Axioms. V Is a vector space and F a field, $X, Y, Z \in V$ and $a, b \in F$. Adapted from (Weisstein).

Coupled with a *Vector Space*[7], this specifies a *Vector Space over Field* which axioms are written in Table 2. Without such *Vector Space over Field* it is formally impossible to perform any *Algebra* (Weisstein); hence impossible to count over real numbers, vectors, matrices, etc. Every piece of mathematics that is commonly used in physics or biology is void without these axioms; but the axioms themselves are unprovable. Despite that, they are the root of many scientific theories and that is why one judges the usefulness of a set of axioms to the usefulness of its theorems and not the opposite. The model-based biology I am suggesting would rely on such low-level statements to specify the basic notions of biological systems. Then, the statements would be combined together and with further hypotheses relevant to the domain of a target biological system. It would then be possible to derive new statements corresponding to the biological implications of the biological system.

An example of model-based biology

The following is an example of logic statements by John H. Woodger in his book *Biology and Language* (Woodger, 1952) derived from William Harvey's natural language statements in his work *Anatomical Disquisition on the Motion of the Heart and Blood in Animals* (1628). Starting with the hypothesis (at the time of Harvey) that a heart is a muscle, Woodger writes:

$$\forall x\, Heart(x) \supset Muscle(x) \quad H1$$

[7] A vector space alone is only a "*set that is closed under finite vector addition and scalar multiplication.*" In short, in a Cartesian space, this correspond to a component-wise addition of two vectors, and the multiplication of the scalar on each component. (Weisstein)

In English, $H1$ reads as "For all x, if x is a heart, then x is a muscle". Which is indeed a more formal way of saying that a heart is a muscle. Then, in conjunction with the following statement (known to be true at Harvey's time) that a muscle in action is hard:

$$\forall x\ \big((Muscle(x)\ \&\ Action(x)) \supset Hard(x)\big)\ S1$$

We can deduce from $H1$ and $S1$ that "A heart in action is hard":

$$\forall x\ \big((Heart(x)\ \&\ Action(x)) \supset Hard(x)\big)\ S2$$

The symbols $\{\forall, \&, \supset\}$ are inherited from Boolean Logic while the symbols $\{Heart,\ Muscle,\ Action\}$ are inherited from Biology. This is the kind of biology where one starts from statements and manipulates the statements to deduce new laws. Of course, this biological knowledge appears quite trivial to nowadays. But what matters is the capacity of a deductive framework to suggest knowledge without actually observing a heart firsthand.

The validity of results in model-based biology

In this approach, the skepticism of biologists (probably built up by their experience in solving "how-questions") who might want to ask, "How do you verify that your model is true?" is uncalled for. In fact, when biologists ask this question about the validity of the model, the "truth" they speak of is usually associated with the predictive capabilities of a model. They judge the usefulness and truthfulness of a model to the accuracy of its solutions compared with the biological phenomena they have observed on Earth. Once again, they focus on the point cloud and not the overall shape of the solution space. Whereas, if you encode the problem in your model as I suggest, then the validity of the model is self-contained, and it is harmless to keep making biological hypothesis without data within this frame. It is possible that none of the solutions within the frame are observable on Earth. In fact, if one defines large problems, it is very likely that it is not, because Earth is only representative of a subset of the constraints that leads to a subset of biological problems. For example, it is unlikely (but maybe not impossible) to find biological systems with functions similar to fishes if the constraint associated with large water areas are absent from the environment. Biomedicine is probably the only subfield of biology that should strongly require that predictions of a model match the observations on Earth. But even so, it does not disqualify the "model-based" approach, as I already explained that answering "what-questions" is a more general way to answer "how-questions". Therefore, I believe the premise of scientific work in biology should be to identify and understand the problems rather than phenomena.

About model-based biology in this thesis

In this thesis, I focus on tools to achieve MBB because I believe this is the easiest way to talk to biologists. The theoretical framework of mathematics has roughly 5000 years of history. It probably will not take biologists another 5000 years to reach an equivalent quality for the theoretical framework of biology because we are (hopefully) more knowledgeable of other concepts which can inspire us. Hence, if one accepts to consider that model-based might be a viable way to do biology, where to start? What symbols to use? What language? What ground theory? What tools? As I already mentioned, many scientists have tried before to develop such deductive framework for biology but it has never persisted. Conincidently, I note that the abstract level of the frameworks was rather high which is usually not the best medium to communicate with biologists. That is why I am taking the direction of intuitive easy-to-use, direct feedback tools. To enable biologists to see, manipulate, to get a feel, and share with other biologists what MBB is like. The tools I developed are software to explore new ways to build biological models, simulate them, and analyze them collaboratively, or alone.

In the following sections of this introduction, I will present modeling approaches and well-established approaches that relate to my endeavor to practice MBB. Then, I will focus on a set of technologies which, as far as I know, have never been used for modeling and simulations in systems biology until my work, namely real-time technologies. I argue that these technologies offer multiple advantages to practice model-based science in general and, hence, are adapted for MBB.

Basic Theoretical Methods in Systems Biology

The aim of this section is dual. First, it is the occasion to present state-of-the-art modeling methods that will be mentioned repeatedly in the rest of this thesis. Second, and the most significant in light of what I discussed in the previous section, it is to raise awareness about the implications of using a specific modeling method on our conceptualization and perception of a target biological system. It is even more necessary as there are no techniques or tools in theoretical system biology that has not been imported from other fields.

Modeling with differential equations

A differential equation defines an unknown function by its derivative. Differential equations are pure mathematical constructs which inherits from the versatility of mathematical symbols to encode problems, models, and solutions. If I stopped here, differential equations would appear as a very good sole candidate to practice MBB. And I will indeed use them later on in this thesis. But, as differential equations are very well defined, they are accompanied by theoretical constraints that are often ignored when modeling biological systems. In the following, I will focus on first order ordinary differential equations (ODE) as these are the most common in biology.

A first order (ODE) is of the form $y' = f(x, y)$ (Hairer and Wanner, 1993) with a given function $f(x, y)$. Thanks to the work of Cauchy in the 19th century (Cauchy, 1823), we know a unique solution may exist for arbitrary first-order differential equations when both f and $\partial f / \partial y$ are continuous on an open set in $\mathbb{R}^2$. In that case, a function $y(x)$ is a unique solution of $y'(x) = f\big(x, y(x)\big)$ for all x under the initial condition $y(x_0) = y_0$. This is a very important result as it guarantees that a phenomenon modeled this way will always yield a solution provided that the function f is chosen very carefully. I find it interesting that modelers in biology seldom take care to verify that this condition holds for all the differential

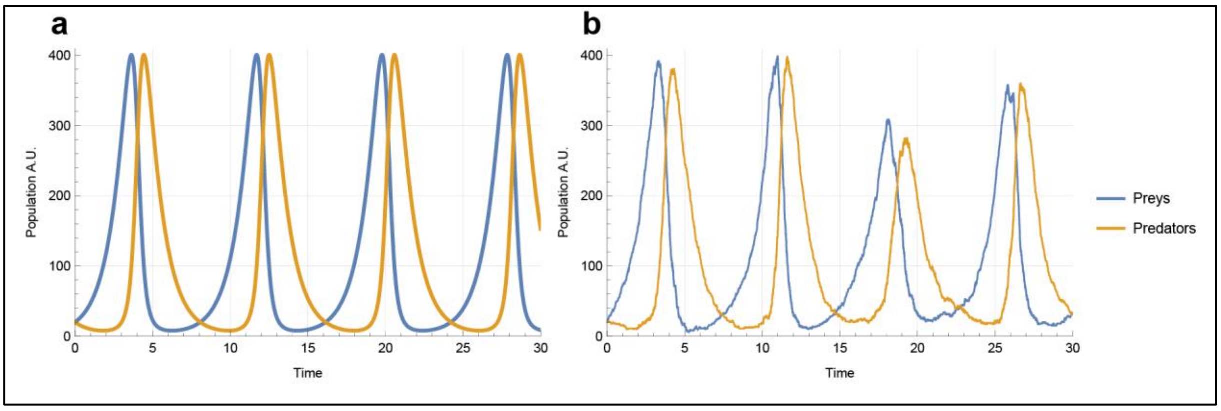


Figure 1: Ordinary Differential Equations (ODEs) vs. Stochastic Systems (SSs) implementation of Lotka-Volterra prey-predator model (Lotka, 1925; Volterra, 1926). The Lotka-Volterra model tries to capture the interactions between preys and predators. Preys can replicate natuarlly and die from being eaten by the predators. Predators grow only when eating preys and can die naturally. The equation system is $\frac{dprey}{dt} = \alpha.prey - \beta.prey.pred$ *and* $\frac{dpred}{dt} = \gamma.prey.pred - \delta.pred$*. Both simulations were realized with* $\alpha = 1$, $\beta = 0.01$, $\gamma = 0.01$, *and* $\delta = 1$*; the seed of the random number generator in Mathematica was set to 12357. a) Solved using ODEs. b) Solved using an SS. This is one of the stochastic realization of the simulation of the SS which happens to look similar to the ODE solution in value range. Other solutions may have higher or lower values, but the qualitative dynamics over time between ODEs and SSs are conserved if equivalent parameter values are used.*

equations they write. Of course, they often reuse and assemble well-known equations which have proven harmless in that regard but there are also instances of models where the functions are adapted forms and no care is given whatsoever to verify the condition (Chance *et al.*, 1960; Novak *et al.*, 2001); the equations are justified from a biology viewpoint but not from a mathematical one.

The physical interpretation of a first-order differential equation is that it represents the speed at which an entity varies relative to a variable (i.e., the derivation variable). Hence, using differential equations to model a system biases the conceptualization of the system as a set of variable-dependent transformations. In the case of biological systems, the set of variables is often limited to time or space. This limitation is the result of a more general research bias that modelers cultivate to always conceptualize a biological system as a physico-chemical system[8]. Consequently, in biology, first order ODEs are has been used to model time-dependent physico-chemical changes at various scales such as for reactions (Michaelis and Menten, 1913; Johnson and Goody, 2011), population (Lotka, 1925; Volterra, 1926) (see Figure 1.a for an example of the dynamics), epidemiology (Ross, 1916; Ross and Hudson, 1917a, 1917b; Kermack *et al.*, 1927), and more. It is noteworthy that in all these

[8] Here is another interesting quote from J. H. Woodger (Woodger, 1929) when he discusses the role of calculations and explanations in biology (p83): "*It is evident that physics and chemistry, having developed earlier than the other sciences, have, so to speak, set the fashion in this respect. The hypothetical entities of physical science have been imagined on the basis of perceptible physical objects. Consequently when biology began to develop it found an explanatory apparatus already to hand with which to pursue analysis. Had biology developed first it is at least possible that its infraperceptual entities might have been conceived on a biological model, and physics might have employed the notion of the organized body for its analysis. Physics would then perhaps have borrowed from biology, instead of vice versa.*"

examples, it is not trivial to identify the initial condition $y(x_0) = y_0$ because a modeler does not have control over the biological system and may be able to observe the current values of variables but does not have access to the initial ones. This implies that a modeler is often forced to estimate the initial values; sometimes thanks to established biological knowledge, some other times via unprovable hypotheses. In any case, the estimation becomes an *ad hoc* constraint that limits the outreach of the solutions of the model. This is not an issue in so far as the model does not focus on its general predictive capacity.

The condition on continuity of f also has a strange effect that might preclude the interpretation of the numerical solution of the ODE model of a biological system. In fact, if it is not much of an issue in physics to abstract quantities to floating point numbers to quantify abstract concepts (e.g. free energy, entropy, diffusion, debit, etc.), it is not so evident in biology where the goal is often to quantify concrete concepts. For example, when modeling an ecological system, what does 0.8 rabbit and 12.6 wolves mean? In such cases the value of the solution lies more in its dynamics over the derivation variable (e.g., time) than its exact numerical value at some points, and its predictive usefulness is limited to the qualitative changes observed over a large open set of the derivation variable. For example, is the population of wolve forecasted to decrease anytime soon and what would be the decrease factor.

Finally, modeling with ODEs also implies that "everything happens together at the same time". Indeed, a system of n differential equations defined by $y' = f(x, y)$ (with the vectors $y = (y_1, y_2, \dots, y_n)^T$ and $f = (f_1, f_2, \dots, f_n)^T$) is valid for every x. Hence, if ODEs are used to model reaction rates in a cell, it implies that any product A of a reaction R_1 that is also the reactant of a reaction R_2 is consumed as soon as it is produced. This likely is not the case in reality. Physically, A is only "consumed" in R_2 when it is in contact with the other reactants (and probably an enzyme as catalyst). Even if A is produced close to the reactants (and/or the enzyme) of R_2 there must be a delay, albeit imperceptible for humans. That is why we find, sometimes, mentions in the hypothesis of a model using ODEs that it assumes the concentration of the enzymes to be larger than that of the reactants. This assumption implies that it "does not take long" for an enzyme to come into contact with the reactants, hence ignoring the physical distance.

Modeling with stochastic events

A stochastic event (SE) is used to represent the occurrence of a phenomenon based on its probability. As for ODEs, stochasticity finds its theoretical ground in mathematics and its usage in biology inherits a large base of symbols to encode problems, models, and solutions. The nature of the encoding stands, however, in opposition to the determinism of ODEs as everything relates to probabilities. As a consequence, even if analytical solutions exist to describe SEs, they will only ever describe the likelihood that a variable takes some value but never which one it takes until the event is realized. In that way, SEs provide fewer ways to probe the phenomenon they represent than ODEs. On the other hand, SEs are better adapted to encode unary representations thanks to their discrete nature. For example, I mentioned previously that ODEs abstract the count of individuals due to the condition on continuity of the function. SEs are free from this constraint, and they happen "every once in a while" as per their probability and the event itself is not tied to a particular numerical effect. Hence, part of the modeler's work is to decide what should happen when the event occurs which, typically in biology, allows to increment or decrement quantities down to the smallest unit. Thus, it is possible for SEs to represent phenomena requiring concrete counts of entities.

Stochastic systems (SSs) are a set of stochastic events, and their realization is the sequence of triggered SEs. It uses the continuity of the axis variable to ensure that there exists a small interval $[x, x + \delta($ during which only one event is realized. Stochastic systems in the case of biological applications are also subject to the bias of always conceptualizing biological systems as physical systems, so the axis variable is often time or space.

The range of application subjects of stochastic systems in biology overlaps with the subjects for ODEs. We find them in biochemistry (Gillespie, 1976, 1977), ecology (Getz, 1976) (see Figure 1.b for one potential realization of the simulation of the SS corresponding to a Lokta-Volterra model), epidemiology (Tuckwell and Williams, 2007; Allen, 2008; Jacopin *et al.*, 2020), and so on. It is actually not rare that a study with a primary model using ODE would also present some solutions of an equivalent of the model encoded as an SS, or vice versa. The goal when doing both is to observe an average behavior of the model with the ODEs while the randomness of a SS allows to observe a different behavior every time a solution is computed.

Gillespie greatly contributed to the modeling of a system of chemical reactions thanks to a SS (Gillespie, 1976, 1977, 1992). His modeling started from the probability of collision between molecules undergoing Brownian motion involved in a reaction with the form R_μ : $S_a + S_b \rightarrow some\ products$ during the infinitesimal time $[t, t + \delta($ in a well-stirred and thermodynamically stable environment. We can demonstrate (Gillespie, 1976, 1992) that this probability follows the form $P(R_\mu)_{t+\delta} = c_\mu \delta$ where c_μ is "the specific probability constant for reaction R_μ". Gillespie demonstrated that this stochastic process was an exact match to the master chemical equation.

The major implication of using SSs is that it assumes the knowledge of the probability function of each SE, and this is far from obvious. In most cases, similarly to how a modeler would build up ad hoc functions in ODEs, modelers will write ad hoc probability functions. In the case of a system of chemical reactions, these are called "*propensities*" and Gillespie wrote them as the "*number of distinct molecular reactant combinations for R_μ found to be present in V at time t*" (Gillespie, 1976). Hence, for the reactions:

- $S_i \longrightarrow some\ products$
- $S_i + S_j \longrightarrow some\ products$
- $2S_i \longrightarrow some\ products$

The propensities write as N_i, $N_i * N_j$, and $N_i(N_i - 1)/2$ respectively. It is noteworthy that these propensities follow the law of mass action of reactions of order 1 and 2. This is a direct consequence of the condition that only elementary reactions may occur during the time $[t, t + \delta($ and reactions of orders different from 0, 1, or 2, can only be sequential combinations of the elementary ones. However, a common modeling practice in the field is to write the propensity function as $a_i(\boldsymbol{X}) = f_i(\boldsymbol{X}, t)$ where f_i is the deterministic rate function that would be used in and ODE version of the model. Indeed, the flux-compliant formalism of ODEs offers a much more convenient way to define the reaction rates. The theoretical ground for this practice is discussed in (Wu *et al.*, 2011) and they found three sufficient conditions which would allow it. The following is a direct quote from the paper:

1) *f is a linear function*
2) *the reaction is monomolecular*
3) *all X_i in the system are noise-free variables, i.e., without (or with ignorable) fluctuations, which implies that the covariance of any two participating reactants is zero (or close to zero).*

If none of these sufficient conditions are verified, then the propensity is not compatible with the master chemical equation and the theoretical ground of the method vanishes. It is not an issue as long as the modeler remembers it and brings forward another rational to justify the structure of the equation.

Modeling with rules

A rule is a map to describe a transformation between states without committing to a numerical or simulation method to solve the transformation. That is why both ODEs and SSs can be used to numerically realize the rule. Rule-based modeling (RBM) is the practice to describe a process by a set of rules over variables undergoing change through this process. RBM is a term which appears mainly in theoretical systems biology (Danos and Laneve, 2003, 2004; Blinov *et al.*, 2004; Hlavacek *et al.*, 2006; Faeder *et al.*, 2009; Danos *et al.*, 2012; Harris *et al.*, 2016; Boutillier *et al.*, 2017) although similar methodologies are employed to describe processes in different fields such as cellular automaton (Wolfram, 2002) or agent-based modeling (Bonabeau, 2002; Jacopin *et al.*, 2021). RBM technologies, in particular the κ-*language*, have put in a lot of effort to characterize its theoretical framework on the basis of set theory and has defined its own symbols to encode problems, models and solutions (Danos and Laneve, 2003, 2004; Danos *et al.*, 2012; Boutillier *et al.*, 2017).

Interestingly, the emergence of RBM – in the late 90s, early 2000 – among other attempts at devising a formal language to describe biological systems (e.g., based on π-calculus (Regev *et al.*, 2000; Priami *et al.*, 2001; Regev and Shapiro, 2002)), coincides with the birth of the field of systems biology. Early papers on the two major RBM technologies κ-*language* (Danos and Laneve, 2003) and *BioNetGen* ((Blinov *et al.*, 2004)) which still remains today motivate their work by mentioning the increasing amount of experimental data being collected and the overwhelming complexity of interaction networks in biological systems. In fact, RBM was the pre-machine learning era attempt to satisfy the holistic ambitions of system biology. RBM is indeed very efficient at reducing a collection of low-level biological phenomena (e.g., post-translational modification) to a macro representation (i.e., the rule). For example, the UniProt (The UniProt Consortium, 2023) entry P04637 of the protein p53 indicates 23 possible post-translational modifications. This represents 2^{23} states. Thanks to RBM, this state space can be compressed to 23 rules which, in *BioNetGen*, would read as `p53_Ser6Phospho: p53(Ser6~0) -> p53(Ser6~P) kSer6P`[9]. Essentially, this decreases the representation complexity of the state's space transitions from exponential to linear. In that way, it is using reductionism to solve holism because there is no loss of information in this encoding.RBM is not devoid of issues, however. For example, both mainstream technologies κ-*language* and *BioNetGen* are focusing on molecular scale phenomena, thus are excellent to encode large-scale metabolic or interaction networks, but struggle, for example, to encode polymerization reactions or to account for compartments. Despite the real potential of RBM, its development has slowed down from 2010 onward and these problems were, to the best of my knowledge, never solved.

Modeling with software

This category of modeling is a bit different from the three precedent approaches because it does not exist alone and is relevant only in so far as it uses some lower-level modeling layers to encode the effect of processes. In fact, it is encouraged in software engineering to develop libraries with accessible interfaces promoting the combination of several libraries inside a unique software. Hence, software becomes a heterogeneous entity that makes the best of every subpart depending on execution requirements. For example, I mentioned earlier that ODEs are better suited for compartment-based models whereas SSs have the possibility to describe events down to the smallest unary elements of a system. As a consequence, it has become standard to have both an ODE solver and an SS solver in modeling software in order to use the most appropriate one depending on the nature of the model. Some software even allow that both are used at the same time (Takahashi *et al.*, 2004) and synchronized (to some extent) to represent metabolism with ODEs and gene interactions (i.e., for complexation of enhancers or inhibitors) with SEs.

The **Constraint-Based Reconstruction and Analysis (COBRA)** (Becker *et al.*, 2007; Ebrahim *et al.*, 2013; Heirendt *et al.*, 2019) library is a tool suite to generate genome scale metabolic models with language bindings in Matlab or Python. It starts by gathering multi-omics, physiology, and biochemistry data to automatically reconstruct metabolic network. This can be followed by a manual curation before applying physico-chemical constraints such as

[9] This is a very simple example of the rule syntax that would describe the phosphorylation of the residue serine at position 6 in the sequence. Most likely, the phosphorylation rule would be preceded by a rule describing the binding of the enzyme operating the phosphorylation. Here is a possibility: (next page)

```
p53_Bind: A(a) + p53(b, Ser6~0) -> A(a!1).p53(b!1, Ser6~0) kBind
```

```
p53_Ser6Phospho: A(a!1).p53(Ser6~0) -> A(a) + p53(b, Ser6~P) kSer6P
```

thermodynamics, mass conservation, rate laws, further experimental data. The constraint-based approach aims to reduce the degrees of freedom of the solutions space of a biological system. Indeed, relying on the constraints and the theory backing each one of them, the generated models have meanings in a well-identified frame (solution space is restricted).

The **COmplex PAthway SImulator (COPASI)** (Hoops *et al.*, 2006) library focuses on biochemical network modeling that can be simulated using a variety of popular methods such as ODEs or SSs with multiple language bindings in C#, Python, and Java. It supports arbitrary detection and trigger of events (even within ODEs system) and provides some tools to make up for difficulties associated with simulation methods such as parameter estimation which is very relevant for ODEs.

E-Cell (Tomita *et al.*, 1999; Takahashi, Ishikawa, *et al.*, 2003; Takahashi, Sakurada, *et al.*, 2003; Kaizu *et al.*, 2020) is a modeling and simulation environment focusing on the integration of multi-bioprocesses within a model. It is noteworthy that the first version of the software (Tomita *et al.*, 1999) claimed to rely on RBM to encode models a few years before the emergence of *κ-language* and *BioNetGen*. The first version of the system was also used to build the first attempt at a whole-cell model using 127 genes of *Mycoplasma genitalium*. Currently, ECell is in the fourth version of its tool suite and provides a custom RBM system, support for ODEs and SSs, and can perform single-particle simulations in multiple dimensions; all with a Python interface.

The **VIVARIUM** (Agmon *et al.*, 2022) library is a modern take on a multi-framework simulation environment that E-Cell originally intended. It arguably possesses the most flexible architecture to encode composite models made from the aggregation of other models that may be running on different frameworks (ODEs and SSs, but also others such as solid body physics). Vivarium was used to build agent-based whole-cell models of *Escherichia coli* colony (Skalnik *et al.*, 2023). I find the case of Vivarium very interesting from a research practice point of view because it is one of the rare ones that emphasizes the architecture of the software as a driving force for modeling. The paper takes the time to introduce design choices because they are believed to enhance the user experience. And, in this case, users are likely biologists that want to build models. Hence, **it is not only about what the software can do but also a reflection on how software may be designed to do a better job at modeling in the future**.

The reason why I mention software as a modeling approach is because programming languages are, in my opinion, underutilized when it comes to creating brand-new symbols to encode a particular system. Usually, a programming language is indeed only viewed as an interface for humans to write instructions directing computer resources into performing sets of operations to produce desired effects. But in so doing, any software creates symbols (variables, functions, classes, etc.) with meaning in the context of the software they are part of. Hence, software could be the container of symbols used to encode problems, models, and solutions. A major difference between the encodings from software and the three previous approaches is that the latter have unique theoretical frameworks whereas the former's theoretical framework must be redefined for every software.

To summarize, all these modeling approaches have the capacity, to some extent, to encode problems, models, and solutions. Hence, they are, to some extent, all adapted to answer "what-questions" and to power the MBB I have in mind. Despite having this capacity, they keep being used to answer "how-questions." Biologists do not like ODEs because of the mathematics backing them up, but because they are a formal expression for fluxes. Likewise, biologists do not appreciate SSs because of the mathematics backing them up, but because they are a formal expression for unary events. For biologists, it is not so much about what the framework can rightfully express but the fact that a particular piece of the framework happens to be adapted to model a target. Biologists use ODEs or SSs to satisfy physical representations. It is the same for RBM which, as I already mentioned, was biased toward molecular descriptions from the start. On the other hand, software is only thought of as tools to implement the lower-level frameworks (ODEs, SSEs, RBM) and not as a source to produce new self-contained valid frameworks. Moreover, existing simulation software in systems biology suffers from a static design which corners the modeler into the cycle *build → simulate → log → analyze*.

To incite biologists into shifting their focus from considering "how-questions" to "what-questions", we need a different modeling methodology. We need to lead the modeler to care more about the problem he is studying than the specific solution he is implementing. We need tools that give him the power to easily iterate on versions of the models, test multiple alternatives, check for the integrity of the models, and share with colleagues. In the following sections of this dissertation, I will defend that **real-time technologies have the means to help me develop this different modeling methodology**.

II. Real-Time Technologies

This section serves as an overview of Real-Time (RT) technologies to understand the constraints of RT technologies and the reasons why I chose them to support MBB. RT systems are computational systems, usually integrated within bigger systems, whose criteria for a task execution's correctness not only include the accuracy of the computation but also the guarantee that the task is finished before a deadline. Research in the field started in the 60s, as soon as computers spread in specialized research laboratories or companies. Since then, the research in the field has spanned across several subjects as the raw performances of computers evolved, analog and digital communications developed, demand for rare faulty control systems rose, costs and performance of personal computers shriveled, Control Processor Units (CPU) went from a unique to multiples cores, and a new sector of non-critical real-time applications opened. In the beginning, the main concern was simply to figure out how to make a computer respond to outside signals (which could have been from humans or other machines) promptly without disrupting already running processes. Then, this concern evolved toward strict time-fault-free control systems in industrial plants or avionics and how the RT computer system integrates into a super system. This involved concurrent research on task scheduling algorithms (Liu and Layland, 1973; Altilar and Paker, 1998; Davis and Burns, 2011), cyber-physical systems (when a computer running simulations is connected to non-computer components) (Bloem and Naigus, 1988; St. John *et al.*, 1987), methodology development for RT system design (Sorenson and Hamacher, 1975; Gomaa, 1984), guarantee of execution completion (Puschner and Koza, 1989), parallel and distributed execution (Avello *et al.*, 1993; Fujimoto, 2001), and so on.

Basics on Real-Time Systems

This section on the basics of RT systems covers some terminology, an introduction on scheduling algorithms, fault-tolerance, and a history of design methods.

Terminology

RT systems are separated between *hard* and *soft*. The difference in denomination stems from the strictness of the condition on which a real-time system should respect the deadline. In the case of hard RT systems, the computation of the tasks must always finish before a set clock tick (Kopetz and Steiner, 2022e). Going over the deadline is called *overrun* and will often result in a system failure. Indeed, hard real-time systems are often associated with time-critical constraints such as the feedback control of an industrial plant, navigation tools in a plane, power grid regulation, or a braking system in a car. Failure of any of these will likely result in severe material damage. Conversely, a soft RT system is not expected to produce a critical failure upon overrun. Instead, the usefulness of the simulation is degraded and evaluated against a *quality of service* parameter (Buttazzo, 2005) or *quality of user experience* (Kopetz and Steiner, 2022e). Problematics related to soft RT systems are more recent than hard RT systems and match the joint decrease of costs of microprocessors and

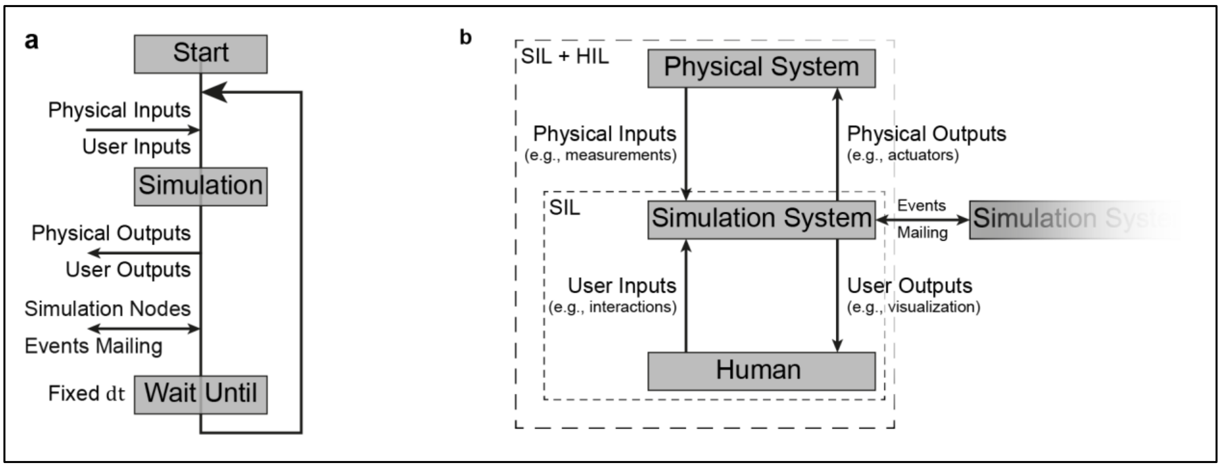


Figure 2: Conceptual structure of real-time systems. a) Typical sequence execution of the simulation loop of an RT system. In case the execution of all tasks took less time than the deadline dt, the process waits until then. b) Typical communications between a human operator, the simulation, and physical components of the RT system. If there is a simulation, the system is often qualified as software-in-the-loop (SIL). If there is a physical part, it often is hardware-in-the-loop (HIL).

the rise of multimedia purposes (streaming, virtual reality, Internet, smartphones, video games, etc.).

An RT simulation corresponds to the execution of an RT system in which clock time ticks are synchronized with the physical time such that one second of simulation maps to one second in reality. The consensus is that a simulation step starts by reading potential inputs from the other parts of the bigger system in which the RT system is integrated (e.g., input from sensors in a plant or human actions on operatable devices), performs necessary computations, sends the computation's results back to any parts of the system that might need it (e.g., a valve operator or an engine injection system), receive external events, and wait until the end of the step (Cellier and Kofman, 2006; Menghal and Laxmi, 2012). The schematic of the loop is in Figure 2.a.

Several measurements exist that are particularly relevant for hard RT systems to reason about how well an RT system and its simulations are indeed real-time. The first of which is called *latency jitter* and matters greatly for *Hardware-In-the-Loop* (HIL) (Cellier and Kofman, 2006; Menghal and Laxmi, 2012) RT systems. As the name implies, HIL real-time systems necessarily integrate components external to the RT simulation computer with which the computer exchanges data as indicated above (see Figure 2.b) during a simulation time step. The latency jitter corresponds to the delay between the time at which the external measurement was taken, and the time at which it reaches the components in the RT computer that will use it. However short this delay is, the value that reaches the computer will always be a little inaccurate. In the case of *Software-In-the-Loop* (SIL) RT systems (see Figure 2.b), everything is within the same computer but an artificial jitter (Kopetz and Steiner, 2022d) is usually set between simulated sensors and the RT process to simulate the real values as closely as possible. Other measures attempt to keep track of the *reliability*, *safety*, *maintainability*, *availability*, and *security* (Kopetz and Steiner, 2022e) of RT systems.

Tasks execution scheduling to enable real-time

RT systems rely on scheduling algorithms to execute sets of tasks within deadlines. The requirements and constraints for scheduling tasks in an RT system vary greatly whether they are hard or soft.

Scheduling algorithms for the hard RT systems are usually developed using the Worst Case Estimation Time (WCET) (Liu and Layland, 1973; Buttazzo, 2005; Kopetz and Steiner, 2022c) as a reference because a single overrun might have catastrophic consequences. Studies to find other metrics than WCET exist (e.g., maximum execution time (Puschner and Koza, 1989)), but there is always a trade-off between assumptions about the properties of the tasks and the RT system which may lead to ad hoc solutions. Getting those time estimates can be manageable on simple, well-isolated, cases analytically (source code and hardware specification analysis) or experimentally (execution measurements). However, the number of interacting components grows significantly in complex tasks with multiple computation processes relying on shared resources (typically data values). In addition, it is always possible that the *administrative tasks* of the computer (i.e., the mandatory tasks dictated by the operating system of the computer) account for a non-negligible overhead which, despite a valid WCET of the simulation alone, leads to overruns. Hence, there is always a certain level of uncertainty about the WCET which motivates us to overestimate them by "*typically more than 20%*" (Buttazzo, 2005). WCETs are typically done pre-simulation run or adapted during the development process of the hard RT system. In that case, static scheduling algorithms are used to search for the best execution order of all tasks without disrespecting the deadlines. Alternatively, there also exist dynamic algorithms (Liu and Layland, 1973) that process tasks according to set priorities, and the execution order of those tasks might be different from one simulation step to another. For example, (Caccamo *et al.*, 2000) describe an algorithm in which tasks are executed within servers with various budgets whose deadlines are indexed on the WCET of the tasks. The idea is to let a server help with the execution of non-finished tasks if it is finished dealing with its set of priority tasks. To that end, every server is allocated a time budget, and it can execute as much as possible of the task from another server until the end of its budget. A little later (Caccamo *et al.*, 2002) achieved a different purpose with the same algorithmic base. The idea is to modulate the accuracy of the output of the task depending on how much budget is available in a server. Typically, this means a fast, low-cost, and low-precision version of the task is first computed. Then, if there is enough time left, higher precision methods are executed and replace the low-precision evaluation. This can be very useful to make the most of the WCET independently of how long one execution of the task takes.

Nothing prevents soft RT systems from using WCET but it would be a waste of resources (Caccamo *et al.*, 2002) because *nothing bad* would happen if a soft RT system overruns. Consequently, using the time execution of the average case might be more appropriate. Many scheduling algorithms soft RT systems (Buttazzo, 2005) trade a few overruns against much better performances on average. In addition, there are applications for soft RT systems, such as in real-time graphics rendering, which do not have any deadlines at all. Instead, the system will always try to minimize the time of a simulation step to emphasize fluidity and interactivity. The soft RT systems I developed during my Ph.D. research are among these and will be presented in later sections.

Finally, one should note that scheduling problems got much more complicated since the shift from single-core to multi-core CPUs in consumer-grade computers. Indeed, the evolution of architecture of CPUs is also a driving force to develop different scheduling algorithms (Fujimoto, 2001; Davis and Burns, 2011; Maiza *et al.*, 2019)

Fault-tolerant real-time systems

Despite anyone's best effort, an RT system is bound to fail at some point. As a consequence, RT systems include hardware and software techniques to reduce the risk of failures or mitigate their repercussions.

The sources of failures are numerous, starting with the digital nature of computers where a single faulty bit in a calculation can produce drastically different results. Such errors can easily be the result of internal hardware fault from "natural" obsolescence and software bugs, or external fault such as power supply variations, radiations, electromagnetic interferences, and wrong input data (Kopetz and Steiner, 2022a). Environmental faults are often *transient faults* (intermittent malfunctions of components in the RT system) which might degenerate into permanent faults inevitably leading to permanent component failures. Should the failure be irrecoverable, the last resort for the RT system is to trigger *fail safe* or *fail operational* countermeasures (Kopetz and Steiner, 2022e). A fail-safe system is a default idle state of the system guaranteeing that no harm can be done, while a fail-operational system remains active and keeps providing basic services, maybe even everything except the failed component. A typical example (Kopetz and Steiner, 2022e) of a fail-safe RT system is a railway signalization network where one failure switches the whole network to a state where every train must stop. As for a fail-operational system, it typically (Kopetz and Steiner, 2022e) corresponds to a plane navigation system where the failure of one component must certainly not interrupt everything else.

A solution to limit the risk of reaching complete failure of a component is called *protective redundancy*. This was identified early on in the field and implemented in many RT systems at the National Aeronautics and Space Administration (NASA) (Aviz, 1969). The gist of it is to have multiple identical components synchronously processing the same tasks connected to voters to decide which result to select. Soon after, protective redundancy and other methodologies for fault-tolerant hardware such as concurrent error detection, or rollbacks to uncontaminated execution streams were adapted for software (Hecht, 1976). The Maintainable Real-Time System (MARS) (Kopetz *et al.*, 1989) is an example of a thorough RT system with industrial applications which implemented hardware and software redundancy. MARS was used for railway-control systems and added a self-checking feature. Interestingly, this would cause the RT system to shut down at the first encountered fault without attempting to recover. Hence, both transient and permanent faults would result in a shutdown.

Later, as distributed RT systems with multiple RT computers and sensors spread, the time synchronization between the clocks of each RT computer became a major issue. Indeed, every clock undergoes a clock drift which desynchronizes them from other clocks[10]. Harsh environmental conditions can significantly inflate the clock drift. Desynchronization implies that a set of time-ordered events is not guaranteed to be processed in the correct order in all RT computers. Thus, despite known limitations (Kopetz and Steiner, 2022b), an RT system with multiple sensors or computation unit must periodically resynchronize all clocks. Even so, some clocks may be found faulty, and simply resetting them across an RT system would not be enough. Therefore, special fault-tolerant time synchronization algorithms were developed (Dolev *et al.*, 1986; Kopetz and Ochsenreiter, 1987; Kopetz and Steiner, 2022b).

As I already mentioned, the architecture and protocols of modern multi-core CPUs are different from the CPUs that were used at the beginning of the research on real-time systems. Technologies such as direct memory access, pre-fetching, or cache management were introduced in an attempt to reduce the average execution time of arbitrary tasks but, in doing so, also introduced non-deterministic task execution and sometimes accumulation (i.e., *overload*). As a consequence, the analytical or experimental methods to estimate execution timings became less and less reliable leading to unexpected underestimations of WCET and the obsolescence of the scheduling algorithms. The first basic reaction to these unpleasant surprises was to overestimate execution deadlines even more to avoid major task accumulations and subsequent, long, overruns which could lead to system failures. Scheduling algorithms henceforth moved from purely optimization-motivated to include prevention of system failures by overrun. Previously developed algorithms were then revisited to adapt to new CPU architectures with an emphasis on getting rid of the pre-runtime estimations such as WCET altogether while still building fault-tolerant RT software. For example, the method in (Xu, 2020) adapts software and hardware redundancy to protect against both software and hardware permanent failure, while still managing overruns and ensuring recovery before and after a failure. Another example is in (Pazzaglia *et al.*, 2021) which handles the sporadic accumulation of tasks that may result from the modern architectures also without pre-runtime tasks' time estimations.

[10] Quartz clocks in normal conditions drift by about $10^{-6}\ s.s^{-1}$. Room temperature or radiation levels can severely degrade the clock drift. A typical quartz is expected to drift about 1 PPM per Celsius degree within a range of $[-20°C, 70°C]$ according to the chart at: https://www.jauch.com/blog/en/ask-the-engineer-how-temperature-sensitive-are-quartz-crystals/. This implies that the clock drift of a computer varies depending on its activity.

Design methodology of real-time systems

The research community in RT systems quickly investigated the best ways to streamline the engineering process of SIL+HIL RT systems, in order to master both economic and safety issues.

Different teams and labs in the community shared early on the architecture design of their respective RT system, emphasizing which decision was taken to better deal with their requirements. For example, (Aviz, 1969) introduces how they managed to design a Self-Testing And Repairing (STAR) computer by implementing protective redundancy at every level and how to monitor, catch, and repair its components failures. The central monitoring system is called the Test And Repair Processor (TARP). Even the TARP itself is a redundant entity; it is responsible for scanning all byte streams encoding the computations for corruption. The paper also goes into great detail about the relationship between the software and hardware redundancy protocols to make sure that, when a fault is detected, the code is rolled back to prior execution (thanks to redundant data storage), the execution is attempted on the same hardware, and switched to a redundant component if it fails again. This work is one of the first to describe in detail the special architecture of a system implementing what will later be called fail-operational systems that I already mentioned.

RT systems such as the STAR required extensive finances, expertise, and time and were more akin to artisanal masterpieces built to satisfy very specific objectives. However, according to Sorenson and Hamacher (Sorenson and Hamacher, 1975), the majority of RT systems were developed from a general purpose hardware of the time which lead to ill-proportioned systems. The authors indeed agree that computers must be the tool supporting the RT system, and not the RT system constraining the computer to a certain type of computation. That's why the authors recommend starting to design any RT system by laying out the environment's objectives first and start writing code as late as possible. This way, they hope the designers will take as few design decisions based on what their computer can or cannot do. Moreover, they argued that programmers were re-using software practices that were successful for conversational command-based programs running on consoles. Unfortunately, these programs are non-RT software. Thus, the practices were unable to deliver on the timing constraints at the level of the program of a hard RT system. In addition, despite several working RT systems and papers deliberating on the best solutions to implement individual components of an RT system, there was yet much of any tool to properly encode the control and execution loops of the hard RT systems in a chemical plant or a space shuttle. The contribution of their work was to introduce a framework to encode RT systems from a *top-down* approach, starting with the description of the environment in which the RT system would run, finishing with the tasks of the components of the RT system. This encoding was supported by a new high-level language Real-time User's Design and Implementation Language (RUDIL) dedicated to writing real-time systems and their computation centers. The authors also developed an RT system *writing system* including an emulator of the physical machines to be included in the target RT system, and a custom resource management meeting the software requirements of the RT system. Of course, the design of the resource manager followed a *bottom-up* approach to focus on start from an abstract description of the computer on which the RT system will run. This gives the designers the power to focus on the RT system and not the computer since the *writing system* would manage the low-level data. Finally, the RUDIL linker would translate the RT system written in RUDIL into operations the *writing system* can process. According to the paper, using RUDIL and the linker allows to smooth 5 layers of computer abstraction encoded in the *writing system* away. RUDIL can therefore be used to build RT systems more easily and was a step forward toward more and better RT systems.

Even if RUDIL helps describing the environment and the components of an RT system before starting to write the code, it does not give help deciding upstream how an RT system should be designed. Of course, it is doubtful that the engineers of the STAR computer did not think about its architecture before actually building it. RT systems engineers made tools to accelerate the construction of an RT system, but they had yet at that time to come up with a formal guidance on how these tools could be streamlined into an efficient process. Bennet touched on this matter in his technical report on RT systems (Bennett, 1980) when emphasizing the awareness for *software engineering* in RT systems research. To my knowledge, Gomaa made a major contribution in this area. His first paper (Gomaa, 1984) on this subject presents a software design method called Design Approach for Real-Time Systems (DARTS) based on *structured design*[11] applied first on how data moves within a target RT system. Bennet had also identified the importance of data transits and status (i.e., is it shared among several processes?) for parallel or distributed RT systems[12] that he calls *multi-programming*. Gomaa argues that tasks of the RT system are identified thanks to the loosely coupled data transformations rather than by the functional purpose of the tasks such as in the Mascot design method (Simpson and Jackson, 1979; Simpson, 1986). This allows to identify which transformations have risks to change the value of the same data should they be grouped in different tasks with overlapping execution durations. Totally data-independent tasks can then be executed in parallel without any worries for data corruption[13] which participates to alleviate the time constraint of RT systems. The DARTS method starts with the specification of the system (the set of features of the perfect RT system); then build a diagram of the state transitions of the system (how to move between features); then the data flow (the data associated to each sate transition); then, the tasks (a group of data

[11] Structured design is a system design approach that aims at decomposing the system into components.

[12] It is clear nowadays that data transit and status is absolutely critical to the integrity of distributed or parallel software whether real-time or not. But the slight advantage of non-RT software over RT ones is that there should be no time-dependent events so there is one less way to trigger processes that might act upon shared data.

[13] In fact, the conflict between multiple transformations that might be using the same data is better known under the term “data race”. Supplementary Figure A2.1 illustrates this challenge.

transformations that “make sense” conceptually in the context of the system while limiting having separated tasks with transformations on the same data); finally a task structures diagram to specify the communication schemes between tasks. In a second paper (Gomaa, 1986), Gomaa enhance DARTS in peripheral design activities such as incremental design schedule, software development environment, better characterization of the events callback sequences, and automated testing through simulation of the integration of the RT system in the target environment.

A very good example of the first 25 years of research in RT systems can be found in the modular space shuttle RT simulation system of NASA (St. John *et al.*, 1987). The goal of this paper is originally to demonstrate how this RT system can be repurposed and enhanced to build a RT simulation system for a year-round inhabited space station (which we know today has become reality). Every physical computation module is made as isolated as possible to be able to plug-in various virtual components that would modify the physics of the simulated shuttle or new hardware corresponding to instruments relevant to the mission’s objectives. The paper is also giving their answer to the problem of developing a modular architecture of the RT system modularity of the system allow support for different versions of the shuttle guaranteeing simulation time and long-term support independent of future simulation’s requirement, task distributed execution, and limited software development time. I find this paper fascinating as it gives deep insight into the benefits of carefully thinking about the architecture of a tool to make sure that that tool is useful to the best of its potential for a long time.

Moving forward, a major critic about the design methodologies based on structured design to identify the components of a system is that, in the case of an RT system, there is nothing that forces the designers to account for real-time constraints. Of course, the designers will eventually think about it, but if the system abstraction and components identification methodology included mentions of how frequently a component is used or how time-critical it is, it would help the designers to consider these as early in the development cycle as possible. That is why new frameworks such as the Hard Real-Time Hierarchical Object Oriented Design (HRT-HOOD) (Burns and Wellings, 1994) or the Time-triggered Message-triggered Object (TMO) (Kim, 1997) were developed[14]. HRT-HOOD was developed as part of a project supported by the European Space Agency and introduces object types with time constraints from which any other object must be based on. Namely, it includes *passive*,

[14] Object-Oriented Design (OOD) is a direct offspring of structured design in the field of software engineering. The purpose is to decompose a system into *functional objects* encapsulating the features of a system, as well as *data object* which mainly group variables that are usually processed together or which “make sense” in the context of the system. In practice, the separation between functional and data objects is very blurry. The “hierarchy” in HOOD implies that objects can be sub-objects of others (e.g., a car and a truck could be sub-objects of a vehicle object). In modern software engineering, this is called “inheritance” and commonly accepted to the point that the “H” never appears in the abbreviations, and we only say OOD.

active, and *protective* which describe whether an object has control over the execution of its tasks and whether it can request execution of tasks in other objects; and it includes *cyclic* and *sporadic*, which describe if the object must execute something on a regular basis or only from time to time. An interesting point brought forward by the authors is that HOOD supposedly helps the whole life cycle of a software with long-term support, maintainability, and reusability of the components across software. Provided this is true[15], it implies that when someone makes a timing analysis of a component of an RT system, the result of this analysis can be transferred. Moreover, if a component is updated, prior timing knowledge about the component should facilitate the timing analysis of the new version. Thus, HOOD could help iteratively build and refine timing heuristics about components every time they are reused in slightly different contexts. Of course, new designers would have trouble about these heuristics, but seniors would not. Kim pursued a similar approach and suggested a more unified structure for objects with his TMO concept. A TMO contains data and, as the name implied, time-triggered services as well as message-triggered services. Time-triggered services are functions that are spontaneously triggered once the clock of the RT system reaches the time associated with the service. Message-triggered services are functions that are requested by other TMOs and may make reservations to spontaneous services of their own TMO. The author argue that this object structure is capable of both RT software and non-RT software, as it is possible to simply not set any time constraints to any spontaneous service. This approach forces engineers to acknowledge the time constraints very early at the abstraction level and hence think about how to actually implement them. This would, hopefully, limit the number of iterative refinement steps of the RT system and eventually accelerate its development.

I would like to make a special mention about a mathematical method to specify and verify requirements of RT systems thanks to a theoretical framework called *duration calculus* (Ravn *et al.*, 1993). Until now I presented non-rigorous abstraction methods of systems which can be helpful to conceptualize but cannot be used to check the integrity of the system beyond the mental model of the designers. So, we can only reason about the abstractions of the RT system informally without any guarantees as there is no theory to back us up. Ravn *et al.* developed a mathematical (logic) way to encode RT systems for control purposes. Thanks to the background of logic theory, it is a formal description that do not let suffer from the ambiguity of natural language. It supports the definition of time, system states, and assumptions about the system (i.e., its features and effects) under the logic theory. The

[15] This idea has had its golden time but, as more and more complicated software was developed in the last 30 years, there is an increasing proportion of examples demonstrating that HOOD has not delivered as much as expected in this area, and no one really manage to take a piece of code from a large software and plug it into a new one without changes.

control law of the RT system is implemented as a *finite state machine*[16] on which duration calculus can be applied to deduce nontrivial state transitions under some predicates (i.e., conditions). Doing so, Kim demonstrates how to deduce the final state of the RT system with predicates simulating the failure of a component. With this approach, qualitative behavior of the RT system can be refined before any implementation.

A question about RT systems that I find very important, but which seem hardly discussed, is how often the RT system really needs to run its execution loop to satisfy its function. Ideally, we would, of course, like to always update to be as precise as possible, particularly when the RT system is controlling a nuclear plant or any other critical system. But this is obviously unreasonable because, independently of all the scheduling algorithms that I already mentioned, a computer's resources are not infinite. Therefore, can we find a rationale method to take informed design decisions about the execution frequency of the RT system? Kaul *et al.* (Kaul *et al.*, 2012) mathematically analyzed this question for the communication frequency between distributed components of an RT system. The paper derives a statistical time of when a component (i.e., *monitor*) in the RT system is left without any update because no *fresh* message has arrived. The optimization target is to minimize this time (*maximize the freshness*) while not overloading the RT system. Interestingly, the paper demonstrates that the best messaging frequency is not "always", so constantly sending messages might not help the RT computer yield any better or more accurate results. This approach is rare in RT systems research but more prolific in networks and data warehouse management research since it is absolutely critical in that field that the data is correctly handled while not overloading the servers.

To conclude this section, I would like to mention the *model-based design* from (Kopetz and Steiner, 2022d) since the name is close from the MBB I defend in this thesis. In RT systems research, the goal of model-based design is to produce an actionable model of a target physical system (typically an industrial plant) so that the RT system can be developed under the assumption that the model is accurate. If this is true, then engineers can constantly check the RT system under development against the actionable model. For example, if the RT system is supposed to control the valves of a water drainage system, then the model would include actionable valves that mimic as closely as possible the valves in the physical world on which the RT system will eventually be deployed. The goal of a model-based design is to detect and optimize the RT system as soon as possible during its development cycle and to not have to wait its deployment in the physical world to discover major issues. In that regard, the RT simulation system of NASA for the space shuttle and space station (St. John *et al.*, 1987) employs model-based design since the RT simulation is used to test the whole RT

[16] A finite state machine is a computation system defined by its list of possible states and the conditions (e.g., inputs, events, etc.) which allow to go from one state to another. For example, a lock requires a key to switch from its *locked* to *unlocked* state; we could easily complexify this by introducing a *permanently locked* if a fraudulent key is used.

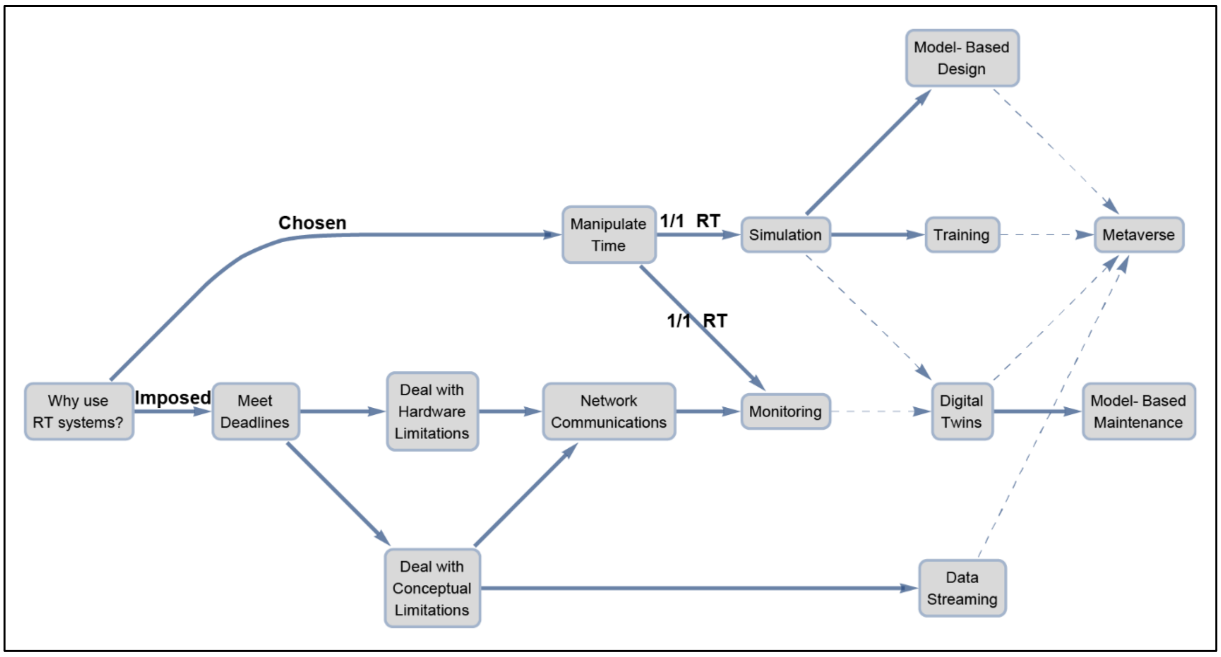


Figure 3: Why use real-time systems? Arrows translate as "used to / used for"; dashed arrows indicate a jump to new engineering fields that could not exist without RT systems. The very early reason to use RT systems was to help meet deadlines despite hardware limitations: RT technologies were more imposed than chosen. The time constraint is also a conceptual limitation in network communications and data streaming; independently of the hardware performance, these tasks must be executed under deadlines (e.g., phone call, music/video streaming, LIDAR signal in autonomous car, etc.). RT systems are also used in situations where it is not the only solution. They are chosen because their architecture allows them to manipulate time scales; this includes a one-to-one (1/1) mapping between the RT system's time and the physical time in reality. This is very practical from monitoring and simulation tasks.

system before the shuttle is sent to flight (as well as to train the astronauts). The major difference with the MBB I am defending is that the guarantees the model is correct are not based on direct observations (such as when replicating the behavior of the valves) but from the theoretical framework used to encode the model.

Applications of Real-Time Systems

In this section, I will explain *why* RT systems were chosen to begin with (see Figure 3), and not only *where* they were chosen. This will be useful to understand the reasons that brought me to consider RT systems as the technology stack to engage with MBB.

When real-time systems are imposed

There are applications where RT systems are necessary. Going back to the modern definition of an RT system, *a system is real-time if it correctly executes computation tasks before a deadline*. The leeway given to respecting the deadline is the criteria to discriminate between hard and soft RT systems. But this leeway didn't exist at the beginning; real-time systems were agreed to be *hard* and nothing else because the concept was specifically introduced to deal with time constraints. On one hand, I think the power of modern consumer grade computers tends to blindfold us and makes us forget that, in the 1960s, the computer itself was a constraint. The Apollo Guidance Computer (AGC) had a mere 2 kilobyte of RAM, 36 kilobyte of memory, 16-bits precision, and a chip running at about 2 MHz. This is at least of the order a thousand times less performant than modern smartphones and the AGC still managed to deliver and assist the astronauts navigate the shuttle in real-time until they landed on the Moon. Therefore, even putting aside the fact that, if the AGC had overrun, hard failed as a result, and likely had killed the crew, the first reason why RT systems are even relevant is because executing tasks fast enough to not let the human operators waiting was not a given. In addition to raw computer performance limitations, some applications cannot succeed without RT systems because they are conceptualized as real-time to begin with. Network communications are in this case because the goal has always been to get as close as possible to the real-time communication of humans meeting face-to-face. Even if it is machines communicating with each other, there is no reason to not process an incoming message as soon as it is received and to react to it once it has been processed. Moreover, beyond my comparison with casual talk between humans, there are cases where prompt communication over a network participates to the success of a real-time system. This is what motivated NASA to upgrade their network communication system in the first half of the 1960s in their effort to pull through the Gemini and Apollo programs (Sollaeg, 1964). Another example of application field which is conceptually dependent on real-time is data streaming. Nowadays, the major type of data streaming is the remote consumption of videos or music from on-demand platforms, An RT system is mandatory at the level of the consumer's computer to decode the incoming data stream and project a fluid video or song to a user (Altilar and Paker, 1998). Methods in real-time signal processing always require some attention because the format of the data encodings or the throughput are regularly update and ever more efficient RT systems are required for the tasks (e.g., visualization of ultrasound data in medicine (Reichl *et al.*, 2009), or 3D volumetric signal for virtual reality (Lee *et al.*, 2020)). Consequently, from the 1960s to today, there are cases where the time constraint is part of the main problem and not just a side effect of algorithmically complex programs. In these cases, RT systems are the de facto solution and *impose* themselves on the engineers.

When real-time systems are chosen

Conversely to the previous paragraph, there are cases where RT systems were willingly chosen among other possibilities. Or, at least, it does not seem like they are being imposed on the engineers. It is only a few years after Apollo that RT systems were identified to also be relevant for side-effects of being able to meet deadlines and not only because the deadlines were part of hardware or conceptual limitations. Indeed, the definition of an RT system only specifies that it must execute tasks before a deadline, but it says nothing about when this deadline should be. As a consequence, computers that consistently deliver results before a deadline of a few milliseconds or a few hours both qualify as RT system. In addition, there is no telling how these milliseconds or hours of computation time in reality influenced the mapping between the simulation time and the real time. It may very well be that the milliseconds of computation time were enough for the simulation to forecast everything that is going to happen in reality in the next few years, or the hours only managed to cover a few seconds. In the former case the computer goes much faster than reality, in the latter case it is much slower. If an RT system can go faster or slower, it can also go just as fast as reality, and this special class of RT systems was identified to be adapted for many technological challenges from the 1970s onwards. For the purpose of my discourse in this thesis, I will denominate such RT system as *1/1-RT system*[17], but, to my knowledge, this term is not used in the community.

Real-time monitoring systems

1/1-RT systems are used for the monitoring of complex infrastructures to take informed decisions about their activity and accurately control or optimize them to avoid accidents or wastes. This classically includes the monitoring of automated factories (Lin and Lee, 1989) to check on the production line and measure the economic impact of the factory. 1/1-RT monitoring systems are also used to control environmental variables such as the water quality of the effluents of a laboratory (Koopman and Yamauchi, 1990) or the pollution of lake Toba in Indonesia induced by tourism (Rahmat *et al.*, 2016). Real-time monitoring for energy production, security, and distribution is also a long-standing challenge with numerous sub-problems that benefits from a 1/1-RT system. It starts with the stability of nuclear reactors (March-Leuba and King, 1987) and finishes with the distribution of energy across a power grid. At the beginning the challenge was "simply" to be efficient and avoid wastes or blackouts at peak load times (Stahlkopf and Wilhelm, 1997) that kept growing alongside our consumption of electricity world-wide. In the early 2000s the fear of blackouts from badly managed supplies rescinded in favor of remote malicious attacks on a power grid; thus RT systems started to also monitor suspicious activity on the grid (Qi *et al.*, 2011; Srivastava *et al.*, 2018). The most recent issue for energy distribution originates from the introduction of renewable energy in the production mix of a country or super system. Unfortunately, renewable energy is depending on environmental conditions and exhibits large output variations as a result which must be balanced in real-time (Beier *et al.*, 2017; Amir *et al.*, 2022).

Real-time simulation systems

In monitoring, the utility of 1/1-RT systems reveals itself at the deployment stage, but being able to map the RT system's time to the time in reality also allows to develop simulations of systems that will behave as fast as they would in reality. Real-time simulation is useful to train humans to face situations and/or to manipulate RT systems they will encounter in the future. Examples abound of RT simulation systems for training purposes. I have already cited the replica of the space shuttle (St. John *et al.*, 1987); other very different examples include, but are not limited to, surgery training (Bro-Nielsen and Cotin, 1996; Cotin *et al.*, 1999), or simulation and visualization of organic molecules (Gandhi *et al.*, 2020). Finally, the approach

[17] 1/1-RT systems are so common today that some publications (Menghal and Laxmi, 2012) assimilate the definition of RT systems to the 1/1 mapping case and classify other systems that may go faster or slower than real-time as being *offline*. But I will not discuss this point further and keep the more general definition of an RT system.

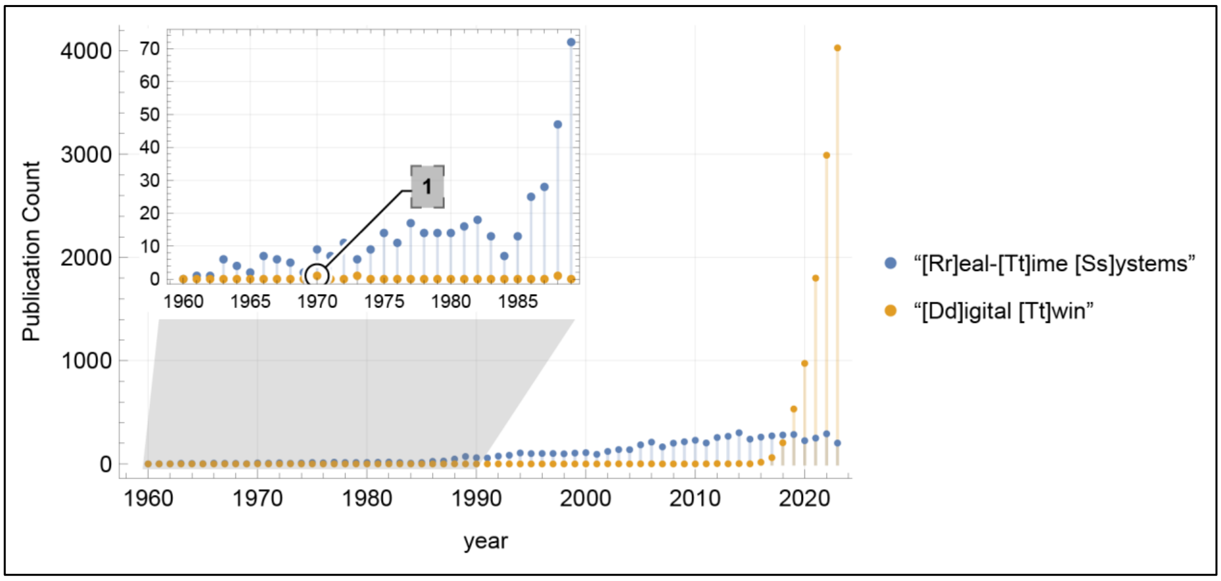


Figure 4: Publication count per year or articles that contain the words "Real-Time Systems" or "Digital Twin" (capital case or not) referenced in Google Scholar between 1960 and 2023 (last count on December 9, 2023). The first found reference of "Digital-Twin" is in 1970 for a thesis entitled "Digital Twin of Cardiovascular Systems". There were also punctual mentions in 1973 and 1987. Other than those, one has to wait 2011 for the explosion of publications on the subject. The rate of publications on the subject of "Real-Time Systems" seem stable (around 250 per year) or with a very small increase since 2010.

of model-based design that I already explained strongly relies on RT simulations because they are interactive – or close to be. The interactivity is, again, a consequence of the one-to-one mapping between the simulation time of the RT simulator and the time in reality. Interactivity – or high responsiveness – allows to rapidly test technical variations of RT systems by simulating faults (i.e., *fault injection*) (Kopetz *et al.*, 1989; Kopetz and Steiner, 2022d) even beyond the normal range of operation, at a fraction of the cost of the same tests performed in reality (Bloem and Naigus, 1988; Menghal and Laxmi, 2012). From a macro perspective, investing in a 1/1-RT simulation system is also the guarantee to have a unified design framework within the engineering team for a few years (Schiano and Silberto, 1986; Koopman and Yamauchi, 1990; Haung *et al.*, 2005) which avoids having to re-train engineers on new tools for every new RT system project. As a consequence, 1/1 RT simulation systems were employed during the development cycle of numerous products in an attempt to shorten the overall development time of the final product (Bloem and Naigus, 1988). The avionics industry is particularly fond of 1/1-RT simulation systems for these economic reasons, and developed many of them to provide reusable RT test platforms (Chelini and Farmer, 1981; Schiano and Silberto, 1986; St. John *et al.*, 1987; Bloem and Naigus, 1988). The modern video game industry is also heavily relying on 1/1-RT systems often called game engines. In fact, as I will describe later in this thesis, a game engine is much more than an RT system.

From real-time systems to digital twins

RT systems are at the origin of a new "buzz word" in engineering research since 2011 that is *Digital Twin* (see on Figure 4 the exponential explosion of associated publications). The field of avionics is, once again, a precursor of this design philosophy (Tuegel *et al.*, 2011;

Glaessgen and Stargel, 2012; Reifsnider and Majumdar, 2013) which objective is to produce an "ultrarealistic" model of each product down to its manufacturing defects once it is out of production (Tuegel *et al.*, 2011). The goal of such a model is to perform high-fidelity monitoring of individual products and to follow their evolution during their whole life cycle. This is the extension of the 1/1-RT monitoring of the factories, industrial plants, power grids, and sewer systems that I have extensively illustrated above, to mobile objects. So, from my point of view, the conceptual jump between RT systems and digital twins is almost inexistant: it is only the miniaturization of monitoring to be able to cover any object of any size. The technological jump is much bigger, however. Indeed, digital twins will only be made possible thanks to the advent of high-frequency and reliable enough sensors and their ever more seamless integration in the Internet of Things (IoT). This trend was predicted by Stankovic in 2003 when he described an RT system tailored to handle dense network of weak sensors with the goal of high frequency monitoring (Stankovic *et al.*, 2003). In addition, digital twins are personalized per product, so there must be one digital twin bound to every replica of an object. This digital twin will then be used to simulate the activity of its real counterpart before it is active in reality. In avionics, for example, the digital twin of a plane will be simulated through variations of the planned flight and, theoretically, everything about the plane will be accounted for, including the possible faults of the navigation system, luggage unintentionally moving in the cargo bay, the stress levels on every cm^2 of the airframe, and so one. These simulations will then be used to predict the components in the plane that are more likely to fail, and the pre-flight maintenance will be adjusted accordingly. Then, data will be collected during the actual flight and directly sent to the RT simulator of the digital twin to feed it with the latest information about the state of the plane; this data can be used to forecast the near future condition of the plane and warn the pilots of any incoming anomalies. Then, once the flight is completed, the post-maintenance of the plane will also be adapted depending on the status of the digital twin. Such approach will inevitably generate massive amounts of RT sensor data as well as simulation data. It pains me, but I am more ready to accept big data to support this design philosophy than I am for biology. That is because digital twins have not, until now, been associated with ludicrous applications and have the potential to increase the security of safety-critical products. Indeed, it mainly concerns planes(Tuegel *et al.*, 2011; Boeing and Bräunl, 2012; Glaessgen and Stargel, 2012), medicine (Masison *et al.*, 2021; Laubenbacher *et al.*, 2022), factories (Schluse and Rossmann, 2016; Schluse *et al.*, 2018), laboratories (Palmer *et al.*, 2021; Zhongcheng *et al.*, 2022), and others. However, the day companies will ship digital twins alongside each smartphone they produce for their clients only for marketing or to sell new services, I will vehemently protest. The takeaway message about digital twins is that they are 1/1-RT simulation systems applied on high-fidelity models, and that they were introduced to propagate the benefits that RT systems brought for model-based design to *model-based maintenance*[18].

From real-time systems to the Metaverse

Digital twins, and RT systems through them, are in turn involved in the development of "the next iteration of the Internet"[19], namely the Metaverse. The Metaverse is originally an informal concept from a science-fiction novel titled "Snow Crash" (Stephenson, 1992); it is depicted as an immersive 3D environment accessible by wearing a Virtual Reality (VR) device and, possibly, additional equipment such suits or gloves that will enhance the immersion. In the novel, the Metaverse exists as an alternative world tightly interwoven with reality impacting the whole society and its macroeconomy. 30 years later, the concept of Metaverse is still fuzzy because evolving along practices (Park and Kim, 2022) and technologies (Ning et al., 2021; Buchholz et al., 2022), and industrials and academics are not yet completely aligned on the definition of the Metaverse. Industrials more involved in manufacturing focus on engineering aspects and define the Metaverse as real-time, immersive, engineering grade with data integrity and traceability, and collaborative (Are We There Yet? A Status Check on the Industrial Metaverse, 2023). Other industrials that sell services among their products focus more on the user experience of the Metaverse through entertainment, socializing, and work (The Metaverse and How We'll Build It Together -- Connect 2021, 2021; Are We There Yet? A Status Check on the Industrial Metaverse, 2023). A qualitative meta-synthesis of various academic definitions of the Metaverse identified a set of 14 "dominant terms" at the top of which is "immersive" (Weinberger, 2022). Academia defines immersion (Ning *et al.*, 2021; Wang *et al.*, 2022; Mystakidis, 2022; Buchholz *et al.*, 2022) in the context of the Metaverse as users embodying avatars (Mystakidis, 2022; Park and Kim, 2022) in a virtual world. Avatars are a virtual representation of users that follow their gestures giving multiple users a shared sense of space, presence and time (Lee *et al.*, 2021). Immersion in the Metaverse is powered by extended reality (XR) technologies. XR includes VR, augmented reality, and mixed reality, each consisting in devices that blend (Milgram *et al.*, 1995) the real and virtual worlds to produce various degrees of immersive experiences. The virtual worlds displayed by XR devices are rendered in real-time thanks to RT simulation software developed using game engines. This implies

[18] To my knowledge, this word is not used by the community; I made it up for the purpose of the comparison with model-based design.

[19] The CEO of Meta, Mark Zuckerberg is a great defender of the idea that the Metaverse will replace the Internet. Here is an extract of his speech during a keynote he gave (The Metaverse and How We'll Build It Together -- Connect 2021, 2021)*: "We've gone from desktop, to web, to phones. From text, to photos, to video. But this isn't the end of the line. The next platform and medium will be more immersive, and embodied internet where you're in the experience, not just looking at it. And we call this, the Metaverse. You'll be able to do almost anything you can imagine, get together with friends and family, work, learn, play, shop, create, as well as entirely new categories that don't really fit how we think about computers or phones today. […] We believe the Metaverse will be the successor to the mobile internet."*

that everything in a Metaverse is, in fact, encapsulated in a real-time process and that is why it can act as the perfect interface for digital twins. The couple *digital twin + Metaverse* is expected to become the new standard of 1/1-RT monitoring systems. Indeed, instead of having a few monitors displaying limited amount of information at a time about a factory, the factory itself will be entirely reproduced in 3D and fully animated to match the real-time condition of every inch of the production lines. Of course, this does not mean that aggregated statistics will disappear, but it is a step forward to benefit from immersion for RT simulation and monitoring. In conclusion, the Metaverse is, with digital twins, a large concept that could not exist without the side effects of RT systems.

In all the examples of applications of RT systems I gave in this section, at least half of them is not used because they are designed to meet deadlines, but because the system to adapt to deadlines allows them to scale the map between the execution time of its tasks and the physical time in reality. Hence, **RT technologies were not used as per what the definition precluded them to do, but they were chosen nonetheless for what the definition implied.** Now that I have explained why RT systems have been used since their creation in the 1960s, I will focus in the next section on how I am suggesting using them for MBB.

III. Characterizing Soft Real-Time Systems for Model-Based Biology

In this section I will motivate the value of soft RT systems to produce tools to build biological axioms informally, interactively, and intuitively to spark the interest of biologists for Model-Based Biology (MBB). Before that, I will summarize the origin and meaning of MBB that I am defending in this work. I opened this thesis dissertation explaining how, in my opinion, biology, and the subfield of systems, biology was obsessed with data. I criticized this obsession arguing first that induction can only yield results at the expense of increasingly large resources consumption and is, therefore, unsustainable, and incompatible with the modern state of our planet. I also suggested, through the words of Sydney Brenner's critics of systems biology, that understanding the mechanisms of biological systems from data alone is intractable and that reverse engineering is likely to fail for biology. I noted that the broad application of machine learning techniques will indeed support the reverse engineering of biological systems, but that will intensify untargeted data acquisition which brings us back to the issue of sustainability. Putting aside these quantitative objections, I pointed out that obsessing over data is blindfolding biologists to explore the solution space of biology through "how-questions" alone instead of also asking "what-questions". As a consequence, biologists are focusing on the biological systems they can observe on Earth which are but a few solutions to generic biological problems. So, answering "how-questions" will only give biologists access to points in the solution space, whereas investigating the biological problem itself through "what-questions" is more likely to inform them on the shape of the solution space. Therefore, answering "what-questions" would have a bigger impact on our understanding of biology than answering a collection of "how-questions". However, as answers to "what-questions" are not necessarily observable on Earth, the problem then arose to figure out how to investigate general biological problems in the absence of observations and I suggested using MBB. MBB as the **definition of symbols which can be combined to encode and deduce biological knowledge.** The purpose of MBB is to explore biology thanks to an initial set of statements or axioms that defines a frame from which biologists can deduce further statements and formal solutions to general biological problems captured in the frame. Of course, the validity of the solutions is guaranteed only within the bounds of the axioms. Then, targeted experiments can be used to confront the derived statements with the biological solutions observable on Earth. Hence, MBB first requires a theoretical framework to encode biological problems, models, and solutions, which can efficiently interface between formal statements and experiment data. Unfortunately, there has been a few attempts to produce such a framework in the past century with little reach in the biology community. That is why, I rather aimed in my doctorate research at producing tools to build axioms informally, interactively, and intuitively in biology and see what kind of biological solutions emerge.

Why Real-Time Technologies for Model-Based Biology?

I decided to use RT technologies over offline simulation methods classically used in systems biology (see again section *Modeling with software* page 23 for examples) because I noted many similarities between the modern applications of RT systems and the goal I am trying to reach. I am looking for a way to easily manipulate symbols encoding heterogeneous biological data, and to quickly receive feedback on how changes on the input symbols influence the behavior of simulation symbols through their outputs. I also want to easily manipulate the time variable in order to investigate the different time scales under which biological systems are established in environment niches. Models of biological systems are also getting increasingly complex, so the architecture must be scalable to afford simulation on such large systems.

The critical deciding factor to use RT systems is obviously the need to manipulate time scales. I have indeed extensively discussed the applications of 1/1-RT systems for simulation and monitoring purposes in the previous section. Hence, I do not plan to use RT systems because they can meet deadlines but because they can manipulate time scales and provide RT interactive user interfaces. In addition, research on RT systems has a strong background in the development of heterogeneous systems, both physical and digital which match the trend in systems biology toward highly heterogeneous models with static data from past experiments, real-time data streams from ongoing automated experiments, data streams from ongoing simulations, and events from user interactions. Support for this heterogeneity is adapted to the definition of various interactive input symbols. Moreover, RT systems are the foundation for digital twins and the Metaverse which are highly complex systems, so I believe they can manage the complexity level a biological system. In addition, RT systems are often used to perform model-based design of other RT systems or themselves. Even if I have already explained that the model-based design in the field of RT systems engineering is slightly different from the model-based biology I am suggesting, I believe the benefits of the former regarding the early detection of issues and hypothesis testing still apply to the later. Consequently, I expect RT systems to help shorten the iterative development of viable models of biological systems and allow a fast generation of model alternatives in the solution space of a biological system instead of focusing on a handful. And last but not least, it is ignored in every modeling approach and software that I am aware of, that a living biological system is a real-time system. Indeed, it receives input from the outside world, process this input, and adapt its internal machinery accordingly. Moreover, it does so under soft deadlines which a prolonged overrun (the system cannot react fast enough) leads to a critical failure. The reasons why state-of-the-art modeling software do not account for this fact is because no biologist has had any use for interactive real-time simulations, and the sampling technology to receive inputs from cells as fast as we monitor a nuclear plant or car engine is not yet available. But I specifically wish for interactive simulations, and it is only a matter of time before autonomous robotic experiment platform can reliably deliver measurement streams.

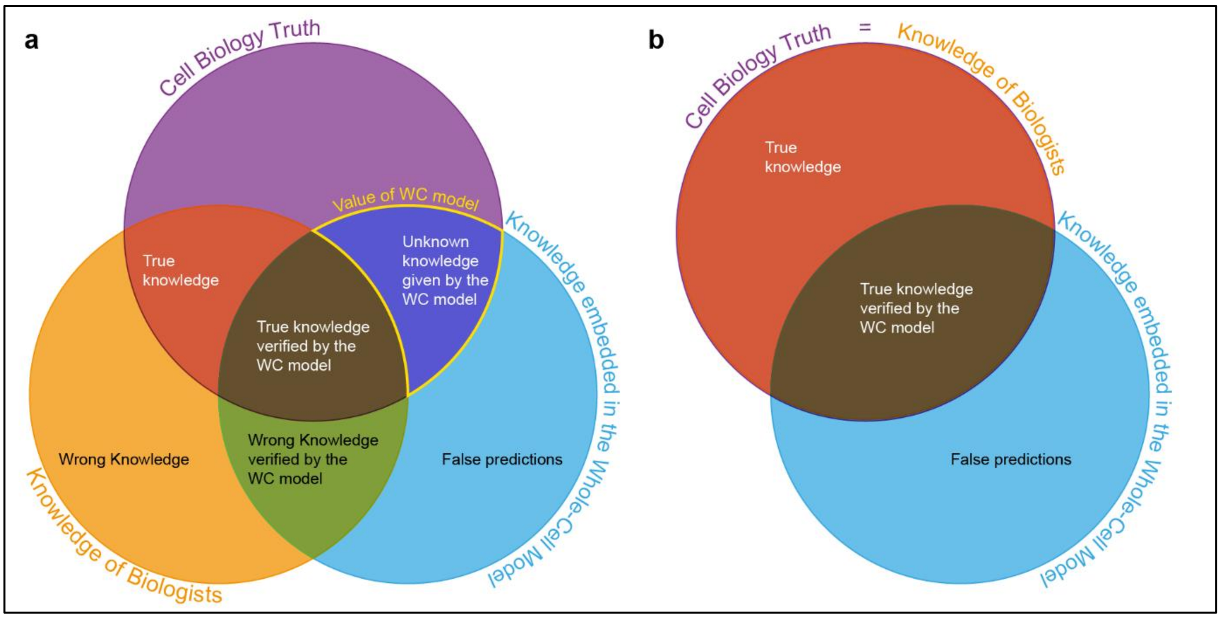

a
Cell Biology Truth
Value of WC model
True knowledge
Unknown knowledge given by the WC model
True knowledge verified by the WC model
Wrong Knowledge
Wrong Knowledge verified by the WC model
False predictions
Knowledge of Biologists
Knowledge embedded in the Whole-Cell Model
b
Cell Biology Truth = Knowledge of Biologists
True knowledge
True knowledge verified by the WC model
False predictions
Knowledge embedded in the Whole-Cell Model

Figure 5: Nature of the intersections between the knowledge of biologists, the knowledge embedded in a whole-cell model, and the truth about cell biology. a) Biologists do not know the whole truth about cells, so if the whole-cell model is ill-formed, there will be biology to learn from it. b) In the limit case where biologists have learned everything about cell biology, models can only tell us things we already know or make errors because simulations of the model cannot be exactly what we know. So, the more knowledge we acquire about cell biology, the less whole-cell model will be useful.

Imagining a Perfect Real-Time System for Whole-Cell Modeling

In this section I imagine how real-time systems would help achieve whole-cell (WC) models to illustrate how various levels of integration of RT systems in the design and life cycle of WC models would be useful for systems biology as a whole.

Background of Whole-Cell Modeling

An ideal WC model is a one-to-one *in silico* description of everything composing a real cell. It is preferred to be bottom-up, starting from the genotype of a cell, it accurately outputs its phenotype through the myriad of interactions and regulation of its molecular machinery. It is equally the dream of holists and the nightmare of reductionists. WC models result from the integration of cellular phenomena and components that are classically studied in isolation. The goal of this integration is to simulate the emergence of cellular behavior that would otherwise be impossible to get when modeling isolated parts. WC modeling has been in the mind of a few since the very beginning of systems biology, notably with the E-Cell project (Tomita *et al.*, 1999) (see section *Modeling with software*, page 23). Rich of his experience with the E-Cell project, Tomita deemed WC simulation to be one of the "grand challenge of the 21st century" (Tomita, 2001). This statement has proven to be true as WC models are still rare more than 20 years later. Of course, there has been progress since the E-Cell project (127 genes of *Mycoplasma genitalium*) with the completion of a WC model of *Mycoplasma genitalium* which was including a complete genome (525 genes) (Karr *et al.*, 2012). More recently a partial WC model of *Escherichia coli* including 1214 genes (43% of the well-annotated genes) was published (Macklin *et al.*, 2020) followed by its integration in simulations of bacterial colonies (Skalnik *et al.*, 2023). Concurrently, alternative WC models targeting minimal viable cells (but bigger than the E-Cell) were also published (Rees-Garbutt *et al.*, 2020; Thornburg *et al.*, 2022). At every step, all these models are the state-of-the-art of what modelers in systems biology can do. They require extensive expertise to build and the software platform to simulate them are much more involved than the average software otherwise published in the community of systems biology. In fact, this level of complexity is one of the reasons why there are so few groups working today on developing technologies to support WC modeling. The investment to build a decent model from scratch is frightening and hinders the democratization of WC models.

WC models are advertised more for their predictive capacities, than for their qualitative behaviors or what they tell us about fundamental functions in life. Indeed, WC model are pushed as platforms for model-based design of biological organisms in synthetic biology (Rees-Garbutt *et al.*, 2020), or for hypothesis generation (Karr *et al.*, 2012; Macklin *et al.*, 2020; Thornburg *et al.*, 2022; Skalnik *et al.*, 2023). The success of these applications is highly dependent on how well a WC model will simulate everything happening in the real cell it simulates. In practice, experimental biologists expect WC models to deliver a perfect one-to-one match on every element of the model. 100% and nothing less. This is impossible today, however, and existing WC models successfully increase our body of biological knowledge only because it is poor enough in the first place (see the highlighted yellow area in Figure 5.a). Even if a WC model does not cover everything biologists know, it happens that what was encoded inside can reach biological knowledge that biologists have yet to acquire, and it is this part that makes WC models currently useful for hypothesis generation.

Unfortunately, WC models are data-based. As such, they are inheriting the faults of reverse engineering voiced by Sydney Brenner and which I narrowed down to the intractability of big data approaches for biology at the beginning of this thesis. Accordingly, raising the predictive capacity of WC models will get increasingly more difficult and costly as it improves over experiment tests. Indeed, the more complex a system of interest, the more potential interactions, and the more experiments required to test every hypothesis. Suppose that we have variables $\forall i \in \mathbb{N}, s_i = \{0,1\}$ and a set of such variables $\forall n \in \mathbb{N}, S^n = \{s_1, \ldots, s_n\}$, that represents 2^n configurations that must be tested. So, every time a WC models will be used to generate hypothesis within that domain, the number of tests will increase by a factor of 2 (i.e., 2^{n+1}, 2^{n+2}, 2^{n+3},...). This is a gross worst-case scenario where all variables have an effect on each other. Thankfully for experimentalists, outside knowledge helps them put constraints and can drastically reduce the number of experiments. Nonetheless, data-based WC models are within a self-reinforcing loop to more data to encode more complex phenomena, that will generate new hypotheses, which will require more data than the previous batch to verify, and so on. Consequently, we will eventually reach a maximum rate of hypothesis testing because of the colossal number of mandatory experiments and data acquisition. Moreover, as biological systems are not closed systems but open-ended because of evolution, the rate of hypothesis testing must be high enough to allow us to catch up, and then keep up, with the new states of biological systems. Only then may we hope to reach a

100% match between the body of biological knowledge and what a WC model can tell us about a cell.

In the event this happens, however, the limits of using empirical and unformal simulation methods will kick in. For example, the complete model of *Mycoplasma genitalium* from Karr *et al.* integrates 28 submodules. This may seem relatively little but there was (and there still is) a lack of formally established techniques to simulate that many modules and the hundreds or thousands of equations they contain. Although the exact number of equations is not given for this model of 525 genes of *Mycoplasma genitalium*, the 1214 genes of *Escherichia coli* (Macklin *et al.*, 2020) is based on "19,000 parameter values […] a system of over 10,000 mathematical equations […]". The solution chosen at the time, and which is still used today (Macklin *et al.*, 2020; Agmon *et al.*, 2022; Skalnik *et al.*, 2023) is to approximate that the modules are independent over small time scales (e.g., 1 sec) such that we can sequentially integrate the modules over this short duration. This approximation is good enough that both models successfully helped uncover new biological knowledge, but one must keep in mind that we have no way to investigate how much the approximation impacts the accuracy of the results. Other existing methods of WC simulation such as spatial lattices (Thornburg *et al.*, 2022) or the particle simulations with millions of agents[20] are also empirical to some degree. We know they work because the simulations they generate has helped biologists, but there is no way to know how well they perform in theory or how they rank between each other. This lack of formal proof is a conceptual limit of modern WC simulation technologies and there are arguably very little resources invested in finding solutions for it. If the biological knowledge increases but the technology to run WC models hits a glass ceiling, we will reach a point where the WCM will only predict things we already know or make errors (see Figure 5.b). WC simulation methods must then improve concurrently to the accumulation of knowledge or, in the absence of perfect simulation results, at least be coupled with a method to evaluate the error made by the WC model without having to perform experiments to find out.

Perfecting WC models is then contingent on technologies that can lower cost of development, increase the hypothesis testing rate, and accommodate formal simulations or self-analysis. Integrating WC model design and simulation within RT systems can help with these three at different levels depending on how deep the integration is.

Integration level 1: predictive SIL with RT interactive WC simulations asynchronous to experiments.

This is the entry level of integration focusing on adapting the WC models' simulation algorithm to the real-time paradigm. Experiments are performed independently of the WC simulations; it can be concurrent to the simulations, but the experiments measurements are not periodically fed to the RT simulations in a way that the RT simulations cannot

[20] Unpublished system by Dr. Kazunari Kaizu at the Laboratory for Biologically Inspired Computing, RIKEN BDR.

periodically optimize its parameters based on the data trace. However, being real-time, WC simulations become interactive both during the design phase and the hypothesis generation phase. Both phases will likely be interwoven at some point similarly to how early integration of RT simulation in the engineering of RT systems led to the identification of shortcomings of the said system (see section on *Design methodology of real-time systems*, page 31). So, early simulations will result in the update of the model. Compared to classic WC simulations, the modeler will be in complete control of the time scale. He will be able to play, pause, resume, go forward, go backward to leverage the manual analysis of the characteristics of the model. Furthermore, the models can be updated while they are running, and experimental data can be manually added and compared to the simulation while the model keeps running. Automated real-time analysis can also be implemented to help detect anormal signals and support the identification of theoretical issues in the structure of the model. This level of integration of WC models in a RT system helps lower their cost of development by accelerating the design cycle on the side of the model and provide tools for self-analysis.

Integration level 2: generative SIL with RT interactive WC simulations with no experiments

This level of integration is identical to the previous one, but the purpose of the integration is different. It is not to provide a simulation framework where biologists can easily compare the prediction of the simulations with experiments and adjust the structure of the model accordingly. Rather it is to explore the biological solution space, it is to provide an interactive simulation framework where biologists can proceed by trial and error, ask more general questions than usual, and think outside of the biology they can observe on earth. This is a tool to investigate “what-questions”. What are the conditions or emergence of information storing in cells? What is a minimal cellular system? What is the drive of cellular evolution? What are the abiotic conditions leading the systematic emergence of cellular systems? And so on. As such, the modelers will probably manipulate more abstract objects in the model space and value the comparison of alternatives over the optimization of one solution. Once again, the benefits of using RT systems over classic offline simulations is the interactivity to accelerate the exploration of the solution space and to quickly get feedback about what is plausible and what is not. It is also a perfect playground to analyze the implications of various WC simulation techniques and to develop more formal methods in which computation errors are analytically defined or very easily measurable.

Integration level 3: predictive SIL with RT interactive WC simulations and HIL synchronized experiments.

This level of integration inherits everything from level one, but the WC simulation receives scheduled periodic measurements from autonomous experiment hardware. The received data can be used in at least two ways. In the first case, it is to periodically assess the quality of the WC simulations relative to the output of the experiments. This can be done with the intent to optimize the design of the WC model or to give feedback on hypotheses generated before the start of the experiment. In the latter case, the modelers already have some confidence in the predictive capacity of their model. In the second case, the purpose is to monitor cells and to update their digital twin encoded as WC models. Monitoring can then be subdivided into two categories. We could have a digital twin of an archetype cell of a species that receives the aggregated data from different cells. Or, we could have a digital twin of one cell within the population. In both cases the modelers have high confidence in the predictive capacity of their WC model. In addition, the latter case is yet very challenging as the state-of-the-art single cell sequencing techniques are destructive, but the next generation should solve this problem (Tang, 2022). This level of integration is the equivalent to monitoring systems for factories or nuclear plants in the field of industrial engineering (see *Real-time monitoring systems*, page 38). This level of integration is the one that will increase the rate of hypothesis testing, allowing experiments to be conducted at any time of the day, all year round. It is the only solution to hope producing the amount of data that will be necessary to heighten and sustain the predictive capacity of WC models.

I will keep referring to these three levels of integration, although the target is not necessarily WC models.

Transferring Real-Time Technologies to Model-Based Biology

I have argued in the past few sections why RT systems could be useful for model-based biology, now I will discuss the conceptual and technical challenges that pertains to RT systems and that I will have to deal with in order to reap the benefits of RT systems.

I am first looking for an RT system that can help quickly iterate on the design of models of biological systems and which remains operatable even as the size of models grow. In the frame of this thesis, I focus on the levels of integration 1 and 2 presented in the previous section. That is to say, I will focus on SIL RT systems for predictive and generative purposes. I am personally more interested in the latter because this is what my definition of model-based biology is about, but I cannot ignore the appetence of biologists for predictive models. Moreover, satisfying both do not put so much stress on the architecture as we will see in later sections. The third level of integration of RT systems with HIL is much more demanding to implement and this thesis do not contain results about it because there are already enough to say about the first two levels. Regardless, I will keep discussing it from a theoretical perspective when relevant to not gloss over the future challenges.

In this thesis, I do not want to consider the problematic of the design of my RT systems only under the lens of the question “Does it output the correct result?”[21]. Unfortunately, this is usually happening for many software programs in scientific fields aside from computer science. In fact, the pressure scientists are facing to solve their problems at hand leads them to disregard how well they are using the tools from other fields that are helping them. I believe this issue is not as frequent when the tools are hardware. I think the grounding of

[21] Although it is, of course, mandatory to answer this question.

hardware in reality raises the users' awareness of the costs involved in case of misuse. For example, bad management of reactants can have dire effects on the number of experiments one can conduct. Or wrongly focusing a microscope is not really an option. Whereas the immateriality of software and the growing facility with which one can code makes users believe that software design is without consequences, and that what truly matters is only to get the correct computation result. Behaving this way is no different from using computers and algorithms only as fancy calculators. This might have been true until the 90s when consumer grade computers had a straightforward architecture. But modern computers' architecture got more elaborated as a result of optimizing their performance without increasing the frequency of the CPU. Consequently, calculating something as simple as a sum can be at least 10 times faster if you write code adapted to the computer's architecture[22]. Beyond these technological aspects, the effect of the design of a scientific software program on its everyday ease of use is also largely ignored. Very little care is given to its User Interface (UI) and User eXperience (UX). For hardware, it would be unimaginable to not strive for the best UI and UX possible because it is directly linked to the survivability of the product on the market. For example, a microscope which focus ring or controls for the position of the plate would be inaccessible to the user without taking his eyes of the binoculars would be doomed. For consumer grade software, UI and UX mostly correspond to what users can see on a monitor and is also a major concern for software companies. But for scientific software I argue that UI and UX concerns spread deeper into the architecture. In effect, scientific software must be reusable for the community or, at least, it must be explorable. It will hardly be possible for any scientific software which architecture was not carefully questioned during its design step. This idea that the design of a software matters not only to ensure the correctness of the computation, but also how well one can use it, maintain it, and extend it is acknowledged as well in works I have already cited. (St. John *et al.*, 1987; Masison *et al.*, 2021; Agmon *et al.*, 2022).

That is why, I believe more consideration about the design step of scientific software is required to discuss how efficiently the architecture of a scientific software powers scientific novelty. What are the trade-offs users should be aware of when they decide to do task T with software with architecture A instead of software with architecture B? This is even more true in cases such as mine where RT systems are far from the mainstream modelling and simulation technologies in systems biology. I believe it is critical to evaluate and communicate about whether the challenges contingent to the design of RT systems in heavy engineering fields are all transferred when designing RT systems in systems biology. Do all design constraints stay relevant in our field? Are there additional ones? For this purpose, I

[22] It is indeed very simple to observe. Using the same programming language, one can write code that will be better fitted to the architecture of the computer and be about 10 times faster than different version. Moreover, the fast code is as simple to write as the slow code. And I am not talking about using multi-cores vs. single core; in this case, we can easily reach x100 speed increase (if not x1000) to compute a sum.

will use the following citation from the paper describing the RT system by NASA for the space shuttle and the space station (St. John *et al.*, 1987):

> *"But how should the simulation be structured in order to support a wide variety of vehicles and subsystems in fully integrated configurations? How should the computer systems be configured such that simulation uptime is maximized and future study requirements are independent of hardware configuration? How should the simulation be distributed across the computer facility to minimize software development time, to produce a high degree of confidence during integration, and to allow for straightforward operation?"*

These questions all focus on delivering the best tool possible to improve the long-term usage of the tool and the quality of the output.

"But how should the simulation be structured in order to support a wide variety of vehicles and subsystems in fully integrated configurations?"

Translating this question to my case involves replacing "*vehicles and subsystems*" with "*biological systems"*. Then the question evidently asks about the versatility of the simulation system and the capacity to run simulations for various encodings of biological systems. I discussed already two major numeric simulation methods in systems biology in sections *Modeling with differential equations* and *Modeling with stochastic events* pages 18 and 20 respectively. ODEs and SSs are more adapted to subsets of biological targets or systems. In addition, the input data for simulations of biological systems is heterogeneous with, for example, static tables of float numbers for parameters or markup files encoding models such as SBML (Keating *et al.*, 2020). Therefore, it is worth considering an architecture which allows to easily add or remove, say, modules, that may encapsulate functions or transformations on other modules which, themselves, encapsulate functions or raw data. Then the versatility of the simulation system would emerge and be guaranteed by the number of possible combinations of modules. I followed this idea of modules for both the RT systems I implemented in the course of my PhD research.

"How should the computer systems be configured such that simulation uptime is maximized [...]?"

This question is about the requirements of the RT system such that implementation is fault-tolerant, and a simulation continues to run for as long as possible without errors. I believe the relevancy of this question highly depends on the level of integration of the target RT systems.

If it stays at levels 1 or 2 for predictive or generative SIL, then I believe the requirements for fault-tolerant architecture is low. Of course, it is always something we can wish for which will improve the reliability and reusability of the system over its life cycle. But, for one thing, RT systems for systems biology have soft real-time constraints. Indeed, for my objective in this thesis, I chose RT technologies for the interactivity they enable as in data streaming or video games and not because I needed a system that could consistently meet deadlines. I believe there will be no damage impaired in case of overrun. The reason we might want to include fault-tolerant systems is to enhance the reliability of RT simulations results with protective redundancy of data to protect against failure of the electric power of the computer, for example. We might also want to implement a checkpoint system to roll back the computation of a task in case a self-check system detected an anomaly such as excessive numeric error. In the case of level 3 of the integration of an RT system, fault-tolerance is much more significant. Indeed, as level 3 includes HIL, the real-time constraint gets stricter.

Level 3 RT systems must guarantee that the simulation will not be lagging and stay synchronized with the physical time in reality (or faster if we consider forecasting). Overruns could not be as critical as in the case of a nuclear plant, but it might imply financial loss for any downtime usage of the autonomous robot performing the experiments. Truthfully, the likelihood of overrun with modern RT systems (like the ones in this thesis) delaying modern robots is low because the former is orders of magnitude faster than the latter unless the simulation is incredibly complex (it may be the case for lattice-based or agent-based WC simulations). As I focused on levels 1 and 2 in this thesis, the need for fault-tolerant methods was low. Moreover, it would have been very time consuming, so I decided to focus on other aspects, and I did not implement fault-tolerance.

"How should the computer systems be configured such that [...] future study requirements are independent of hardware configuration?"

This question is about the link between the hardware and the software. It asks two implicit questions: i) Will the hardware running the simulation change and how does it impact the tasks scheduling, execution, and the aptitude to meet the deadlines? ii) Will the hardware external to the RT simulation computer likely change and impact the communication channels or network protocols between the two?

Question i) is relevant for the SIL part of all three levels. I already mentioned on several occasions that the evolution of computer hardware had severely impacted the design process and final architecture of RT systems (e.g., switch from single core to multi-cores CPU). So, preparing for the case where the nature of the hardware might change even if it was not the intention of the designers is not uncalled for. Moreover, even though it might be branded as niches applications, it is not impossible to wish to run an RT simulation on a low-power computer. It could simply be in an effort toward green computing to save energy for the sake of fighting against climate change. But it could also be for the purpose of a mission where the energy supply is strictly regulated. A possible example is the International Space Station. It is well-known that biological experiments are performed on the ISS to study the adaptation of plants or microbes to living in space. One of these experiments may eventually involve an RT simulation system to help the astronauts to pre-analyze the samples before they are sent back on Earth. Or, alternatively, a (level 3) RT system might be installed in the ISS to automatize the biological experiments and free the time of astronauts for different tasks. However, a low-power computer also means that the frequency and computation power will be decreased compared to conventional hardware. Such a decrease will inevitably impact the ability of the RT system to meet deadlines and must therefore be carefully considered in the design steps: what is the range of expected simulation hardware?

Question ii) is mainly relevant for level 3 and in the case that simulation hardware is distributed. It is about the modularity of the hardware in general. Since hardware is never perpetual and the field of autonomous robots for biological experiments is moving fast, a robot might be superseded every 3-5 years. It is true that robots cost tens of thousands of dollars and it is unlikely that research labs have the funds to renew their fleet of robots this frequently (but companies might). Ignoring technological obsolescence, a laboratory with several autonomous systems specializing in different types of experiments might want to connect the RT simulation systems to any of the robots. Therefore, whatever the cause, if the hardware sending periodic data to the RT simulation computer changes, it is possible that the nature of the data or the communication protocol changes with it. In the case of one of the RT systems I implemented during my research, the simulation hardware has limited computation power compared which prevents it to do much more than what it is built for. Hence, the RT system was coupled with an external computation unit to delegate some work.

"How should the simulation be distributed across the computer facility to minimize software development time […]?"

I must admit that I could not fully understand the meaning of this question. I could not figure out how the distribution of the simulation on the computer facility influences software development time. In particular, I am unsure of what the author meant by "computer facility". Is it simulation hardware? Does it include human resources? In the former case, I do not understand how distributing the simulation will minimize the development time. Maybe because distribution allows us to run more tests and figure out bugs and system faults. In the latter case the link between manpower and simulation distribution puzzles me. Thus, I could not translate this question to my case.

"How should the simulation be distributed across the computer facility […], to produce a high degree of confidence during integration […]?"

This question is about the possible loss of the integrity of data when it transits over a computing system. Scientific research and engineering both undergo strict constraint regarding the accuracy of computation and simulation results. For science, accuracy raises the usefulness of experiments and increases the likelihood of enriching a body of knowledge. For engineering, it is to improve the faculty to take informed decisions regarding design problems. This question is linked to how easy it is to verify or validate computations in a simulation and eventually to how well the integration methodologies are supported by a theoretical framework. If we can design an RT system based on simulation techniques that can be formally analyzed, then we can hope to compute a degree of certainty in the simulation's results. Formalization can happen at different points in an RT system. There is the input of data, the computation itself, and the output of the simulation.

For the input, I already discussed the difficulties associated with the clock drift leading to possible difficulties to restore the original order of occurrence of events which were timecoded with two different clocks. In the case of RT systems for biology, I think this problem might occur for level 1 and 2 of integration in the situation where different modules of a model are solved on different CPU and, therefore, have different clocks. In that case, timecoded events on one module might have trouble to be exactly synchronized on another. I do not know the details of the distribution scheme, but I would expect this might happen in the WC colony model (Skalnik *et al.*, 2023) because the design paper of Vivarium (Agmon *et al.*, 2022) mentions heavily relying on CPU distribution for the simulation. Without hardware distribution, the synchronization problem for input data might be emulated when submodules are approximated to be time independent over some time scales such as for the WC model of *M. genitalium* and *E. coli* (Karr *et al.*, 2012; Macklin *et al.*, 2020). In such case, if an event is simulated at the beginning of the time step, it will only be accounted for by the other modules at the beginning of the next time step. Of course, in the case of the WC models mentioned above, the modules were separated in a way to limit this situation, if not to completely eradicate it. But one must then be aware that the design pressure relieved from designing an RT system relying on short time-step independence is, in fact, traded-off to more pressure on the design of the model. It might also happen in the case of level 3 integration if we consider an experimental setup where several autonomous systems periodically send measure data to the RT simulation or monitoring system. However, with modern sampling technology in biology, the clock times between the two autonomous robots must be very desynchronized for a time order error to occur. Indeed, biochemical processes are mainly at the order of minutes or above and we can expect clock drift for embedded quartz clocks in normal conditions is about $10^{-6}\ s.s^{-1}$. So, unless the robotic system is recording at frequency higher than the total clock drift over the course of the experiment, time integrity is likely preserved. High frequency sampling experiments such as measuring the current propagation in cells of tissues might be problematic, however.

For the computation itself, loss of data integrity might happen because of bounded numeric precision and float number calculations. The theory to measure calculation errors has been extensively studied for ODE numeric integration. Using ODEs as the computation formalism of a model therefore provides ways to take informed decisions about simulation results. Modern ODE solvers often rely on the computation of the error to adapt the size of the integration time step of the system[23]. This approach allows to integrate over large time steps when the error is small and to integrate over small time steps when the error is large. The former typically happens when the numeric solution is not varying a lot over time and, conversely, the latter happens when the numeric solution abruptly changes values. In biology, the former happens over steady or quasi-steady states, while the latter may happen after a regulation factor has been introduced in the system resulting in the activation or

[23] The theory is briefly described in Appendix A1.

inhibition of a chemical species. To my knowledge, equivalent techniques to measure the output error of SSs do not exist. However, the quality of the randomness of the *pseudo–Random* Number Generator (RNG) used as the basis of the SSs can be measured. By now, some RNG are known to be of poor quality and vice-versa. Whether for ODEs or SSs, increasing the accuracy of the computations involves sacrificing some computation time and it is always a trade-off between executing the tasks fast enough to satisfy the deadline and having good enough simulations.

As for the integrity of the output data of RT simulations, I am talking about the ability to replay or analyze *a posteriori* the simulations with the assurance that every step of the simulation is clearly distinguishable from others and that no recorded data is corrupted. As I already mentioned, this integrity is often achieved thanks to protective redundancy of the hardware that stores the data: several copies of the simulation output is saved on different devices to reduce the probability of corrupted data. This approach appears to me just as relevant when integrating RT systems in biology. I will add, however, that independently of making sure the data is correctly logged, there is a need for tools to analyze the trace of the data. Indeed, in the case of a research activity, building a model is an iterative process and every step of this process might be discussed among pears at any time. Therefore, there is a need for non-destructive, yet, iterative, recording format of the data of models being designed.

In the case of the RT systems I developed, I faced the need to adapt existing off-line algorithms to enable input integrity of data within the simulation. Namely, the correct detection and timely execution of events in all different modules. I selected a popular variable time-step size ODE solver to integrate ODE systems, and I studied how to include traceability of the output data of my RT systems.

"How should the simulation be distributed across the computer facility [...] to allow for straightforward operation?"

In my understanding, this question is the complement of the first question on the modularity of the simulation software for the sake of better simulations. Here, it is about the modularity of the simulation for the sake of better usage. This question is the expression (37 years ago) of my intention to also consider the influence of the architecture of my RT systems on the UI and UX of the system. In the original paper, the answer to this question involves the architecture of the SIL part but also the HIL part as the RT system of the space shuttle is a physical replica of the actual shuttle. According to the paper, the architecture of the shuttle was so easily modular that testing alternative models of the airframe of the shuttle for the flight simulation or the new on-board radio was only a matter of loading definition files or plugging off and on a few cables (I give a bit more details in section *Design methodology of real-time systems* page 31). In this context, a good architecture design drives the development cycle of the product. Alternative versions of the hardware and software can be seamlessly installed. This will power the model-based approach where designers can try models early on. Overall, an architecture that promotes modularity speeds up production, because we can easily perform tests and start manipulations early in the development cycle.

For levels 1 and 2 of the integration of RT systems for biology, this matter corresponds to the seamless update of the model encodings or parameter values. Currently, models of biological systems are manipulated exclusively via files (e.g., SBML); hence, when a modeler has an idea and wishes to modify the structure of the model, he must adapt the code inside the file. Sometimes, the Graphic User Interface (GUI) of the software performing the simulations permits to update parameters, reactions, and other concepts encoded in the file. But the modifications are rarely accounted for by a simulation unless it is stopped and started back at the beginning. Instead, if we consider the simulation of a WC model, users could add or remove cellular compartments, drop a carbon source in the simulation environment, knockout a gene, induce a signal cascade, and so on. The purpose is to design the RT system in such a way that users can test anything they want while the simulation is still running.

In the case of level 3, I believe this question is linked to the variability of the hardware I already discussed. The challenge is slightly different however as, here, the problem is to willingly test various versions and not to target some range of hardware. In biology, this would be more akin to testing various experimental protocols which might result in a change of data input in the RT simulation system, even if it is formatted the same way as before. For example, it could be testing which protocol is more adapted between counting a bacterial population by measuring the absorbance or by image recognition. In both cases, the nature of the information sent to the RT simulation system is a count, but the method to generate the value is clearly different. Only minimal changes should be allowed on the side of the RT simulation system, if any.

For one of my RT systems, I implemented this type of modularity to be agnostic toward external computation hardware that executes tasks too heavy for the RT computer running the simulation.

Additional conceptual considerations when designing

The last question from St John *et al.* is inviting to examine the architecture of RT systems for the sake of its accelerated development and long-term life cycle. Specifically, how engineers should design RT systems so that they can keep shaping and using them for a long time. But what about the reverse modality? How should engineers design an RT system so that it shapes its users back and orients them toward specific behaviors? Another way to phrase it would be: How to design an ecosystem that guides the users in the tasks the software is meant to help them accomplish? These questions arguably reach beyond the field of RT technologies, and I am sure I could find many conceptual papers on the subject in the field of applied psychology or recently trending *nudges* in marketing. I believe I could find similar questions in architecture as well (in the sense of architecture of buildings or cities, not software or systems) where architects often consider how to shape spaces so that the utility of the space reveals itself to users.

I do not wish to venture in these fields at this point of my thesis, however. But I want to point out that, including this problematic, as complicated as it may be and seemingly unrelated to the primary objective of a software, could have large-scale benefits in the early design process of an RT system. For example, engineering and research in RT systems are seldom individual tasks. On the contrary, the deployment scale or the criticality of the applications warrant collaborative efforts to increase the likelihood of success of the system. Developing a new RT system from scratch or updating an existing one is always an expensive process; RT systems for engineering tasks are meant to be used during several years (Schiano and Silberto, 1986; St. John *et al.*, 1987; Kopetz *et al.*, 1989; Koopman and Yamauchi, 1990; Haung *et al.*, 2005) and incidentally define the outline of every future engineering tasks that will use it. During the life cycle of an RT system, engineers and designers all share a common tool and language. Hence, in my opinion a reasonable question would be: how do you design an RT system so that it invites its designers and end-users to collaborate for the sake of making it even better? How do you shape this RT system so that several users can coexist in the user space and complement each other? Indeed, if an RT system has a non-negligeable lifetime, it should help different teams work well together and not only run simulations or monitoring tasks.

Biology, like other sciences, is fragmented into several subfields to study life at different scales. If we consider that RT systems can be used in biology, my previous question about collaboration also holds. For instance, what is the architecture of an RT system that would promote inter-specialty collaboration and the mutualization of resources and ideas for faster development of models in biology? For the level 1 integration, such collaborative RT system would be like a “Google Document for modeling”. Several biologists would connect to the RT simulation system and work together in real-time. Experimentalists would provide the data, modelers would connect solvers and arrange the structure of the model, experimentalists again would comment the structure of the model based on their knowledge and observations, visualization specialists would handle the display of relevant variables and output data of the simulation, and statisticians would formally compare simulations’ output with experimental data. In the case of level 2, the above would stay valid but artificial life researchers would probably take the place of fundamental biology experimentalists. Here, MBB and the exploration of the solution space of a model would aim to provide explanations about the emergence of life and biological functions. The level 3 of a collaborative RT system would make use this time (and contrary to what I have been advocating in this thesis), of the capacity of RT systems to meet deadlines to efficiently schedule experiments over a crowd of laboratory robots (Yachie and Natsume, 2017). The purpose would be to maximize the uptime of the robots that have been jointly purchased or are simply shared.

One of the RT systems presented in this thesis was specifically developed with the objective to provide an environment where users could work alone, but also with collaborators.

Additional technical considerations when designing

Until now I have only mentioned technological challenges of implementing an RT system as consequences of conceptual challenges. In this section I will concentrate on technological challenges that are valid when implementing an RT system for biology but that are not present in state-of-the-art offline modeling software in systems biology. The primary source of technological challenges is the real-time constraint.

Whatever application it is for, the paradigm to execute a heterogeneous set of tasks under a deadline prevents from using many algorithmic solutions that were working fine in an offline system. Especially when the set of tasks is dynamic, either because some are aperiodic or their definition might change during the simulation, it becomes impossible to use static scheduling algorithms ahead of the simulation. In the worst case, the order of execution of the tasks must be reevaluated at the beginning of every step of the simulation. The data storage of the simulation elements must also be always dynamic. Indeed, with an interactive RT simulation system, the parameters, rules, or equations defined in a model file (e.g., SBML) might be changed by the user at any point of the simulation to test a different configuration. Updating a value is not going to be much different in an RT simulation software compared to an offline one. But adding or deleting a value will change the memory layout and might trigger moving an entire table of values from one point of the memory to another. This copy will evidently take time off the execution and put more constraint on respecting the deadline. Moreover, if a component of the model is deleted from the simulation, every other component that was using its output will not receive data anymore or received the data from another object that took its place in memory[24]. For example, biological model files often define equations for reaction rates. This reaction rate might be a composition of numeric constants, parameters defined in the file, chemical species quantities, or other functions (e.g., $f'(t) = 0.5 * k^2 * [X] + g(t)$). In an offline simulation software, the code corresponding to the equation would be generated at the beginning of the simulation and statically linked with the values (i.e., point to the memory location) of k, $[X]$, and $g(t)$. But, if the user decides to internal representation of $g(t)$ or delete parameter k, or species $[X]$, what should be done to compute the value of f'? Deletion might result in memory error and simulation crash. It is, of course, possible to implement countermeasures, but should it be a fail-safe or fail-operational system? What are the coding constraints of both? How to implement it so that it has a minimal impact on the objective to meet the deadlines?

[24] That is called a stale pointer in C++. This kind of error is very language specific. But it is easy to encounter similar bugs in any language when creation/deletion of objects is involved.

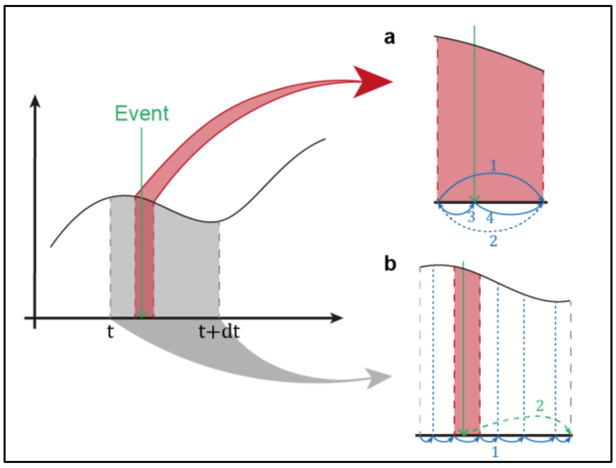


Figure 6: Event execution options during simulation integration. The grey area represents one RT time step. The red area represents one integration time step. a) Higher precision of the timing of execution of simulation events. The algorithm backtrack to the start of the step (2), integrates until right before the event is supposed to execute + do execute it (3), then resumes integration (4). b) The algorithm waits for the end of the RT time step. to execute the event.

Another challenge stems from the size of the RT time step (i.e., the deadline) relative to the size of the integration time steps of the numeric solvers. This mainly concerns continuous solvers such as the ODE solvers which may require a much smaller time step than the RT time step to limit the local error. In fact, it is rarely possible to ask the ODE solver to integrate for more than a few milliseconds or even microseconds for biological systems. But the RT time step is not limited in size and could be of any time order according to the definition of an RT system. The solution, in that case, is to have the ODE solver integrate within the boundary of the RT step using an adequate integration step size to limit the local error (corresponds to subdividing the grey area into smaller steps such as the red one in Figure 6). But then, the problem arises to correctly detect and synchronize the execution of events within the integration process. Is it necessary to adapt the ODE integration because an event was detected? In which case, the event detection algorithm must be triggered at the end of every integration step, then the integration must be rolled back to the time right before the event, execute the event, and resume the integration (see Figure 6.a). Or is it possible to apply events only at the scale of the RT time step (see Figure 6.b)? In which case, events detected in the past RT time step are simply approximated to have happened right before the beginning of the new RT time step. The two options have dramatically different implication on the computation cost of an RT time step with the former being much more expensive, but also much more accurate. The latter option is the choice that was made for the WC model of *M. genitalium* (Karr *et al.*, 2012), and which, to my knowledge, stays true in Vivarium (Agmon *et al.*, 2022) for in-between modules events.

In the last section of my thesis, I concretely describe the RT systems I developed. I give the major architecture details and how they relate to my vision of MBB. I will describe use cases and give running examples.

Game-Engine-Inspired Real-Time Systems

I will conclude this major section by presenting Game Engines (GEs) as their architecture, features, and development cycle practices inspired the RT systems that I developed.

Generators of "soft real-time interactive agent-based computer simulation"

Jason Gregory, author of the book *Game Engine Architecture* (Gregory, 2018a), analyzed that scientists would probably call video games "*soft real-time interactive agent-based computer simulation*" (third edition, page 9). A GE is a reusable software packing all the core tools and technologies used to author video games. Therefore, a GE is a generator of SIL soft RT systems. This characteristic only should suffice to realize the potential of a GE-like software for MBB. Indeed, my vision of MBB involves quickly iterating over a model through RT simulations to optimize it (biologists who want predictions), or to generate viable model alternatives (for me who wants to explore the solution space of biology). Thus, I need a generator of RT simulations and, as I discussed previously, the deadline constraints for the RT simulation of a biological model are soft which makes GEs a perfect match in this respect. Moreover, I am explicitly interested in the interactivity of the RT simulations for a biological model to obtain immediate feedback of the choices made by the modeler and further support quick design iterations. Iterating over a model will also likely involve a lot of trial and errors, updating values, and adding or deleting components of the systems. This type of data-wise dynamic environment is common in video games where many objects are managed, created, and deleted as the player interacts with the simulation. This is what Gregory included in the "*agent*-based" denomination in the definitions of video games. GEs therefore seem to also have achieved support for dynamic simulations that I am looking for. In addition, these agent-based simulations are rarely small. On the contrary, there is a fierce fight in the video game industry to make stunning simulations thriving with details even on embedded hardware with limited computation power. This underlines the capacity of GEs to scale up should there be a need for large-scale simulations. As models in biology becomes increasingly more complex (e.g., WC models), the guarantee that GEs can also successfully deliver for these extreme scenarios should be very appealing to any modeler in systems biology. GEs are can also scale up to integrate very heterogeneous data types (e.g. audio, 3D models, textures, physics models, code scripts, and so on)[25]. Likewise, biological simulations must integrate heterogeneous data ranging from model files (e.g., SBML) to raw data from experiments or other simulations. Heterogeneity in biology is also accentuated by the provenance of the data from databases such as BiGG (King *et al.*, 2016), MetaNetX (Ganter *et al.*, 2013; Moretti *et al.*, 2016, 2021), or BioModels (Malik-Sheriff *et al.*, 2020), for which models are not harmonized and translations are non-trivial. Finally, GEs are highly optimized software with the belief that a well-designed architecture enhances the way it is used. Notably with the modularity of the GE itself which helps engineers extending the capabilities of the engine; this practice is called "engine tooling". This is not a direct requirement for MBB but, as I have already argued, I believe it is a good practice for scientific software engineering. Therefore, GEs successfully solve similar ranges of problems than the ones I am facing to make MBB a reality.

---

[25] Gregory gives the following list section 7.2 page 493 (third edition) (Gregory, 2018a): "*Every game is constructed from a wide variety of resources (sometimes called assets or media). Examples include meshes, materials, textures, shader programs, animations, audio clips, level layouts, collision primitives, physics parameters, and the list goes on. A game's resources must be managed, both in terms of the offline tools used to create them and in terms of loading, unloading and manipulating at runtime*." He gives a slightly more general list a bit earlier in the book (section 7, page 481).

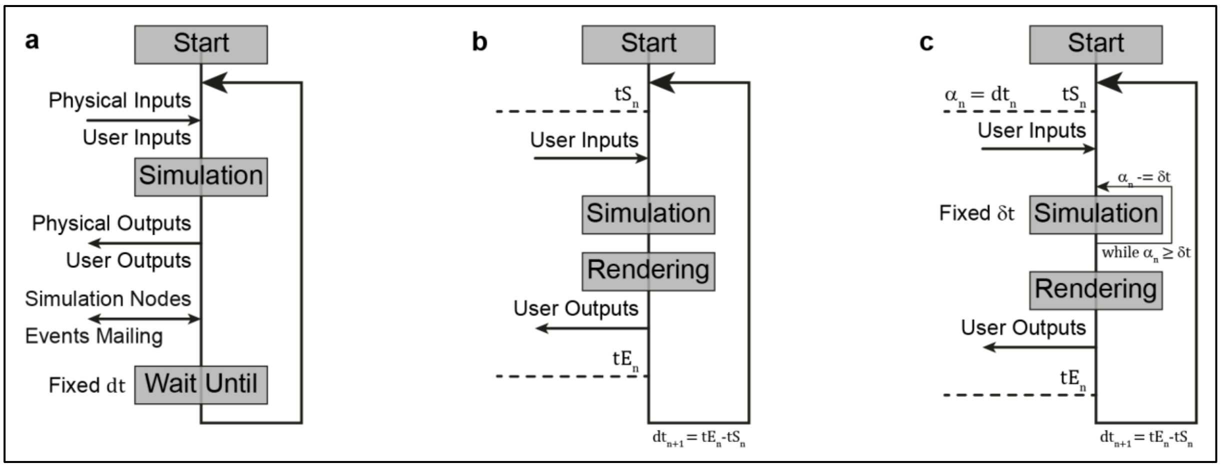


Figure 7: Various execution sequence of real-time loops. a) Same as Figure 2.a, provided for comparison; typical sequence execution of the simulation loop of an RT system. In case the execution of all tasks took less time than the deadline dt, the process waits until then. b) Basic variable time step size. The time budget of the next step is set to the execution time of the step that just finished to be processed. It assumes the execution of the next step is not going to be much different from the past one. This will naturally stabilize to on the best update rate the hardware can afford. Simulation and rendering have the same time steps. Main issue of this architecture is that the quality of the simulation depends on the power of the hardware because small dt implies higher precision for the integration. c) Semi fixed time step size. Compared to the loop in b), the simulation has its own fixed time step. The integrity of the simulation is better maintained. The difficulty with this approach is to select δt such that even low-end computers can finish to execute the simulation in less time than dt_n.

About the real-time simulation loop of a game engine

The RT simulation loop of a GE is traditionally called the Game Loop (GL). GEs are soft RT systems which may not have to guarantee the execution of one GL before a deadline, but instead execute as many GL as possible. That is because one of the marketed criteria for the success of a game is how smooth it is on a target hardware, which is equivalent to achieving a high refresh rate of the game simulation and the visual rendering (hence as many GL as possible). At the beginning of the millennial, the target refresh rate of video games on consoles would often be of 30 Frames Per Second (FPS) which effectively gives a time budget of 33 ms to execute a simulation step. Later, with the advent of more powerful hardware on consoles, video games are often advertised to run at a stable 60 FPS (16 ms). On a computer, however, consumers expect the game to run as fast as their hardware allows it for the best experience possible. In that case, it is not rare to have machines that run a particular game at more than 60 FPS, and even up to 300 FPS (3 ms) for some competitive titles. Even in these cases, consumers expect the frame rate to be stable enough in all circumstances of the simulation so that variations are unnoticeable. That being the case, unless the game is highly optimized, or the hardware is not saturated by the simulation, the higher the frame rate, the more complicated it is to keep it stable. The reason is easy to understand: video games have variable amounts of work to do every GL, if the simulation allocated a tight time budget for one GL based on the average workload, but there is sudden spike of activity, there will be a transient drop of frame rate. Overruns for games therefore initiate a decrease in quality of the simulation and user experience. In extreme cases, critics from consumers and negative reviews are bound to severely impact the sales of a game and impair a loss of income. Be that as it may, no life-threatening damages will ever ensue for the consumers, so GEs are indeed soft RT systems. Consequently, a GL may have to "*Wait Until*" if the frame rate is set such as on consoles (similarly to RT loops in heavy engineering, see Figure 7.a), or start processing the next update immediately such as on computers (this is the case in Figure 7.b & Figure 7.c).

Another specificity of GLs over simulation loops in heavy engineering RT systems is the presence of a specific rendering task concluding the GL (see Figure 7.b & Figure 7.c). In practice this task takes a big share of the time budget. Indeed, rendering the state of the simulation on a display is the major output for video games and high visual quality is often a selling point for games. So, it is not surprising that most of the time budget is dedicated to the rendering at the expense of the simulation. However, it is unreasonable in the scientific context represented by this thesis to allocate more time for stunning visual graphics if it means decreasing the precision of the simulation. This is one of the major hurdles that must be dealt with when adapting a GE architecture to scientific simulations. In this work, one solution I implemented is to decouple the computation resources used for scientific simulations from the resources used for graphic rendering. The second solution is simply to use a low-cost rendering scheme to give more importance to the simulation task. Finally, a classic GL do not contain any communication with physical components because games are expected to be self-contained on the machine of the consumer (computer, smartphone, or console alike). There is also no need to synchronize with other simulation systems that might be running on distributed nodes, and which would be computing complementary simulation data. Indeed, if a video game happens to be a multiplayer experience over a network, the communications are processed as part of the simulation task. To my knowledge, there is no game that would rely on the aggregated computation resources of a set of connected machines. In this work, I will only be concerned about level 1 and 2 of the integration of RT systems in modeling for biology, so no communications with hardware will be required.

As I mentioned, the target frame rates of the GL are set provide the experience possible for consumers depending on the computation resources their hardware has. One might chose to implement slightly different flavors of the update policies of the GL as a consequence (Fiedler, 2004; Nystrom, 2014; Gregory, 2018c). The easiest solution that is also reasonably agonistic of computer hardware decouples the simulation update from the rendering update. The former will be computed at a fixed integration time step, while the latter will be updated only after simulation has finished all its integration steps (see Figure 7.c). Nystrom described this algorithm as the simulation loop "*playing catch up*" to the dt. The main benefit of this approach is to make the simulation computation deterministic across all hardware, whether low-end or high-end. Indeed, the alternative solution where both updates are not decoupled (see Figure 7.b) will result in the high-end hardware doing more updates than the low-end hardware. In such case, we will have $dt_{low} > dt_{high}$, so the local error of the simulation integration will be higher for the low-end hardware. In the decoupled solution (see Figure 7.c), however, the simulation integration step is always δt, which is fixed in advance by the engineers. Consequently, both low-end and high-end hardware should perform very close computations. The delicate point then becomes to find a value for δt which is not too small that the low-end hardware might be overwhelmed. In fact, if δt is too small, then the computation time to integrate the simulation from t to $t + dt$ might become greater than dt. If this happens, the simulation will lag and be asked to integrate bigger dt at every step, which will result in bigger computation time, and so on. A security to avoid this "*spiral of death*" (Fiedler, 2004) is to set a maximum threshold μ for dt(Gregory, 2018c). This can protect against transient expensive simulation updates on a hardware which normally has no problems. But if we set $dt = \mu$ all the time because the hardware is never powerful enough to catch up to the real-time, the simulation will be slower than real-time by the factor dt/μ every step. On another note, Fiedler also argues (Fiedler, 2004) that, because of floating point errors during calculations, a different number of integration step between low-end and high-end hardware on the coupled solution (see Figure 7.b) will produce different rounding error accumulations as well.

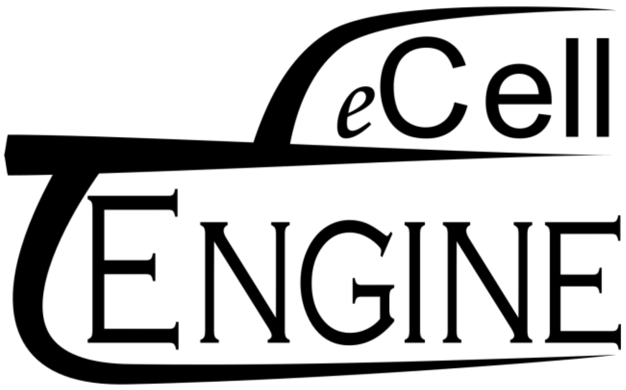
The game engine frameworks I used

The RT systems I developed in my thesis are using two different game engine frameworks. One of them is entirely custom and inspired from general architecture and pattern designs encountered in GEs. This is the one that I developed as a tool to support my vision for levels 1 and 2 of RT biological simulations. I had to make concessions, however, due to the gruesome size and complexity of building a typical modern game-engine-like software from scratch. The second RT system uses the game engine Unity (Unity Technologies, 2022b) to benefit from the already existing support for 3D simulations. Unity is a professional game engine which is now used in many other fields than the video game industry such as architecture, automobile manufacture, cinema, robotics (Unity Technologies, 2020b), and AI (Unity Technologies, 2020a, 2022c; Juliani *et al.*, 2018). I developed this second RT system to investigate how the architecture of an RT system can promote RT collaboration among biologists in an immersive 3D virtual world.

IV. Using Soft Real-Time Systems for Model-Based Biology

The RT systems I developed during my PhD thesis are tools supporting my vision of MBB. Namely, tools that motivate biologists to consider the solution space of biology instead of punctual observations. To do so, my RT systems were designed to easily test alternative versions of the model and to provide users with as much flexibility as possible as to what kind of model they want to build. The flexibility was achieved by giving the possibility to manipulate the passing of time of the simulations, change parameter values at will, add or remove symbols. Components of my RT systems were inspired both by RT systems geared for monitoring and simulation in heavy engineering. I was particularly impressed by the thoughts, concepts, and methods researchers on RT systems developed to rationalize the design of RT systems. In particular how they strived to include a working system as soon as possible in the design cycle to perform tests early on and quickly identify the shortcomings of the current version. In addition, I explored how to design an RT system so that it motivates the users to work collaboratively to increase the final quality of the model. My RT systems fall within levels 1 and 2 of the integration of an RT system with biology. Although I did keep in mind the possibility of extending them to level 3, there is no direct support. The RT systems I developed are not the most optimized, because their goal is first to introduce RT simulations in the modeling framework of biologists. In a way, this work is equivalent to the early research on RT systems for heavy engineering in the 1960s: it is first important to understand the advantages by examples before we can identify what should be optimized.

Designing a Soft Real-Time Simulation Engine: ECellEngine



ECellEngine is meant to help biologists explore the solution space of biology rather than focusing on specific solutions. It can be used to build, and analyze models while the simulation is running in real-time. The source code of *ECellEngine* is accessible publicly on GitHub (https://github.com/ecell/ECell_Engine).

Figure 8: Logo of ECellEngine.

User requirements: build, play, learn

ECellEngine must give the possibility to assemble heterogeneous biological data, easily interact with this data in real-time to for edition and analysis.

Build: assembling heterogeneous data | This concept is illustrated by the blue section denominated "*Build*" in Figure 9. The first responsibility for *ECellEngine* is to support the variety of data used in biology. This starts with models which have either been built from scratch or extracted from popular model file format in systems biology such as SBML. In addition, *ECellEngine* shall also be capable of processing raw experiment data files (e.g., csv

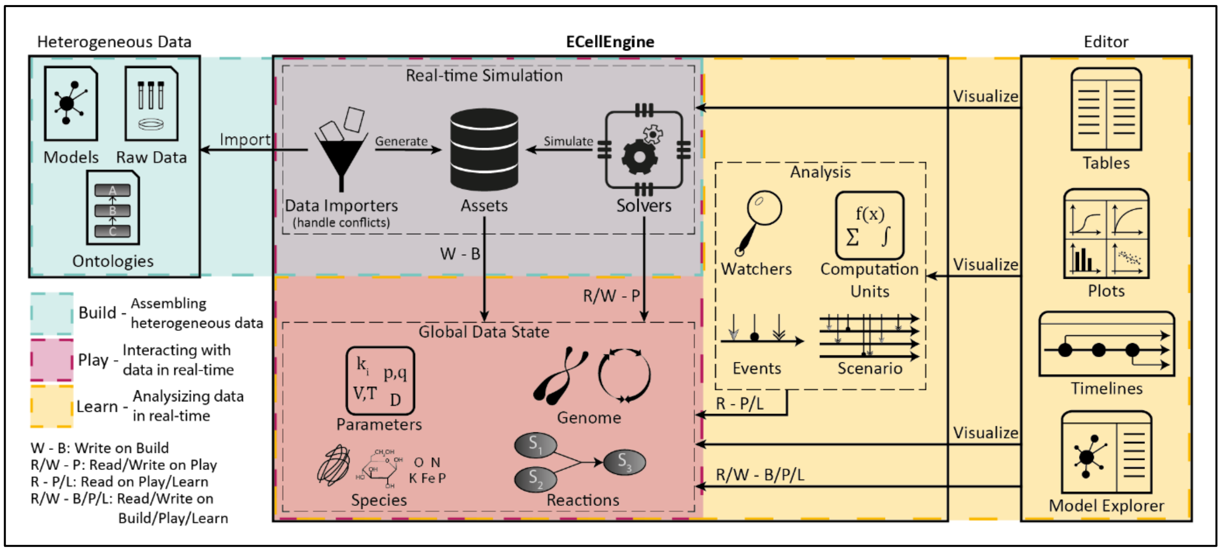


Figure 9: High-level structured view of the user's requirements for ECellEngine. It has three main parts. "Build" corresponds to the components involved with the real-time construction of biological models. "Play" corresponds to the components involved in the interactive real-time simulation. "Learn" corresponds to the components involved in the real-time analysis of the simulation.

or other table-like format) or ontologies. Once the content of the files has been translated to data structures internal to *ECellEngine*, they shall be collectively referred assets (as it is the custom in a GE). Given the diversity of the files potentially relevant to define a model, *ECellEngine* must, of course, implement an extensible file importer that can be easily accept new parsing strategies. Moreover, it is possible that different model files will refer to similar variables but using different names. Or, on the contrary, different variables will have colliding names. The probability of such cases is non negligeable because model files stored on databases are not harmonized at the level of their respective content. To address this issue, the asset system should perform a conflict check on each import and reports anomalies to the modeler. The system should also be extensible to rely on popular ontologies for the biology community in order to suggest curated interactions between entities identified in independent model files. Finally, as will be expressed in the following sections as well, the assets will be used very dynamically as a consequence of the interactivity. Therefore, ECellEngine must ensure easy access to the assets via a global and very open data state. The data state centralizes and exposes the data contained in the assets imported while building a model. This way, any processes related to a simulation can query, update, and display the data.

Play: Interacting with data in real-time | This concept is illustrated by the red section denominated "*Play*" in Figure 9. The next responsibility of *ECellEngine* is to effectively support my vision for MBB thanks to interactive real-time simulations. For this, I want *ECellEngine* to hand over full control of the time axis of simulations to modelers, giving them the opportunity to pause, play, resume, accelerate, decelerate, go forward or backward, advance simulations step-by-step or continuously. This feature will be coupled with the possibility to update the values exposed by model assets that were added to the simulation environment and that participate to the integrity of models. In addition to updating values, modelers shall be allowed to add new assets to the simulation environment or to delete already existing ones. Modelers shall also be given the possibility to switch from stochastic to deterministic solvers at run-time whether they are, for example, more interested in the diversity of the simulations or steady states. Such switch will invariably require parameter conversion that *ECellEngine* must account for.

Learn: Analyzing data in real-time | This concept is illustrated by the orange section denominated "*Learn*" in Figure 9. The final responsibility of *ECellEngine* is to leverage real-time interactions. Interacting with simulations in real-time is very different from the mainstream model design cycle in systems biology. In fact, the usual approach enforced by other software is to sequentially describe the model, run a simulation, log everything, and analyze the log. Then, depending on the results of the analysis, update the content of the model and restart the cycle. But as *ECellEngine* allows to do the first two steps of the cycle in real-time, the boundaries between the steps get very blurry and it is only natural that the output of the data might also be analyzed in real-time. Furthermore, my vision of MBB includes iterating through the design of a model, and it is reasonable that, at some point, the modeler will want to compare alternatives and not just overwrite the previous version of the model. Consequently, *ECellEngine* shall support the instantiation of alternative simulation environment that branch out from an already running version of a model. Then, *ECellEngine* should offer the possibility to display the state of each model; for this purpose, RT plots seem adapted enough. These features will be easier to implement if *ECellEngine* do possess a highly open data state per simulation as I mentioned previously. (see the connection between the red section "*Play*" and the orange section "*Learn*" in Figure 9). However, I can easily imagine that manual RT analysis of RT simulations can quickly become tiring, especially if the simulations of several alternatives are running concurrently. To help with this problem, *ECellEngine* shall include software objects that I will call *watchers* and *events* to help automatically track elements of the simulation and trigger responses to help the modeler manage the simulation. For example, the response could be to record the numeric values of the last x seconds of simulation, to update the value of a parameter, or pause the simulation and let the user decide. This kind of automatic RT data analysis will likely be enriched quite fast as *ECellEngine* is used on real modeling tasks. Finally, I expect RT interactions with simulations to lead to the necessity to remember every modification a modeler has made to a model in order to understand how the final version was reached. To this end, *ECellEngine* shall also include data structures that I will call *scenario* which contain a record of everything that happened during a simulation (manual or automatic modifications) to allow to replay the sequence in *ECellEngine* or to analyze it on a third-party tool.

Elements of architecture

The code base is much too large[26] to be discussed in its entirety here, so I will focus on three aspects of the architecture that are related to challenges of RT systems or requirements of this project.

Open, accessible, and flat data state | *ECellEngine*'s data layout was designed to facilitate the requirement "Play" which involves potentially adding, editing, and removing a lot of components from the simulation space. Conventionally in Object Oriented Programming (OOP), the data relative to an object is stored within the boundary of this object. This approach to data layout in a software is often the modern default choice and is heavily taught in computer science classes because it makes it straightforward to translate one's mental model of a problem to code (as long as we are using an OOP language). Indeed, mechanisms in the physical world are built from well-defined and carefully shaped lump of atoms, and we usually judge the brilliancy of an engineer by his capacity to assemble these parts together. As such, our physical –artificial– world is overflowing with such items. So, OOP has been extensively used in software engineering since its advent in the early 80s and the majority of modern code bases are using OOP one way or another. The field of RT technologies is not exempt as illustrated by the couple of design methodologies (Gomaa, 1986, 1984; Burns and Wellings, 1994) inspired by the object paradigm that I described earlier in section *Design methodology of real-time systems* page 31. Although very useful on small to medium scale projects, some properties and coding practices associated with OOP might suddenly increase the cost of upgrading and maintaining a code base. The one that interests me in the present context is the pressure that OOP applies to isolate and restrict the flow of data in a code base along a single axis with design practices such as *inheritance*[27]. The principle of inheritance is to define an object through his existential relationships "*is-a*" to other objects. A nice example is in (Gregory, 2018b): **Car** $\xrightarrow{is-a}$ **Terrestrial Vehicle** $\xrightarrow{is-a}$ **Vehicle** or **Boat** $\xrightarrow{is-a}$ **Aquatic Vehicle** $\xrightarrow{is-a}$ **Vehicle.** This type of relationships produces a graph commonly called *Class Hierarchy*. The above graph is all good until someone suggests extending it to incorporate the object **Amphibious Vehicle.** This object is both terrestrial and aquatic, so what should we do? Maybe we can inherit from both terrestrial and aquatic. But maybe we can't because some data in both classes have the same name (e.g., engine type). So, we need to fix the data layout of both classes to avoid collision. But the next problem is that every part of the software that was using the fixed

[26] The project has about 20,000 lines of code (not including documentation) over 100+ files and more classes. The project is in C++ which is rather verbose, so the size is not that big for a "complex" software. Nevertheless, it has become big enough that describing the whole architecture in this thesis would be counterproductive. The full documentation can be found online (https://ecell.github.io/ECell_Engine/index.html)

[27] Inheritance could, of course, be traded for *composition* in this example. But I did not want to discuss all problems and alternative solutions in this section; simply illustrating one of the limits of OOD which influenced the data layout of *ECellEngine*.

data now also needs to be updated. And so on. Real world situations can be much more complicated and many game engine engineers have written compelling pieces about these kind of issues (Fabian, 2018; Gregory, 2018b). The bottom line is that accessing data in a strict OOD can become cumbersome and eventually result in accreting design patterns to solve the problems of other design patterns[28]. In biology, models tend to be expressed using OOD because the mental model of biologists is shaped by centuries of classification work in botany, anatomy, histology, cytology, etc... As such, a common mistake is to use OOD to literally translate the knowledge layout of a biological object to the layout of the biological model on a computer. An example would be to separate molecular species in memory because they are only found in the cytosol, the Golgi apparatus, or the nucleus of a cell, respectively. Another example would be to separate them in memory because they possess different qualities which biologists categorized as proteome or metabolome. In effect, it is essential to understand that a computing system could not care less about the labels we put on data: it is only bits. Consequently, OOD is only a methodology to help humans keep as much as possible of the structure of a virtual complex system in their head. OOD do not help the computer. On the contrary, OOD undermines computational efficiency because it easily produces *memory fragmentation,* and it becomes very time consuming for the CPU to fetch all data relevant to a computation. It is much more important for a computer that the frequently updated data are close together in memory. With all these issues in mind, I spent a great deal of time considering the data layout of the mandatory biological concepts to be included in *ECellEngine* to trade-off my mental model of biology knowledge, my mental model of the architecture of the software, and the data processing efficiency for the processing unit. *ECellEngine* therefore defines an object called `DataState` which is unique to every simulation. The `DataState` contains every dynamic elementary object defining the model being simulated. An elementary object might be a `Species`, `Reaction`, `Parameter`, `Equation`, `Operation`, `LogicOperation`, `Event`, or `Trigger`. These elements are stored in C++ standard implementation of hash tables `unordered_map`. Hash tables are data structure that allows to retrieve their elements given a unique key. Of course, it incurs memory fragmentation compared to the likes of arrays, but hash tables are very practical to easily find individual objects. I needed this property to easily modify or delete existing objects, as per my requirements for trial & error in MBB. But, as expected, recent performance analysis indicate that this layout is not the best for the simulation in the update loop. In the future, it might be worth considering maintaining an array-like (i.e., non-fragmented) copy of the `DataState` for the simulation, despite the possible memory cost in large-scale models and the dangers to have desynchronized values between what the modeler manipulates (from the hash tables) and what the simulation processes (from the array-like data structure).

[28] Richard Fabian wrote a yet unpublished new book focusing on this misuse of design patterns that I was given the first draft to read for an informal review.

Simulation loop | The RT simulation loop follows a modified "Catch up" structure as discussed in Figure 7.c (page 61). The noticeable difference is that we allow the δt to vary depending on the integration strategy of the simulation. In particular, if the numeric solver used expects the model to be defined as differential equations, the default ODE solver implements a Dormand-Prince 5(4) version of the explicit Runge-Kutta method[29]. This method allows to calculate a variable time step size based on an approximation of the numeric local error. Since the calculation of the approximation of the local error is the same on low-end or high-end hardware, the original argument in favor of a fixed δt for determinism of the simulation is conserved with this approach. And, of course, it has the added benefit of guaranteeing a certain level of precision. If the numeric solver used expect the model to be defined as a stochastic system, the default stochastic solver implements a Gillespie *Next Reaction Method*. In this case, since the time of the next reaction depends on the sample of a *pseudo*-RNG, determinism can only be guaranteed if two runs of a simulation use the same random seed. As for the detection and trigger of simulation events, I opted for the highest precision option represented in Figure 6.a (page 59). In effect, both solvers will integrate their step, check is an event should have been triggered within the last integration step and backtrack to this exact moment. The current code to implement this strategy is extremely cumbersome. Moreover, as I already mentioned, this option is the most expensive one and it might become problematic in the future for large-scale models where many events must be checked. I plan to eventually refactor this implementation. Finally, in the simulation engine layer of the code, we completely removed the rendering step. Consequently, it is up to a third-party using my simulation engine to include a rendering or not. I demonstrated this practice by also implementing a GUI for the engine that I will now describe.

Integration of the simulation engine in the editor | The code base for the simulation engine is separated from the code base used for rendering. In fact, the code base for the simulation can be used independently of any GUI. For example, it could be simply controlled from a terminal prompt. However, that would not be very useful and completely undermine the purpose of implementing an RT simulation to achieve interactivity. Thus, I also developed a separated GUI to control the simulation engine. It is based on the C++ library *Dear ImGui*[30] with *glfw*[30] for the windows and input backend, and *Vulkan*[30] for the rendering. The update loop of the GUI (henceforth referred to as *the Editor*) follows the structure represented by Figure 7.c (page 61). Essentially, the user's mouse and keyboard inputs are polled by *glfw*, the simulation loop runs for all active simulations, and rendering is serviced by *Vulkan* and handed over to *glfw* to draw the application's windows. There are no outputs for the user. However, an issue with this kind of loop in our context is that, if the simulation

[29] Details are in Appendix A1.

[30] *Dear ImGui*: https://github.com/ocornut/imgui; *glf*: https://www.glfw.org/; Vulkan: https://vulkan.lunarg.com/

takes time to compute, the editor will also become sluggish. This is acceptable in a video game because we usually don’t want the user to be able to do anything while the simulation has not finished to integrate. But, in the present case, there is no reason to penalize the user of the editor if a simulation is heavy and cannot be as fast as the physical time. In fact, the user should be able to interact with the editor whatever the status of the simulation. To reach this new level of decoupling, the next version of the editor’s update loop will run the in parallel of the simulation. This requires a powerful messaging system to be able to send commands to the simulation asynchronously from the editor. For example, when a modeler updates, creates, or delete a component of the simulation, the corresponding command will be sent to the simulation engine, queued, and executed once the current step has been integrated. The base infrastructure of this messaging command system is already implemented in *ECellEngine*, and part of the communication between the editor and the engine are using it. Putting this matter of the synchronization of the two RT loops aside, the question arises to decide the interaction scheme of the user with the editor to design models and control simulations efficiently and easily. This problem is a recurrent one for GEs and falls under the choice of the *scripting interface* (Gregory, 2018b). A scripting interface in a GE is either typically a programming language or a GUI that gives game designers the power to manipulate the simulation without the skills or the knowledge of the engineers who developed the engine. In Unity, the scripting language is C#; in Unreal Engine (Epic Games, 2022), the scripting language is a node-based interface called blueprints (Unreal is especially famous for this interface which allows to make entire games without a line of C++). A node-based interface is much more user-friendly than code and participates to quick iteration design, so I naturally went in that direction. Enriching the catalogue of nodes requires the skills to code in C++, but using the nodes is accessible to anyone. In short, the nodes of the editor represent data structures from the simulation engine (e.g., molecule, reaction, parameter, model, solver, event, …) which can be connected to modify the behavior of the simulation. Modifications resulting from the graph are eventually sent to the simulation engine using the asynchronous messaging system and are, therefore, accounted for at the end of the integration steps.

Designing a Collaborative Soft Real-Time Modeling Environment: Kosmogora + ECellDive

Figure 11: Logo of Kosmogora

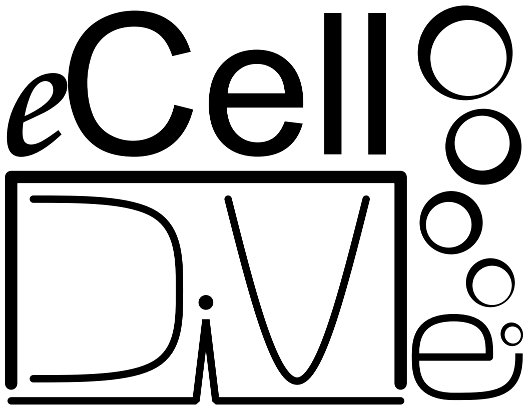


Figure 10: Logo of ECellDive

Kosmogora and *ECellDive* are meant to help bringing biologists from different fields together thanks to the Metaverse to speed up the iterative design cycle of biological models. The source code of *Kosmogora* and *ECellDive* are accessible publicly on Github (https://github.com/ecell/kosmogora and https://github.com/ecell/ECell_Dive respectively). The content of this section was adapted from my paper "*An architecture for collaboration in systems biology at the age of the Metaverse*" (Jacopin *et al.*, 2024).

User requirements: collaborative biology modeling in the Metaverse

I explained in section *From real-time systems to the Metaverse* page 41 how the Metaverse is a product of RT systems by integrating elements from simulation, monitoring, digital twins, and data streaming. But, as I explained as well, the definition of the Metaverse has yet to stabilize and there exist many interpretations or focus of interest. In the following, I will propose conceptual requirements for a scientific metaverse based on my understanding of the various definitions in academia and the trends I observed in the private sector.

Proposition of 7 requirements for a scientific metaverse | I identified seven points among the different concepts discussed by others (Are We There Yet? A Status Check on the Industrial Metaverse, 2023; Wang *et al.*, 2022; Ning *et al.*, 2021; Buchholz *et al.*, 2022; The Metaverse and How We'll Build It Together -- Connect 2021, 2021; Mystakidis, 2022; Lee *et al.*, 2021; Park and Kim, 2022; Weinberger, 2022) which appear necessary to qualify an RT system of scientific metaverse including, in my case, a metaverse for biology (see Figure 12).First, there are the technological aspects with (1) real-time, (2) immersive and (3) multiplatform. There seems to be a consensus on these three between the private sector and academia. This comes down to the technology used to leverage metaverses and is quite independent of the systems biology field per se; these three are requirements. Next, I believe it is necessary to provide a reliable environment to process experiment data and simulations. An environment much like the one voiced by heavy engineering companies where the integrity of the data transformations is good enough to generate added value

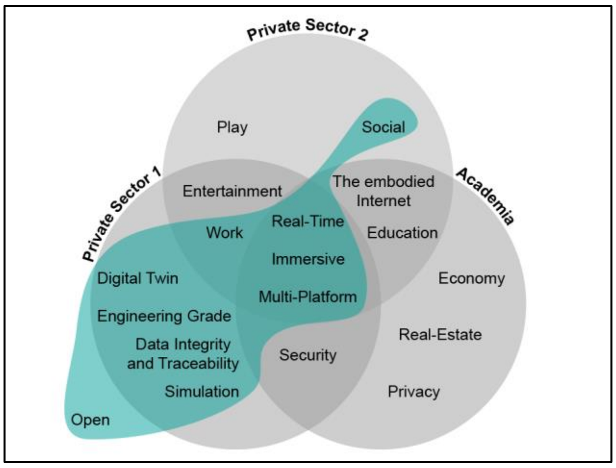


Figure 12: Distribution of the center of interests of different actors of the construction of the Metaverse. The "private sector 1" represents companies with activities more geared toward heavy engineering. The "private sector2" is represented by companies that also sell services. The turquoise area delimits the concepts which I think apply for a scientific metaverse.

compared to taking decisions outside of the Metaverse. I believe the term (4) engineering-grade (high simulation accuracy, data integrity, and data traceability) that we hear from the side of private sector is an appropriate denomination. I also reckon that (5) collaborative and (6) social apply to a scientific metaverse. For (5) collaborative, I mean close relationships among the collaborators during the whole timeframe of the collaboration. The collaborators will likely share data and actively engage in discussions about their projects. In (6) social, I extend the spectrum of human interactions via asynchronous (e.g., newsfeeds, articles) and real-time media (e.g., talks, posters, conferences) with peers beyond the frame of collaborators. I also believe communications with the public must be included in this requirement as press releases, interviews or open classes all falls under the aforementioned media. Finally, I add (7) open. I say that I "add" it because I have hardly read papers from the academia or heard the private sector explicitly communicate about this point. Nevertheless, (7) open(-ness) is gaining more traction every day as a modern science practice via, for example, the implementation the Findable, Accessible, Interoperable, Reusable (FAIR) data principles. Therefore, I believe that any metaverse that calls itself scientific in the future shall be (7) open. Moreover, in a scientific metaverse, I do not think that (7) open(-ness) can be about data alone. As the Metaverse will indeed reach high levels of hardware integration due to the spread of laboratory automation (King *et al.*, 2009; Sparkes *et al.*, 2010; Coutant *et al.*, 2019; Ochiai *et al.*, 2021; Kanda *et al.*, 2022) and digital twins of laboratories (Zhongcheng *et al.*, 2022; Palmer *et al.*, 2021; Rukangu *et al.*, 2021), a scientific metaverse will likely integrate equipment in the virtual space. It will become possible to run experiments in the real world while interacting with virtual elements in a scientific metaverse. Therefore, I foresee the presence of (7) open equipment in an open scientific metaverse. As an analogy to FAIR data, FAIR equipment could start with a public collection of the virtual counterparts of research institutions (Findable). Visitors could take a virtual tour or watch on-going experiments (Accessible). Visitors could also manipulate the digital twins of equipment provisioned for educational or research purposes (Interoperable). Manipulations could be re-run to verify results of protocols or adapted for different experiments (Reusable). Of course, I also foresee difficulties to implement this level of (7) open(-ness) because, unlike data produced by experiments, equipment is directly related to an investment cost. So, I expect high barriers against open equipment, but I still believe it is feasible as this already happens today for the network of telescopes on the planet. Next, I will give more details about requirements (4) engineering-grade and (5) collaborative which I specifically targeted with *Kosmogora* + *ECellDive*.

About (4) engineering-grade | A metaverse is engineering-grade if it guarantees the integrity of data and simulation of complex tasks where mistakes can be costly at best and dramatic at worse. First, I believe the same requirements as outside apply regarding simulations in a metaverse. Specifically, usage of rationally designed models, awareness of integration errors, and understanding of the frame of validity of a simulation method. I think an added difficulty in performing simulations integrated in a metaverse is the coupling with immersive 3D rendering. As I presented for *ECellEngine*, rendering steps are usually the most expensive in an update loop. Therefore, care must be taken to not unwillingly make any concessions about the accuracy of the simulation because the immersive visualization was being too expensive. A solution to this potential problem is to separate the computation resources used for the simulation from the resources used for the immersive visualization. Second, I think there might be issues with preserving the integrity of the mental model of a user in a scientific or professional metaverse. It might be more relevant in the former case as knowledge in science is usually considered less *certain* and the practice of a scientific activity in a metaverse might add to the complexity because of the sheer size and heterogeneity of the data sources one will be able to access in a metaverse. Subsequently, I think a scientific practice in the Metaverse will require counter measures to protect the integrity of the mental model of scientists about the subject of their investigation. However, I do not doubt that scientists will eventually adapt to this practice as they familiarize themselves with it and as the quality of the counter measures rises._Another issue to realize engineering-grade metaverses is about the traceability of its user's scientific decisions. As my vision of MBB includes a high degree of model versions as a result of the exploration of the solution space of biology, or the optimization of a particular model for its predictive capacities, there is a need to keep track of the modifications. This is arguably not new as traceability has been increasingly recognized as a mandatory scientific practice for the last couple of decades to allow better peer-review and fight against frauds and falsification of data. But the current scope of the practice enforcing traceability will be too limited in a data intensive environment such as a metaverse. I identified three components in knowledge tracing. First, there is the path of states as a whole – is it possible to identify every intermediate state that led to the current state? I call this "temporal traceability." In biological knowledge bases, this is manifested by maintaining the possibility of accessing every version of a file. UniProt (The UniProt Consortium, 2023) or BioModels (Malik-Sheriff *et al.*, 2020) implement such traceability. Second, there is the state itself – how is one state different from another? I call this "differential traceability." To our knowledge, only UniProt provides access to such tracing in biology. Outside biology, Git (Chacon and Straub, 2014) is a well-known versioning system that enables both temporal and differential traceability. Finally, there is the transition between two states – what were the actions that led to commit to a new state? Here I do not focus on the analysis a posteriori of the choices (Dou *et al.*, 2009; North *et al.*,

2011) but rather on their manifestation during real-time collaboration. I call this "real-time traceability." To our knowledge, no biology databases provide such information. This is to be expected, as online databases are "static" environments where knowledge is pushed to (and pulled from) but never created. Text editing platforms, such as Google Documents, are examples that enable real-time traceability. In my opinion, temporal and differential traceability could be leveraged with appropriate data files. Of course, such file would include the authors that contributed to reaching a specific state. They would also enable non-destructive modifications of the original file in order to facilitate backtracking by simply re-importing a previous version. At this point, this type of files would not be conceptually much different from the commit reports in Git. Except that the nature of the objects that might be tracked in a scientific metaverse is much more diverse that plain text files. Moreover, these files must be mobilizable in real-time. A trade-off must be found between raw human-readability, compression (for environmental sustainability), and fast machine readability. The file format I created for this purpose will be discussed in sections *Elements of architecture* page 75 (paragraph "*Biological data management by Kosmogora*").

About (5) collaborative | I believe that the modalities of collaboration we are currently used to compared to the ones we can expect in a scientific metaverse are different enough to warrant careful consideration. Indeed, even if the recent pandemic has raised our awareness toward virtual collaboration tools, they are mainly geared toward remote and asynchronous communications and not real-time. In addition, RT interactions in a virtual environment allows for richer interactions powered by visual effects that cannot be reproduced with RT interactions in a real environment. For example, let's suppose that two collaborators are seated at a table and having a meeting in the real world. It is very common that, at some point in the meeting, one of the collaborators will make use of a physical object to help him convey his thoughts to his interlocutor. The object can be a notebook, a whiteboard, a tablet, a computer, or yet another possibility; the nature of the object does not matter. What matters is that there is only one physical instance of the object in the world at this meeting. Therefore, ignoring potential optic illusions, both collaborators see the same object and, should it be passed from one to the other, can exclusively interact with the object in a sequential way. However, as soon as the object share between the collaborator switches from being physical to virtual (e.g., RT shared editable documents such a rich text file or a slide presentation) several consequences ensue. First, they may not see exactly the same object because a collaborator does not have the same visualization clearance as the other or, simply, network lag and desynchronizations of content distort the representation of the object. Second, they now have the possibility (supposing the software allows it) to interact with the object concurrently in real-time. In which case, there is a possibility that their actions overlap and result in a deterioration of object leading to a decrease in the quality of their collaboration. In a virtual world of the Metaverse, the risk of dissonant collaboration necessarily increases because the meeting space itself has been virtualized. Therefore, the collaborators cannot be certain that they are interacting with who they are supposed to be and that what their interlocutor is sharing with them is the whole truth. Consequently, a scientific, or any professional metaverse, requires a very high degree of trust between the parties involved in the collaboration. I do not doubt that scientists will respect this assumption, but one malicious intent is enough. Putting aside this unpleasant scenario, I believe the virtualization of a collaboration environment also has much potential to increase the quality of the collaboration. In fact, assuming that the collaborators behave professionally and stay mutually aware (Biocca *et al.*, 2003), the virtualization gives the possibility to greatly accelerate the iterative design cycle of a model by concurrently working on alternative versions. This is the offspring of the asynchronous *branches* in Git with a RT 3D immersive virtual world. A scientific metaverse shall give the possibility to users to create alternatives of models in real-time, pull the work from collaborators to apply the modifications on their own branch, and push their work to other collaborators for review. Thus, where UI/UX was only supposed to support a single flux of activity for physical objects in reality, it shall now accommodate multiple users in a virtual world, leveraging collaborative work while helping avoid that some disrupt progress by mistake. Finally, another interesting notion that comes with the Metaverse is the persistence of the virtual world. Persistency is the property of a virtual object to not reset after users log out. Hence, users that connect back in the Metaverse will find it identical to when they left unless other users modified it in the meantime. I believe the quality of long-term collaboration would increase if a certain level of persistency was applied to the modeler's activity and decisions. This would be analog to archeological traces of human activity in the real-world. Of course, the traces in the virtual world shall not be as complex to interpret in order to effectively improve the collaboration. Instead, I propose the trace to be limited to major design decisions. The long-term of such milestones would become a medium for indirect communication and collaboration.

Elements of architecture

The code base of *Kosmogora* and *ECellDive* is also quite large[31], I will focus here on aspects that are fundamentally different or absent from what was discussed for *ECellEngine*.

Kosmogora-ECellDive communication | *ECellDive* communicates with *Kosmogora* through HTTP to delegate data management and simulation tasks. *Kosmogora* is implemented in Python and utilizes the *Uvicorn* package (Encode, 2022). Every module in *ECellDive* that interacts with *Kosmogora* (i.e., for importing, saving, and simulating) uses an Application Programming Interface (API) to build Uniform Resource Locators (URLs) for HTTP requests. The basic structure of the URLs contains the IP address and the port to reach *Kosmogora* separated by a colon; then a page of the URL is the name of the query to run in *Kosmogora*;

[31] The project has about 30,000 lines of code (not including documentation) on over 100+ C# scripts, and 50+ other assets including hand-built 3D models, simple textures, icons, shaders, and so on. A detailed description of the main systems can be found online (https://ecell.github.io/ECell_Dive/articles/Dev/before_you_start.html).

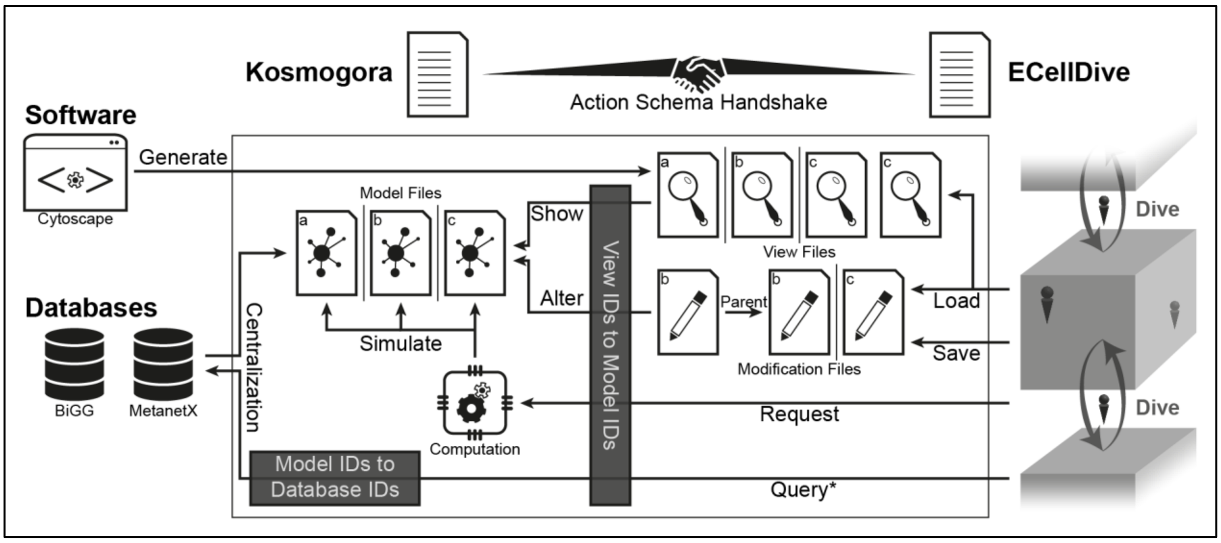


*Figure 13: Centralization and management of biological data by Kosmogora Kosmogora manipulates three types of files: model files, view files, and modification files. Model files are gathered from online databases and the respective IDs are linked to allow cross referencing. View files are used in ECellDive to represent all or a portion of a model file; one model file has at least one view file. View files can be generated by external tools such as Cytoscape[47]. Modification files record alterations made by users in ECellDive while manipulating the model (e.g., knockout of a reaction in a metabolic network model). Modification files may have a parent-child relationship. When a child file is imported into ECellDive, all modifications recorded in the parent are also imported. This lineage of modification files allows tracing the temporal evolution of a model ("temporal traceability"). Modification files are lists of entries describing the modifications and can be compared with each other ("differential traceability"). Finally, Kosmogora-like servers must implement a list of functions declared by ECellDive; the compatibility is checked in the action schema handshake. *As of Kosmogora 1.1.X, the queries do not use the databases' API but process locally downloaded content.*

and finally, the last page describes the parameters of the query (if any). In *ECellDive v0.11.X-alpha*, users can ask to see the list of models and view files, to import view files, to see the list of modification files, to save new modification files and to perform the Flux Balance Analysis (FBA) of a metabolic network model. Although I developed *Kosmogora* for *ECellDive*, it could be replaced with an alternative server that would also provide data management services for *ECellDive*. This is a one-to-many map from the virtual world to the external services in order to arbitrarily grow or vary the execution of features in the virtual world. Hence, users can connect to any server if it implements the set of HTTP requests required by a module. The API of a *Kosmogora*-like server is checked by *ECellDive* against its "server action schema" describing the mandatory subsets of commands for each module. If every command in the subset is present in the list of commands implemented by a *Kosmogora-like* server, then the module is unlocked in *ECellDive*. Furthermore, if users connected to multiple *Kosmogora*-like servers implementing the API for a module, then they can choose which to use. Thanks to API checking system, I have decoupled the RT rendering system from the hardware and software that manage the data and performs simulations. This promises flexibility over a few versions of *ECellDive* and is my answer to the design question *"How should the computer systems be configured such that [...] future study requirements are independent of hardware configuration?"* page 52.

Biological data management by Kosmogora | *Kosmogora* uses classic file formats and simulation packages from the field of systems biology to avoid creating yet another formalism. The metabolic pathway model file is an SBML file (Keating *et al.*, 2020) and the view file is a CyJson file (Shannon *et al.*, 2003). Currently, I use *COBRA* (Heirendt *et al.*, 2019) in *Kosmogora* to run simulations. *Kosmogora* uses *modification files* to alter *model files*, and *view files* to project a part or all the content of a *model file*. A *model file* can exist with zero or more *modification files* (in Figure 13, model (a) has no associated modification files, model (b) has two, and (c) has one). Modification files are YAML files with fields declaring the user saving the file (author), date, root model, and a list of modifications. They can optionally include a reference to a parent *modification file* (in Figure 13, modification files of model (b) have such a relationship). Wherever a parent is referenced, the new modification file appends all the modifications stored in the parent file upon loading in *ECellDive*. One *view file* is the interface to one model file. A *model file* has one or more *view files* (in Figure 13, model file (a) and (b) have only one matching view file while (c) has two view files). The format of *view files* can vary to adapt to the most popular format of each research community. A *view file* must however store enough information to instantiate the Game Objects (GOs) that will embody the elements of the model in *ECellDive*. For example, if a *model file* describes a graph (e.g., metabolic network), then the *view file* requires information about the positions of the nodes and the sources and targets of the edges. A *view file* may also include additional information such as labels and metadata of the objects that are present in the model. However, not enforcing a general format for the *view files* has implications. Indeed, for any new *view file* format added to *Kosmogora*, a matching module must be added in *ECellDive* to correctly parse it. *Modification files* are independent of *view files* which are only spatial projections of *model files*, the core information is the model. Model views may react to *modification files* only if they both contain entries matching the same element in the *model file*. If a modification file contains modifications about parts of a model that are currently not included in the view file, they will still be accounted for. For example, if a *model file* contains three reactions R_1, R_2, R_3 and the view file only includes information about R_1 and R_3, only these two reactions will be displayed in *ECellDive*. But, if the user applies a *modification file* which indicates to knockout R_2, then this modification is still applied even if it is invisible with the current loaded *view file.* Conflicts between *modification files* are avoided by always giving priority to the last loaded file. Indeed, I consider that any modification in the $n\text{-}th$ loaded file and targeting an element e of a model will override any modification that also targeted e and when one of the $n-1$ previous files were loaded. Similarly, when users build genealogies of modification files with the parent—child system, the parent is always the last file loaded. Hence, any modification in the $n-1$ previous modification files that are not shared with the $n\text{-}th$ file is considered "new" and will be explicitly recorded when saving the $(n+1)\text{-}th$ modification file.

Working with the game loop in Unity | The VR scenes of *ECellDive* are managed by Unity's GL. The behavior of a GO is defined by the combination of components attached to it. Unity has prebuilt components to cover classic aspects of a game such as user inputs, physics, graphics, etc. However, programmers can also implement custom components using C# scripts. A component is created by defining a class deriving from MonoBehaviour that is part of Unity's code base. I wrote dozens of custom components for *ECellDive* (refer to the online documentation for a full description). Finally, GOs can have parent—child relationships to further increase the complexity of the behavior of a parent GO. The data and action modules in *ECellDive* are GOs to which I added components to customize their behavior accordingly. Some of the components are part of Unity's XR package, while others are custom-written C# scripts. These scripts implement features related to the user interface (e.g., highlight, grab, move) as well as the core features of the modules. I wrote two C# classes to facilitate the implementation of new features in *ECellDive* that are called modules (see Table 4, page 84, for more details). The first is `Module`, the second is `GameNetModule`. The former is used when a module should only be visible by the user who added it in the collaborative virtual environment, the latter is used to share the module over the multiplayer network with all connected collaborators. For example, the server action modules (see Figure 14) in *ECellDive* to communicate with *Kosmogora* inherit from *Module* because none of them require simultaneous access or interaction. Conversely, a data module (see Figure 14) is shared among all users, so it inherits from `GameNetModule`. The metabolic pathway data module implemented for this paper uses the default layout of the metabolic network based on the (X, Y) coordinates of the nodes (i.e. metabolites) stored in a CyJson file inherited from the desktop software *Cytoscape* (Shannon *et al.*, 2003). Since every user must be able to dive into this CyJson data, the `CyJsonModule` inherits from `GameNetModule`.

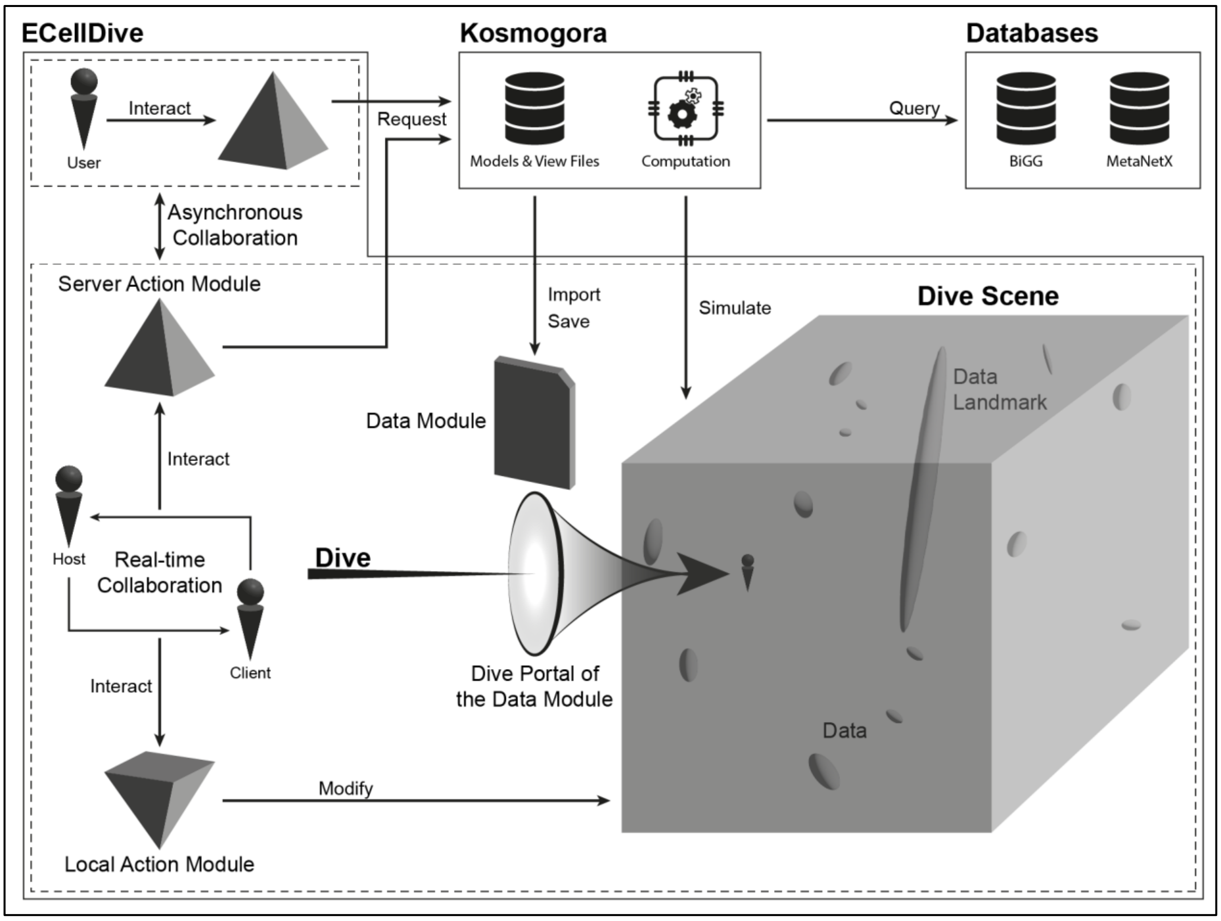


Figure 14: High-level interaction schemes in ECellDive between users and 3D objects in dive scenes. Data files, portals, data, interactions with the server, and interactions with data are based on tangible 3D objects called modules. In contrast to windows on a screen, 3D objects in the virtual world of a dive scene participate in the feeling of immersion. Those can also appear as "landmarks" in the 3D world. That is, data in the dive scene are physically more prominent than the rest and can be clearly identifiable by users similar to a mountain on the horizon.

ECellDive virtual space division in dive scenes | *Dive Scenes* are a concept representing a portion of space in which users can navigate, add modules, interact with data, and instantiate portals to go to other dive scenes (see Figure 14). They protect the integrity of the mental model of the users by giving them an abstract unit to delimit the seemingly infinite space of a virtual environment. In Unity, a scene is an asset containing a hierarchy of GOs that make up the content of a virtual space. The hierarchy of GOs defines the logic driving anything that is happening in the space, including user's movement or interaction. Despite the apparently good match between our concept of *Dive Scene* and a *Unity Scene* there are constraints in how scenes are managed in Unity. For one, as far as I could tell, scenes assets are built in the application (and it's reasonable). This implies that you can only add scenes in the Editor and that you must know what the *Unity Scenes* will contain in advance which is incompatible with our vision of dynamic *Dive Scene* when users dive into newly added data. So, in fact, in *ECellDive*, players diving from one scene to another never leave the Main *Unity Scene*. Our `DiveScenesManager` keeps track of which GO of the *Unity Scene* belongs to which *Dive Scene* and, when a user dives, the manager hides the GOs of the previous *Dive Scene* and shows the GOs of the new *Dive Scene*. Fellow divers on the

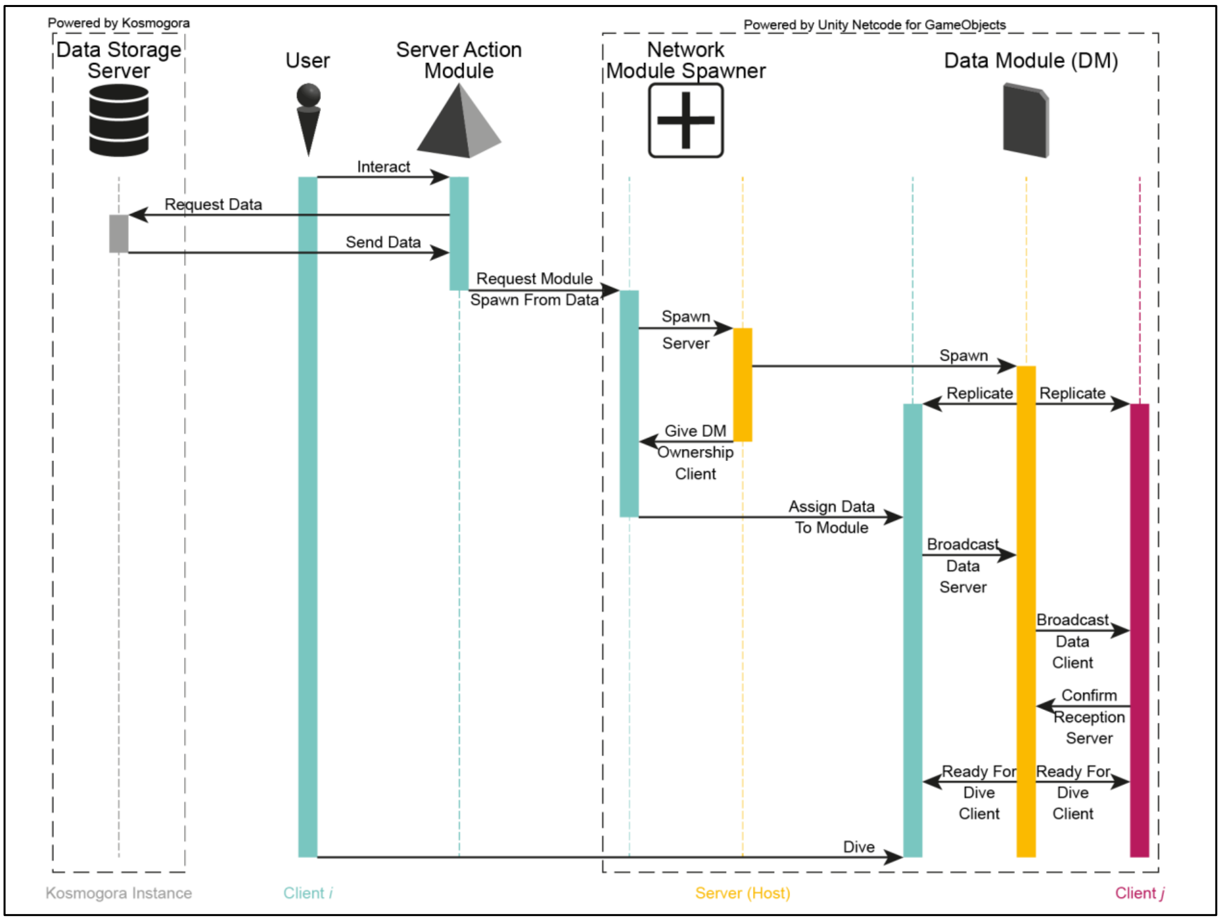


Figure 15: Simplified sequence diagram for data import in the Client/Host work session of ECellDive. A Client in the work session interacts with a server action module targeting a data storage to request for data. Once the data is received, it triggers a cascade of calls in the work session to create a 3D object encapsulating the data (data module) for all clients in the session. After every client confirms that it received all the data, any client can dive into the data module to visualize its content and further interact with it.

multiplayer network are also hidden and showed depending on the *Dive Scene* they are currently exploring.

Sharing large biological data over the multiplayer network in ECellDive | I used Unity's package *Netcode for GameObjects* (Unity Technologies, 2022a) to implement a host/client architecture in which a user, the host, runs the server that will synchronize the virtual world for all other connected collaborators (the clients). This network solution does not require additional hardware because the instance of the server runs directly on the device of the host. However, as there is no specialized hardware, the number of synchronized clients is limited by the computational power and network bandwidth of the host. *ECellDive* runs on Meta Quest 2, and it is recommended to not exceed four users to avoid latency and ensure a good experience in a dive scene. In this framework, a user creates an instance of a collaboration session at a specific IP address, and clients (collaborators) can join the session. In a work session, clients can import, edit, and save data with the link between *Kosmogora* and *ECellDive*. Future metaverses will have a much larger network infrastructure than *ECellDive* with dedicated servers to oversee the real-time collaboration of more than four users. For example, current multiplayer video games can manage real-time lobbies of several dozens of users without latency issues. In single-player mode, the user is their own host, while in multiplayer mode, a player can host others over a local area network. Connection over the internet with the host/client architecture is theoretically possible but requires every host to configure port forwarding on their router. This can prove complicated if the host is within the network of a laboratory where the router's configuration is probably under strict management. Connection of a client to a host is protected by a password set by the host. When a client connects, he will exchange data with the server (the host) to synchronize the state of its *dive scene*. This operation is simplified to some extended thanks to the API of *Netcode for GameObject* but I encountered difficulties when sharing large data files (e.g. models and views). Indeed, *Netcode for GameObjects* was not designed with this use case in mind. Rather, it is meant to support multiplayer games where small communications between the server and the clients are the norm. This is an issue in our case because I need to synchronize large data files: when a client decides to import a module, the content of that module must be shared to all other clients. A simplified version of the sharing sequence between users is depicted in Figure 15. A client contacts Kosmogora through a server action module (left half of the figure); then, when the client receives the data, it is automatically forwarded to the server that will broadcast it to the other users connected to the session (right half of the figure). When the data is large, the "Broadcast Data Server RPC" handles the partitioning into smaller chunks and sends one per frame. Then, the chunks are re-assembled on the side of the receiving clients.

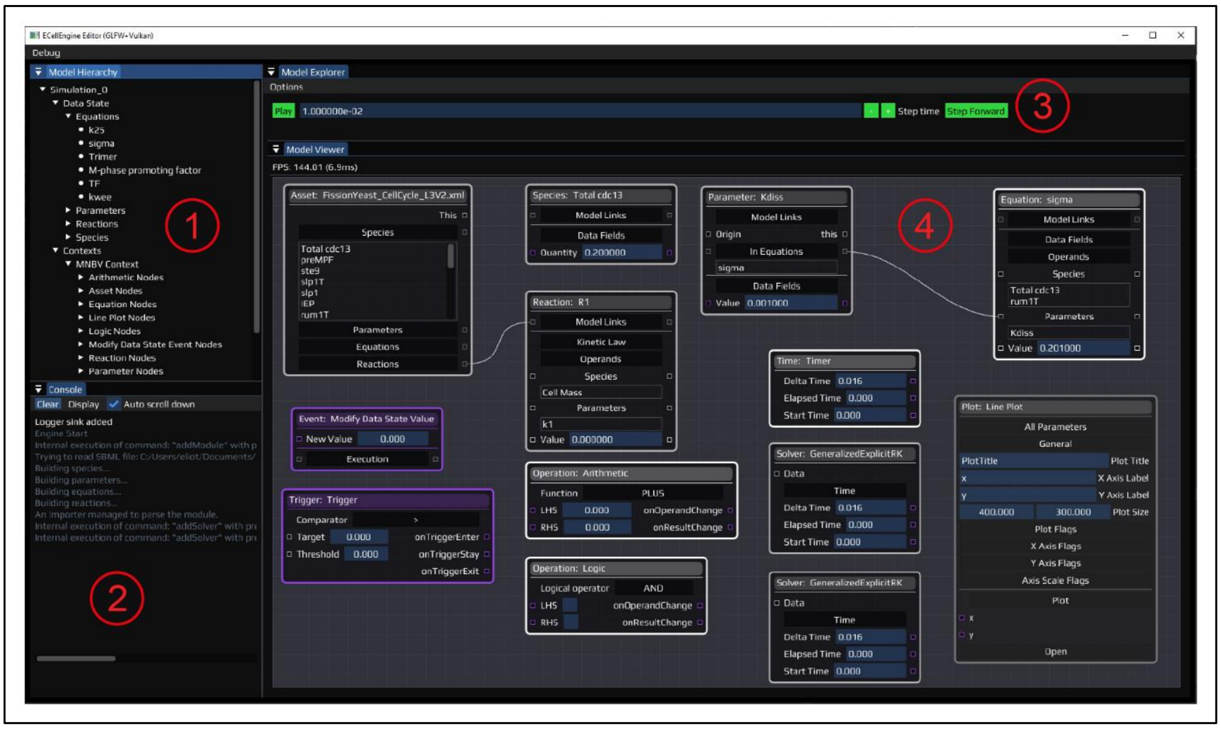
Satisfying Soft Real-Time Systems Requirements to Benefit Model-Based Biology

This section reports on the efforts to implement the user requirements discussed in the previous sections and their effects on practicing MBB.

User experience for iterative biological model design

I implemented a GUI editor for *ECellEngine* to effortlessly manipulate the RT simulation engine before and during simulations. I designed it to report useful information in real-time to help modelers take decisions regarding the structure of the model. The editor notably includes a graphic scripting interface inspired by the node-based scripting interfaces in GEs[32]. Nodes in this interface can encapsulate biological concepts, code routines or visualization components to directly modify or probe the simulation (see the nodes in Figure 16, point 4).

[32] It is also known under the term of flow-based programming.



```
Logger sink added
Engine Start
Internal execution of command: "addModule" with p
Trying to read SBML file: C:/Users/eliot/Documents/
Building species...
Building parameters...
Building equations...
Building reactions...
An importer managed to parse the module.
Internal execution of command: "addSolver" with pr
Internal execution of command: "addSolver" with pr
```

Figure 16: Screenshot of the editor of ECellEngine. At point 1 is a hierarchy exploratory of the model data loaded and the other nodes in the simulation space. Point 2 is a text console for message outputs. Point 3 are the controls to launch, pause, stop the simulation, and manipulate the time axis. Point 4 is the node-based scripting language to edit simulations and model.

The sections *COPASI vs. ECellEngine* page 93, and *CellDesigner vs. ECellEngine* page 93 give some elements of comparison with already existing well-known GUI in systems biology. In *ECellDive*, iterative model design benefits from the support for real-time and asynchronous collaboration inspired from the Metaverse paradigm. The virtualization of the collaboration environment and the VR immersion gives all collaborators the opportunity to concurrently engage in the construction of a model while still retaining a sense of presence of the other collaborators due to the use of avatars. In addition, as the modification files are stored on the server and not locally on one user's device, anyone can asynchronously access and further build upon the modifications or backtrack at any time. Finally, it is possible to apply multiple modification files. This last point is similar to layers in image editing software; a modification file is a layer that is applied onto the original data and layers can be combined thus allowing mixing of ideas when testing variations of a model. This solution is possibly not as powerful as the RT branch pull/push collaboration method I mentioned in the paragraph "About (5) collaborative" but is still versatile enough to distribute and integrate work among collaborators.

Heterogeneity | My approach to handle heterogeneity in each software was to offer a set of built-in representations of data, biological concepts, and functions that modelers could add to a simulation space and compose together to modify or probe the content of a model. Table 3 gives the list of the nodes that can be used in the scripting interface of the editor of

Category	Node	Description
Asset	Equation	Gives access to the root of the tree encoding an equation of a model.
	Model	Encapsulates all equations, parameters, reaction, and species encoded in a SBML model.
	Parameter	Gives access to the float value of a parameter of a model.
	Reaction	Gives access to the reactants and products of a reaction of a model.
	Species	Gives access to the quantity of a species of a model.
	Value Float	Defines an arbitrary float.
Event	Modify Value in `DataState`	Events to change a float value in the `DataState` is a condition is satisfied.
	Trigger	Watch conditions and propagates the result.
Math	Arithmetic operation	Performs addition, subtraction, multiplication, or division on a pair of floats.
	Logic Operation	Performs AND, OR on a pair of Booleans.
Plot	Line Plot	Plots data streams as lines. Requires a unique stream for the X axis. Multiple streams can be received for the Y axis.
Solver	Gillespie Next Reaction Method	The corresponding algorithm. It receives a model as input.
	ODE Explicit Runge-Kutta	Implements a DOPRI5(4). It receives a model as input.
Time	Simulation Time	The data structure used to advance the time in the simulation space. It might be different from the time of the solvers.

Table 3: Nodes available in the scripting interface of the editor of ECellEngine

ECellEngine, and Table 4 gives the list of the modules that can be added to the virtual world in *ECellDive*. The category "Asset" for the editor of *ECellEngine*, are the nodes giving access to the corresponding data structures on the side of the simulation engine. The nodes in this category are slightly biased toward the SBML format due to its popularity in systems biology, but this is not a fatality. These nodes allow to break free from the classic table views in mainstream software. Modelers can compose the asset nodes with others from the categories "Event", "Maths", or "Plot". The purpose of these compositions is to automatically detect state changes in the model, to react to them, and to visualize them. The

Type	Name	Description
Data Module	CyJson Graph	Encapsulate the view file of a graph using the format CyJson from the software *Cytoscape*.
Local Action Module	Dive Travel Map	A graph representing how a diver navigated in the dive scenes.
	Group By	Automatically make colored groups according to metadata of the elements loaded in the scene.
Server Action Module	API Checker	Checks that a *Kosmogora*-like server implements the HTTP API requested for ever server action modules in *ECellDive*.
	Flux Balance Analysis	Requests an FBA for the model currently loaded in the dive scene.
	Modification Handler	Imports and saves modification files associated to a model loaded in one of the dive scenes.
	Reaction Info Query	Queries reaction information from remote database.
	Remote Importer	Import view files stored in *Kosmogora*.

Table 4: Modules available in ECellDive.

modules in *ECellDive* were not built with composition in mind. Instead, I focused on the benefits of a 3D immersive virtual world to make them mimic tangible 3D objects that I are used to manipulate in reality. In UI/UX research, the notion of tangible UI (Ishii and Ullmer, 1997) describes objects in reality that may modify the virtual world. In VR, tangible UI is approached by associating 3D shapes to functions and haptic feedback in the controllers. 3D objects give a sense of presence, which I believe is better adapted to biologists because they are used to working with physical tools in laboratories. Therefore, the heterogeneity is dealt with by adding tools in the virtual world that correspond to a set of functions.

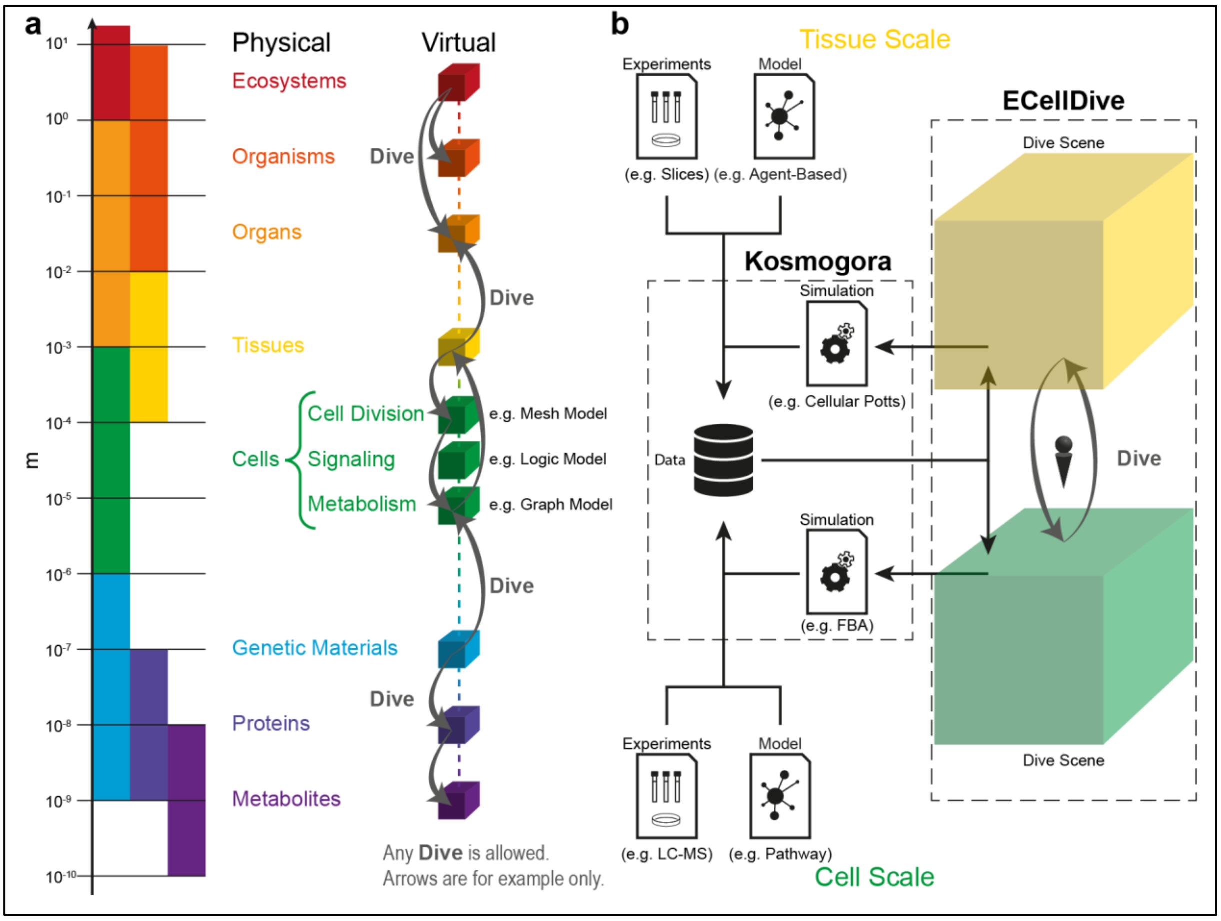


Figure 17: Diving between scenes. a) Using the scale hierarchy of biological subfields for the mental model of the architecture of data and knowledge in a metaverse of systems biology. This aims to help the user understand where they stand in the metaverse of biology similarly to a world map. b) Zoom of the relationship between data centralization and simulation in Kosmogora and its access in ECellDive in conjunction with the user's movement between dive scenes. The data imported into the dive scenes make them fit either the tissue or cell scale.

Integrity | The architecture of *ECellEngine* or *Kosmogora* + *ECellDive* was adapted in order to guarantee the integrity of data and simulations. First, I decoupled the simulation loops from the rendering loops to reduce the computational impact of the latter on the accuracy of the former. Second, the format of the modification files implemented in *Kosmogora* + *ECellDive* participates to maintaining the integrity of the Metaverse. The modification files allow storing information about a model while being disjoint of the original model file so, there is no risk to corrupt it. Users are then free to try many modifications and to build many alternatives of the model in a non-destructive way. Third, *ECellDive* implements protections for the integrity of the mental model of biologists in a metaverse of biology. Knowledge in biology is indeed very diverse and spread across subfields targeting different scales of life. With ecosystems at the top within the order of meters, and metabolites at the bottom within the order of nanometers, it is hardly possible to be up to date on every subjects. To answer issue, I simply followed this scale hierarchy to navigate biological data and knowledge. In *ECellDive*, I project the physical targets of biology to virtual levels that I call "dive scenes." This does not match the way data are stored within *Kosmogora* or *ECellDive* but rather, dive scenes are concepts to help biologists build a mental model of the biological landscape they are exploring. There can be as many dive scenes as needed, and they are not preset for ecosystems or metabolites; a dive scene becomes what users import in it. Figure 17.a shows three examples of phenomena at the scales of cellular biology (cell division, signaling, metabolism), which could be attributed to three independent dive scenes to separate modeling approaches (3D mesh model, logic model, and network model). The action of moving from one dive scene to another is a "dive". This allows for a swift change of scales to facilitate knowledge connections between the different subfields of biology. Users can then access knowledge outside of their area of expertise.

This is illustrated in Figure 17.b where a user is moving from a dive scene containing data and a model about a tissue, to another dive scene containing data and a model about a cell. All data transit through *Kosmogora* and can originate from experiments or simulation results hosted on online databases. I implemented a "Dive Travel Map" in *ECellDive* to display the dive scenes a user has visited, and in which order much like in Figure 17.a. This travel map can help users build and maintain their mental model of the biological knowledge they are exploring. In the context of MBB, I believe these solutions help modelers understand where their models stand in the landscape of biological knowledge even if they are not directly relying on experiment data.

Traceability | I provided support in *Kosmogora* + *ECellDive* for all three types of traceability mandatory for a scientific metaverse. Temporal and differential traceability are enabled in *Kosmogora* thanks to the fields recording authorship, date, and lineage in the YAML. In addition, as modifications are stored in a list, it is easy to compare multiple modification files; thus, leveraging differential traceability. RT traceability is enabled thanks to the RT immersive collaboration in *ECellDive*. Compared to the other two traceability types, there is not data stored on *Kosmogora* or *ECellDive* for RT traceability. Rather, real-time traceability arises as soon as two or more users are following each other's actions and is contingent to mutual awareness (Biocca *et al.*, 2003). It follows from their discussions, actions, and decisions about the data. In *ECellDive*, real-time traceability is made possible by the fact that a user can see others' movements; all users can see others interact with data modules; all users can see others make groups of modules.

Interactive simulations

ECellEngine and *ECellDive* are both running within RT loops with small deadlines, henceforth producing interactive simulation spaces. *ECellEngine* includes RT versions of the classic numeric solvers for ODEs and SSs. It is then possible to feed in the system of rate functions of a biological model and to simulate it using any of these two schemes. Adding model definitions and choosing which solver to user to simulate the models is made as simple as connecting two nodes in the scripting interface of the editor. The content of a model can also be unfolded to have access to low-level variables, especially their quantities, which can be further connected with mathematic operation nodes, plotting nodes or event nodes. This versatility facilitates the manipulation of the structure of the model as I was initially aiming for in the context of MBB. As discussed in the section on user requirements, *ECellEngine* also allows to manipulate the time axis to the extent of playing, pausing, resuming, and stopping the simulation. Finally, nodes in the scripting interface can be added and removed even when the simulation is running to avoid having to stop the simulation every time modelers wish to edit the model.

In *ECellDive*, the interactivity with the 3D virtual world is guaranteed by the fact that I used Unity to develop it. The interactivity of the biological simulation is more complicated to achieve however because, as I decided to use external computation resources to perform the simulation, there always is a bit of latency ensued by the network communications with *Kosmogora*. Currently, the heavier task of performing a FBA on a network of a couple hundred reaction takes a bit less than one second to be visualized in *ECellDive*. I consider this latency acceptable for now and I know there is plenty of room for optimization if it becomes an issue in the future.

Integrated analysis

In *ECellEngine*, immediate analysis is powered by the trigger and event nodes which allow to automatize the detection of relevant state changes in the model. Numeric feedback is provided thanks to RT plots updated with the data provided by the modeler. RT plots are embedded in nodes in the scripting interface of the editor and can easily be set up by connecting a stream of data for the X and Y axis. Currently *ECellEngine* only uses the line plot of the library *implot*[33] compatible with *Dear ImGui*. However, I carefully laid the groundwork to be able to include more graphs in ulterior versions of the editor. Thanks to these integrated analysis tools, modelers do not need to export the simulation's output to start learning about the behavior of their models. This also participates in increasing the speed of model design.

In *ECellDive*, the VR immersion pushes the boundary of integrated analysis a step further. Virtual environments of metaverses are infinite and, regardless of the size of the real-world room the user is in, he can cross kilometers virtually without moving physically. Although this "infiniteness" is similar to the panning of a 2D canvas on a monitor such as for the scripting interface in the editor of *ECellEngine*, the "infiniteness" of a VR environment is more versatile. For example, users are not constrained to a third-person view when looking at data encapsulated in a plot anymore. Instead, data can be mapped in the whole virtual environments to build a landscape users can explore in first-person view. In a "landscape of data" extrema would appear as "landmarks" similar to mountains on the horizon (see in Figure 14, page 79).

For both the editor of *ECellEngine* and *ECellDive*, the explicit presence of a rendering step in the RT loop encourages to use animations. This is especially true for ECellDive since it was implemented with Unity and any GE provide some support for RT visual effects. Thus, virtual animations are customary in 3D virtual worlds to help users and to add contextual

[33] Available at: https://github.com/epezent/implot

information. Continuing with the metaphor of a landscape, exploration can be tedious without a map or a compass. In the case of a “landscape of data” animated virtual objects (i.e., compass) would guide users toward noteworthy regions of the landscape. The core concepts for efficient data visualization (e.g., colors, shape, size, layout, …) on a PC monitor are still valid in a virtual world but we gain degrees of freedom due to the infinite space, the third dimension, and the real-time component. In *ECellDive*, animations are used as a proxy to numeric values corresponding to fluxes (i.e., after an FBA).

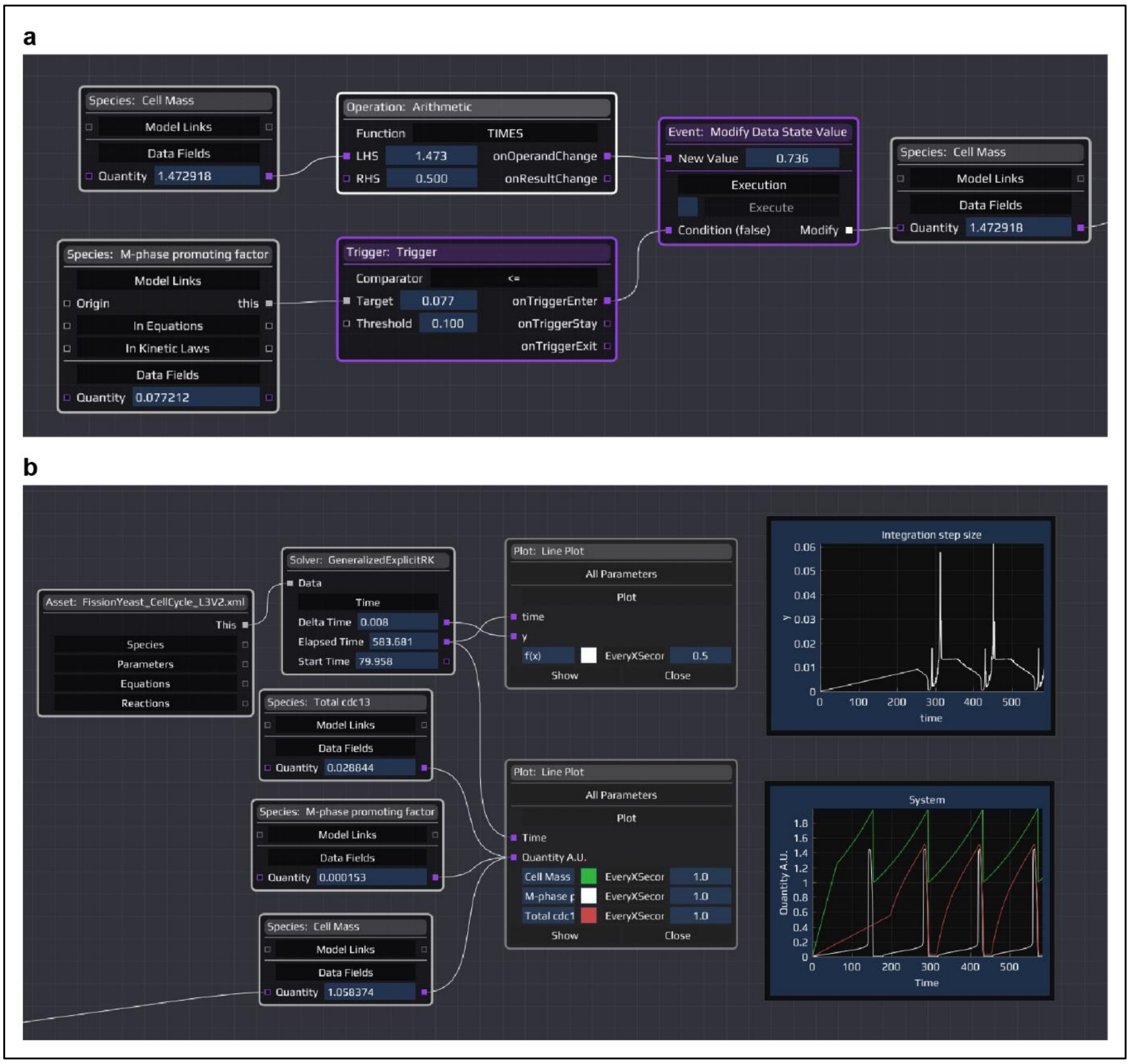
Soft Real-Time Systems in Action for Model-Based Biology

This section contains user examples for both *ECellEngine* and *Kosmogora* + *ECellDive*. For the former, I used a published model to show how it can be explored and modified with the node-based scripting API. In the latter, I used a model of the central metabolism of *Escherichia coli* to demonstrate the usage of the FBA module.

A published model with ECellEngine

The model I selected for this demonstration is an ODE encoding of the regulation by cyclin-dependent protein kinases of the division cycle of fission yeast (Novak *et al.*, 2001). The model consists in a set of nine ODEs, six assignment rules, ten species, 52 parameters, to represent 19 reactions. This is a fairly small model which still can mobilizes most data structures and features currently implemented in *ECellEngine* and its editor.

Events are not yet automatically extracted from the SBML files in *ECellEngine.* The reason that an event in SBML is a couple of a passing condition and the effects that should follow once the condition is verified. This effectively translates as a set of nodes in the node-based scripting language of *ECellEngine*. The graph and the nodes are indeed an editor-only construct and not an engine data structure. Currently (and I don’t see reasons to change in the near future), a node in the editor corresponds to one data structure in the engine. Hence, if there are no data structure in the engine side to encapsulate all the data of a component of a model, I need to create a new one. Creating new data structures is fortunately not an issue in the current architecture—I did my best to make it that way—but I have to be careful of which data structures I decide to add to not bloat the scripting API and complexify the maintenance of the code base. In the current case, I prefer to avoid having one data structure that specifically represents SBML encodings of events. Instead, we can manage to translate many events from various model files in biology if we implement two nodes, one for the condition checking, and one for the effects to apply. The inconvenient of that solution is that there may be uncertainties regarding how the detection of the event and the application of the effects are connected (see in Figure 18.a that the *Trigger* node has several possible outputs. So, in the current implementation, events must be added to the simulation space manually after importing a model in *ECellEngine*.



a

Species: Cell Mass
Model Links
Data Fields
Quantity 1.472918

Operation: Arithmetic
Function TIMES
LHS 1.473 onOperandChange
RHS 0.500 onResultChange

Event: Modify Data State Value
New Value 0.736
Execution
Execute
Condition (false) Modify

Species: Cell Mass
Model Links
Data Fields
Quantity 1.472918

Species: M-phase promoting factor
Model Links
Origin this
In Equations
In Kinetic Laws
Data Fields
Quantity 0.077212

Trigger: Trigger
Comparator <=
Target 0.077 onTriggerEnter
Threshold 0.100 onTriggerStay
onTriggerExit

b

Asset: FissionYeast_CellCycle_L3V2.xml
This
Species
Parameters
Equations
Reactions

Solver: GeneralizedExplicitRK
Data
Time
Delta Time 0.008
Elapsed Time 583.681
Start Time 79.958

Plot: Line Plot
All Parameters
Plot
time
y
f(x) 🟩 EveryXSecor 0.5
Show Close

Species: Total cdc13
Model Links
Data Fields
Quantity 0.028844

Species: M-phase promoting factor
Model Links
Data Fields
Quantity 0.000153

Species: Cell Mass
Model Links
Data Fields
Quantity 1.058374

Plot: Line Plot
All Parameters
Plot
Time
Quantity A.U.
Cell Mass 🟩 EveryXSecor 1.0
M-phase p ⬜ EveryXSecor 1.0
Total cdc1 🟥 EveryXSecor 1.0
Show Close

Figure 18: A model of the division cycle of fission yeast (Novak et al., 2001) in ECellEngine. a) Focus on the construction of the event defined in the model using the node-based scripting interface. b) Focus on the visualization of the outputs of the simulation using the node-based scripting interface.

The two events in the model are defined as follows $if\ flagMPF == 1\ \&\&\ MPF \leq 0.1, then\ \left\{CellMass \longleftarrow \frac{CellMass}{2}, flagMPF \leftarrow 0\right\}(E1)$ and the second event is $if\ MPF > 0.1, then\ \{flagMPF \longleftarrow 1\}\ (E2)$. Even if I wrote the events using the $if/then$ syntax, it is not to be interpreted that the events are triggered as long as the conditions are true. In fact, these events should trigger a reaction only for the one step the condition became true. This corresponds in *ECellEngine* to using the *onTriggerEnter* as the output of the *Trigger* node. In the original model the usage of the parameter $flagMPF$ is a modeling trick to protect from triggering $(E1)$ very early in the simulation (within the first 2 min). This is indeed not something we are looking for with the initial conditions $[MPF]_{t=0} = 0.2$ and $CellMass_{t=0} = 1$. If the authors didn't make use of the $flagMPF_{t=0} = 0$, the value of $CellMass$ would be set to 0.5 very early on in the simulation. This does not seem consistent in the biological context of a fission yeast. It is indeed more natural to consider that

$CellMass = 1$ for one yeast cell and $CellMass = 2$ for 2 yeasts cell, and not 0.5 and 1 respectively. So, in essence, $flagMPF$ is only useful for the very beginning of the simulation.

In *ECellEngine,* however, I could ignore $flagMPF$ by simply connecting the *Trigger* node to the *ModifyDataStateValue* event shortly after $[MPF]_t$ has decreased below the threshold 0.1. This is possible only because I am using an RT simulation environment: everything does not have to be set and perfect at the beginning. I agree, however, that it might not appear rigorous enough for a scientific experiment. But, even though the feature is not yet implemented in *ECellDive*, this kind of actions will be recorded in the scenario data structure I mentioned in the section *User requirements: build, play, learn* page 64 (in "Learn") to not forget it and so that anyone can replay the exact procedure. The *ModifyDataStateValue* is responsible for triggering the division of $CellMass$ by two. In input, it takes the signal of the Trigger; but also, the result of the actual arithmetic operation $0.5 * CellMass$. This value is then forwarded to the $CellMass$ *Species* data structure in the simulation engine via the connection to the input pin of the node.

As it can be observed in the Figure 19.a, there are multiple nodes that represent the value of $CellMass$ (top left and far right). In fact, the output of the *ModifyDataStateValue* event node could have been connected to the input of the *Species* $CellMass$ node on the left side of the graph. But it would have lacked visibility. Figure 19.b shows the part of the graph I added to visualize the results of the simulation. In particular, the line plot on the bottom right corner shows $[CellMass]_t$ in green, the $[MPF]_t$, and another variable, $[CDC13^T]_t$, that I added by curiosity. The straight line for the early time of $[CellMass]_t$ between $t = 0\,min$ and $t \sim 50\,min$ is an artifact due to the addition of $[CDC13^T]_t$ to the plot when the simulation had already been running for some time. Also, the original unit the time variable in the model is *minutes*, but I accelerated the simulation such that $1sec$ of real-time corresponds to $1min$ of simulation of the system to avoid having to wait the

Finally, I plotted the evolution of the variable integration time step size on the line plot on in the top right corner of Figure 19.b**.** This is a demonstration of the monitoring of the quality of the simulation. We can observe on the plot that the integration step size (i.e. *Delta Time* from the ODE solver node) is the smallest when there are sharp variations in the ODE system (about every 140 min), which is good news because it means the integration system "is taking more precautions in complicated parts of the simulation" (of course, this is just an image).

A flux balance analysis with Kosmogora + ECellDive

I used a published model (King *et al.*, 2015; Rowe *et al.*, 2018) (iJO1366) to perform an FBA (Varma and Palsson, 1994; Orth *et al.*, 2010; Edwards *et al.*, 2002), a common method in systems biology to analyze the theoretical throughput of metabolic pathways that are in a steady-state. I retrieved the central metabolism of *Escherichia coli* downloaded from Escher (King *et al.*, 2015; Rowe *et al.*, 2018) and stored it in *Kosmogora*. From there, I imported it into *ECellDive* as a data module and dived into it (see Figure 19.a).

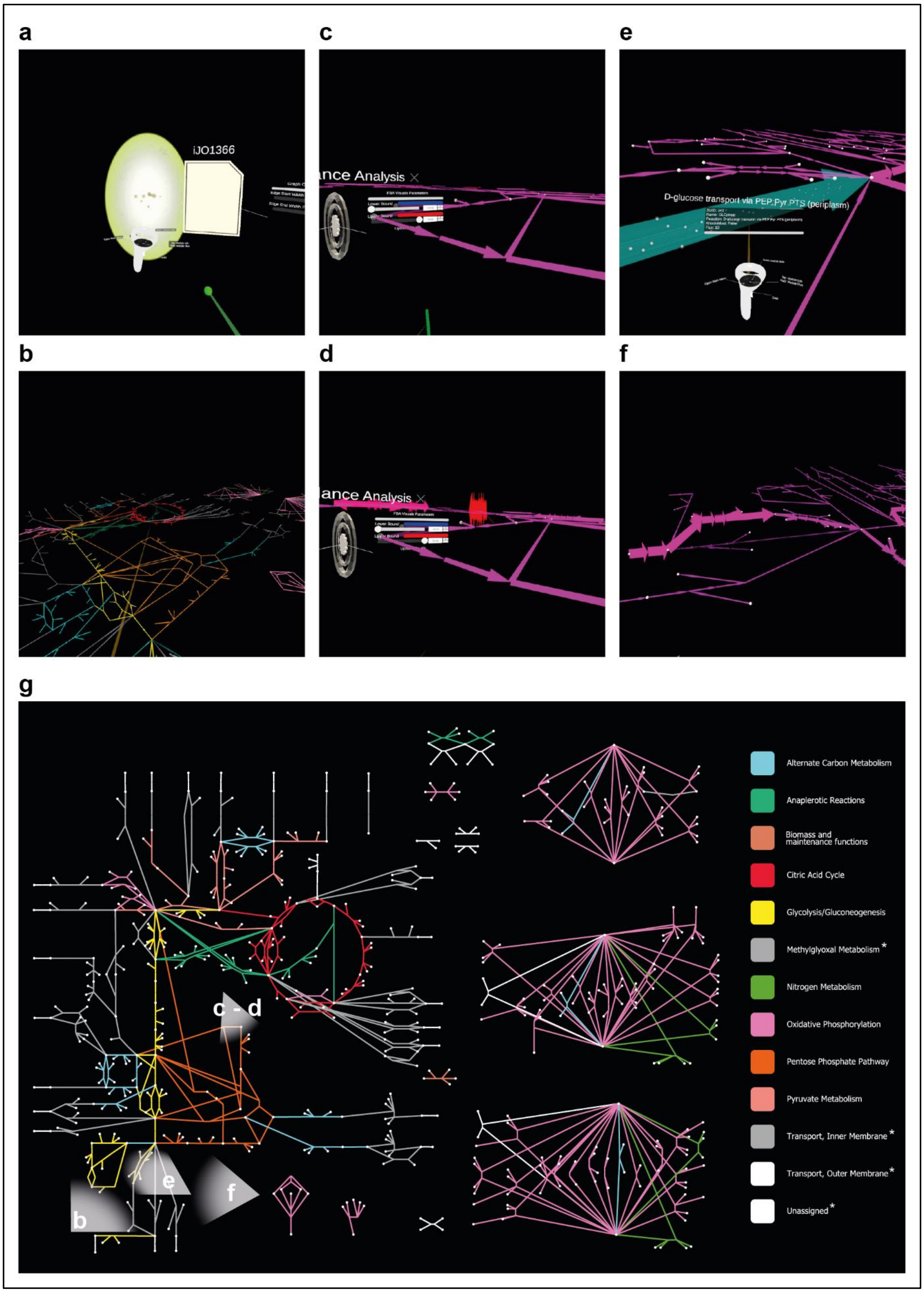

a
c
e
b
d
f
g
iJO1366
ance Analysis
D-glucose transport via PEP:Pyr PTS (periplasm)
ance Analysis
c - d
e
f
b
Alternate Carbon Metabolism
Anaplerotic Reactions
Biomass and maintenance functions
Citric Acid Cycle
Glycolysis/Gluconeogenesis
Methylglyoxal Metabolism *
Nitrogen Metabolism
Oxidative Phosphorylation
Pentose Phosphate Pathway
Pyruvate Metabolism
Transport, Inner Membrane *
Transport, Outer Membrane *
Unassigned *

Figure 19: Illustration of the main steps of a flux balance analysis in ECellDive on model iJO1366. a) The portal used to dive into the representation of iJO1366 encoded in the view file b) A high view of the pathway after color customization of the edges to group them according to the main subsystems of this metabolism. c) We performed a Flux Balance Analysis (FBA); the minimal flux value is -45 and the maximal value is +55. Colors interpolate between blue for low values and red for high

*values. Most of fluxes have value 0 (hence, the pink). d) Manipulating the width of the edges is much more efficient than relying on colors only. VR immersion enables the concept of "landmark" facilitating the spotting extrema as we would for mountains in nature: the red fluxes are clearly visible. e) Zoom on a flux; the D-glucose transport through the periplasm. The FBA gave it a value of 10. The white points on the images are animated particles which debit is proportional to the value of the flux. f) We knocked out the D-glucose transporter and the flux was redirected. g) An overview of iJO1366's network made in Cytoscape (Shannon et al., 2003). This is not available in ECellDive. It is provided here only to help locate where the user and its field of view in panels a-d. *Colors for the groups were taken to match the situation in panel b), the color conflicts were a choice of the user at the time.*

In this example, two dive scenes are defined by the root space where the user is dropped when launching *ECellDive*, and the scene defined by the metabolic pathway.

By default, the metabolic pathway is mapped on a plane. Biologists are familiar with seeing metabolic maps in 2D, and typically newcomers expect the TCA cycle to appear as a circle because that is how it is taught in textbooks. Familiarity with the visualization scheme is an important consideration that must be considered to ease the transition between traditional desktop applications and virtual worlds. Once users are ready, local action modules can customize the layout of the imported biological system.

For example, when the *groupby* action module is added to the *dive scene*, it automatically detects the parts of the model that can be grouped according to their metadata. For a metabolic pathway, the objects suitable for automatic grouping are the metabolites and the reactions. Users can then group edges to highlight functional subsystems of the pathway (e.g., TCA cycle in red in Figure 19.b) and nodes to understand their positions in cellular compartments (e.g., cytosol).

Users can also interact with an action module to perform FBA remotely on *Kosmogora*. After the FBA is processed by the server, the fluxes values are mapped to the color or width of the edges of the metabolic pathway (see Figure 19.c & Figure 19.d).

This is similar to the conventional visualization in applications running on a computer (King *et al.*, 2015; Rowe *et al.*, 2018), but large flux values are now akin to mountains on the horizon due to the immersion provided by a VR device; this illustrates the concept of "landmarks." sers can clamp the width of edges to balance between spotting individual fluxes and global readability. Flux values can also be visualized dynamically due to animated particles (see Figure 19.e), which are instantiated on edges in proportion to each flux value to map the flow rate of particles to the quantity of metabolites involved in the reaction symbolized by an edge. Finally, the numerical value of the flux can be obtained by opening the information panel attached to every edge. Users can interact with reactions to simulate knockout experiments, which can influence the FBA and reroutes the fluxes (see Figure 19.e & Figure 19.f).

Comparisons

The real-time property of *ECellEngine* and *Kosmogora* + *ECellDive* makes them intrinsically different from mainstream modelling software in biology so it is hard to fairly compare them together. The examples of software in Table 5 and Table 6 do overlap on the basis of the application targets, but the practices are radically different. It is probably more appropriate to think of my RT systems as a parallel solution. Be that as it may, I give some elements of comparison in the next four sections.

COPASI vs. ECellEngine

COPASI is very well established in the community of systems biology to simulate and edit models of bio-chemical networks. It is currently in its fourth major version and provides reliable services for time course simulations, parameter optimization, sensitivity analysis, model building, plotting, and so on. Supplementary Figure A2.2 gives two screenshots of the GUI showing the interface to setup a plot and perform a simulation of the cell division cycle model of fission yeast (Novak *et al.*, 2001) . The model is identical to the one used previously in section *A published model with ECellEngine* to illustrate the node-based scripting interface of *ECellEngine* on a real case. Clearly, *ECellEngine*'s approach to interacting with the model has nothing to do with the tabular view of COPASI. In the latter, it takes several clicks to reach the plot panel, add new plots, new curves, select when we want to draw it, decide of the X and Y axis, validate the selection then, finally, go back on the tab to run the simulation. On the contrary it is a matter of few *click & drag* in *ECellEngine*. Moreover, it is disturbing that one must leave the options of the simulation to edit the options for plotting. In my understanding, COPASI is very good when it comes to having the best of individual methods to solve modeling problems one-by-one. The goal is not to integrate seamlessly the pipeline but to regroup under a unique platform a tool suite for many frequently used modeling techniques. Given that the trend towards larger biological models requiring the integration of multi-scale or multi-type simulation techniques, it is possible the popularity of COPASI will progressively decrease in the future.

CellDesigner vs. ECellEngine

CellDesigner overlaps with *ECellEngine* as it also uses visual elements to support the construction and editing of biological systems. The intention of CellDesigner is to consider the biological systems as engineering processes which can be accurately described using the same process diagrams as for engineering tasks. The result of this approach is a software where designers can build network schematics representing a cellular system. Supplementary Figure A2.2.a gives a screenshot of the automatically generated schematics after I opened the cell division cycle model of fission yeast (Novak *et al.*, 2001) that I used previously. CellDesigner is an excellent tool to match the mental model of modelers to a machine-readable file format after translation by CellDesigner. The tool also allows to run simulation by connecting to one of three simulation backends (including a wrap

Name	Software Type	Field/Subject	Description
CellDesigner **(Kitano *et al.*, 2005)**	GUI	Cellular Biology Modeling	Generating & modifying biological models' interactive schemas
COPASI **(Hoops *et al.*, 2006)**	API + GUI	Bio-Chemical Networks	Many methods to solve problems associated with bio-chemical networks (simulation, parameter optimization, …)
E-Cell Project **(Takahashi, Sakurada, *et al.*, 2003; Kaizu *et al.*, 2020)**	API	Multi-Scale Modeling	A software interface for multi-simulations (rule-based modeling, 2D/3D particle simulations, …)
Vivarium **(Agmon *et al.*, 2022)**	API	Multi-Scale Modeling	A software interface for multi-simulations (constraint-based, bio-chemical reactions, solid-body physics, …)
SOFA **(Faure *et al.*, 2012)**	API + GUI	Physics-Based Simulation	Interactive physics simulations of 3D rendered models

Table 5: Software whose objectives or technologies overlap with ECellEngine.

of COPASI, see Supplementary Figure A2.2.b). However, the modeling approach in CellDesigner is strictly descriptive. It differs from *ECellEngine* in that it is not a scripting language, and you cannot mix the model description with logic gates, code routines, or dynamic visualization.

Nanome vs. Kosmogora + ECellDive

Nanome is VR collaborative software to study molecular dynamics, protein-ligand interactions and dockings. It is certainly more advanced than *ECellDive* from an aesthetic perspective and the RT human interactions in a virtual workroom are more elaborated thanks to better avatars and gestures. However, my understanding of this software is that its outreach of functions and domain of application makes it a translation of what equivalent standalone software on PC. Of course, *Nanome* greatly benefit from VR and immersion for the visualization molecule interactions compared to a 2D monitor. But there does not seem to be a reflection about what it implies to practice this kind of activity in the greater frame of the Metaverse. That is why I have, along other examples of software listed in Table 6, labelled it as *non-Metaverse-ready*. Given that the Metaverse is still only a future construct, I qualify a software to be *metaverse-ready* if it implements features to account for some of the requirements of the Metaverse such as I discussed in the paragraph "*Proposition of 7 requirements for a scientific metaverse*". In that respect my couple of software *Kosmogora + ECellDive* is closer Nvidia's *Omniverse*.

Name	*Actor*	*Platform*	*Field/Subject*	*Metaverse -Ready*[1]	*Description*
Nanome (Nanome Inc, 2023)	*Industry*	*VR*	*Biology, Chemistry, Pharmaceutics*	*No*	*A collaborative tool for molecular design and analysis*
VR-Omics (Bienroth *et al.*, 2023; Ramialison *et al.*, 2023)	*Academia*	*Desktop 2D & 3D + VR*	*Omics data visualization*		*Desktop to VR pipeline integration for omics data processing and visualization*
Molecular Rift (Norrby *et al.*, 2015)	*Academia*	*VR*	*Biology, Chemistry, Pharmaceutics*		*A tool for molecular structure visualization*
Cytoscape (Shannon *et al.*, 2003)	*Academia*	*Desktop 2D & 3D (via plug-in)*	*Network Data (Biology, Social Sciences, …)*		*Popular tool for display and analysis of network data*
Graphi*a* (Freeman et al., 2022)	*Academia*	*Desktop 2D & 3D*	*Network Data (Biology, Agritech, Social Networks, …)*		*Tool for display and analysis of large-scale network data*
SpaceTim*e* (Xia et al., 2018)	*Academia*	*VR*	*Real-time collaboration*		*Research project for new collaboration schemes and techniques in real-time VR*
DataHop (Hayatpur et al., 2020)	*Academia*	*VR*	*Data visualization*		*Research project for immersed data visualization where plots are laid out according to the analysis steps of the user*
NVIDIA Omniverse (NVIDIA, 2022)	*Industry*	*Desktop & XR*	*Virtual world creation*	*Yes*	*Large development tool suite for engineering-grade digital twins (industrial and scientific) and metaverse applications*
Volvo Truck R&D (Horton and Wurster, 2022)	*Industry*	*Desktop & VR*	*Automobile engineering R&D*		*In-house R&D system for collaborative and immersive design of new trucks*

Table 6: Tools overlapping with concepts also present in Kosmogora+ECellDive.

[1]Metaverse-Ready means that the tool includes metaverse constraints directly in its specifications; it does not mean that the tool is a Metaverse or is future-proof against the evolution of the definition of the Metaverse. For example, Nanome and VR-Omics are very good immersive visualizations, but they do not include concepts in their architecture to integrate with the larger scale of a Metaverse. NVIDIA Omniverse and the Volvo Truck R&D were designed to support this larger scale, which makes them "metaverse-ready" in our definition.

NVIDIA Omniverse vs. Kosmogora + ECellDive

Omniverse is a huge tool suite maintained by Nvidia to help develop the Metaverse. It originally started as collaborative development platform for artists and designers, but it now also includes simulation platforms for digital twins of robots, climate dynamics, computer vision problems, and so on. Of course, the amount of resources invested to develop and maintain *Omniverse* is eon larger than what I could invest alone during my PhD. But it does not change the fact that *Kosmogora + ECellDive* are investigating scenarios and work situations that are also at the heart of the collaborative tools in the ecosystem of *Omniverse.* Notably, it includes research on non-destructive modes of iteration over versions of a project. This problem seems to have been mainly addressed using a file format known as Universal Scene Description (USD) which allows to preserve the trace of versions for 3D models in addition to being fast, and extensible. USD files are solving problems that I addressed via the modification files.

V. Closing Remarks

I opened this dissertation with a critic of the gluttony of biologists for experimental data that, I expect, has likely been met with greater reluctant perplexity than genuine curiosity. At least, this is the prediction based on my few experiences during the three years of my PhD[34] where I was given the opportunity to strike a conversation with experimental biologists in systems biology about this matter. Long story short, it usually led nowhere. Be it for their blindsided behavior and faith in observations, my inability to explain my point of view as to what a theoretical framework for biology would look like, the possible insignificance of the matter, or a mix of these three and beyond, I could never maintain a meaningful exchange for more than a few minutes. The point on which appeared to crystallize all their perplexity was the matter of the verification and validation of a research method that would produce biological knowledge in the absence of the corresponding repeatable observations. From there, any attempts I made to motivate and illustrate that a body of axioms was strong enough to define a self-contained space of validity, never managed to interface with their conception of biology research should be conducted. The idea that unprovable statements could be the foundation to rock-solid biological theories was judged, at best, a pipe dream, at worse, a grave scientific malpractice. Even when I avoided talking about a theory-only biology, and instead focused the discussion on the uncomfortable fact (for me) that over-relying on data was limiting us to answer, "how is this problem being solved?" (i.e., "how-questions"), and ignore "what is the problem?" (i.e., "what-questions"), did not seem to raise much concern. In effect, rather than considering it as a problem that could be fixed, my interlocutors recognized it as a neutral quality of induction in biology. Neither bad nor good, simply a consequence to account for. Then, my arguments about the intractability of reverse engineering in inductive biology was invariably met with something along the line of "ML techniques will solve that". And my final argument about the debatable sustainability of deep ML approaches did not shake the ambient unwavering optimism that the benefits of scientific discoveries outweigh the possible costs for the planet.

Somewhat sadly, I then discovered that it was easier to have meaningful discussions on this topic outside the sphere of influence of systems biology. The first researcher whom I discussed with about the relative weight of theoretical approaches over observations was a physicist's turned biologist for the time of a poster at the International Conference for Systems Biology in October 2022. His research was about the use of maximum entropy principle to explain the position of retinal cones (Beygi, 2023). It turned out easier to discuss with a researcher that was more adept at composing and aggregating abstract layers of knowledge to formulate questions about phenomena. This made me realize that I was probably better off honing my arguments with researchers in fields peripheral to biology.

[34] In fact, it is only 2 years since my first year coincided with the COVID pandemic and I carried out my research entirely from my parent's home in France.

This is how I was introduced to the field of Artificial Life. There, I found, and was recommended, many papers which did not hesitate to cast aside big data to focus on foundational properties of biotic systems thanks to theoretical frameworks (Montévil and Mossio, 2015; Hernández-Orozco *et al.*, 2018; Kauffman, 2020) or *in silico* experiments (Chan, 2020). Astrobiologists and geochemists which research overlaps with the subject of the origin of life are also used to perform a lot of hypothetical reasoning (Bartlett and Wong, 2020; Wong *et al.*, 2022; Kauffman, 2020; Kauffman and Roli, 2023) and to constantly deal with a shortage of data in their field. It is a research practice in a realm of uncertainty and open variables that would, I assume, make system biologists uncomfortable.

Discussions with researchers outside of the field of systems biology has not changed my mind about the long-term inadequacy of induction in biology. If anything, it reinforced my opinion that alternative and complemental practices are possible and a goal worth pursuing. In this thesis, I investigated my alternative of choice, namely model-based biology powered by soft real-time systems. I defended that soft RT systems possess properties inherent to their necessity to execute tasks in a timely manner that can be transferred to modeling in systems biology for the better. Specifically, the possibility to manipulate the time axis to control the flow of the simulation for in-depth analysis of the characteristics of the simulated biological system; the interactivity of the simulations which can be designed to allow dynamic update, creation, and deletion of components for faster design iteration of biological models; and the support of the integration of heterogeneous input data, physical components, and virtual components, for the integration of automated hardware for biology experiments with synchronized simulations. All three of which can be used to accelerate the exploration of the solution space of biological systems by quick trial and errors. The exploration of the solution space can be used to target both “how-questions” and “what-questions”, even though I am more in favor of the latter. I effectively implemented two RT systems during my PhD research to demonstrate my claims. The first, *ECellEngine*, is a soft RT simulation system for biological models. Its main purpose is to benefit from the RT framework while building, simulating, and analyzing plausible biological systems via the node-based scripting interface in the editor. The second, *Kosmogora* + *ECellDive*, is a couple of systems leveraging the benefits of RT collaboration for the iterative design of models of biological systems in a scientific metaverse.

The work presented in this thesis can be extended in many ways. First, neither *ECellEngine* nor *Kosmogora* + *ECellDive*, are perfect and both could benefit from longer development time to make them more accessible, dependable, and versatile. This is particularly true for *ECellEngine* which some of the requirements identified in the section *User requirements: build*, *play*, *learn* page 64 and in Figure 9 page 65 are not yet implemented. This includes the generation of branches for non-destructive simulation alternatives. Such branches would be very useful to protect the integrity of simulation data and compare the design choices made between multiple versions of a biological model.

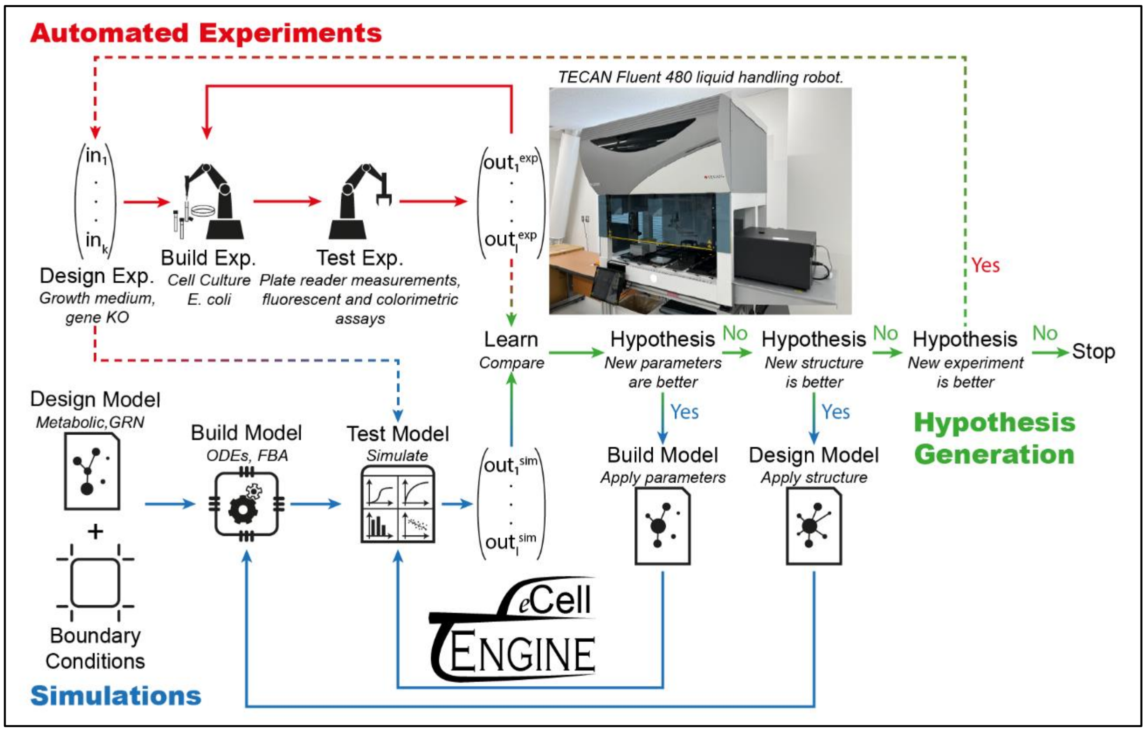


Figure 20: The integration of ECellEngine within a bigger loop involving automated laboratory hardware. The dotted arrows indicate asynchronous communications between the simulation component led by ECellEngine and the physical components.

ECellDive is also currently lacking the possibility to simulate in the opposite direction of the time axis to backtrack simulations to a time of interest for the modelers. Finally, an inconvenient missing piece is the possibility to export edited models into SBML format but also into a non-destructive file format to store the history of the modifications. For this task, I wish to use the USD file format promoted for the description of virtual worlds in the Metaverse by Nvidia in *Omniverse* and other big companies such as *Pixar*, *Google*, and *Apple*. The USD file format is a high-end professional asset and the learning curve is quite steep. Although initially developed to describe alternative versions of 3D environments within a file, the USD library allows to define custom fields. I originally planned to customize USD to export from *ECellEngine* and store them in memory. Unfortunately, I could not finish the integration within the frame of my PhD. This must be achieved to truly make *ECellEngine* usable by other researchers.

Finally, a direction to build on top of my research is to integrate automated biology experiment hardware with the RT simulation software I developed in this research as illustrated in Figure 20. This type of integration would power what is commonly called "*closed-loop AI scientists*" to let autonomous systems **Design → Simulate → Compare → Hypothesize**, and then the cycle restarts. The comparison between the simulation and the experiment outputs would allow to generate hypotheses about the biological system which will likely involve *AI* technologies (hence, the name). Of course, the purpose of this type of integration is to optimize a model to match as much as possible all the observations acquired during autonomous experiments. Given that the update of the model and the acquisition of data are both present in the cycle, it is neither the inductive biology I criticized, nor the model-based biology (i.e., deductive biology) I defended. I hope this could be an intermediate setup where the acquisition of data might be indeed large in the long-term, but, at least, bounded and directed thanks to the model's simulations. Eventually, whether this system leans more toward induction or deduction, will depend on whether the hypothesis generation policy gives more weigh to the experiment results or the model's simulation. The design illustrated in Figure 20 would give more weight to the knowledge acquisition by updating the model first. Indeed, there are two internal loops (in blue) trying to optimize the parameters and the structure of the model, before generating a new batch of experiments (dotted arrow green to red). This way, it is the structure of the model that influences the experiments. It could have been the opposite by giving priority to the exploration of the input parameter space of the experiments (the value of $(in_1, in_2, \dots, in_k)^T$), and only then updating the model after much data has been acquired. In the future, we might observe a gradient between a full inductive and a full deductive *closed loop AI scientist*.

Appendices

A1. Primer on Numerical Integration of Ordinary Differential Equations

A solver for Ordinary Differential Equations (ODEs) was implemented in *ECellEngine* in order to simulate models encoded as ODEs. The following is an summary highly inspired by the two reference textbooks written by Hairer and Wanner (Hairer and Wanner, 1993, 1996). This primer contains the essential of the knowledge I used (and that anyone uses) to implement an algorithm derived from Runge-Kutta's method to numerically solve ODEs.

Newton on Euler's initial value problem

It is known from Newton's work that Euler's initial value problem:

$$y' = f(x, y), \quad y(x_0) = y_0 \quad (1.1)$$

can be solved with better accuracy than Euler's method (error bounded by Ch where C is a constant and h the step size) if f happens to be independent from y. In that case, (1.1) rewrites as:

$$y' = f(x), \quad y(x_0) = y_0 \quad (1.2)$$

and has the solution:

$$y(x^*) = y_0 + \int_x^{x^*} f(x)dx \quad (1.3)$$

A way to integrate from x to x^* is to iterate to successively approximate the values $y_i(x)$ using the midpoint rule $y_i(x_i + h_i) \approx y_i + h_i f\left(x_i + \frac{h_i}{2}\right)$ until we reach:

$$y(x^*) \approx y^* = y_{n-1} + h_{n-1} f\left(x_{n-1} + \frac{h_{n-1}}{2}\right) \quad (1.4)$$

Explicit fixed step-size Runge-Kutta method

Runge extended this method to the problem (1.1), which for the first step and an arbitrary h reads:

$$y_0(x_0 + h) \approx y_0 + hf\left(x_0 + \frac{h}{2}, y\left(x_0 + \frac{h}{2}\right)\right) \quad (1.5)$$

Further approximating $y\left(x_0 + \frac{h}{2}\right)$ with a euler step, we can rewrite the right side of (1.5) as:

$$y_0 + hf\left(x_0 + \frac{h}{2}, y_0 + \frac{h}{2} f(x_0, y_0)\right) \quad (1.6)$$

The iteration form is better visible if we set $k_1 = f(x_0, y_0)$ and $k_2 = f\left(x_0 + \frac{h}{2}, y_0 + \frac{h}{2}k_1\right)$ to obtain $y_1 = y_0 + hk_2$.

Further expanding this s times to approximate y_1 starting from y_0 forms an *s-stage Explicit Runge-Kutta (ERK) method* for (1.1). Generally, the expansion initializes with:

$$k_1 = f(x_0, y_0)$$

$$k_i = f\left(x_0 + c_i h, y_0 + h\left(\sum_{j=1}^{i-1} a_{i,j} k_j\right)\right)$$

and continues as and the approximate solution:

$$y_1 = y_0 + h \sum_{i=1}^{s} b_i k_i \quad (1.7)$$

with the function values $a_{i,j}$, b_i and c_i real coefficients. In some cases, it is useful to assume $c_i = \sum_{j=1}^{i-1} a_{i,j}$.

It is conventional to represent the coefficients as in the table below:

$$\begin{array}{c|ccccc}
\mathbf{0} & & & & & \\
\boldsymbol{c_2} & \boldsymbol{a_{21}} & & & & \\
\boldsymbol{c_3} & \boldsymbol{a_{31}} & \boldsymbol{a_{32}} & & & \\
\vdots & \vdots & \vdots & \ddots & & \\
\boldsymbol{c_s} & \boldsymbol{a_{s1}} & \boldsymbol{a_{s2}} & \cdots & \boldsymbol{a_{s,s-1}} & \\
\hline
 & \boldsymbol{b_1} & \boldsymbol{b_2} & \cdots & \boldsymbol{b_{s-1}} & \boldsymbol{b_s}
\end{array}$$

Appendix Table 1: Generic Runge-Kutta function values tableau.

The variations of Runge-Kutta methods boils down to finding different sets of parameters which will have slightly different effects on the stability or precision of the method at equal s or with higher orders s. Here are the values usually associated to “The” Runge-Kutta method:

$$\begin{array}{c|cccc}
\mathbf{0} & & & & \\
\mathbf{1/2} & \mathbf{1/2} & & & \\
\mathbf{1/2} & \mathbf{0} & \mathbf{1/2} & & \\
\mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \\
\hline
 & \mathbf{1/6} & \mathbf{2/6} & \mathbf{2/6} & \mathbf{1/6}
\end{array}$$

Appendix Table 2: Function values of the classic Runge-Kutta Method (order 4)

Explicit variable step-size Runge-Kutta method

The hope behind variable step size is to make bigger steps when the integration method makes small local errors (the error made when computing y_1) and to make smaller steps after the local error has exceeded a set threshold. In practice, thanks to this approach, we observe that the integration method “goes fast” when the solution does not vary a lot and it “goes slowly” when the solution has sharp variations. Of course, there are some optimizations to avoid growing or shrinking the step size too fast depending on the result. One way to compute the error is simply to compute the difference between two approximations y_1 and $\hat{y}_1$, where one of them is of higher order s. The "trick" to benefit of variable step-size at a low computational cost is to avoid having to compute k_i values twice (for y_1 and $\hat{y}_1$). The idea is then to find function values that can be used for both approximations:

$\mathbf{0}$					
$\boldsymbol{c_2}$	$\boldsymbol{a_{21}}$				
$\boldsymbol{c_3}$	$\boldsymbol{a_{31}}$	$\boldsymbol{a_{32}}$			
$\vdots$	$\vdots$	$\vdots$	$\ddots$		
$\boldsymbol{c_s}$	$\boldsymbol{a_{s1}}$	$\boldsymbol{a_{s2}}$	$\dots$	$\boldsymbol{a_{s,s-1}}$	
	$\boldsymbol{b_1}$	$\boldsymbol{b_2}$	$\dots$	$\boldsymbol{b_{s-1}}$	$\boldsymbol{b_s}$
	$\boldsymbol{\hat{b}_1}$	$\boldsymbol{\hat{b}_2}$	$\dots$	$\boldsymbol{\hat{b}_{s-1}}$	$\boldsymbol{\hat{b}_s}$

Appendix Table 3: Generic Runge-Kutta function values for double approximations

Then, we find y_1 of order p as:

$$y_1 = y_0 + h\sum_{i=1}^{s} b_i k_i \quad (1.9)$$

And $\hat{y}_1$ or order $\hat{p} = p + 1$ or $\hat{p} = p - 1$:

$$\hat{y}_1 = y_0 + h\sum_{i=1}^{s} \hat{b}_i k_i \quad (1.10)$$

A famous set of solutions comes from Dormand-Prince (DOPRI) where $p = 5$ and $\hat{p} = 4$. This is the solution implemented in *ECellEngine*. The function values are:

$\mathbf{0}$					
$\frac{1}{5}$	$\frac{1}{5}$				
$\frac{3}{10}$	$\frac{3}{40}$	$\frac{9}{40}$			
$\frac{4}{5}$	$\frac{44}{45}$	$-\frac{56}{15}$	$\frac{32}{9}$		
$\frac{8}{9}$	$\frac{19372}{6561}$	$-\frac{25360}{2187}$	$\frac{64448}{6561}$	$-\frac{212}{729}$	
$\mathbf{1}$	$\frac{9017}{3168}$	$-\frac{355}{33}$	$\frac{46732}{5247}$	$\frac{49}{176}$	$-\frac{5103}{18656}$

$\mathbf{1}$	$\mathbf{\frac{35}{384}}$	$\mathbf{0}$	$\mathbf{\frac{500}{1113}}$	$\mathbf{\frac{125}{192}}$	$\mathbf{-\frac{2187}{6784}}$	$\mathbf{\frac{11}{84}}$	
$\boldsymbol{y_1}$	$\mathbf{\frac{35}{384}}$	$\mathbf{0}$	$\mathbf{\frac{500}{1113}}$	$\mathbf{\frac{125}{192}}$	$\mathbf{-\frac{2187}{6784}}$	$\mathbf{\frac{11}{84}}$	$\mathbf{0}$
$\boldsymbol{\hat{y}_1}$	$\mathbf{\frac{5179}{57600}}$	$\mathbf{0}$	$\mathbf{\frac{7571}{16695}}$	$\mathbf{\frac{393}{640}}$	$\mathbf{-\frac{92097}{339200}}$	$\mathbf{\frac{187}{2100}}$	$\mathbf{\frac{1}{40}}$

Appendix Table 4: Function values for DOPRI5(4)

The method requires 7 stages to reach order 5.

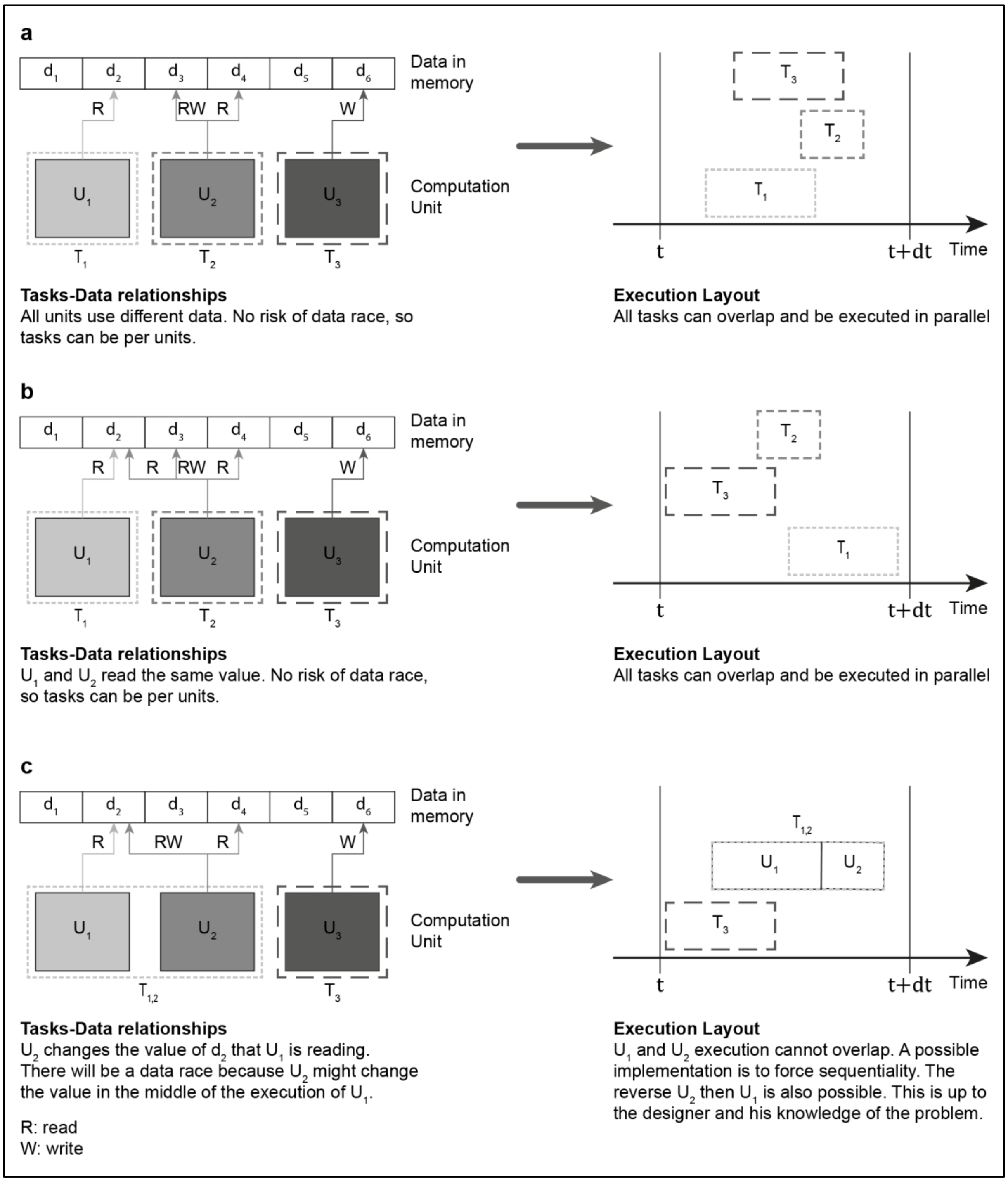
Implicit Runge-Kutta formula

The general expansion defining an *s-stage Explicit Runge-Kutta* system can be further expanded to include terms from stages beyond the current one. This is called an implicit Runge-Kutta method when there exists at least one i such that $\forall(i,j) \in [1,s]^2 \mid i \leq j, a_{i,j} \neq 0$. In effect, it means there are non-zero coefficients above the diagonal of the function values table. The final form for the k_i that includes both the explicit and implicit definitions up to stage s is:

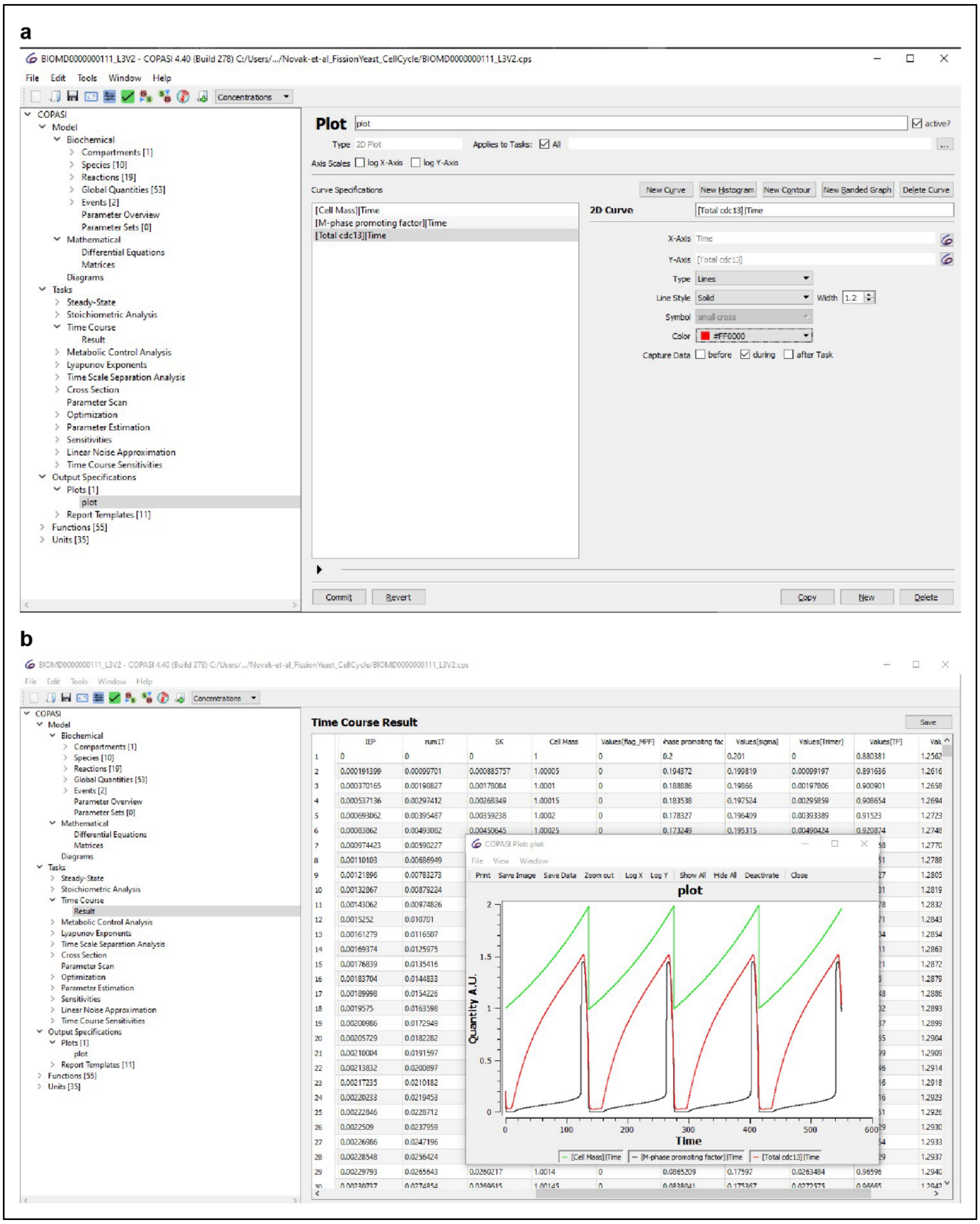
$$k_i = f\left(x_0 + c_i h, y_0 + h\left(\sum_{j=1}^{s} a_{i,j} k_j\right)\right) \quad (1.11)$$

It can be proven that implicit methods have a much larger stability space that explicit methods. A method is stable when the numerical solution does not exhibit oscillating or explosive behavior. Implicit methods are also more accurate than explicit ones. In comparison, explicit methods are less computationally expensive and easier to implement.

A2. Supplementary Figures


a
d_1 d_2 d_3 d_4 d_5 d_6
Data in memory
R RW R W
U_1 U_2 U_3
Computation Unit
T_1 T_2 T_3
Tasks-Data relationships
All units use different data. No risk of data race, so tasks can be per units.
T_3 T_2 T_1
t t+dt Time
Execution Layout
All tasks can overlap and be executed in parallel
b
d_1 d_2 d_3 d_4 d_5 d_6
Data in memory
R R RW R W
U_1 U_2 U_3
Computation Unit
T_1 T_2 T_3
Tasks-Data relationships
U_1 and U_2 read the same value. No risk of data race, so tasks can be per units.
T_2 T_3 T_1
t t+dt Time
Execution Layout
All tasks can overlap and be executed in parallel
c
d_1 d_2 d_3 d_4 d_5 d_6
Data in memory
R RW R W
U_1 U_2 U_3
Computation Unit
$T_{1,2}$ T_3
Tasks-Data relationships
U_2 changes the value of d_2 that U_1 is reading. There will be a data race because U_2 might change the value in the middle of the execution of U_1.
R: read
W: write
$T_{1,2}$ U_1 U_2 T_3
t t+dt Time
Execution Layout
U_1 and U_2 execution cannot overlap. A possible implementation is to force sequentiality. The reverse U_2 then U_1 is also possible. This is up to the designer and his knowledge of the problem.

Supplementary Figure A2.1: Basic schematic to introduce the concept of data race between functional units. Data race occurs when multiple units are using shared memory and one of them is trying to update the value of data in memory that might be read by any other unit. In such cases it is possible that the value of the data being modified changes in the middle of the execution of units reading the data. a) No data race. b) no data race. c) Data race between U1 and U2 so they must be strictly organized one to the other. Another solution not showed here is to interrupt the execution of every unit before reading data that might be modified until the unit modifying it has finished its job.



Plot plot ✓ active?

Type 2D Plot — Applies to Tasks: ✓ All

Axis Scales ☐ log X-Axis ☐ log Y-Axis

Curve Specifications
- [Cell Mass]|Time
- [M-phase promoting factor]|Time
- [Total cdc13]|Time

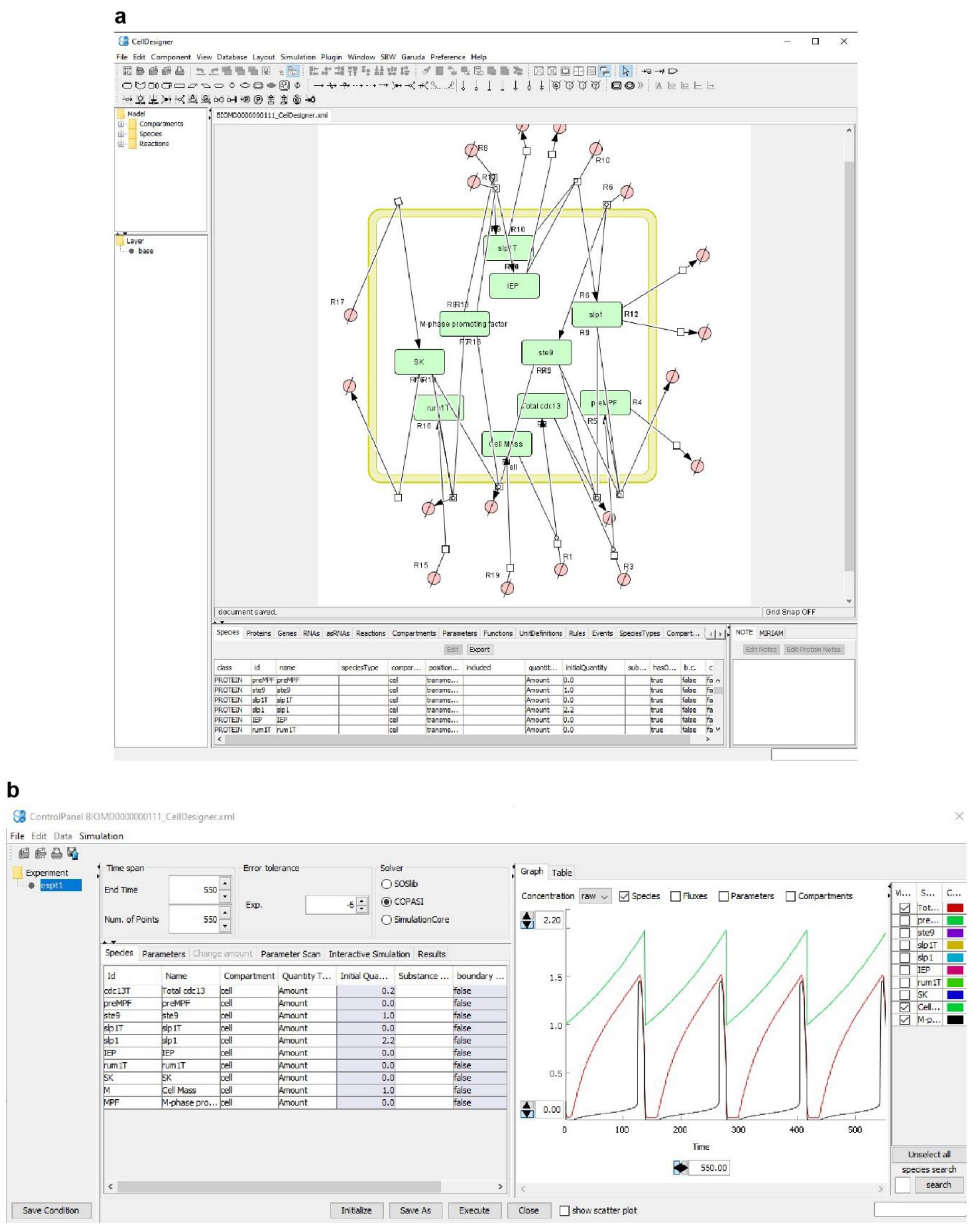
2D Curve [Total cdc13]|Time

X-Axis	Time
Y-Axis	[Total cdc13]
Type	Lines
Line Style	Solid — Width 1.2
Symbol	small cross
Color	🟥 #FF0000
Capture Data	☐ before ✓ during ☐ after Task

Time Course Result

	IEP	rum1T	SK	Cell Mass	Values[flag_MPF]	hase promoting fac	Values[sigma]	Values[Trimer]	Values[TF]	Valu
1	0	0	0	1	0	0.2	0.201	0	0.880381	1.2562
2	0.000191399	0.00099701	0.000885757	1.00005	0	0.194372	0.199819	0.00099197	0.891636	1.2616
3	0.000370165	0.00198827	0.00178084	1.0001	0	0.188886	0.19866	0.00197806	0.900901	1.2658
4	0.000537136	0.00297412	0.00268349	1.00015	0	0.183538	0.197524	0.00295859	0.908654	1.2694
5	0.000693062	0.00395487	0.00359238	1.0002	0	0.178327	0.196409	0.00393389	0.91523	1.2723
6	0.00083862	0.00493082	0.00450645	1.00025	0	0.173249	0.195315	0.00490424	0.920874	1.2748
7	0.000974423	0.00590227							58	1.2770
8	0.00110103	0.00686949							51	1.2788
9	0.00121896	0.00783273							27	1.2805
10	0.00132867	0.00879224							31	1.2819
11	0.00143062	0.00974826							78	1.2832
12	0.0015252	0.010701							71	1.2843
13	0.00161279	0.0116507							04	1.2854
14	0.00169374	0.0125975							11	1.2863
15	0.00176839	0.0135416							21	1.2872
16	0.00183704	0.0144833							5	1.2879
17	0.00189998	0.0154226							48	1.2886
18	0.0019575	0.0163598							02	1.2893
19	0.00200986	0.0172949							37	1.2899
20	0.00205729	0.0182282							55	1.2904
21	0.00210004	0.0191597							99	1.2909
22	0.00213832	0.0200897							46	1.2914
23	0.00217235	0.0210182							16	1.2918
24	0.00220233	0.0219453							16	1.2923
25	0.00222846	0.0228712							51	1.2926
26	0.0022509	0.0237959							29	1.2930
27	0.00226986	0.0247196							54	1.2933
28	0.00228548	0.0256424							29	1.2937
29	0.00229793	0.0265643	0.0260217	1.0014	0	0.0865209	0.17597	0.0263484	0.96596	1.2940
30	0.00230737	0.0274854	0.0269615	1.00145	0	0.0838041	0.175367	0.0272575	0.96665	1.2942

Supplementary Figure A2.2: Screenshots from the graphic user interface of COPASI. a) The interface to add visualization after running a simulation. b) The results after simulating using an ODE solver the model of cell division cycle of the fission yeast from (Novak et al.*, 2001).*



document saved.

Species | Proteins | Genes | RNAs | asRNAs | Reactions | Compartments | Parameters | Functions | UnitDefinitions | Rules | Events | SpeciesTypes | Compart...

Edit Export

class	id	name	speciesType	compar...	position...	included	quantit...	initialQuantity	sub...	hasO...	b.c.	c
PROTEIN	preMPF	preMPF		cell	transme...		Amount	0.0		true	false	fa
PROTEIN	ste9	ste9		cell	transme...		Amount	1.0		true	false	fa
PROTEIN	slp1T	slp1T		cell	transme...		Amount	0.0		true	false	fa
PROTEIN	slp1	slp1		cell	transme...		Amount	2.2		true	false	fa
PROTEIN	IEP	IEP		cell	transme...		Amount	0.0		true	false	fa
PROTEIN	rum1T	rum1T		cell	transme...		Amount	0.0		true	false	fa

Time span
End Time 550
Num. of Points 550

Error tolerance
Exp. -6

Solver
SOSlib [✓ COPASI] SimulationCore

Species | Parameters | Change amount | Parameter Scan | Interactive Simulation | Results

Id	Name	Compartment	Quantity T...	Initial Qua...	Substance ...	boundary ...
cdc13T	Total cdc13	cell	Amount	0.2		false
preMPF	preMPF	cell	Amount	0.0		false
ste9	ste9	cell	Amount	1.0		false
slp1T	slp1T	cell	Amount	0.0		false
slp1	slp1	cell	Amount	2.2		false
IEP	IEP	cell	Amount	0.0		false
rum1T	rum1T	cell	Amount	0.0		false
SK	SK	cell	Amount	0.0		false
M	Cell Mass	cell	Amount	1.0		false
MPF	M-phase pro...	cell	Amount	0.0		false

[✓ Species] Fluxes Parameters Compartments

Vi...	S...	C...
✓	Tot...	🟥
	pre...	🟩
	ste9	🟪
	slp1T	🟨
	slp1	🟦
	IEP	🟥
	rum1T	🟩
	SK	🟦
✓	Cell...	🟩
✓	M-p...	⬛

Supplementary Figure A2.3: Screenshots from the graphic user interface of CellDesigner. a) The default model view after opening the model of cell division cycle of the fission yeast from (Novak et al.*, 2001). b) The simulation panel using COPASI as the backend simulation software.*

References

Agmon,E. *et al.* (2022) Vivarium: an interface and engine for integrative multiscale modeling in computational biology. *Bioinformatics*, **38**, 1972–1979.

Akman,O.E. *et al.* (2008) Isoform switching facilitates period control in the Neurospora crassa circadian clock. *Mol Syst Biol*, **4**, 164.

Allen,L.J.S. (2008) An Introduction to Stochastic Epidemic Models. In, Brauer,F. *et al.* (eds), *Mathematical Epidemiology*, Lecture Notes in Mathematics. Springer, Berlin, Heidelberg, pp. 81–130.

Altilar,D.T. and Paker,Y. (1998) An optimal scheduling algorithm for parallel video processing. In, *Proceedings. IEEE International Conference on Multimedia Computing and Systems (Cat. No.98TB100241).*, pp. 245–248.

Amir,M. *et al.* (2022) Intelligent based hybrid renewable energy resources forecasting and real time power demand management system for resilient energy systems. *Science Progress*, **105**, 00368504221132144.

Are We There Yet? A Status Check on the Industrial Metaverse (2023) GTC Digital Spring.

Avello,A. *et al.* (1993) A simple and highly parallelizable method for real-time dynamic simulation based on velocity transformations. *Computer Methods in Applied Mechanics and Engineering*, **107**, 313–339.

Aviz,A. (1969) Design Methods for Fault = Tolerant Navigation Computers National Aeronautics and Space Administration, Jet Propulsion Laboratory, Pasadena, California.

Bartlett,S. and Wong,M.L. (2020) Defining Lyfe in the Universe: From Three Privileged Functions to Four Pillars. *Life*, **10**, 42.

Becker,S.A. *et al.* (2007) Quantitative prediction of cellular metabolism with constraint-based models: the COBRA Toolbox. *Nat Protoc*, **2**, 727–738.

Beier,J. *et al.* (2017) Energy flexibility of manufacturing systems for variable renewable energy supply integration: Real-time control method and simulation. *Journal of Cleaner Production*, **141**, 648–661.

Bender,E.M. *et al.* (2021) On the Dangers of Stochastic Parrots: Can Language Models Be Too Big? 🦜. In, *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, FAccT '21. Association for Computing Machinery, New York, NY, USA, pp. 610–623.

Bennett,S. (1980) On Real Time System Design The University of Sheffield.

Beygi,A. (2023) Universality of Form: The Case of Retinal Cone Photoreceptor Mosaics. *Entropy*, **25**, 766.

Bienroth,D. *et al.* (2023) Spatially Resolved Transcriptomics Mining in 3D and Virtual Reality Environments with VR-Omics. 2023.03.31.535025.

Biocca,F. *et al.* (2003) Toward a More Robust Theory and Measure of Social Presence: Review and Suggested Criteria. *Presence: Teleoperators and Virtual Environments*, **12**, 456–480.

Bitensky,M.W. (1986) Sequencing the Human Genome, Summary report of the Santa Fe Workshop Department of Energy, Office of Health and Environmental Research.

Blinov,M.L. *et al.* (2004) BioNetGen: software for rule-based modeling of signal transduction based on the interactions of molecular domains. *Bioinformatics*, **20**, 3289–3291.

Bloem,D. and Naigus,R. (1988) Real-time simulation - A tool for development and verification. In, *Flight Simulation Technologies Conference*. American Institute of Aeronautics and Astronautics, pp. 244–249.

Boeing,A. and Bräunl,T. (2012) Leveraging multiple simulators for crossing the reality gap. In, *2012 12th International Conference on Control Automation Robotics Vision (ICARCV)*., pp. 1113–1119.

Bonabeau,E. (2002) Agent-based modeling: Methods and techniques for simulating human systems. *Proceedings of the National Academy of Sciences*, **99**, 7280–7287.

Boutillier,P. *et al.* (2017) Incremental Update for Graph Rewriting. In, Yang,H. (ed), *Programming Languages and Systems*, Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, pp. 201–228.

Brenner,S. (2012) Life's code script. *Nature*, **482**, 461–461.

Brenner,S. (2002) NATURE'S GIFT TO SCIENCE.

Brenner,S. (2010) Sequences and consequences. *Philosophical Transactions of the Royal Society B: Biological Sciences*, **365**, 207–212.

Bro-Nielsen,M. and Cotin,S. (1996) Real-time Volumetric Deformable Models for Surgery Simulation using Finite Elements and Condensation. *Computer Graphics Forum*, **15**, 57–66.

Brunnsåker,D. *et al.* (2023) High-throughput metabolomics for the design and validation of a diauxic shift model. *npj Syst Biol Appl*, **9**, 1–9.

Buchholz,F. *et al.* (2022) There's more than one metaverse. *i-com*, **21**, 313–324.

Burns,A. and Wellings,A.J. (1994) HRT-HOOD: A structured design method for hard real-time systems. *Real-Time Syst*, **6**, 73–114.

Buttazzo,G.C. ed. (2005) Soft real-time systems: predictability vs. efficiency Springer, New York.

Caccamo,M. *et al.* (2000) Capacity sharing for overrun control. In, *Proceedings 21st IEEE Real-Time Systems Symposium*., pp. 295–304.

Caccamo,M. *et al.* (2002) Handling execution overruns in hard real-time control systems. *IEEE Transactions on Computers*, **51**, 835–849.

Callebaut,W. (2012) Scientific perspectivism: A philosopher of science's response to the challenge of big data biology. *Studies in History and Philosophy of Science Part C: Studies in History and Philosophy of Biological and Biomedical Sciences*, **43**, 69–80.

Cauchy,A.L. (1823) Résumé des leçons donnees a l'Ecole Royale Polytechnique sur le calcul infinitésimal, par m. Augustin-Louis Cauchy... Tome premier A Paris : de l'Imprimerie Royale chez Debure freres, libraires du Roi et de la bibliotheque du Roi, rue Serpente, n. 7, 1823.

Cellier,F.E. and Kofman,E. eds. (2006) Real-time Simulation. In, *Continuous System Simulation*. Springer US, Boston, MA, pp. 479–518.

Chacon,S. and Straub,B. (2014) Pro Git 2nd ed. Apress, USA.

Chan,B.W.-C. (2020) Lenia and Expanded Universe. MIT Press, pp. 221–229.

Chance,B. *et al.* (1960) Metabolic control mechanisms. 5. A solution for the equations representing interaction between glycolysis and respiration in ascites tumor cells. *J Biol Chem*, **235**, 2426–2439.

Chelini,J. and Farmer,R.H. (1981) Real-time flight management avionics software system. In, *Digital Avionics Systems Conference*. American Institute of Aeronautics and Astronautics, St. Louis.

Collins,F. and Galas,D. (1993) A New Five-Year Plan for the U.S. Human Genome Project. *Science*, **262**, 43–46.

Collins,F.S. *et al.* (1998) New Goals for the U.S. Human Genome Project: 1998-2003. *Science*, **282**, 682–689.

Cotin,S. *et al.* (1999) Real-time elastic deformations of soft tissues for surgery simulation. *IEEE Transactions on Visualization and Computer Graphics*, **5**, 62–73.

Coutant,A. *et al.* (2019) Closed-loop cycles of experiment design, execution, and learning accelerate systems biology model development in yeast. *Proceedings of the National Academy of Sciences*, **116**, 18142–18147.

Danos,V. *et al.* (2012) Graphs, Rewriting and Pathway Reconstruction for Rule-Based Models. In, D'Souza,D. *et al.* (eds), *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012)*, Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, pp. 276–288.

Danos,V. and Laneve,C. (2003) Core Formal Molecular Biology. In, Degano,P. (ed), *Programming Languages and Systems*, Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, pp. 302–318.

Danos,V. and Laneve,C. (2004) Formal molecular biology. *Theoretical Computer Science*, **325**, 69–110.

Darwin,C.R. (1859) On the origin of species by means of natural selection, or the preservation of favoured races in the struggle for life. 1st edition. John Murray, London.

Davis,R.I. and Burns,A. (2011) A survey of hard real-time scheduling for multiprocessor systems. *ACM Comput. Surv.*, **43**, 35:1-35:44.

Derks,J. *et al.* (2023) Increasing the throughput of sensitive proteomics by plexDIA. *Nat Biotechnol*, **41**, 50–59.

Dolev,D. *et al.* (1986) Reaching approximate agreement in the presence of faults. *J. ACM*, **33**, 499–516.

Dou,W. *et al.* (2009) Recovering Reasoning Processes from User Interactions. *IEEE Computer Graphics and Applications*, **29**, 52–61.

Dritschel,H. *et al.* (2018) A mathematical model of cytotoxic and helper T cell interactions in a tumour microenvironment. *Letters in Biomathematics*, **5**, S36–S68.

Dudziuk,G. *et al.* (2019) Biologically sound formal model of Hsp70 heat induction. *J Theor Biol*, **478**, 74–101.

Ebrahim,A. *et al.* (2013) COBRApy: COnstraints-Based Reconstruction and Analysis for Python. *BMC Syst Biol*, **7**, 1–6.

Edwards,J.S. *et al.* (2002) Metabolic modelling of microbes: the flux-balance approach. *Environ Microbiol*, **4**, 133–140.

Encode (2022) Uvicorn: an ASGI web server, for Python.

Epic Games (2022) Unreal Engine.

Fabian,R. (2018) Data-oriented design: software engineering for limited resources and short schedules Richard Fabian.

Faeder,J.R. *et al.* (2009) Rule-Based Modeling of Biochemical Systems with BioNetGen. In, Maly,I.V. (ed), *Systems Biology*, Methods in Molecular Biology. Humana Press, Totowa, NJ, pp. 113–167.

Faure,F. *et al.* (2012) SOFA: A Multi-Model Framework for Interactive Physical Simulation. Springer, p. 283.

Fiedler,G. (2004) Fix Your Timestep! *Gaffer On Games*.

Fisher,R.A. (1930) The genetical theory of natural selection Clarendon Press, Oxford.

Freeman,T.C. *et al.* (2022) Graphia: A platform for the graph-based visualisation and analysis of high dimensional data. *PLOS Computational Biology*, **18**, e1010310.

Fujimoto,R.M. (2001) Parallel and distributed simulation systems. In, *Proceeding of the 2001 Winter Simulation Conference (Cat. No.01CH37304).*, pp. 147–157 vol.1.

Gandhi,H.A. *et al.* (2020) Real-Time Interactive Simulation and Visualization of Organic Molecules. *J. Chem. Educ.*, **97**, 4189–4195.

Ganter,M. *et al.* (2013) MetaNetX.org: a website and repository for accessing, analysing and manipulating metabolic networks. *Bioinformatics*, **29**, 815–816.

Genome Sequencing Workshop (1986).

Gershenson,C. (2023) Emergence in Artificial Life. *Artificial Life*, **29**, 153–167.

Getz,W.M. (1976) Stochastic equivalents of the linear and Lotka-Volterra systems of equations— a general birth-and-death process formulation. *Mathematical Biosciences*, **29**, 235–257.

Gillespie,D.T. (1976) A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *Journal of Computational Physics*, **22**, 403–434.

Gillespie,D.T. (1992) A rigorous derivation of the chemical master equation. *Physica A: Statistical Mechanics and its Applications*, **188**, 404–425.

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

Glaessgen,E. and Stargel,D. (2012) The Digital Twin Paradigm for Future NASA and U.S. Air Force Vehicles. In, *53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference*. American Institute of Aeronautics and Astronautics, Honolulu, Hawaii.

Goldbeter,A. *et al.* (1990) Minimal model for signal-induced Ca2+ oscillations and for their frequency encoding through protein phosphorylation. *Proc Natl Acad Sci U S A*, **87**, 1461–1465.

Gomaa,H. (1984) A software design method for real-time systems. *Commun. ACM*, **27**, 938–949.

Gomaa,H. (1986) Software development of real-time systems. *Commun. ACM*, **29**, 657–668.

Gregory,J. (2018a) Game engine architecture Third edition. Taylor and Francis, CRC Press, Boca Raton.

Gregory,J. (2018b) Runtime Gameplay Foundation System. In, *Game Engine Architecture*. Taylor and Francis, CRC Press, Boca Raton, pp. 1039–1158.

Gregory,J. (2018c) The Game Loop and Real-Time Simulation. In, *Game Engine Architecture*. Taylor and Francis, CRC Press, Boca Raton, pp. 525–558.

Hairer,E. and Wanner,G. (1993) Solving Ordinary Differential Equations I Springer, Berlin, Heidelberg.

Hairer,E. and Wanner,G. (1996) Solving Ordinary Differential Equations II Springer, Berlin, Heidelberg.

Hajjar,G. *et al.* (2023) Scaling-up metabolomics: Current state and perspectives. *TrAC Trends in Analytical Chemistry*, **167**, 117225.

Harris,L.A. *et al.* (2016) BioNetGen 2.2: advances in rule-based modeling. *Bioinformatics*, **32**, 3366–3368.

Haung,S.-H. *et al.* (2005) Web-based real time power system dynamic performance monitoring system. In, *Fourtieth IAS Annual Meeting. Conference Record of the 2005 Industry Applications Conference, 2005.*, pp. 2651-2656 Vol. 4.

Hayatpur,D. *et al.* (2020) DataHop: Spatial Data Exploration in Virtual Reality. In, *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*. Association for Computing Machinery, New York, NY, USA, pp. 818–828.

Hecht,H. (1976) Fault-Tolerant Software for Real-Time Applications. *ACM Comput. Surv.*, **8**, 391–407.

Heirendt,L. *et al.* (2019) Creation and analysis of biochemical constraint-based models using the COBRA Toolbox v.3.0. *Nat Protoc*, **14**, 639–702.

Hernández-Orozco,S. *et al.* (2018) Undecidability and Irreducibility Conditions for Open-Ended Evolution and Emergence. *Artificial Life*, **24**, 56–70.

Hlavacek,W.S. *et al.* (2006) Rules for Modeling Signal-Transduction Systems. *Science Signaling*, **2006**, re6–re6.

Hoops,S. *et al.* (2006) COPASI—a COmplex PAthway SImulator. *Bioinformatics*, **22**, 3067–3074.

Horton,S. and Wurster,J. (2022) The Journey to Collaborative Virtual Workspaces: PopUp-XR at Volvo Group Trucks Technology.

Ishii,H. and Ullmer,B. (1997) Tangible bits: towards seamless interfaces between people, bits and atoms. In, *Proceedings of the ACM SIGCHI Conference on Human factors in computing systems*, CHI ’97. Association for Computing Machinery, New York, NY, USA, pp. 234–241.

Jacopin,E. *et al.* (2024) An architecture for collaboration in systems biology at the age of the Metaverse. *npj Syst Biol Appl*, **10**, 1–11.

Jacopin,E. *et al.* (2020) Factors favouring the evolution of multidrug resistance in bacteria. *J. R. Soc. Interface.*, **17**, 20200105.

Jacopin,E. *et al.* (2021) Using Agents and Unsupervised Learning for Counting Objects in Images with Spatial Organization: In, *Proceedings of the 13th International Conference on Agents and Artificial Intelligence*. SCITEPRESS - Science and Technology Publications, pp. 688–697.

Jeppesen,M.J. and Powers,R. (2023) Multiplatform untargeted metabolomics. *Magnetic Resonance in Chemistry*, **61**, 628–653.

Johnson,K.A. and Goody,R.S. (2011) The Original Michaelis Constant: Translation of the 1913 Michaelis–Menten Paper. *Biochemistry*, **50**, 8264–8269.

Juliani,A. *et al.* (2018) Unity: A General Platform for Intelligent Agents.

Jumper,J. *et al.* (2021) Highly accurate protein structure prediction with AlphaFold. *Nature*, **596**, 583–589.

Kaizu,K. *et al.* (2020) E-Cell System version 4.

Kanda,G.N. *et al.* (2022) Robotic search for optimal cell culture in regenerative medicine. *eLife*, **11**, e77007.

Karr,J.R. *et al.* (2012) A Whole-Cell Computational Model Predicts Phenotype from Genotype. *Cell*, **150**, 389–401.

Kauffman,S. (2020) Answering Schrödinger's "What Is Life?" *Entropy (Basel)*, **22**, 815.

Kauffman,S.A. and Roli,A. (2023) A third transition in science? *Interface Focus*, **13**, 20220063.

Kaul,S. *et al.* (2012) Real-time status: How often should one update? In, *2012 Proceedings IEEE INFOCOM.*, pp. 2731–2735.

Keating,S.M. *et al.* (2020) SBML Level 3: an extensible format for the exchange and reuse of biological models. *Molecular Systems Biology*, **16**, e9110.

Kermack,W.O. *et al.* (1927) A contribution to the mathematical theory of epidemics. *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, **115**, 700–721.

Kim,K.H. (1997) Object structures for real-time systems and simulators. *Computer*, **30**, 62–70.

King,R.D. *et al.* (2009) The Automation of Science. *Science*, **324**, 85–89.

King,Z.A. *et al.* (2016) BiGG Models: A platform for integrating, standardizing and sharing genome-scale models. *Nucleic Acids Research*, **44**, D515–D522.

King,Z.A. *et al.* (2015) Escher: A Web Application for Building, Sharing, and Embedding Data-Rich Visualizations of Biological Pathways. *PLOS Computational Biology*, **11**, e1004321.

Kitano,H. *et al.* (2005) Using process diagrams for the graphical representation of biological networks. *Nat Biotechnol*, **23**, 961–966.

Koopman,S. and Yamauchi,R.K. (1990) Real-time sewer effluent monitoring system. Las Vegas.

Kopetz,H. *et al.* (1989) Distributed fault-tolerant real-time systems: the Mars approach. *IEEE Micro*, **9**, 25–40.

Kopetz,H. and Ochsenreiter,W. (1987) Clock Synchronization in Distributed Real-Time Systems. *IEEE Transactions on Computers*, **C–36**, 933–940.

Kopetz,H. and Steiner,W. (2022a) Dependability. In, Kopetz,H. and Steiner,W. (eds), *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Springer International Publishing, Cham, pp. 143–175.

Kopetz,H. and Steiner,W. (2022b) Global Time. In, Kopetz,H. and Steiner,W. (eds), *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Springer International Publishing, Cham, pp. 57–85.

Kopetz,H. and Steiner,W. (2022c) Real-Time Scheduling. In, Kopetz,H. and Steiner,W. (eds), *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Springer International Publishing, Cham, pp. 247–267.

Kopetz,H. and Steiner,W. (2022d) System Design. In, Kopetz,H. and Steiner,W. (eds), *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Springer International Publishing, Cham, pp. 269–305.

Kopetz,H. and Steiner,W. (2022e) The Real-Time Environment. In, Kopetz,H. and Steiner,W. (eds), *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Springer International Publishing, Cham, pp. 1–29.

Lander,E.S. *et al.* (2001) Initial sequencing and analysis of the human genome. *Nature*, **409**, 860–921.

Laubenbacher,R. *et al.* (2022) Building digital twins of the human immune system: toward a roadmap. *npj Digit. Med.*, **5**, 1–5.

Lee,K. *et al.* (2020) GROOT: a real-time streaming system of high-fidelity volumetric videos. In, *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, MobiCom ’20. Association for Computing Machinery, New York, NY, USA, pp. 1–14.

Lee,L.-H. *et al.* (2021) All One Needs to Know about Metaverse: A Complete Survey on Technological Singularity, Virtual Ecosystem, and Research Agenda.

Lehman,J. and Stanley,K.O. (2015) Investigating Biological Assumptions through Radical Reimplementation. *Artificial Life*, **21**, 21–46.

Leonelli,S. (2019) The challenges of big data biology. *eLife*, **8**, e47381.

Leonelli,S. (2014) What difference does quantity make? On the epistemology of Big Data in biology. *Big Data & Society*, **1**, 2053951714534395.

Lin,C.E. and Lee,L.A. (1989) A PC-based real time measurement system for factory automation on quality control and production control. In, *6th IEEE Conference Record., Instrumentation and Measurement Technology Conference*. Washington, DC., pp. 57–61.

Liu,C.L. and Layland,J.W. (1973) Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *J. ACM*, **20**, 46–61.

Lotka,A.J. (1925) Elements of Physical Biology Williams & Wilkins.

Macklin,D.N. *et al.* (2020) Simultaneous cross-evaluation of heterogeneous E. coli datasets via mechanistic simulation. *Science*, **369**, eaav3751.

Maiza,C. *et al.* (2019) A Survey of Timing Verification Techniques for Multi-Core Real-Time Systems. *ACM Comput. Surv.*, **52**, 56:1-56:38.

Malik-Sheriff,R.S. *et al.* (2020) BioModels—15 years of sharing computational models in life science. *Nucleic Acids Research*, **48**, D407–D415.

March-Leuba,J. and King,W.T. (1987) Development of a real-time stability measurement system for boiling water reactors. In, *Trans. Am. Nucl. Soc.; (United States)*. Dallas.

Masison,J. *et al.* (2021) A modular computational framework for medical digital twins. *Proceedings of the National Academy of Sciences*, **118**, e2024287118.

Mayer-Schönberger,V. and Cukier,K. (2014) Big data: a revolution that will transform how we live, work, and think First Mariner Books edition. Mariner Books, Houghton Mifflin Harcourt, Boston.

Mendel,G. (1865) Versuche über Pflanzen-Hybriden. *Verhandlungen des naturforschenden Vereines in Brünn*, **Bd.4 (1865-1866)**, 3–47.

Menghal,P.M. and Laxmi,A.J. (2012) Real time simulation: Recent progress & challenges. In, *Controls and Computation 2012 International Conference on Power, Signals*., pp. 1–6.

Michaelis,L. and Menten,M.L. (1913) Die Kinetik der Invertinwirkung. *Biochem Z.*, **49**, 333–369.

Milgram,P. *et al.* (1995) Augmented reality: a class of displays on the reality-virtuality continuum. In, *Telemanipulator and Telepresence Technologies*. SPIE, pp. 282–292.

Montévil,M. and Mossio,M. (2015) Biological organisation as closure of constraints. *J Theor Biol*, **372**, 179–191.

Moretti,S. *et al.* (2016) MetaNetX/MNXref – reconciliation of metabolites and biochemical reactions to bring together genome-scale metabolic networks. *Nucleic Acids Research*, **44**, D523–D526.

Moretti,S. *et al.* (2021) MetaNetX/MNXref: unified namespace for metabolites and biochemical reactions in the context of metabolic models. *Nucleic Acids Research*, **49**, D570–D574.

Mystakidis,S. (2022) Metaverse. *Encyclopedia*, **2**, 486–497.

Nagasaki,M. *et al.* (1999) Bio-calculus: Its Concept and Molecular Interaction. *Genome Informatics*, **10**, 133–143.

Nanome Inc (2023) Nanome: Creating Powerful, Collaborative, and Scientific VR Tools.

Nijhout,H.F. *et al.* (2004) A mathematical model of the folate cycle: new insights into folate homeostasis. *J Biol Chem*, **279**, 55008–55016.

Nikolov,S. *et al.* (2020) The role of cooperativity in a p53-miR34 dynamical mathematical model. *Journal of Theoretical Biology*, **495**, 110252.

Ning,H. *et al.* (2021) A Survey on Metaverse: the State-of-the-art, Technologies, Applications, and Challenges.

Norrby,M. *et al.* (2015) Molecular Rift: Virtual Reality for Drug Designers. *J. Chem. Inf. Model.*, **55**, 2475–2484.

North,C. *et al.* (2011) Analytic provenance: process+interaction+insight. In, *CHI '11 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '11. Association for Computing Machinery, New York, NY, USA, pp. 33–36.

Novak,B. *et al.* (2001) Mathematical model of the cell division cycle of fission yeast. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, **11**, 277–286.

Novak,B. and Tyson,J.J. (2022) Mitotic kinase oscillation governs the latching of cell cycle switches. *Current Biology*, **32**, 2780-2785.e2.

NVIDIA (2022) NVIDIA Omniverse.

Nystrom,R. (2014) Sequencing Patterns: Game Loop. In, *Game Programming Patterns*. genever benning, pp. 123–138.

Ochiai,K. *et al.* (2021) A Variable Scheduling Maintenance Culture Platform for Mammalian Cells. *SLAS TECHNOLOGY: Translating Life Sciences Innovation*, **26**, 209–217.

Orth,J.D. *et al.* (2010) What is flux balance analysis? *Nat Biotechnol*, **28**, 245–248.

Palmer,C. *et al.* (2021) Virtual Reality Based Digital Twin System for Remote Laboratories and Online Practical Learning. *Advances in Manufacturing Technology XXXIV*, 277–283.

Park,S.-M. and Kim,Y.-G. (2022) A Metaverse: Taxonomy, Components, Applications, and Open Challenges. *IEEE Access*, **10**, 4209–4251.

Pazzaglia,P. *et al.* (2021) Adaptive Design of Real-Time Control Systems subject to Sporadic Overruns. In, *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE).*, pp. 1887–1892.

Priami,C. *et al.* (2001) Application of a stochastic name-passing calculus to representation and simulation of molecular processes. *Information Processing Letters*, **80**, 25–31.

Puschner,P. and Koza,Ch. (1989) Calculating the maximum execution time of real-time programs. *Real-Time Syst*, **1**, 159–176.

Qi,H. *et al.* (2011) A Resilient Real-Time System Design for a Secure and Reconfigurable Power Grid. *IEEE Transactions on Smart Grid*, **2**, 770–781.

Rahmat,R.F. *et al.* (2016) Real time monitoring system for water pollution in Lake Toba. In, *2016 International Conference on Informatics and Computing (ICIC)*., pp. 383–388.

Ramialison,M. *et al.* (2023) Spatially Resolved Transcriptomics Exploration in 3D desktop and Virtual Environments with VR-Omics - Application.

Ravn,A.P. *et al.* (1993) Specifying and verifying requirements of real-time systems. *IEEE Transactions on Software Engineering*, **19**, 41–55.

Rees-Garbutt,J. *et al.* (2020) Designing minimal genomes using whole-cell models. *Nat Commun*, **11**, 836.

Regev,A. *et al.* (2000) Representation and simulation of biochemical processes using the π-calculus process algebra. In, *Biocomputing 2001*. WORLD SCIENTIFIC, pp. 459–470.

Regev,A. *et al.* (2017) The Human Cell Atlas. *eLife*, **6**, e27041.

Regev,A. *et al.* (2018) The Human Cell Atlas White Paper.

Regev,A. and Shapiro,E. (2002) Cellular abstractions: Cells as computation. *Nature*, **419**, 343–343.

Reichl,T. *et al.* (2009) Ultrasound goes GPU: real-time simulation using CUDA. In, *Medical Imaging 2009: Visualization, Image-Guided Procedures, and Modeling*. SPIE, pp. 386–395.

Reifsnider,K. and Majumdar,P. (2013) Multiphysics Stimulated Simulation Digital Twin Methods for Fleet Management. In, *54th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*. American Institute of Aeronautics and Astronautics, Boston.

Ross,R. (1916) An application of the theory of probabilities to the study of a priori pathometry.—Part I. *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, **92**, 204–230.

Ross,R. and Hudson,H.P. (1917a) An application of the theory of probabilities to the study of a priori pathometry.—Part II. *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, **93**, 212–225.

Ross,R. and Hudson,H.P. (1917b) An application of the theory of probabilities to the study of a priori pathometry.—Part III. *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, **93**, 225–240.

Rowe,E. *et al.* (2018) Escher-FBA: a web application for interactive flux balance analysis. *BMC Systems Biology*, **12**, 84.

Rozenblatt-Rosen,O. *et al.* (2017) The Human Cell Atlas: from vision to reality. *Nature*, **550**, 451–453.

Rukangu,A. *et al.* (2021) Virtual Reality for Remote Controlled Robotics in Engineering Education. In, *2021 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*., pp. 751–752.

Schiano,C. and Silberto,J. (1986) Grumman's real time computing system for avionics testing. In, *3rd Flight Testing Conference and Technical Display*. American Institute of Aeronautics and Astronautics, Las Vegas.

Schluse,M. *et al.* (2018) Experimentable Digital Twins—Streamlining Simulation-Based Systems Engineering for Industry 4.0. *IEEE Transactions on Industrial Informatics*, **14**, 1722–1731.

Schluse,M. and Rossmann,J. (2016) From simulation to experimentable digital twins: Simulation-based development and operation of complex technical systems. In, *2016 IEEE International Symposium on Systems Engineering (ISSE)*., pp. 1–6.

Shannon,P. *et al.* (2003) Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome Res*, **13**, 2498–2504.

Simpson,H. (1986) The Mascot method. *Software Engineering Journal*, **1**, 103–120.

Simpson,H.R. and Jackson,K. (1979) Process synchronisation in MASCOT*. *The Computer Journal*, **22**, 332–345.

Sivakumar,K.C. *et al.* (2011) A systems biology approach to model neural stem cell regulation by notch, shh, wnt, and EGF signaling pathways. *OMICS*, **15**, 729–737.

Skalnik,C.J. *et al.* (2023) Whole-cell modeling of E. coli colonies enables quantification of single-cell heterogeneity in antibiotic responses. *PLOS Computational Biology*, **19**, e1011232.

Slavov,N. (2021) Increasing proteomics throughput. *Nat Biotechnol*, **39**, 809–810.

Sollaeg,D.L. (1964) NASCOM Real-Time System National Aeronautics and Space Administration, Goddard Space Flight Center.

Sorenson,P.G. and Hamacher,V.C. (1975) A Real-Time System Design Methodology*. *INFOR: Information Systems and Operational Research*, **13**, 1–18.

Sparkes,A. *et al.* (2010) Towards Robot Scientists for autonomous scientific discovery. *Autom Exp*, **2**, 1–11.

Srivastava,A.K. *et al.* (2018) Graph-theoretic algorithms for cyber-physical vulnerability analysis of power grid with incomplete information. *Journal of Modern Power Systems and Clean Energy*, **6**, 887–899.

St. John,R.H. *et al.* (1987) Real-time simulation for space station. *Proceedings of the IEEE*, **75**, 383–398.

Stahlkopf,K.E. and Wilhelm,M.R. (1997) Tighter controls for busier systems [power systems]. *IEEE Spectrum*, **34**, 48–52.

Stankovic,J.A. *et al.* (2003) Real-time communication and coordination in embedded sensor networks. *Proceedings of the IEEE*, **91**, 1002–1022.

Stephenson,N. (1992) Snow crash Bantam Books, New York.

Strubell,E. *et al.* (2019) Energy and Policy Considerations for Deep Learning in NLP. In, *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Florence, Italy, pp. 3645–3650.

Takahashi,K. *et al.* (2004) A multi-algorithm, multi-timescale method for cell simulation. *Bioinformatics*, **20**, 538–546.

Takahashi,K., Ishikawa,N., *et al.* (2003) E-Cell 2: multi-platform E-Cell simulation system. *Bioinformatics*, **19**, 1727–1729.

Takahashi,K., Sakurada,T., *et al.* (2003) E-CELL System Version 3: A Software Platform for Integrative Computational Biology. *Genome Informatics*, **14**, 294–295.

Tang,L. (2022) Sequencing single cells without killing. *Nat Methods*, **19**, 1166–1166.

The Metaverse and How We'll Build It Together -- Connect 2021 (2021) Meta.

The UniProt Consortium (2023) UniProt: the Universal Protein Knowledgebase in 2023. *Nucleic Acids Research*, **51**, D523–D531.

Thornburg,Z.R. *et al.* (2022) Fundamental behaviors emerge from simulations of a living minimal cell. *Cell*, **185**, 345-360.e28.

Tomita,M. *et al.* (1999) E-CELL: software environment for whole-cell simulation. *Bioinformatics*, **15**, 72–84.

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

Tuckwell,H.C. and Williams,R.J. (2007) Some properties of a simple stochastic epidemic model of SIR type. *Mathematical Biosciences*, **208**, 76–97.

Tuegel,E.J. *et al.* (2011) Reengineering Aircraft Structural Life Prediction Using a Digital Twin. *International Journal of Aerospace Engineering*, **2011**, e154798.

Turing,A.M. (1952) The chemical basis of morphogenesis. *Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences*, **237**, 37–72.

Unity Technologies (2022a) Netcode for GameObjects.

Unity Technologies (2022b) Unity Game Engine.

Unity Technologies (2020a) Unity Perception Package.

Unity Technologies (2020b) Unity Robotics Hub.

Unity Technologies (2022c) Unity SynthHomes: A Synthetic Home Interior Dataset Generator.

Varma,A. and Palsson,B.O. (1994) Metabolic Flux Balancing: Basic Concepts, Scientific and Practical Use. *Nat Biotechnol*, **12**, 994–998.

Venter,J.C. *et al.* (2001) The Sequence of the Human Genome. *Science*, **291**, 1304–1351.

Volterra,V. (1926) Variazioni e fluttuazioni del numero d'individui in specie animali conviventi. *Memor. Accad. Lincei, Ser.*, **6**, 31–113.

Wang,Y. *et al.* (2022) A Survey on Metaverse: Fundamentals, Security, and Privacy. *IEEE Communications Surveys & Tutorials*, 1–1.

Weinberger,M. (2022) What Is Metaverse?—A Definition Based on Qualitative Meta-Synthesis. *Future Internet*, **14**, 310.

Weisstein,E.W. Algebra. *MathWorld--A Wolfram Web Resource.*

Weisstein,E.W. Vector Space. *MathWorld--A Wolfram Web Resource.*

Wolfram,S. (2002) The World of Simple Programs. In, *A new kind of Science*. Wolfram Media, Champaign, IL, pp. 51–113.

Wong,M.L. *et al.* (2022) Searching for Life, Mindful of Lyfe's Possibilities. *Life*, **12**, 783.

Wong,M.L. and Prabhu,A. (2023) Cells as the first data scientists. *Journal of The Royal Society Interface*, **20**, 20220810.

Woodger,J.H. (1929) Biological Principles Routledge (Republication in 2010).

Woodger,J.H. (1952) Biology and Language: An Introduction to the Methodology of the Biological Sciences Including Medicine. The Tarner Lectures 1949-50 1st ed. The University Press, Cambridge.

Woodger,J.H. (1962) Biology and the Axiomatic Method. *Annals of the New York Academy of Sciences*, **96**, 1093–1116.

Woodger,J.H. (1937) The axiomatic method in biology The University Press, Cambridge.

Wu,J. *et al.* (2011) Constructing stochastic models from deterministic process equations by propensity adjustment. *BMC Syst Biol*, **5**, 187.

Xia,H. *et al.* (2018) Spacetime: Enabling Fluid Individual and Collaborative Editing in Virtual Reality. In, *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*, UIST '18. Association for Computing Machinery, New York, NY, USA, pp. 853–866.

Xu,J. (2020) Effectively Handling Primary and Backup Overruns and Underruns in a Real-Time Embedded System That Tolerates Permanent Hardware and Software Failures. In, *IECON 2020 The 46th Annual Conference of the IEEE Industrial Electronics Society*., pp. 2267–2274.

Yachie,N. and Natsume,T. (2017) Robotic crowd biology with Maholo LabDroids. *Nat Biotechnol*, **35**, 310–312.

Zhongcheng,L. *et al.* (2022) Web-based digital twin online laboratories: Methodologies and implementation. *Digital Twin*, **2**.

Academic Accomplishments

Publication

Jacopin Eliott, Sakamoto Yuki, Nishida Kozo, Kaizu Kazunari, Takahashi Kouichi, (2024) *An architecture for collaboration in systems biology at the age of the Metaverse*, npj Systems Biology and Applications, **10**, 1—11, doi: https://doi.org/10.1038/s41540-024-00334-8

Conference Participation

ECellDive: Exploring Biological Systems in Virtual Reality – **MBSJ 2022** – *Poster + Flash Talk*

Delivering Virtual Reality and Gaming Technologies to the Field of Systems Biology – **ICSB 2022** – *Poster*

ECellDive: Exploring Biological Systems in Virtual Reality – **ISMB BioVis 2022** - *Poster + Flash Talk*