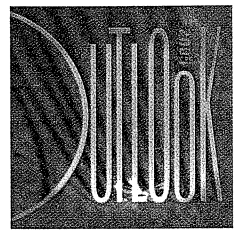
Reactive Animation: Realistic Modeling of Complex Dynamic Systems



Reactive animation combines state-of-the-art reactivity and state-of-the-art animation by linking advanced tools in the two areas. Architects of complex reactive systems and front-end designers then have a bridge between a system's appearance and what it does.

Sol Efroni
National Cancer Institute Center for Bioinformatics

David Harel

Irun R. Cohen
Weizmann Institute of Science

Complex systems come in many varieties. Some are transformational, of an input-process-output type, repeatedly carrying out their prescribed work for each new set of inputs. The complexity of these systems stems from their computations and from data flow, and is in the range of what existing tools can manage.

A far more problematic class of complex systems comprises large systems that are heavily control- or event-driven. Such systems—dubbed *reactive*[1,2] because their role is to react to various kinds of events, signals, and conditions in intricate ways—are often concurrent and distributed. Many exhibit time-critical aspects and hybrid behavior that is predominantly discrete but also has continuous aspects. A reactive system's most distinguishing feature is that no one asks it to produce some final values upon terminating; rather, the system maintains an ongoing interaction with its environment.

Reactive systems can be relatively small, such as cell phones, PDAs, and digital watches, or far more complex, such as the systems in automotive and aerospace applications. Reactive systems can be control systems, such as telephone and communication controllers, air traffic systems, and industrial robots, or large-scale interactive software packages, such as word processors and mail programs. Biological systems, such as the brain and immune system, might be viewed as the epitome of reactive complexity, yet both systems are accessible.

Whatever their size and function, all reactive systems must maintain intricate dynamic relationships with their environment, reacting properly and on time to mouse clicks, pressed buttons, temperature changes, receivers being hung up, cursors moving on a screen, and so on.

Reactivity exacerbates the problems complex system developers already see, as "The Challenge of a Reactive System" sidebar explains. Reactive system complexity continues to challenge both system architects and front-end designers. System architects must understand the computational/behavioral problem and carry out the complex tasks of modeling (specifying) and analyzing the system's reactive behavior and then designing the implementation. Front-end designers must build intuitive and appealing visual interfaces for simulation, analysis, testing, and deployment.

The scientific community within software engineering has equipped system architects with extensive tools—aids for analysis and verification as well as a variety of visual languages and methodologies convenient for building visual system models, such as Petri nets,[2] sequence charts,[3] and Statecharts.[4] The Unified Modeling Language (UML) has collected many of these languages, often in modified or

Published by the IEEE Computer Society
0018-9162/05/$20.00 © 2005 IEEE

The Challenge of a Reactive System

The structure of a reactive system consists of many interacting, often distributed, objects. Very often the structure itself is dynamic, with objects being created and destroyed during the system's life. Thus, reactivity's primary challenge is in how to specify the system's behavior over time, clearly and correctly, and in ways that someone can easily and reliably implement, analyze, and verify.[1-3] What will happen and when? Why will it happen, and what else will it cause to happen in its wake? Can other things happen in the interim? Are certain things mandatory but others merely allowed to happen? What are the time constraints on something happening? What if things don't happen as expected? What things are not allowed to happen under any circumstances?

As these questions imply, reactivity is not an exclusive characteristic of man-made computerized systems. It is present in biological systems, as well, which despite being much smaller than humans and their homemade artifacts, can be quite a bit more complicated. Perhaps the most daunting challenge is not the precise, yet realistic representation of reactive systems that humans design using reasoning and logical constructions, but the precise, testable, and realistic representations of reactive systems that a meandering evolution "designs."

The complexity of reactive biological systems such as the brain and the immune system is amplified manyfold by the unforeseen and unforeseeable functions and interconnections of the component parts that constitute the system. The logic of evolution is not always clear to human minds. For example, a typical component of a reactive biologic system performs more than one function (pleiotropism); connections between components are variable (degeneracy); and any function seems to be preformed by alternative choices of component parts (redundancy). The modeler of such a system must represent the true richness of the evolved system in its pleiotropism, degeneracy, and redundancy while revealing the properties of the system that emerge from its complexity.[4] The explication of emergence is the challenge.

References

1. D. Harel and A. Pnueli, "On the Development of Reactive Systems," *Logics and Models of Concurrent Systems*, K.R. Apt, ed., NATO Advanced Science Institute Series, vol. F-13, Springer-Verlag, 1985, pp. 477-498.
2. A. Pnueli, "Applications of Temporal Logic to the Specification and Verification of Reactive Systems: A Survey of Current Trends," in *Current Trends in Concurrency*, J. de Bakker et al., eds., LNCS, vol. 224, Springer-Verlag, 1986, pp. 510-584.
3. R.J. Wieringa, *Design Methods for Reactive Systems: Yourdon, Statemate, and the UML*, Morgan Kaufmann, 2002.
4. I.R. Cohen, *Tending Adam's Garden: Evolving the Cognitive Immune Self*, Academic Press, 2000.

extended form, which has likely contributed to its popularity as a multifaceted development framework.

Some of the system architect's tools are true "reactive engines," with full executable semantics, in that the tools can run the models and generate final implementation code. Some can also link to verification tools, allowing hard analysis. These engines are thus quite helpful, and their usefulness grows as UML continues to standardize and improve.

Similarly, front-end designers have adequate tools for their work, which can generate animated graphical designs with high quality and performance at an increasingly lower cost. The Web is becoming a driving force behind the development of improved animation tools, and these tools in turn offer designers what they need to create realistic user interfaces.

Although system architects and front-end designers have their respective quality toolsets, their functions remain largely separate. Since reactive systems are highly dynamic by nature, a crucial question becomes how to combine precise modeling and analysis with an animated front end. In more general terms, how do you append knowledge to beauty or an automobile's engine to its body and external appearance? How do you accurately represent the complex reactive immune system with the realism required to trigger the minds of experimentalists to explore new ideas and move to conducting new experiments? This is the challenge we elected to explore.

The result of our work is *reactive animation*, a patent-pending technology aimed at combining state-of-the-art reactivity with state-of-the-art animation. To implement RA, we propose linking advanced tools for reactive system development with advanced tools for animation—a strategy we believe applies to a wide spectrum of applications that call for complex reactivity and a highly dynamic front end.

FOUNDATIONAL PRINCIPLES

RA is not a new way to animate algorithms;[5] it does not attempt to introduce a method for translating algorithmic complexity into abstract, arbitrary animation. Instead, it uses dynamic user interfaces and animates or dynamically redesigns them to represent the system and its operational

Graphical User Interface versus Front-End Animation

Because reactive complexity has driven modelers to strongly emphasize the visual, most reactive system engines now use visual diagrammatic languages and then execute the models in a diagrammatic animation mode. Tools like Statemate, Magnum, Rose RT, and Rhapsody can animate statecharts, showing color-coded state changes triggered by edge transitions, the dynamics of instance creation and destruction, and more. A graphical user interface commonly complements such running models with the goal of facilitating model manipulation. The GUI structure usually relies on form-based tools with standard interface elements, such as switches and bulbs, drop-down menus, buttons, dials, sliders, and so on.

More realistic interfaces—such as those Altia provides for virtual prototype construction—can enhance GUI tools. To describe an air traffic control panel GUI, for example, architects can easily and intuitively build an interface for the control board, and the GUI software could enhance that by offering standard libraries for interface tasks and events.

Adding nonstandard dynamic components to these interfaces is another matter. Suppose the architects wanted to enrich the standard control panel with elements that depict not only the location, speed, and altitude of various aircraft, but also an aircraft's dynamically changing shape and appearance, *and* they wanted to do this on the background of a realistically animated outside world as seen from the cockpit, *and* they wanted to do it all in a truly reactive and interactive fashion. They would be forced to program most of this separately and tediously in a standard programming language and in a way that fails to exploit the natural and easy connection between the GUI and the underlying reactive engine. Moreover, if this richer kind of interface is to be part of air-traffic simulation, the architects might want to enhance it by including the reactive behavior involved in navigating the aircraft and to equip the interface with the ability to zoom into individual aircraft to see (and perhaps also modify) the decision-making processes, the individual cockpit views, and so on. Animation tools typically provide the underlying components that such animation and interaction tasks require, but it is virtually impossible to create a synergy between these components and standard GUI tools or to use them to connect to the underlying reactive engine.

Popular animation tools for game and animated movie creation include Macromedia's Flash, Director, and Maya (a recent Academy Award winner). Users of these tools can create realistic animations in total animative freedom, which is not possible with either standard GUI tools or form-based developmental environments. Because of their user-oriented design, true animation tools also offer myriad ways to facilitate interactivity. Designers can easily capture user events in a context-sensitive manner and manipulate animated objects using rotation, enlargement, movement, and other actions. Partially as a result of the gaming industry's experience, these tools also offer optimized configurations according to platform and graphic needs. Thus, satisfactory performance comes with relative ease. Animation tools can yield realistic simulations aimed at a broad audience, and player software is available for most platforms.

behavior realistically. RA is also not a methodology that uses notions from reactive systems and visual languages to facilitate the handling of reactivity inside the animation tools themselves.[6]

Rather, RA is about linking the two efforts—reactive systems design and front-end design—by bridging the power of the best tools in the two separate areas. In essence, RA has two arms: One comprises powerful tools and methods for reactive systems development, the heart of which is a rigorous specification of the system's reactivity. The other comprises powerful animation tools to represent that specification as an intuitive, controllable, animated front end. And these animation tools are not simply to provide entertainment; animation is an essential communication channel in the cross-cultural discourse between computer scientists and biologists, between living systems and human understanding.

Technically, RA is based on the view that says a system is a closely linked combination of what it does and what it looks like; from this stem two separate but closely linked paths: reactive behavior design and front-end design. Initially the two paths do not overlap, but they do connect later on when their cooperation becomes essential to understanding the model or facilitating some task.

Designers might first prepare a visual system description and build the visual interface using their preferred animation tool. When this external view requires further dynamic enrichment with the sophistication of a reactive design, the view can link with a reactive design tool.

Alternatively, the design can start with the construction of all or part of a reactive system model, which designers can enrich later by incorporating a high-quality animated front end. The "Graphical User Interface versus Front-End Animation" sidebar explains more about the unique capabilities this kind of front end offers.

IMPLEMENTATION STEPS

Figure 1 illustrates the steps in building an RA implementation. Modelers develop the system's appearance (left path) using the animation tool, which includes the required animation components and the scripting instructions on how the components combine at runtime. Along the mechanism path (right), which leads to a specification, architects use reactive system development tools to deal with the system's architecture and runtime behavior. Through specification, the designer produces a full running model of the system.

Once the animation components and the reactive behavior are operational, the designer can link them through a specific communication channel. When the link is complete, the designer can run the reactive model so that it continuously sends the information animation needs, while at the same time attending to information coming from animation. The user can then view and interact with the running model through the front-end animation, the visual language's diagrammatic animation, or both.

Our initial implementation of reactive animation involved two state-of-the-art tools: Ilogix's Rhapsody (www.ilogix.com/rhapsody/rhapsody.cfm) and Macromedia's Flash (www.macromedia.com/software/flash/). We developed the reactive model in Rhapsody in the usual way, employing mainly statecharts and class diagrams in a manner consistent with the general UML approach.[7] We then analyzed the resulting diagrams using tools built into Rhapsody and analyzed the model's behavior in standard ways.

The next step was to identify specific events and states that we deemed critical to comprehensively understanding the system. We built visual representations in Flash to portray the specific events, for example, building a spreading pulse to signify the biological event of an activated T cell.

We then connected the reactive specification and the animation using TCP/IP,[3] which provides many degrees of freedom. Because TCP/IP allows communication over a network, for example, we could run Rhapsody and Flash on two separate machines that communicate over a local area network. Thus, we had easily distributed processes and the freedom to use custom machines in specific applications.

We are exploring ways to make RA more generic. A program designed using RA is oblivious to whether it is being displayed on a desktop monitor, a handheld computing device, or even a cell phone. If, say, computer game designers create agents as reactive systems, they can use RA to customize the agents' embodiment in the interface to fit the carrying platform. A user playing that game on a desktop could then leave and continue the game on a network-connected handheld. The same idea applies to many other day-to-day systems, such as home appliances.

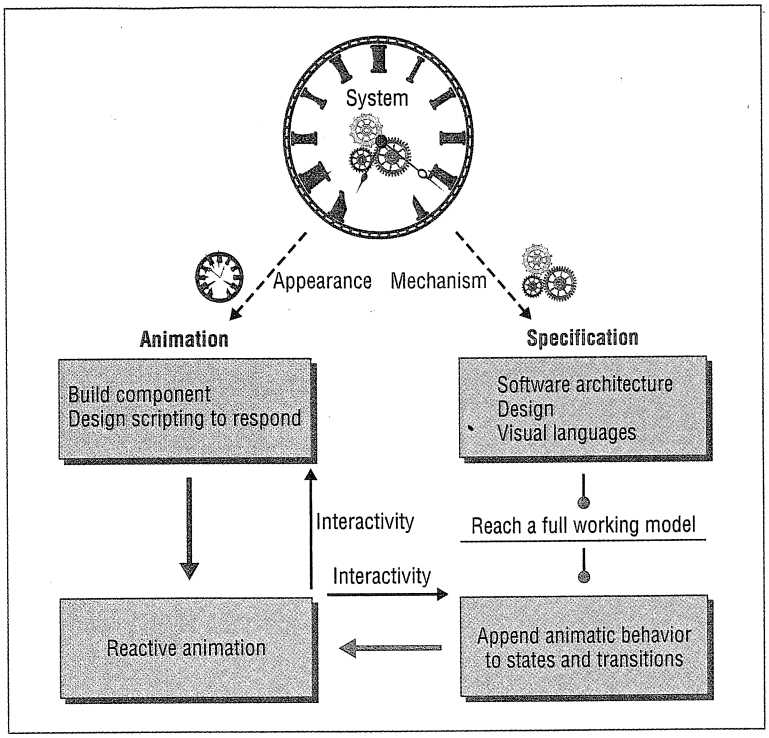


Figure 1. An RA concept diagram. Appearance and mechanism represent two design paths that start separately but eventually connect, thus bridging the gap between what a system looks like and what it does.

A classic example is the difficulty many people have programming their VCRs, a state due in part to the limited possibilities of the VCR's interface (at least in most models). The VCR program itself is a reactive system, however; as such, a designer can endow it with RA through an interface that uses any monitor connectable to a network. Thus, without changing the VCR's reactive program—just allowing it to communicate—any user can build a custom legible interacting interface and run it on the display of choice.

IMPLEMENTATION EXAMPLE

We used RA to model and simulate the development of T cells in the thymus gland, a highly complex process that involves many cells, chemical substances, surface molecules, protein consumption, and more.[8] To build the model, we analyzed and incorporated the information in more than 300 scientific papers using mainly Statecharts and Rhapsody (the reactive engine) linked with Flash (the front end).[9,10] Readers can view several prerecorded video clips of the simulation, including segments of the animation and the statecharts in operation, and some examples of typical interaction at www.wisdom.weizmann.ac.il/~dharel/ReactiveAnimation/.

In this RA model, we view each cell as a reactive object and assign it a detailed statechart, which is

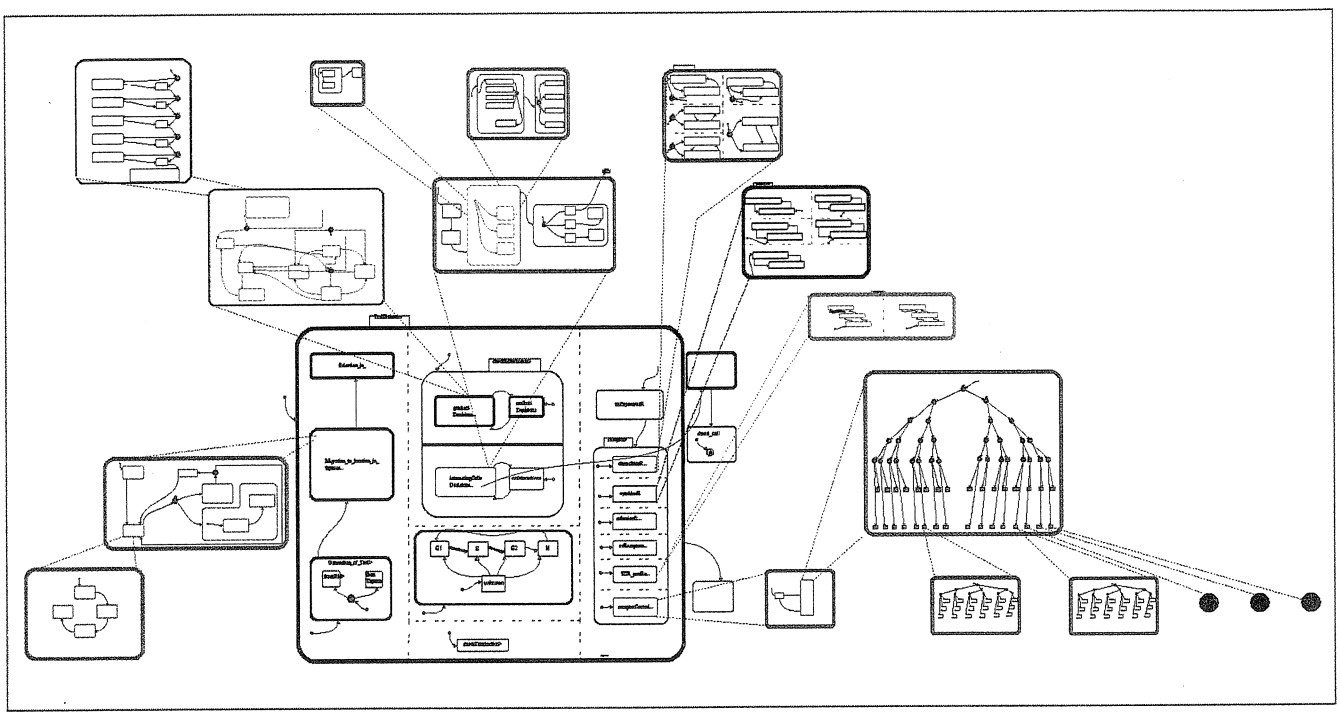


Figure 2. T-cell statechart. The dotted lines depict a zoom into the main statechart's various states.

identical within each cell type. Thus, different cell types correspond to different statecharts. Some of the molecules have their own reactive behavior, while others follow a behavior specified through mathematical equations. In a typical execution, the model generates thousands of cells, each with its statechart instance.

The most complex behavior belongs to the class of developing T cells. Figure 2 shows parts of the statechart of the T-cell class. The statechart specifies behaviors that correspond to how the cell checks its surroundings for chemical gradients and neighboring cells, how it internalizes molecular substances, how it moves, how it interacts with other entities in the model, and how it makes decisions, such as cell death and cell proliferation. Other relevant cells perform other tasks, such as growth, molecular secretion, or molecular consumption.

A small piece of behavior

Over the years, the members of the immunological community, our potential clients, have developed conventions for naming the different cell development stages and have more or less agreed on a visual convention for many aspects of cell description. As an example of how animation can reflect a small piece of behavior, consider a cell's receptors. These molecules, which reside on the cell's surface, are involved in various interactions with other cells.

During development, cells frequently change their receptors' state of expression, and the convention is to use different names for the various receptor profiles. Both the Boolean property of receptor availability/unavailability on a cell's surface and the actual act of dynamic change in the expression of various receptors are of biological relevance. Figure 3 shows three description types: state-based, illustration-based, and text.

To signify the event of a receptor becoming expressed, a small piece of animation depicts the receptor's smooth movement from the cell's interior to its surface. Different receptors have different visual features (color, size, and structure). Flash dynamically assigns these features a location on the cell surface according to spatial considerations or expression level. Figure 3b shows part of this assignment process.

When a receptor changes its expression status because of some event, Rhapsody sends an appropriate message to the Flash animation. We designed the setup so that the animation doesn't simply make the receptor appear or disappear from the cell surface. Rather, the animated receptors are themselves simple components that use their own special behavior to portray the dynamics of their expression changes. A newly expressed receptor will thus move from inside an animated cell to its surface, using a path computed ad hoc, according to the expression level and position of neighboring receptors.

An entire cell

We made several decisions on both sides of the RA scheme—reactive model and animated front end—to produce a dynamic representation of an entire T cell in operation.

As a cell continuously checks its surroundings for molecular signals that direct its movement, it might find an appropriate molecular signal that will tell it to move one micron to the right. The reactive model implements this by changing the location attributes of the cell's instance. This translates easily to animation as ratio-based movement (movement proportional to the cell's physical dimensions).

During this routine check of its surroundings, a cell might find another cell or molecule to interact with. If so, the reactive model will switch to an *interacting* mode, which the cell's statechart models as a state. The running Rhapsody simulation easily identifies an interacting mode and performs a series of actions when the model enters that state.

One action is notifying the Flash animation of the impeding interaction through the communication channel. When Flash receives the message, it looks for the appropriate instance from the pool of active instances and notifies it to display "interaction." The animated instance handles the message as its programming dictates. We programmed the animation in the T-cell development model to display a pulsating red visual gradient, for example. This pulsating gradient (or similar animation) might also prompt the viewer to query the interaction for additional information. The viewer simply clicks on the pulsating cell and selects certain kinds of information at various levels from a drop-down menu.

Once Rhapsody initiates an interaction, it identifies a specific relevant interaction from the pool of available interactions. For example, a T cell might start interacting with an epithelial cell, another important cell type in the thymus, which the front end depicts as lengthy diagonal-line-like forms. Consequently, the engine will select a way to handle the interaction. It can be the user's explicit choice, a computationally random choice, or a more intelligent internal choice.

After it selects the way it will deal with the interaction, the engine instantiates a new instance of the special class we have built to handle the interaction and automatically launches its statechart. Rhapsody now follows this statechart's behavior as part of the simulation, leading to whatever biological results it yields. Once the interaction is complete, the statechart terminates its operation, and Rhapsody eliminates it.

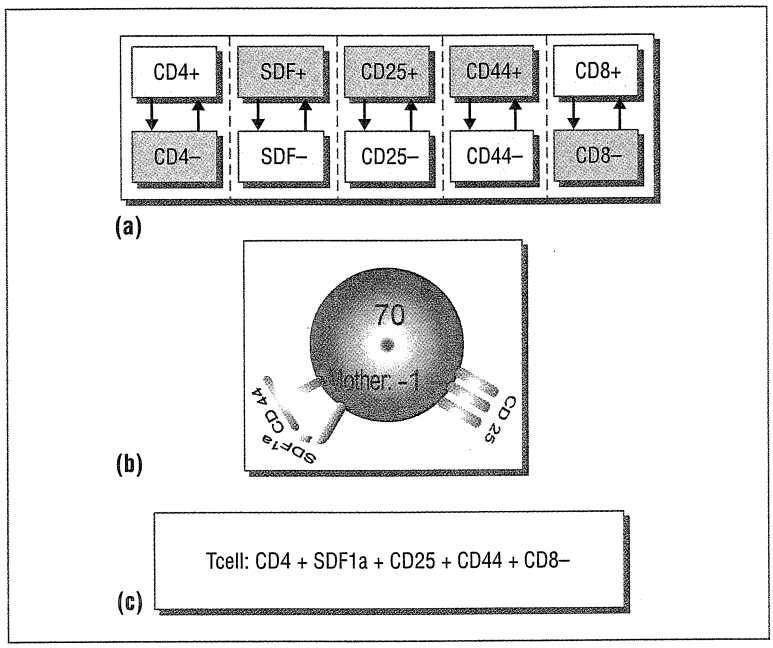


Figure 3. Three ways to describe a phenomenon: (a) state-based presentation, (b) cartoon, and (c) text-based molecular notation.

The full model

Like a real thymus, our full thymus simulation generates numerous developing T cells. In a typical run, it generates a few tens of thousands of cells, each of which is driven by its own highly complex statechart. When these cells run in tandem, they interact with each other, express receptors, consume and respond to events, move around, divide, die, enter and leave specific areas of the thymus, and so on. The resulting movie is never preknown or self-repeating. Rather, like the biology itself, the animation depends on the minute details of the structure of developing epithelial cells, their molecular expression patterns, and other random and random-like events, enhanced by their influence on the large populations of cells.

Figure 4 shows the front-end cell and organ views. The cell view lets users inspect single cells or small cell groups in more detail. The organ view is a zoom-out of the simulated system, incorporating the running reactive model's entire activity. The user switches from one view to the other by clicking on the magnifying glass.

The wealth of immunological data available to biologists has resulted in several theories for explaining specific observations relevant to T-cell behavior in the thymus. We have incorporated these theories into our model. As part of the reactive simulation, the user has access to a catalog of proposed theories and can choose among them (at both the start of simulation and during it); the chosen theory, in turn, can influence the animation and its results.

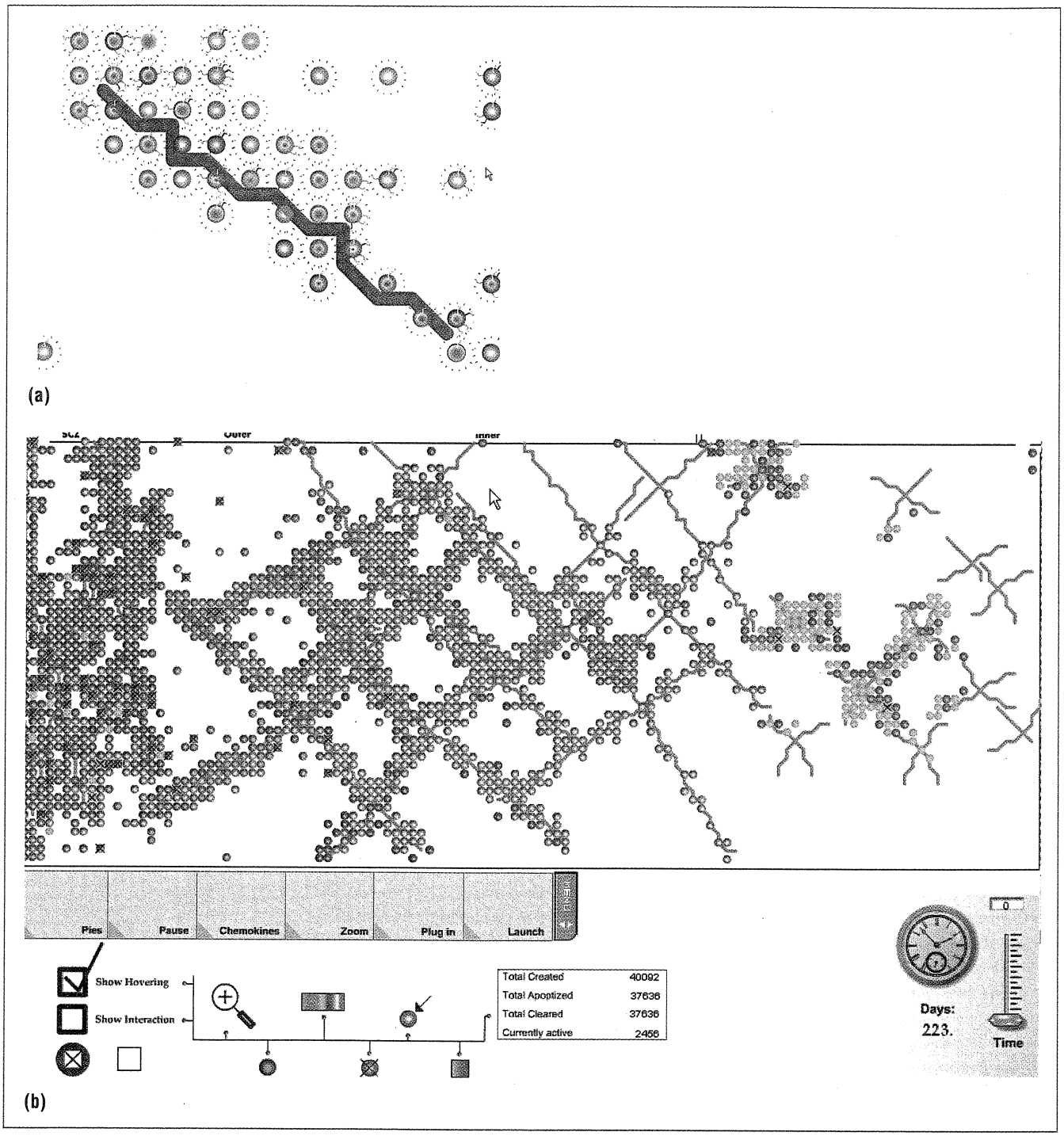


Figure 4. Front-end cell and organ views. (a) Still picture taken from continuously generated animation representing the running reactive simulation. (b) Zoom-out view of the simulated tissue with items that facilitate interactions at various levels.

One such example involves the various theories on the commitment choice that T cells make near their final stages of maturation to become either helper or cytotoxic cells. Different theories suggest different outcomes to the cells' interactions at their double-positive stage (a cell-maturity stage with a host of significant behavioral characteristics). By switching between theories, we can watch and

compare—on the fly—the influences of each theory on the resulting lineage repertoire and on the entire organ's dynamics.

Interacting with the model

The user might want to interact with the running simulation through the front end, to query the simulation or modify it at some point as it runs. The various mechanisms of alerts and animated changes that flow between the reactive engine (Rhapsody) and the front end (Flash) can prompt the user to query specific areas or specific cells in the simulation. For example, a user might want to follow the history of a cell's movements, see a list of a cell's molecular details and its interaction partners, and so on. More statistically complex information is sometimes required as well, such as collective data about many cells and molecules. We have organized the queries according to two types. Simple queries are those the user makes by questioning the running simulation. The other type, queries involving multiple instances, requires a statistical utility.

For simple queries, the user first clicks on a cell, which opens a menu. Figure 5 shows the available options. If the user chooses Interaction Details, for example, Flash sends an appropriate request to Rhapsody, which identifies the proper instance, gathers the needed information, and sends it back to Flash. Flash then displays the information using a specially devised mechanism. The other options—Show Trail, Show TCR Sequence, Link to Parent, and Developmental Stage—work in a similar manner, with more sophisticated means of presentation. Additional queries are available through other parts of Flash.

More complex queries, involving multiple instances, require a more complicated mechanism. For these, we have constructed a communication channel between the Rhapsody simulation and tools, such as Matlab, that are specially designed to handle data of large proportions. When the user asks through Flash (or a similar front end) for this kind of analysis, the communication channel transfers the needed data to Matlab for processing, and Flash displays the results.

The user can manipulate any of several tools to change the simulation's running dynamics. Figure 5 shows some of the choices available at the T-cell level. In the figure, the user has clicked on Change Receptors, which opens the box at right listing possible receptors. The user then clicked on the Chemokine Receptors option.

By choosing which chemokine receptors to express or internalize, the user actually sends a message to the reactive engine, which performs the requested action (if it is valid) and the now-changed simulation resumes. For example, choosing to express the receptor for TECK would change states and features in the reactive simulation. The reactive engine identifies these changes as events that Flash must be aware of. When Flash receives Rhapsody's notification, it allocates movie components to extract relevant movie clips for the expression process of the TECK receptor on the cell's surface and then applies them. In this way, RA encourages and supports experimentation in silico.

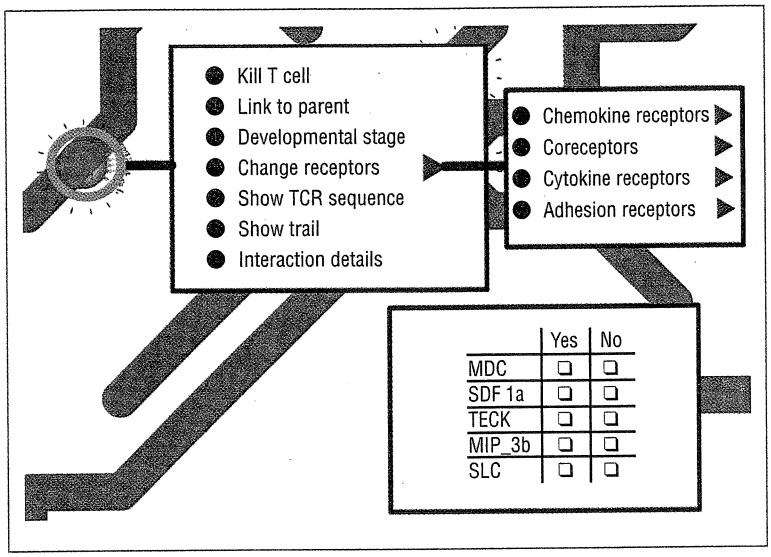


Figure 5. Interactive menu available as the simulation runs. The user has clicked on a T cell and chosen the Change Receptors option. Flash notifies Rhapsody, which makes the change and resumes simulation.

Some insights

No less important, RA explicates emergence. Our model needs only basic molecular and cellular data to generate an anatomically accurate thymic lobule starting from only a relatively few stem cells (evident in the long-run and partial-run movies on our demos Web page). The animation shows that immature cells proliferate at specific zones, while mature cells proliferate at anatomically different locations. This leads to the conclusion that the T-cell repertoire is largely selected through immature interactions by thymocytes that proliferate in a specific zone in the thymic lobule.

This conclusion is entirely novel to thymologists, and we view our ability to convey such a hypothesis to experimentalists as a direct result of RA. Further, RA's interactive nature makes it possible to knock out molecules or cells and observe the effects. The three main in silico experiments we have carried out involved knocking out SDF1, TECK, and MDC. Each such knockout influences the resulting thymus morphology in a different way,

a phenomenon that RA makes visible quite effectively. Others have already experimentally validated in vivo two of the three predictions we made after these simulations. Our prediction regarding the third knockout claims a developmental arrest at the later stages of cellular differentiation and reduced cellularity at specific anatomical locations.

One phenomenon a running RA simulation shows clearly—even in short movies—is that developing cells compete with each other in attempting to reach the arms of epithelial cells and interact with them. We did not incorporate this competition directly into the model; rather, it is a behavioral derivative that emerges from the lower-level data that constitutes the model. The animated representation clearly exhibits traffic jams that occur as the cells jostle each other across molecular gradients.

Cell competition is actually a life-or-death matter: During development, thymocytes require suitable stimulation by interacting with epithelial cells; otherwise they die (through apoptosis, or death by neglect). RA shows us how competition might physiologically regulate apoptosis. When we tweaked the parameters of cell competition by modifying the kinetic constants, we found that eliminating competition allowed too much survival, and this abolished the normal pattern of apoptosis that was concentrated in specific anatomical zones. RA has thus helped clarify the heretofore unappreciated role of cell competition in thymocyte physiology.

We have made additional predictions about the effect of cell competition, cell speed, chemokine consumption, and more, on T-cell survival, which we plan to publish in the near future. The revelation of emergent properties and the in silico experiments that RA makes possible thus arm the experimentalist for a renewed investigation of the real world. In effect, representation and experimentation in silico prompt experimentation in mundo.

Reactive animation offers great promise as a way to enhance the visualization of system behavior, especially for reactive systems, which are a central part of most current computerized technologies. By separating the two facets of the design problem—animation and reactive behavior—we can equip both with state-of-the-art tools.

Gaming lends itself particularly well to using RA. The reactive behavior of game characters is driven by extensive know-how, and gaming can potentially benefit significantly from the experience accumulated in reactive system development. The behavior of the characters might evolve through enhancements to their computer vision, emotional behavior, or playtime capabilities. We believe that using even more powerful approaches and tools to design the reactivity engine and front end will make incorporating such enhancements much easier.

We are currently working on strengthening RA by using 3D animation and evaluating RA on platforms other than Rhapsody and Flash. One plan is to connect the Play-Engine for reactivity[11] with Maya (www.alias.com/eng/products-services/maya/) for animation. We are also aiming to make RA more generic by designing a plug-in for connecting any appropriately abstracted reactive engine with any animation tool. In fact, the separation of interface from computation that RA supports can significantly strengthen mobile computing. The inability to mobilize CPU power and peripheral connectivity often constrains standard kinds of mobile computing. When programmed with RA, software is not bound to its CPU, since RA lets the program run on one CPU while the interface, or the visualization, runs on another CPU and on a different machine.

We also expect to expand our biological models over time, adding technologies, algorithms, and databases. The biological model for the thymus could, with time, assimilate a detailed model of molecular events on the cellular level and could even lead to the realistic RA modeling of a complete animal.[12] Such models could have an architecture based on played scenarios[11] or on a combination of scenarios and state-based behavior. Whatever the case, one benefit will always be the easily shared front end, since RA is intended to be first and foremost a cross-cultural communicator. ■

References

1. D. Harel and A. Pnueli, "On the Development of Reactive Systems," *Logics and Models of Concurrent Systems*, K.R. Apt, ed., NATO Advanced Science Institute Series, vol. F-13, Springer-Verlag, 1985, pp. 477-498.
2. W. Reisig, *Petri Nets: An Introduction*, Springer-Verlag, 1985.
3. *ITU-TS Recommendation Z.120: Formal Description Techniques (FDT)—Message Sequence Chart (MSC)*," Int'l Telecomm. Union (ITU), Geneva, 1996.
4. D. Harel, "Statecharts: A Visual Formalism for Complex Systems," *Scientific Computing Programming*, vol. 8, 1987, pp. 231-274.
5. J.T. Stasko et al., eds., *Software Visualization*, MIT Press, 1998.
6. J. Kaye and D. Castillo, *Flash MX for Interactive Simulation: How to Construct & Use Device Simulations*, OnWord Press, 2002.

7. W.R. Stevens, *The Protocols*, Addison-Wesley, 1994.
8. I.R. Cohen, *Tending Adam's Garden: Evolving the Cognitive Immune Self*, Academic Press, 2000.
9. S. Efroni, D. Harel, and I.R. Cohen, "Toward Rigorous Comprehension of Biological Complexity: Modeling, Execution, and Visualization of Thymic T-Cell Maturation," *Genome Research*, vol. 13, 2003, pp. 2485-2497.
10. I.R. Cohen, "Informal Landscapes in Art, Science, and Evolution," to be published in *Perspectives in Biology and Medicine*, 2005.
11. D. Harel and R. Marelly, *Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine*, Springer-Verlag, 2003.
12. D. Harel, "A Grand Challenge for Computing: Full Reactive Modeling of a Multi-Cellular Animal," *Bull. European Assoc. Theoretical Computer Science*, no. 81, Oct. 2003, pp. 226-235.

Sol Efroni is a postdoctoral fellow at the National Cancer Institute Center for Bioinformatics, where his work includes computational analyses of cellular molecular pathways over high-throughput experiments. He received a PhD in both immunology and computer science from the Weizmann Institute of Science and developed RA during his PhD thesis research. Contact him at sol@nih.gov.

David Harel is the William Sussman Professor in the Department of Computer Science and Applied Mathematics at the Weizmann Institute of Science. A cofounder of I-Logix Inc., he has worked in many computer science areas, including automata and computability theory, logics of programs, database theory, software and systems engineering, visual languages, diagram layout, modeling and analysis of biological systems, and the synthesis and communication of smell. Harel invented statecharts, coinvented live-sequence charts, and was a member of the team that designed Statemate and Rhapsody. Recently, he has codeveloped the play-in/out approach to scenario-based programming and the Play-Engine. He is a fellow of the IEEE and the ACM. Contact him at dharel@weizmann.ac.il.

Irun R. Cohen is the Mauerberger Professor of Immunology at the Weizmann Institute of Science; director of the Center for the Study of Emerging Diseases; a founder and steering committee member of the Center for Complexity Science, Jerusalem; and director of the National Institute for Biotechnology in the Negev, Beer Sheva. Cohen does basic research in immunology and has developed novel immune therapies, such as T-cell vaccination for autoimmune diseases and peptide therapy for type-1 diabetes mellitus, both now in clinical trials. Cohen studies and models design principles of the immune system and has proposed a cognitive theory for immune system behavior. Contact him at irun.cohen@weizmann.ac.il.