Visualizing the Dynamics of Conceptual Behavior Models:
The Vivid OPM Scene Player

Sergey Bolshchikov, Judith Somekh, Shay Mazor, Maxim Monadeev, Shaul Hertz, Mordechai Choder, and Dov Dori

Abstract

Modeling plays an increasingly important role in the lifecycle of systems. Complex systems are difficult to model, preventing users from deeply understanding their intricate behavior. Existing conceptual modeling languages contain behavioral diagrams aimed to describe how the modeled system changes over time. However, they are still static diagrams that do not directly reflect the system's behavior in space and time in a manner that is close to conceived reality. Models that are inherently dynamic can potentially provide system architects and designers, as well as prospective customers, with profound understanding of the behavior of the system under development without requiring knowledge of any specific modeling language. We present a new software module, *Vivid OPM*, which generates from an OPM conceptual model a "video clip" of the system under development and plays it. While requiring relatively little effort on the side of the modeler, this option explicates how the system behaves over time, providing a powerful tool for understanding and communicating any complex system's dynamics.

Index Terms – Object-Process Methodology (OPM), Simulation, Visualization, Dynamic Models, Spatio-Temporal Models.

Introduction

Conceptual modeling is increasingly recognized as a vital stage in the process of developing any complex system. It allows expressing the meaning of terms and concepts used by domain experts to discuss the problem and to find correct relationships between different concepts (Fowler, 1997).

OPM (Dori, 2002) is a comprehensive approach to conceptual modeling for system development, evolution, and lifecycle support. OPM represents the two things that are inherent in a system: its objects and its processes. In OPM, any system is described in terms of things that exist — objects, and things that happen to the objects — processes. Objects are what a system or product is, and they may be stateful, i.e., have states. Processes express what a system does — how it transforms the objects, where transformation means generation of new objects, consumption of existing objects, or change of their state. The OPM model shows the structural relations and procedural links between the system's building blocks at any needed level of detail. The single model provides for clear and expressive animated simulation of the OPM model, which greatly facilitates design-level debugging (OPCAT, 2010).

OPM explicitly addresses the system’s dynamic-procedural aspect, which describes how the system changes over time. However, the resulting model is basically static, and as such, it does not fully reflect the behavior of the system being architected or designed.

OPCAT (Dori et al., 2010), the software environment that supports OPM-based conceptual modeling, does have an animation module which provides for animating the OPM model, but the animation is at the abstract, conceptual level. For example, processes happen to objects, but in the OPM model they exist as conceptual entities that become colored when they are active. We wish to decrease this level of abstraction and bring the dynamic aspect closer to reality by actually playing what the conceptual model expresses formally but not very intuitively. Moreover, the model of a complex system must include all its subsystems and be represented at a sufficient number of abstraction levels. The need to traverse levels of abstractions during the visualization further inhibits users from clearly and completely understanding the behavior of the system as a whole.

Humans grasp moving pictures intuitively. The more visual and dynamic a model is, the deeper and more intuitive the understanding of the system it represent and how the system changes over time. Thus, creation of a visual dynamic model from a static one, which represents changes of objects in space along time by mimicking the actual system’s behavior, is likely to enable better comprehension of the system’s behavior without requiring knowledge of any specific modeling language.

To enable this visual dynamic model rendering, we have developed *Vivid OPM* — a software module that takes an OPM conceptual model and plays a vivid dynamic “clip” of the dynamics of the system under development or research. Prior to pursuing this path of development, we had considered several options for solving the problems of lack of truly dynamic models. These are briefly described below.

Literature Review

In this section we briefly review research related to visual executable models.

Executable UML

Executable UML is a profile of the UML that graphically specifies a system “at the next higher level of abstraction, abstracting away both specific programming languages and decisions about the organization of the software” (Mellor, 2002). The models are testable, and can be compiled into a less abstract programming language to target a specific implementation (Mellor, 2002). Executable UML supports Model Driven Architecture (MDA) through specification of platform-independent models, and the compilation of the platform-independent models into platform-specific models (Mellor, 2002).

A system is composed of multiple subject matters, known in Executable UML terms as domains. Executable UML is used to model a domain at the level of abstraction of its subject matter, independent of implementation concerns. The resulting domain model is represented by domain charts, class diagrams, state charts and action language.

Advantages of executable UML are:

- higher level of abstraction than 3GLs,
- provision for true separation of concerns,
- support of non-deterministic behavior, and
- connection between documentation and programming language, as the models are a graphical, executable specification of the problem space that is compiled into a target implementation.

However, Executable UML has the following disadvantages (Gardner, 2006; OMG, 2010):

- it uses limited number of constructs,
- it permits limited amount of customization,
- it provides only visual notation for activity modeling, and
- its significant activity diagrams are hard to comprehend.

Real-Time Object-Oriented Modeling

Real-Time Object-Oriented Modeling (ROOM) is phase-independent modeling concept, which is divided into two correlated levels (Selic, 1994):

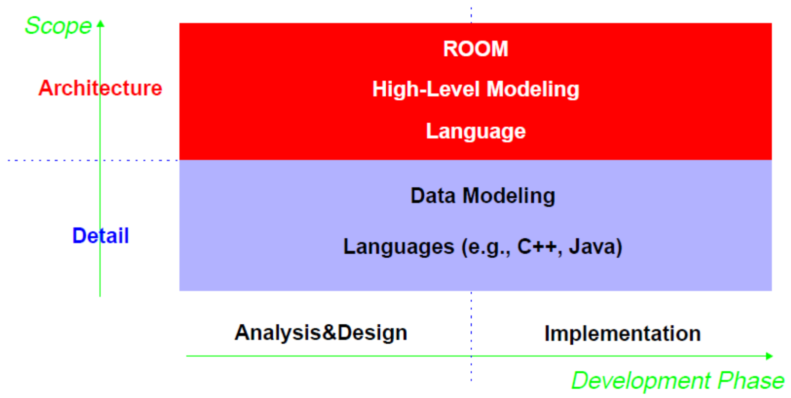
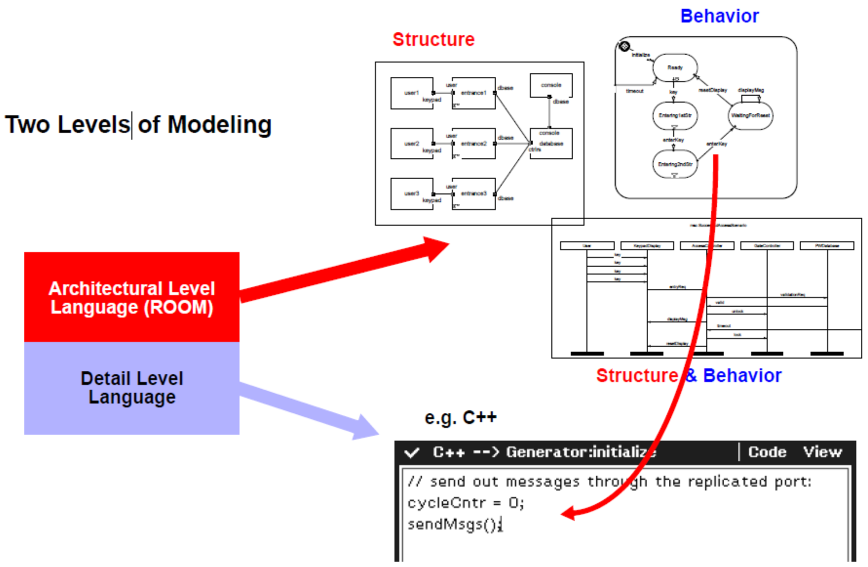


Figure 1. ROOM modeling concepts

Modeling in ROOM consists of two levels:



```
// send out messages through the replicated port:
cycleCntr = 0;
sendMsgs();
```

Figure 2. Modeling in ROOM

ROOM has a number of weaknesses (Smith, 1997):

- long learning curve,
- limit of design choice,
- documenting a design in ROOM is a challenge since many key model relationships are not expressed graphically, and
- The ObjecTime ROOM application is integrated with other third-party tools.

System Dynamics

System Dynamics (SD) is an approach to understanding the behavior of complex systems over time. It deals with internal feedback loops and time delays that affect the behavior of the entire system. What makes using system dynamics different from other approaches to studying complex systems is the use of feedback loops with stocks and flows. These elements help describe how even seemingly simple systems display baffling nonlinearity. SD has certain limitations (Lane, 2000):

- It does not always explain how flows influence stocks.
- Loops can be mislabeled.
- It cannot provide a diagrammatic explanation of all dynamic phenomena.

Live Sequence Charts

Harel and Marelly (2003) have developed a methodology for scenario-based specification of reactive systems, in which the behavior is “played in” directly from the system’s GUI or some abstract version thereof, and can then be “played out”. The approach is supported and illustrated by a tool, the play-engine. As the behavior is played in, the play-engine automatically generates a formal version in the language of live sequence charts (LCS). Play-in focuses on requirements elicitation, enabling non-professional end-users to participate in the process. As the authors note, for more complex and sophisticated features, the user is expected to be more familiar with the language of LSCs, so playing in is like programming in an intuitive, visual and high-level programming environment. Play-out allows even more end users to operate the GUI and validate the requirements by actually operating the application. While there are similarities between Play-in/Play-out and Vivid OPM, the latter focuses on making dynamic complex systems explicit and more understandable.

Vivid OPM Objective and Architecture

Vivid OPM aims to translate a conceptual OPM model into a spatio-temporal model that is driven by the conceptual model and represents the systems at a level that is closer to conceived reality than the level at which the conceptual model is. Vivid OPM comprises two main parts: OPCAT, the OPM-based systems modeling environment, and VisuOPM—a GUI platform that visualizes the system’s dynamic aspect driven by the OPCAT-resident OPM conceptual model of the system. Figure 3 depicts the Vivid OPM architecture.

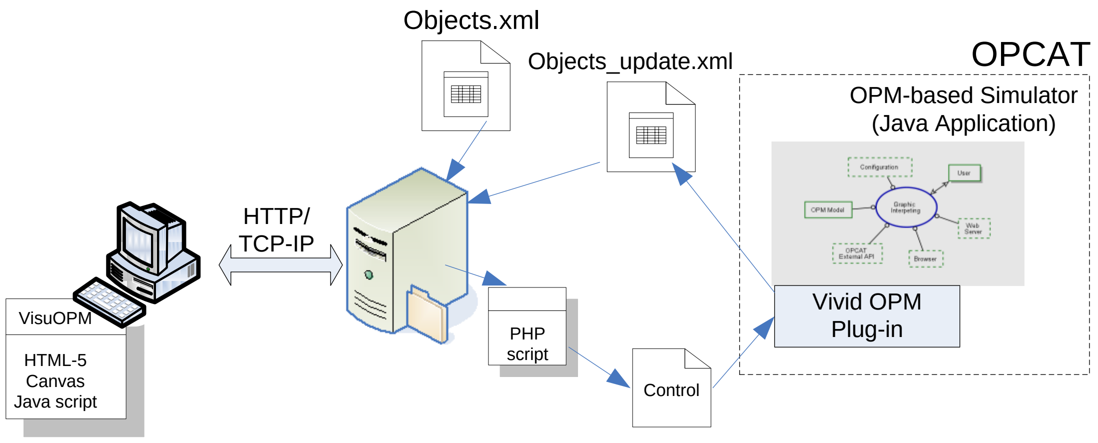


Figure 3. Architecture of Vivid OPM

The OPM-based simulator is a Java application that executes the OPM model. It enables comprehension and evaluation of behavioral aspects of the model of the system under development or research. The Vivid OPM Java plug-in tracks the objects’ state transitions and generates or updates an XML file called *Objects_update.xml*. The plug-in is capable of halting the OPM simulator execution to ensure that changes in objects’ states do not overrun the graphics transition whenever the objects’ control file reports that a previous state change transition has not yet been completed.

Vivid OPM Technologies

The VisuOPM GUI platform is based on a Web browser. The underlying client-server capabilities support communication between VisuOPM and the remote OPM simulator. The graphical visualization is based on the HTML-5 Canvas element technology (W3C, 2010). The Canvas element is currently supported by most Web browsers, including the latest versions of Mozilla Firefox, Google Chrome, Safari, and Opera. While not natively implemented by Internet Explorer 8, support is under development for Internet Explorer 9. However, many of the Canvas element's features can be supported via JavaScript libraries that rely on either Adobe Flash, or more recently, Internet Explorer's built-in support for Vector Markup Language (W3C, 1998).

The Web browser client's task is to visualize graphic elements of the model and to enable the user to control the model visualization. The server side holds all the graphic information and controls the OPM-based simulator. The client-server communication is based on HTML/HTTP/TCP-IP, while the client-server controls are implemented with PHP server-side scripting.

The HTML-5 leverages the AJAX (Asynchronous JavaScript and XML) technology by executing an HTML java script that periodically queries the server for updates in the form of XML files. AJAX provides the means for exchanging data with a server and for updating altered parts of a Web page without the need to reload the whole page. The HTML/AJAX combination allows loading all the graphical data to the client just once and using the small-sized XML transfers to update the client on the OPM model state change.

The basis of the Web client graphical display is an HTML-5 page containing both the JavaScript code and the Canvas element. The HTML-5 page which we have developed is generic, making it suitable for visualization of any OPM model. All the specific information of an OPM model, such as the size of the canvas and the coordinates of each object, is contained in a file called *Objects.xml*. Upon page loading, the Java script gets the *Objects.xml* file from the server. The file contains graphical descriptions of all the OPM objects, their states, and the required graphical dynamic transitions and effects. The Java script executes periodically gets the file *Objects_update.xml* from the server and updates the Canvas element.

The *Objects_update.xml* file contains updated information on the current state of OPM objects. An incrementing integer, assigned for each object, allows the system to check whether the browser is ready to receive the latest output from the simulation. For example, the plug-in should not relay to the browser a 3rd change of state for object A before the browser finished displaying the 2nd change. The JavaScript identifies the OPM objects that are required to be updated and triggers their graphical transitions.

The Canvas element update is executed whenever an object is in transition between states. The JavaScript code executes the transition effect, which can be movement, sizing, coloring, etc.

Attributes such as the speed of transition are described in the *Objects.xml* file for each transition between two object states.

Whenever an object's graphical transition is completed, the Java-script issues a PHP command to trigger a server-side script. The script, in turn, generates for each object a *Control file*, which contains the latest update number that the browser is presenting. For example, after the browser has finished displaying the 2nd change of state of object A, the generated *Control file* will hold the number 2. This synchronization mechanism allows the OPM-based simulator to be aligned with the graphic transition execution.

Vivid OPM Molecular Biology System Example

Biology is one of the possible applications of Vivid OPM. Biological systems, especially at the molecular level are most complex, so their models can potentially provide better understanding. However, a conceptual model of a complex, sophisticated system is inherently complex in itself. Moreover, it is written in a language that is unfamiliar to biologists, making it very difficult for them to gain deep understanding of the system under study. The Vivid OPM example that follows is aimed at presenting the biology researcher with a spatio-temporal model of the molecular biology system which is more graphic and closer to perceived reality than a static conceptual model in an unfamiliar modeling language. The OPM model, presented in figures 4-6, describes the transcription part of the mRNA lifecycle.

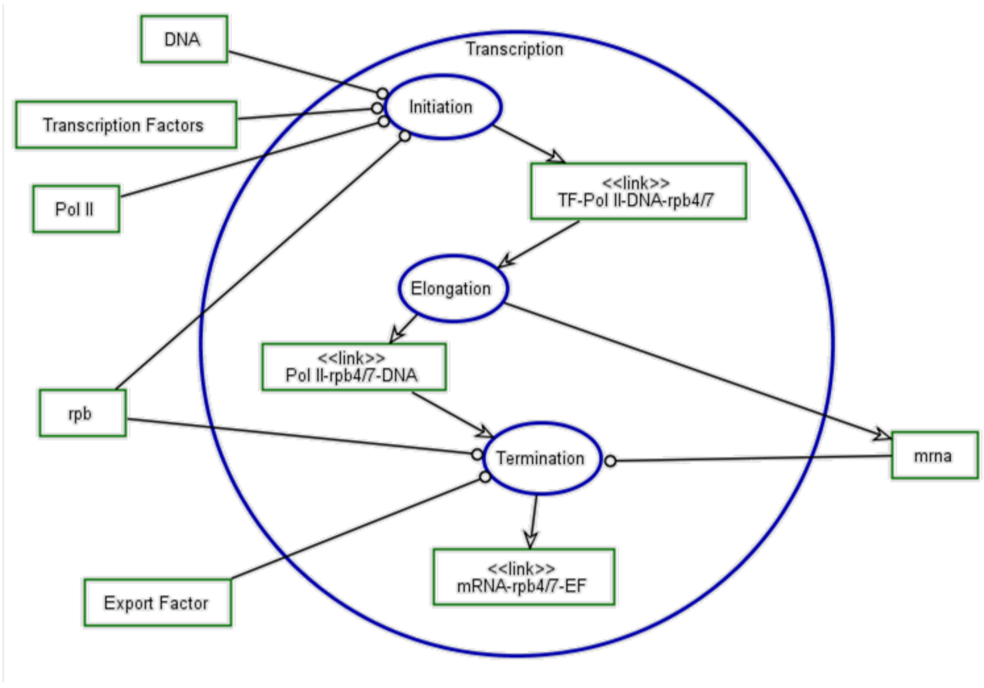


Figure 4. Transcription in-zoomed

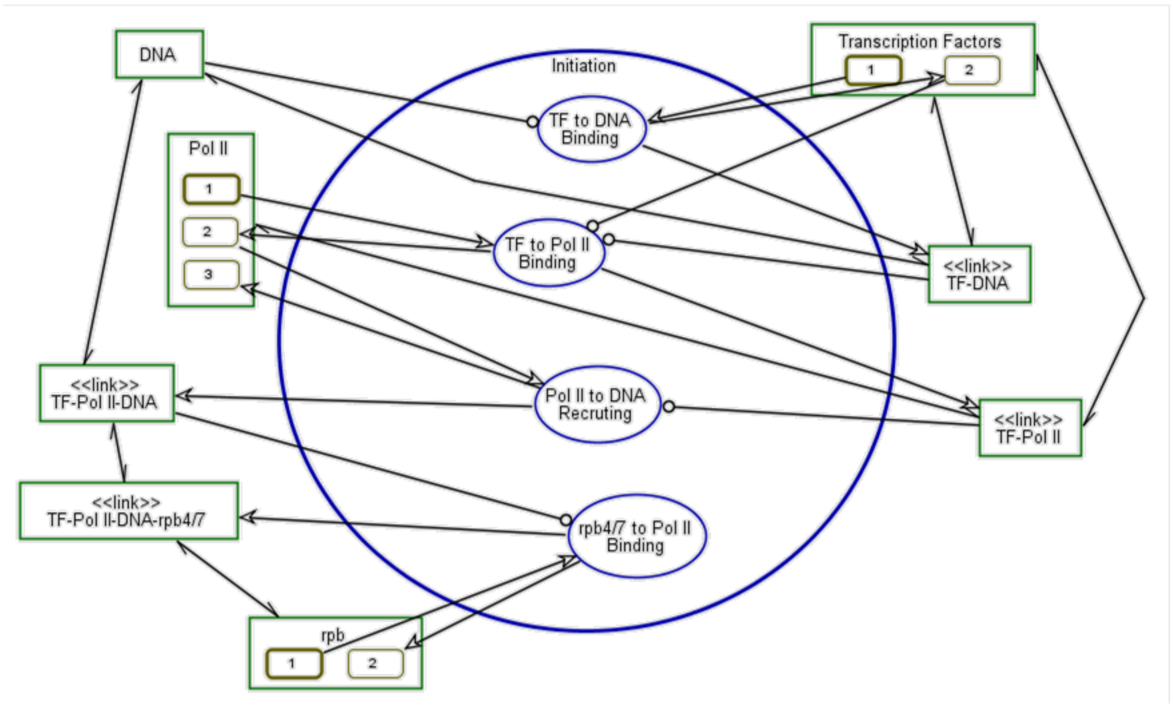


Figure 5. Initiation in-zoomed

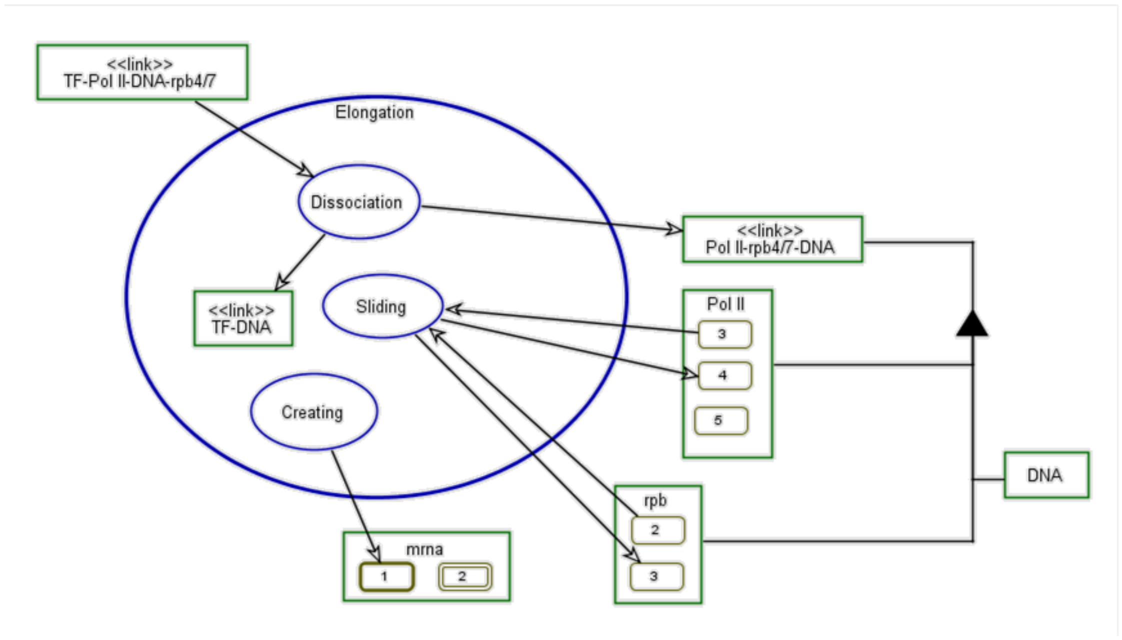


Figure 6. Elongation in-zoomed

The OPM-based formal conceptual model of creating mRNA has been validated by the built-in OPCAT simulation capability. However, the OPM model does not contain the graphical information required to animate a spatio-temporal model, such as icons for objects, the size of the drawing canvas, coordinates and size of objects, and their translation speeds. Using Microsoft Word, the user creates a file containing a canvas in which all the objects in the conceptual model are placed, as shown in Figure 8. Each object has a name and a state number which correspond to its name and states in the OPM model. The MS file is saved as an XML file called *input.xml*. Before activating the animation, the user launches a Python script (residing in a file called *VividOPM.bat*), which converts *input.xml* into the structure required for the animation which is saved into a file called *object.xml* (see Figure 7).

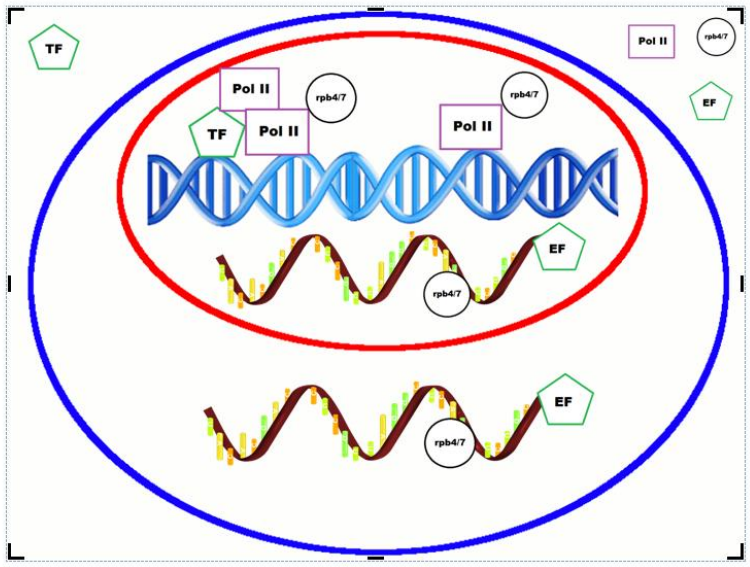


Figure 7. Example of a rendered *objects.xml* file

The Vivid OPM Graphical User Interface is presented in Figure 8. The user can activate and stop the animation simply via the Play and Stop buttons. A sliding bar allows the user to adjust the animation speed. During the animation objects move from their current positions to the places specified in the *input.xml* file.

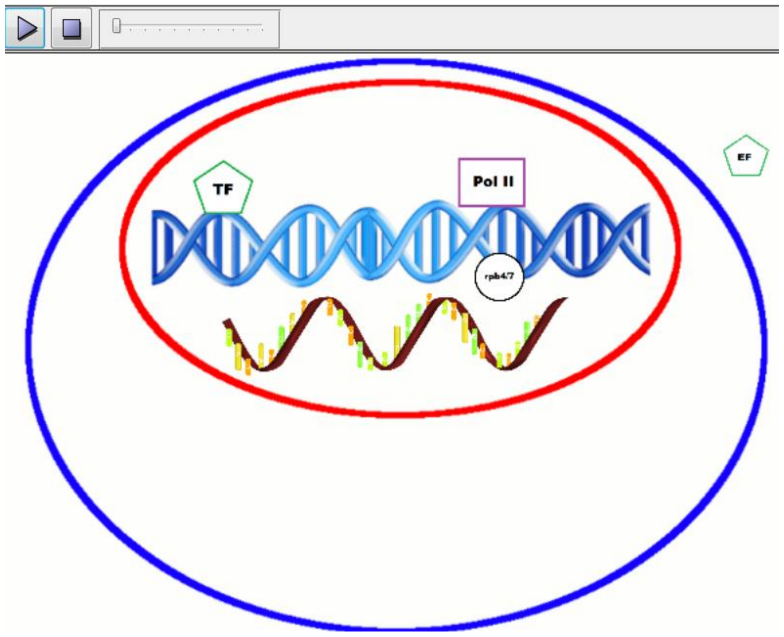
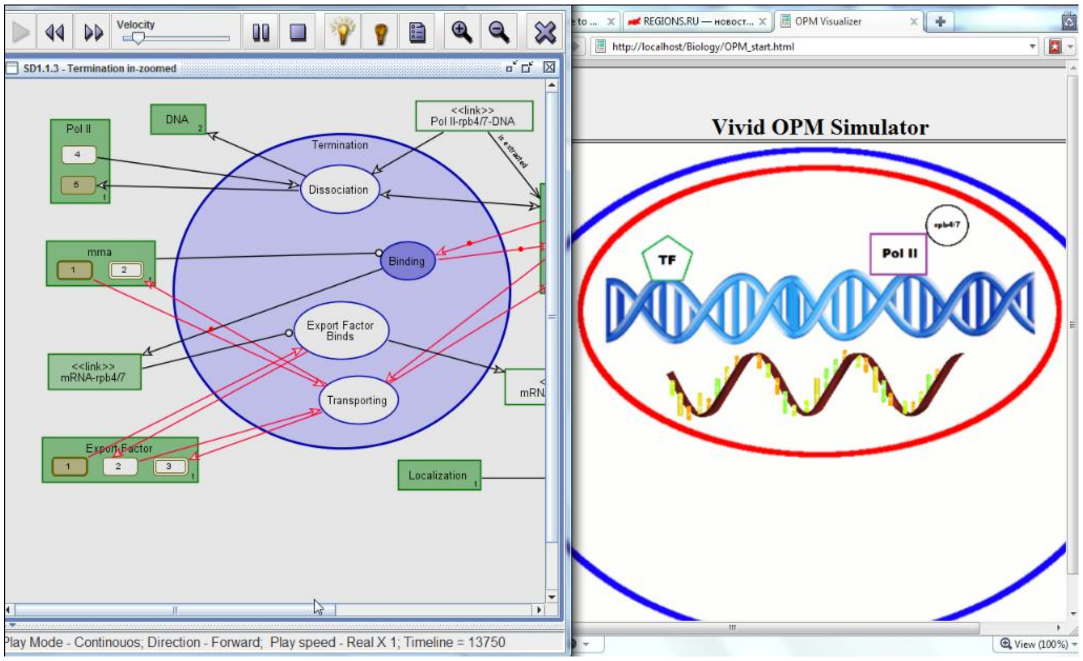


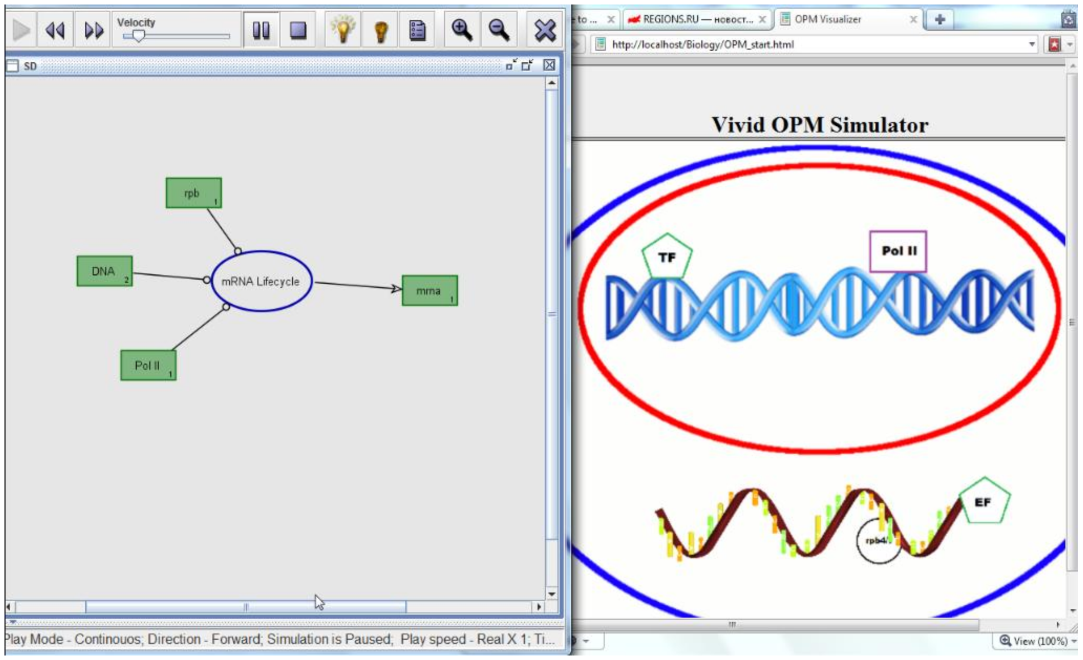
Figure 8. The GUI of the Vivid OPM animation

Figures 9 and 10 show two different points in time during the animation execution. On the right hand side is the Vivid OPM animation, which is driven by the OPM conceptual model, shown on the left hand side. Showing the two models — the conceptual and the spatio-temporal — side by side is helpful for testing the correspondence between the OPM animation and Vivid OPM and in optional. End users are expected to look mainly at the Vivid OPM clip.



Play Mode - Continouos; Direction - Forward; Play speed - Real X 1; Timeline = 13750

Figure 9. The Vivid OPM animation (right) driven by the OPM conceptual model (left) showing the binding process of Pol II to rpb4/7



Play Mode - Continouos; Direction - Forward; Simulation is Paused; Play speed - Real X 1; Ti...

Figure 10. The Vivid OPM animation (right) driven by the OPM conceptual model (left) at the end of the animation execution

Summary and Future Development

We have developed and presented a preliminary prototype of Vivid OPM, a software module which translates an animated conceptual OPM model, which is relatively abstract, into a spatio-temporal clip that is readily understandable by domain experts who need not know the underlying OPM modeling language. The obvious benefit of this presentation and play-out mode is that it provides vivid visualization of the dynamics of the system under study, which is driven by the conceptual model. Any change to the conceptual model is automatically reflected in the spatio-temporal model driven by it. The expected end users of Vivid OPM are scientists and system developers wishing to make the conceptual model concrete and “alive”. We plan to enrich the model in several directions, including improved graphics and multimedia, interaction, and support of 3D objects. Multimedia enhancements include the ability to embed video and sound in the animation. Control of the simulation flow will enable saving the animation as a video clip, adding breakpoints, and playing in reverse.

References

1. B. Selic, G. Gullekson, and. P. Ward, “Real-Time Object-Oriented. Modeling,” John Wiley & Sons, NY, 1994.
2. D. Dori, C. Linchevski, and R. Manor, OPCAT – A Software Environment for Object-Process Methodology Based Conceptual Modelling of Complex Systems. Proc. 1st International Conference on Modelling and Management of Engineering Processes, University of Cambridge, Cambridge, UK, Heisig, P., Clarkson, J., and Vajna, S. (Eds.), pp. 147-151, July 19-20, 2010.
3. D. Dori, Object-Process Methodology – A Holistic Systems Paradigm, Springer Verlag, Berlin, Heidelberg, New York, 2002.
4. D. Harel and R. Marelly, Specifying and executing behavioral requirements: the play-in/play-out approach. Software and Systems Modeling, 2, (2), pp. 82-107, 2003.
5. D. Lane, “Diagramming Conventions in System Dynamics”, The Journal of the Operational Research Society, Vol. 51, No. 2 (Feb., 2000), pp. 241-245.
6. K. Smith, “Real-Time Object-Oriented Modeling”, (http://www.drdobbs.com/184410342), 1997.
7. M. Fowler, “Analysis Patterns: Reusable object models”, Addison-Wesley Longman, 1997.
8. OMG, “Semantics of a Foundational Subset for Executable UML Models”, http://www.omg.org/spec/FUML/1.0, OMG Document Number: ptc/2010-03-14.
9. OPCAT website, www.opcat.com, accessed July 22, 2010.
10. S. Mellor, M. Balcer, “Executable UML: A foundation for model-driven architecture”, chapters 1.2, Executable UML, 1.4 Model Compilers, 1.5 Model Driven Architecture, Addison Wesley, 2002.
11. T. Gardner, L. Yusuf, “Explore model-driven development (MDD) and related approaches: A closer look at model-driven development and other industry initiatives”, (https://www.ibm.com/developerworks/library/ar-mdd3/), 2006.
12. W3C website. A vocabulary and associated APIs for HTML and XHTML, http://www.w3.org/TR/html5/spec.html, W3C Working Draft 24 June 2010.
13. W3C website. Vector Markup Language (VML), http://www.w3.org/TR/NOTE-VML, World Wide Web Consortium Note 13 May 1998.