# OPM MODEL-DRIVEN ANIMATED SIMULATION WITH COMPUTATIONAL INTERFACE TO MATLAB-SIMULINK

Sergey Bolshchikov, Aharon Renick, Judith Somekh, and Dov Dori
*Technion - Israel Institute of Technology, Haifa, Israel*
*snb@tx.technion.ac.il*, *adrenick@tx.technion.ac.il*, *yuditol@techunix.technion.ac.il*, *dori@ie.technion.ac.il*

Keywords: Object Process Methodology (OPM), Simulation, Visualization, Dynamic Models, Spatio-Temporal Models, Matlab

Abstract: Modeling plays an increasingly important role in the life-cycle of systems, starting from the design stage, as well as important in the process of studying an unfamiliar, existing system. Complex systems are difficult to model, making it harder on users to deeply understand their intricate behavior. Moreover, in many cases the system may exhibit complex computational and stochastic behavior, critical to truly representing the system. Existing modeling methodologies contain behavioral diagrams aimed to describe the systems' changes over time, but they are still static and do not reflect the behavior of systems in spatio-temporal space in a manner that is close to conceived reality. Similarly, these methodologies do not offer an ability to fully model the quantitative aspects of the system, or offer such ability at the expense of simplicity or generality of the methodology. We offer two complementing concepts, *Vivid OPM* and *OPM Matlab Layer*, which address the dynamic display and quantitative aspects by enhancing Object Process Methodology (OPM) to better reflects these aspects while keeping the holism and simplicity of OPM.

## 1 INTRODUCTION

A model is an abstract representation of some aspects of interest of a system under study or development, aimed at understanding, communicating, explaining, or designing these aspects (Dori, 2002).

There are many different ways to distinguish models types, including stochastic vs. deterministic models and discrete vs. continuous models. Figure 1 is an Object-Process Diagram (OPD) showing one of them. The differentiation is based on values of three attribute: concreteness, dynamicity, and computability. The concreteness attribute has two values: physical and conceptual. A physical model describes the 3-D structure, and possibly the behavior of the system components and their spatial arrangement. Examples of physical models can be a 3-D mockup of a building, a vehicle, or a city. A conceptual model is intended to describe the concept of how system is structured and/or how it operates. The dynamicity of a model can be d static or dynamic.

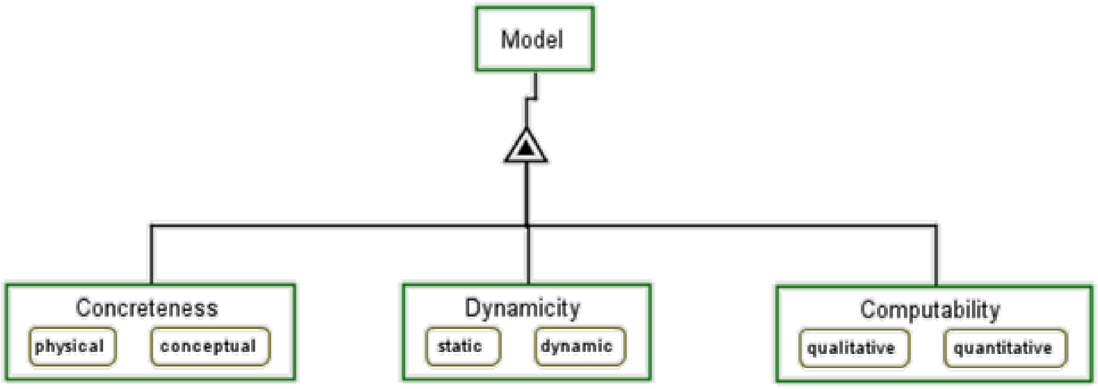


Figure 1: Classification of models by their attributes.

## 2 MODELING METHODOLOGIES

### 2.1 Modelica

Modelica is an object-oriented equation-based modeling methodology. The Modelica language defines and describes the system, its components and relations by a set of mathematical equations. This approach allows direct modeling of the computational aspects of the system, as opposed to other common modeling methods, which are based on different input/output and block-diagram abstractions (Mattsson, 1997).

Although Modelica supports quantitative and stochastic aspects in its models, it does not expense of simplicity and holism of the methodology. Describing the model mathematically requires a significant level of detailed concrete modeling, as opposed to the need for a conceptual description.

### 2.2 Play-in/Play-out

The Play-in/Play-out approach (Harel, & Marelly, 2003) is a methodology for modeling systems, specifically the reactions of the system with the environment and its different sub-systems. In the Play-in/Play-out approach, scenarios are “played in” by the user modeling the system using a tool called the “Play-engine”. The user executes the various actions that can affect the system and the reaction of the system in each case. This is done quite intuitively using a GUI representing the system and its parts. As the behavior is played in, the play-engine automatically generates a set of Live Sequence Charts (LSCs), specifying the behavior of the system.

The play-engine allows specifying an event value as a function predefined in the GUI code (for example, using visual-basic). This can allow modeling complex systems, where the result of an action is not as simple as a constant reaction, but rather has quantitative aspects.

The play-engine provides limited support for non-deterministic actions. It is possible to define more than one possible reaction for each action “played-in”, and allocate each reaction a probability.

### 2.3 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. 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.

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 a 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.

## 2.4 Object Process Methodology

OPM (Dori, 2002) is a comprehensive approach to conceptual modeling for system development, evolution, and lifecycle support. 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.

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. That is why creating a simulation of a concrete visual dynamic model which is driven by a simulation of the corresponding conceptual model is likely to enable better comprehension of the system’s behavior without requiring knowledge of any specific modeling language.

A conceptual model is a powerful tool in the process of understanding a system under design or research. Yet, due to its abstraction, a conceptual modeling does not normally include all the elements of the modeled system. In order to achieve simplicity of the model, the quantitative aspects are suppressed in favor of a more qualitative representation. Thus, it is a challenge to enable quantitative modeling while preserving the simplicity and abstraction level of the model.

# 3 VIVID OPM

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 as shown in Figure 2: 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.

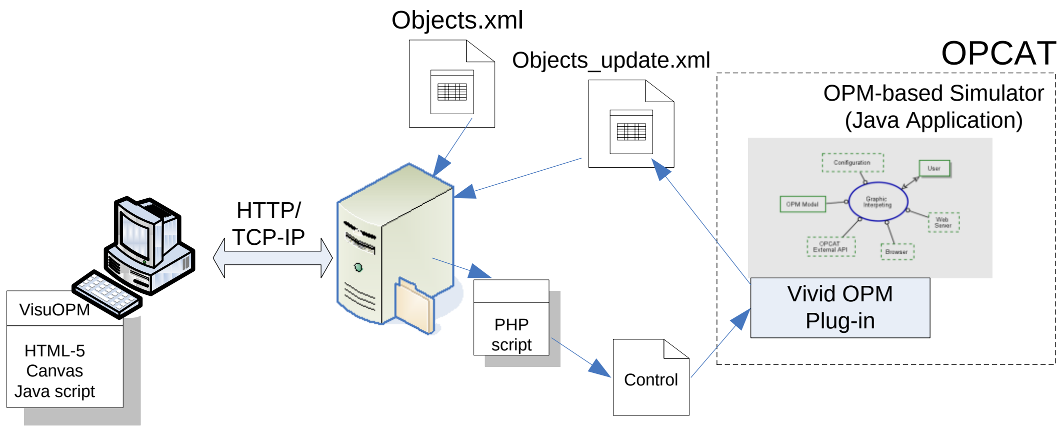


Figure 2: Architecture of Vivid OPM.

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. 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.

## 4 OPM/ML

OPM conceptually models the system, but it does not enable sufficient quantitative simulation of the modeled system. Conversely, MATLAB (Houcque, 2005) is a convenient tool for simulating complex systems, but it does not have advanced conceptual modeling abilities. Adding a numerical computational layer to conceptual modeling of OPM creates a complete conceptual model while also providing for simulating its behavior both qualitatively and quantitatively.

The architecture of OPM Matlab Layer consists of two main parts: OPCAT and the OPM/ML generator — platform that creates Matlab code (m files) from the OPM model, controls the Matlab simulator, and manipulates the OPCAT simulation according to the results in the Matlab Layer.

## 5 EXPERIMENT

### 5.1 Description

The first Vivid OPM and OPM/MATLAB Layer has been demonstrated on a biological model, which describes the transition part of mRNA, presented in figures 3-5. We show how a biological model can be enhanced using the two features. The main enhancements were visualizing the motion of biological processes and adding stochastic behavior, such that some process has a certain probability of occurrence and computing process duration.

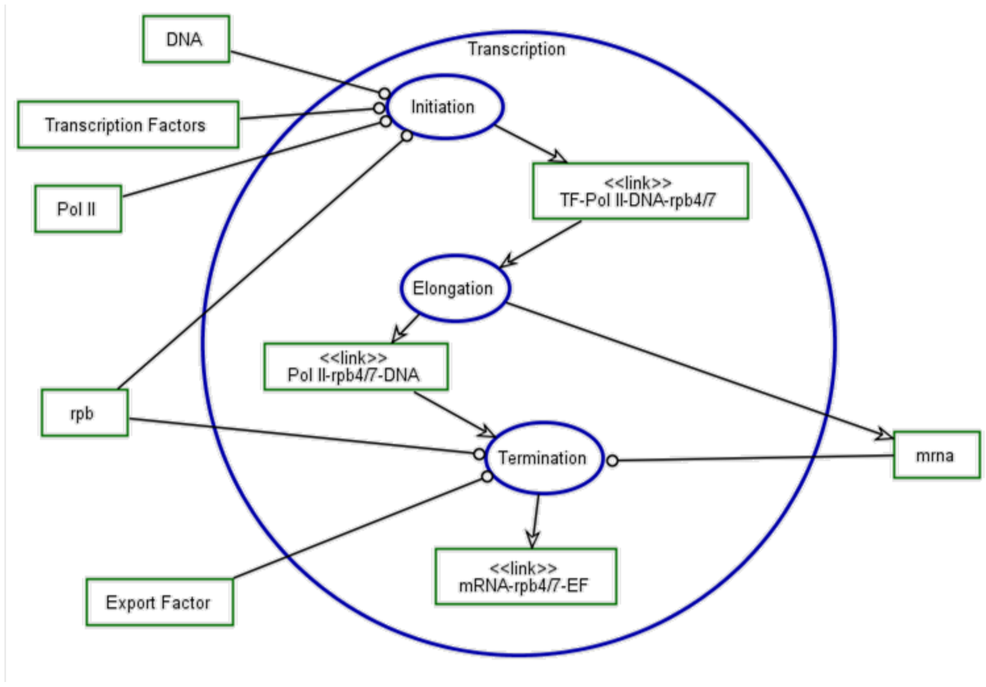


Figure 3: Transcription in-zoomed.

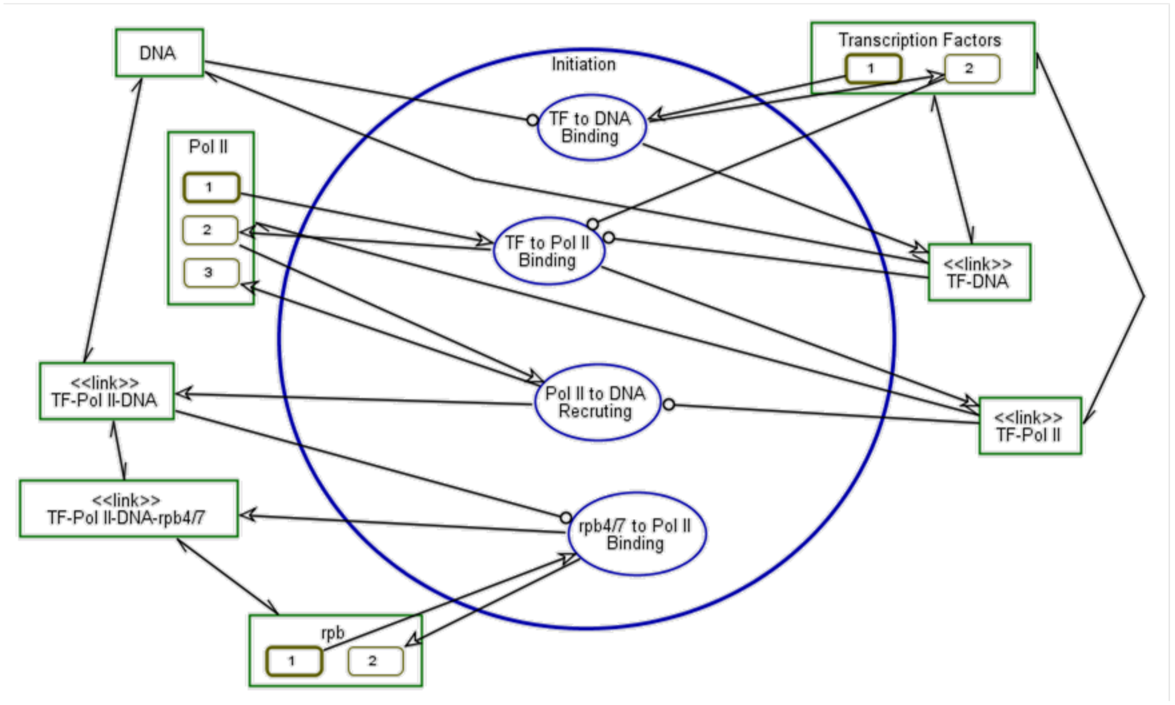


Figure 4: Initiation in-zoomed.

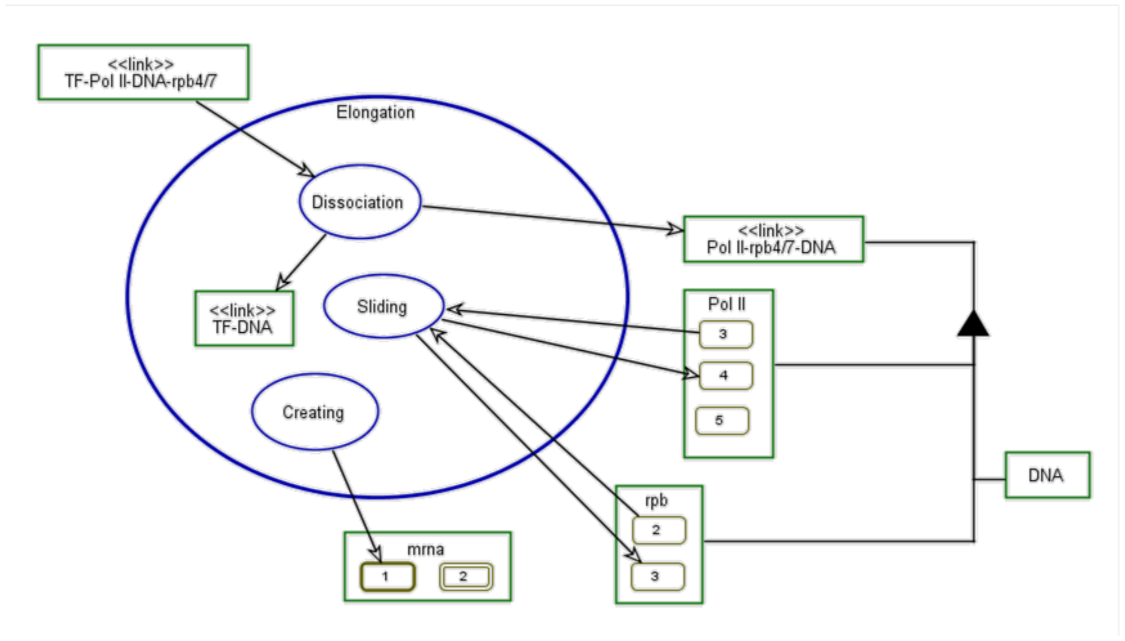


Figure 5: Elongation in-zoomed.

## 5.2 Result

Figures 6 shows 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. Being able to see the two models—the conceptual and the spatio-temporal—side by side is helpful for testing the correspondence between the OPM animation and the movement in space of the system components over time in Vivid OPM. End users are expected to look mainly at the Vivid OPM clip.

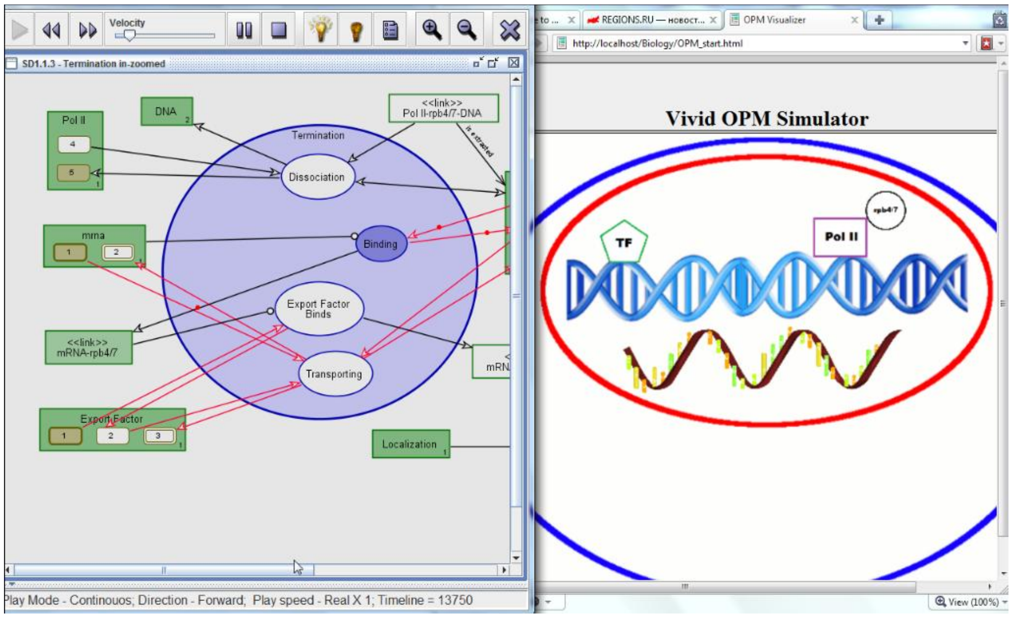

Velocity
SD1.1.3 - Termination in-zoomed
Pol II
DNA
<<link>> Pol II-rpb4/7-DNA
Termination
Dissociation
mrna
Binding
Export Factor Binds
<<link>> mRNA-rpb4/7
Transporting
Export Factor
Localization
Play Mode - Continouos; Direction - Forward; Play speed - Real X 1; Timeline = 13750
OPM Visualizer
http://localhost/Biology/OPM_start.html
Vivid OPM Simulator
TF
Pol II
rpb4/7

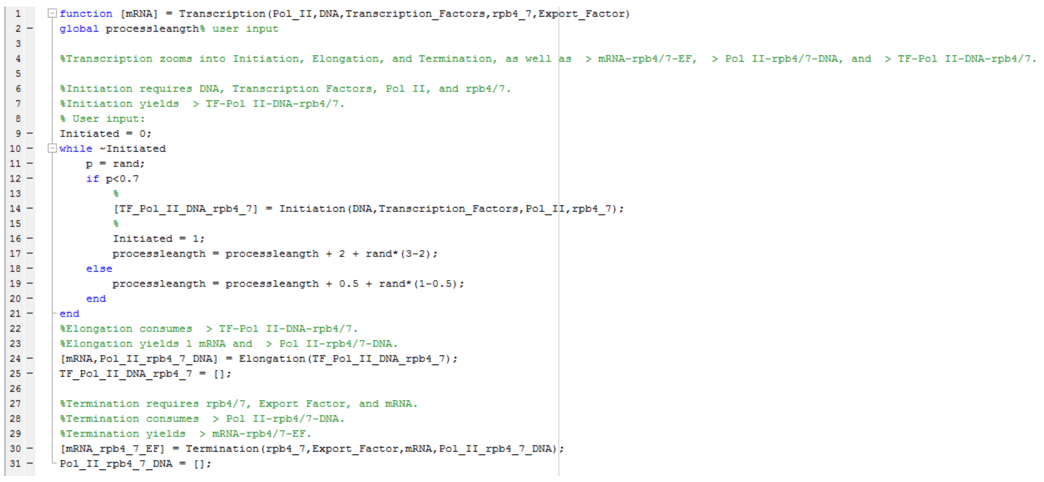
Figure 6: The Vivid OPM animation (right) driven by the OPM conceptual model (left) showing the binding process of Pol II to rpb4/7.

The original OPM model was deterministic, whereas the real biological process being modeled was assumed to have a stochastic behavior. In addition, in the original model there is no meaning to the time it takes for each process to occur. These two features that were not addressed in the original conceptual model of the system were implemented in the experiment as follows.

- The Initiation process is random, with an occurrence probability of 0.7, (a probability of 0.3 that it will not occur).
- The Initiation process has a random duration, which is uniformly distributed from 2 milliseconds to 3 milliseconds.
- A failed Initiation process has a duration uniformly distributed from 0.5 milliseconds to 1 millisecond.

The MATLAB layer was created in two stages. First, the basic MATLAB code was created using the OPL output. This code represented the original OPM model, without the OPM/ML enhancements. For demonstration purpose it was created manually using FreeMat, an open-source solution compatible with MATLAB.

Next, the enhancements where added into the MATLAB layer as presented in Figure 7. It was then possible to automatically run the model simulation many times, collect statistics on different parameters and plot various graphs. For example, we plotted a graph of the total event length over 1000 runs of the mRNA Lifecycle process, as shown in **Error! Reference source not found.**8.



```
function [mRNA] = Transcription(Pol_II,DNA,Transcription_Factors,rpb4_7,Export_Factor)
global processleangth% user input

%Transcription zooms into Initiation, Elongation, and Termination, as well as  > mRNA-rpb4/7-EF,  > Pol II-rpb4/7-DNA, and  > TF-Pol II-DNA-rpb4/7.

%Initiation requires DNA, Transcription Factors, Pol II, and rpb4/7.
%Initiation yields  > TF-Pol II-DNA-rpb4/7.
% User input:
Initiated = 0;
while ~Initiated
    p = rand;
    if p<0.7
        %
        [TF_Pol_II_DNA_rpb4_7] = Initiation(DNA,Transcription_Factors,Pol_II,rpb4_7);
        %
        Initiated = 1;
        processleangth = processleangth + 2 + rand*(3-2);
    else
        processleangth = processleangth + 0.5 + rand*(1-0.5);
    end
end
%Elongation consumes  > TF-Pol II-DNA-rpb4/7.
%Elongation yields 1 mRNA and  > Pol II-rpb4/7-DNA.
[mRNA,Pol_II_rpb4_7_DNA] = Elongation(TF_Pol_II_DNA_rpb4_7);
TF_Pol_II_DNA_rpb4_7 = [];

%Termination requires rpb4/7, Export Factor, and mRNA.
%Termination consumes  > Pol II-rpb4/7-DNA.
%Termination yields  > mRNA-rpb4/7-EF.
[mRNA_rpb4_7_EF] = Termination(rpb4_7,Export_Factor,mRNA,Pol_II_rpb4_7_DNA);
Pol_II_rpb4_7_DNA = [];
```

Figure 7: MATLAB code of Transcription function.

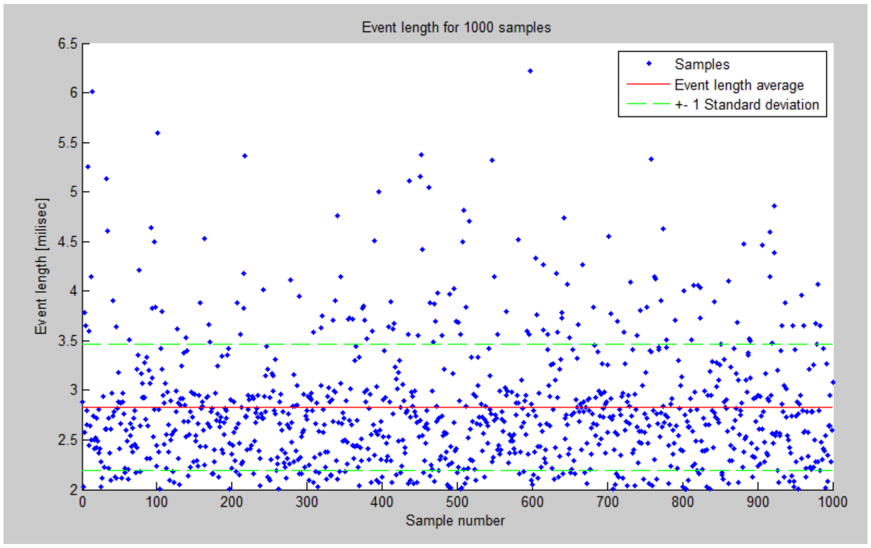


Figure 8: Graph of event length of 'Initiation' for 1000 permutations, created by MATLAB.

## 6 CONCLUSION AND FUTURE WORK

We have developed and presented a proof of concept of Vivid OPM and OPM/ML. Vivid OPM enables translation between conceptual OPM models, 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. OPM/ML significantly enhnaces OPM by adding computational ability to the conceptual model. The expected end users of Vivid OPM and OPM/ML are scientists and system developers wishing to make the conceptual model concrete and "alive", as well as more accurate in different computational aspects.

In future work we plan to integrate the two enhancements to OPM into a comprehensive modeling and simulation framework which will be conceptual, quantitative, and executable in the sense that it will look similar to the "real" system in action.

## REFERENCES

Dori, D., Linchevski, C., and Manor, R., 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.

Dori, D., *Object-Process Methodology – A Holistic Systems Paradigm*, Springer Verlag, Berlin, Heidelberg, New York, 2002.

Harel, D. and Marelly, R., *Specifying and executing behavioral requirements: the play-in/play-out approach*. Software and Systems Modeling, 2, (2), pp. 82-107, 2003.

Harel, D., Marelly, R. *Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-engine*, Springer Verlag, 2003.

Houcque, D. *Introduction to MATLAB for Engineering Students*. Northwestern University, Version 1.2, August 2005.

Mattsson, S., Elmqvist, H. *Modelica – An International Effort to Design the Next Generation Modeling Language*.

Mellor, S., Balcer, M., *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.

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.