

Object-Oriented Behavior Modeling for Biological Systems

Daniel A. Shegogue

Dept. Biostatistics, Bioinformatics and Epidemiology, Medical University of South Carolina
Charleston, SC 29425, USA

and

W. Jim Zheng*

Dept. Biostatistics, Bioinformatics and Epidemiology, and Hollings Cancer Center
Medical University of South Carolina
Charleston, SC 29425, USA

ABSTRACT

To effectively model complex biological systems biologists need methods to organize and analyze the flood of complex genomic and proteomic data being produced. The issue of handling systems complexity has already been addressed in other fields. Specifically the computer science field has defined object-oriented methodologies and architectures to handle the complexity in software systems. To demonstrate how these tools can be used to define the behaviors of a biological system and its components, we have set forth to model the behaviors exemplified by influenza A. Although, the genome of influenza A has been well studied, putting the available information into a format that is usable to biologists poses significant challenges. Here, we have used reverse engineering and literature searches to define behaviors associated with the influenza infection process and its viral components. Using object-oriented principles, static and dynamic models were developed that emulate behaviors found in the viral infection and reproduction processes.

Keywords: object-oriented, behavior, modeling, biology, and UML

*Presenting Author

1. INTRODUCTION

The genome-sequencing projects have provided information about individual components of biological systems – genes and proteins they encode. For many of these components, their functions have been studied to different degrees. These functions represent how these components work, either by themselves or by interacting with other proteins, DNA or membranes in the cell. To understand biological systems it is essential to understand the interactions and dynamics among these biological entities (bioentities). By applying object-oriented modeling, the system-level and component-level behavior in a biological system can be captured. This behavior

will delineate the dynamic process in the cell, and annotate the genome information in unprecedented ways.

The behavior of an object can be roughly defined as the combination of the abilities an object has and its reaction to its environment. These objects comprise part of a larger system behavior. Systems may be comprised of high-order and low-order behaviors, and their components may contain object behaviors defined in terms of event-independent, event and time-dependent and state behavior [1]. Event-independent behaviors are not bound by previous actions taken in the system. For instance, a protein might dimerize with another protein. However, event and time-dependent behaviors may depend on previous system actions or are specified for a particular duration. For example, a protein may only bind to another if it has previously dimerized with itself. In this example the behavior of dimerization was necessary for the initiation of the binding behavior. Finally, state behaviors are conditions that persist for an object for a significant period of time [1] (e.g. changes in a protein's glycosylation or phosphorylation status).

These behaviors can be captured as part of the dynamic processes of an object-oriented model. For an object-oriented system represented using Unified Modeling Language (UML) diagrams, these events are described by sequence, state, and activity diagrams. By applying a software engineering process, we have systematically dissected the influenza viral infection process and captured the behaviors contained in this process in sequence, state, and activity diagrams.

As a disease that affects millions of people annually, the genetic structure and regulation of the influenza virus has been extensively studied. The influenza A genome consists of 8 negative-sense single stranded RNAs that code for 11 proteins. Briefly, upon endocytosis of the virus, viral M2 acidifies the virus causing the release of M1 from the viral ribonucleoprotein (vRNP) [2]. The virus then fuses with the endosomal membrane and the vRNP is released [3]. The vRNP travels to the nucleus where transcription of the genomic RNAs produces viral

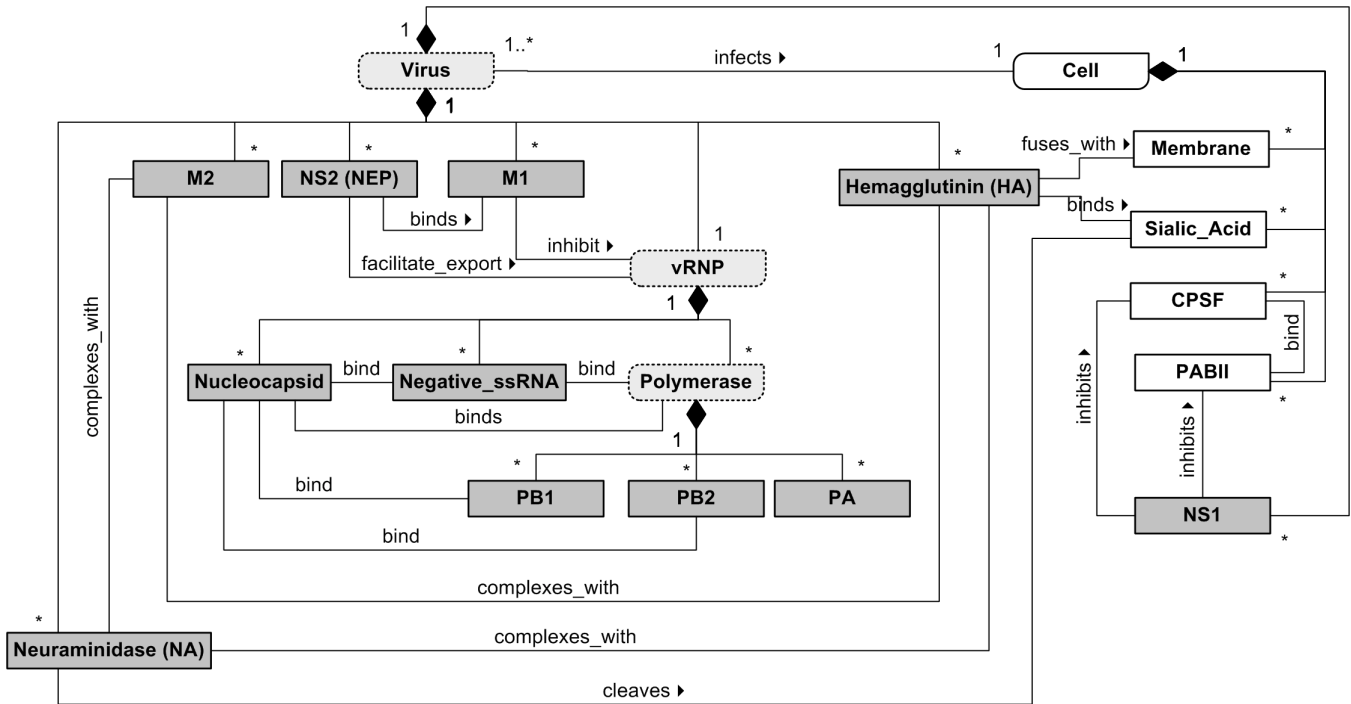


Figure 1. Conceptual diagram of the influenza virus and cell. Major components of Influenza (shaded) virus and viral infection and their associations are modeled as objects. These objects are represented by rectangles within the diagram. Rounded rectangles are high-level objects. Lines with solid diamonds (◆) at the end indicate composition. These are read from the diamond end, for example, as Virus contains vRNP. Other interactions are represented as named, binary associations (—). Interactions may contain cardinalities at their ends indicating how many objects interact with another object. As an example from above, one Virus can infect one Cell, and one Cell can be infected by one to many Viruses. Biological entities have been decomposed into objects. Object attributes and operations are hidden to reduce complexity. This conceptual model demonstrates the interactions that occur during viral infection.

mRNA (vRNA), after stealing a 5' cap from a host mRNA [4]. Viral complementary RNA (cRNA), which does not require priming for transcription, is produced without a cap or poly(A) tail. After the cRNA is encapsulated by the nucleocapsid protein, replication occurs [5, 6]. Messenger RNA is exported from the nucleus and translated in the cytoplasm. Newly synthesized proteins travel to the cellular plasma membrane or reassociate with the newly replicated vRNA. M1 displaces the vRNP from the nuclear matrix [7]. The vRNP assembles with the other viral components at the cell surface. Budding takes place through cleavage of sialic acids by neuraminidase (NA) [8].

While influenza biology is well-studied and relatively well understood, the next step in the annotation and analysis of the influenza genome remains to be taken. Therefore, we propose a methodology for defining the behavior of the influenza A system and its components using object-oriented methodologies and design. We applied a reverse engineering approach to create an object-oriented model that describes the influenza viral infection process. The well-defined software engineering process allowed us to systematically transform existing knowledge about influenza viral infection into an object-oriented model of influenza behavior that is well

represented in UML.

2. METHODS

Conceptual Model Generation

To provide an overview of the system and its interrelationships, a conceptual model, or simplified class diagram, was generated based on the information defined in the requirement-gathering phase. This conceptual model integrated biological information, and represented the viral and cellular components involved in viral infection and their relationships in UML notation. By applying object-oriented analysis, the influenza virus and the cellular components involved were decomposed into objects and component relationships were realized. However, information regarding component properties was hidden. This intermediate artifact defined the organization of the biological system and provides an overview of the components and their relationships. This and other UML diagrams in this study were generated using Microsoft Visio Pro. For a review on UML models see <http://bdn.borland.com/article/0,1410,31863,00.html>.

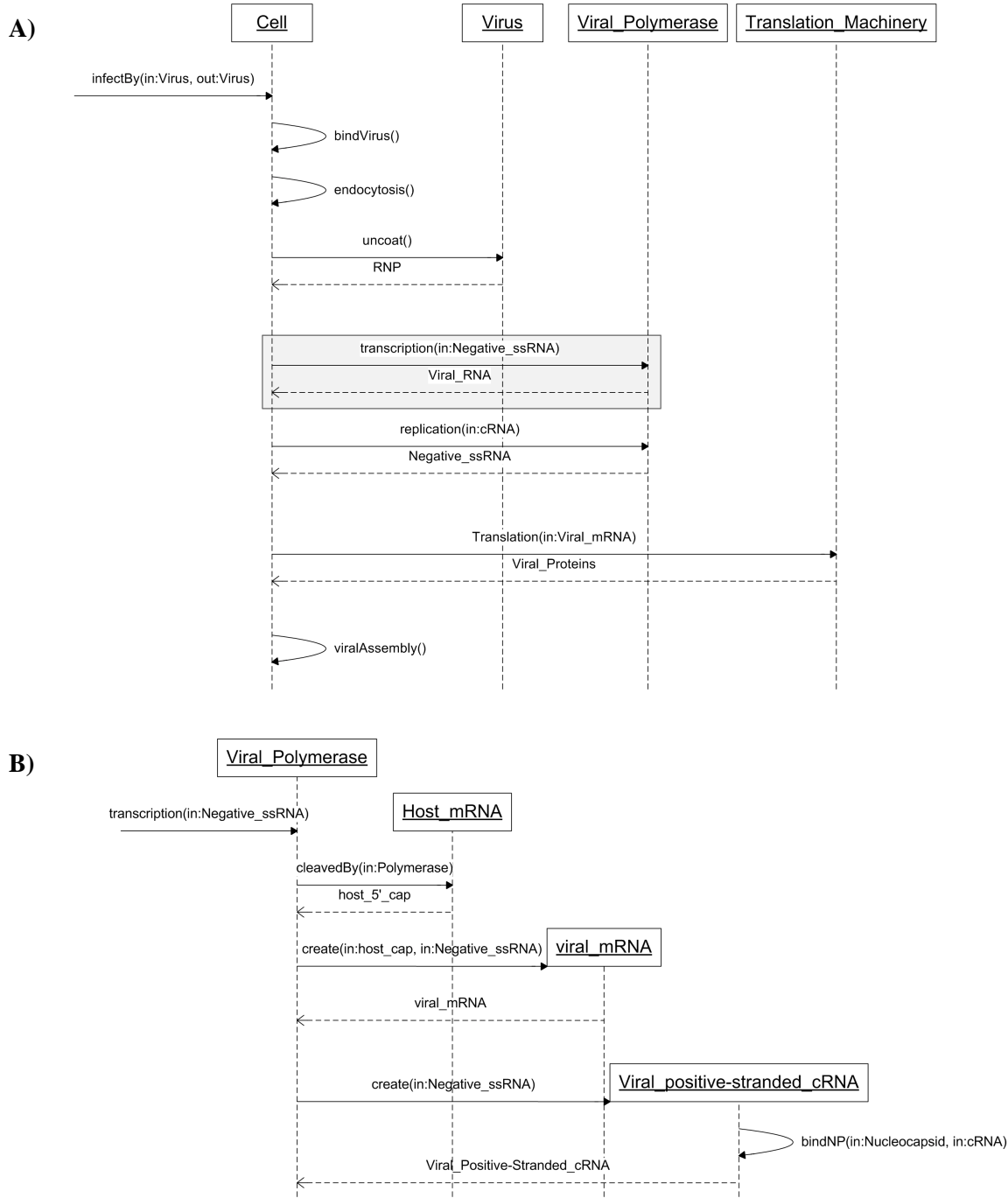


Figure 2. Sequence Diagram for Influenza A. (A) Shows a high-level sequence diagram for the viral infection process. (B) A more detailed view of viral transcription. Bioentities (e.g. cell, virus, translation machinery and viral polymerase) are modeled as objects. These objects are shown at the top of the diagrams with vertical lifelines (|), representing extensions of the objects below. Functions and events associated with these bioentities are modeled as messages sent to the corresponding objects. These message calls, indicated by solid arrows (\rightarrow), are made between objects via connection of their lifelines. Messages contain a message name followed by parameters, which must be passed into the object and/or outputted from the object. Alternatively, an object calling itself (\curvearrowright), may be specified and require no input. Return values, where applicable, are displayed below the message call as a named dotted arrow extending in the opposite direction as the original message call.

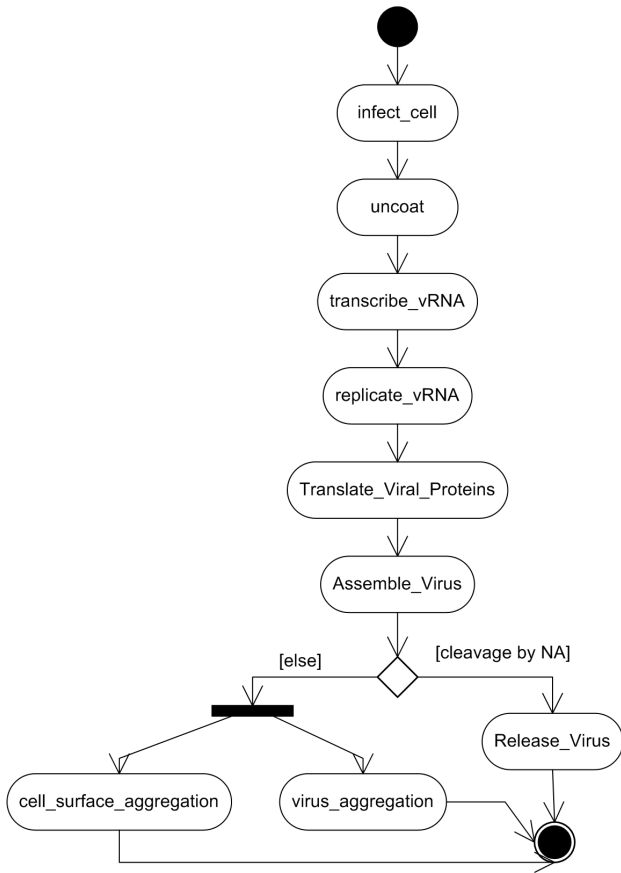


Figure 3. Activity diagram for Influenza A. The infection process consists of several major events; infect cell, uncoat, replicate viral RNA, translate viral proteins, assemble virus and release virus. The events leading to viral particle release are shown in detail. All activity diagrams have a pseudo-start (●) and pseudo-end (●) states. Action states, represented by rounded rectangles (○) are captured during different time periods. The flow of control may also be conditional. For example, decision diamonds (◇) represent a point at which alternative events can occur. At these decision diamonds, guard conditions must be satisfied for the flow of control to continue, or else the flow of control follows an alternative route. NA, neuraminidase; vRNA, viral RNA

State Diagram Generation

State diagrams were created to capture the transitions and different states that a cellular component can exist. In addition, multiple concurrent states can be illustrated using this UML notation.

Sequence and Activity Diagram Generation

Sequence and activity diagrams were used to further capture the dynamic events occurring between influenza A and the host cell into an object-oriented model. To generate these diagrams, objects representing

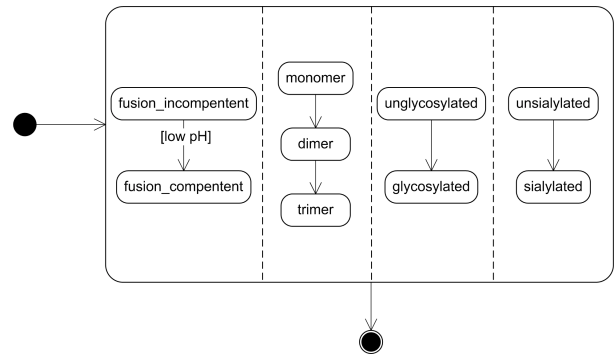


Figure 4. State diagram for Hemagglutinin (HA). The possible state-transitions that the HA protein can undergo are characterized. HA protein starts in an unmodified state. With proper input the protein can undergo a state change from unglycosylated to glycosylated, and monomer to trimer. Composite states may be created that contain concurrent states. These concurrent states occur independently of each other. Vertical dashed lines separate concurrent states. The different states that (○) HA may achieve and the transitions between these states are represented. The state diagram starts (●) and ends (●) with a pseudostate. Arrows connecting states are transitions.

corresponding bioentities were created, and their essential behaviors were captured. Interactions among objects were also identified. For each interaction, a corresponding method was generated. This method specified the behavior for of an object towards its environment. Each sequence diagram captured a sequence of events that occurs during viral infection.

3. RESULTS

Conceptual model generation

During the requirement-gathering phase, we developed a use case that captured the details of each step of viral infection, such as viral binding of cell receptor, viral entry, decoating, viral RNA replication, viral protein generation, and viral assembly and release. Alternative scenarios were also captured in the use case. Based on the use case, use case diagrams were developed.

A conceptual model (Figure 1) was developed to dissect the system and identify components of the system. Simple and complex objects were created from these components and their basic interrelationships were determined. Complex objects may be composed of multiple simple objects. Complex objects may also have different behavior (higher-order behavior) than the individual entities that comprise them. For example, in Figure 1 the ‘virus’ object has a behavior such that it infects a ‘cell’ that is in its presence, whereas no individual component of the virus possesses this behavior.

The conceptual diagram can also illustrate inherited behavior for the modeled objects. For example, proteins might inherit the common behavior 'bind' from the general class of 'proteins'. This behavior, inherited from a common class, may be further refined for a specific purpose. A specific protein that inherits this behavior may act only on proteins that have been phosphorylated. Likewise, a different protein may bind only proteins that have been glycosylated. In this way, multiple objects that have inherited a common behavior may also demonstrate behavior polymorphism. Here, objects although containing a behavior inherited from a common class execute these behaviors in distinct ways. However, the conceptual diagram has limitations. Mainly, the behavior of these components is not captured in a time-dependent manner or event-dependent manner, thus this static behavior can only serve as an intermediate toward defining object behavior.

Behaviors captured using sequence diagrams

To further define the behavior of the objects in the model requires specifying time dependencies. To fully capture the dynamics of the system requires a visualization tool that is capable of describing how behaviors are carried out in response to events, in a prescribed order [9]. In addition, behaviors not dependent on a previous event, such as protein dimerization may also be captured in these diagrams. To specify these behaviors the activities during viral infection were analyzed. These activities were organized and sequence diagrams were developed to capture these activities. The behaviors were decomposed into messages that stipulate how one object interacts with another. Finally, layered structures were applied to organize the viral infection activities into manageable pieces. High-level sequence diagrams (**Figure 2A**) were used to describe major events while lower-level details were captured in additional sequence diagrams (**Figure 2B**). **Figure 2A** shows a high-level diagram for the viral infection process. In this diagram, behaviors captured as messages passed between objects can encapsulate higher-order behaviors. For example, we have expanded the 'transcription' message from **Figure 2A** to show that the transcribe behavior is actually comprised of a series of events as shown in **Figure 2B**. **Figure 2B** is a lower-level sequence diagram that details the actual events and behaviors that occur in the higher-order behavior 'transcription'. Sequence diagrams can thus clarify complex behaviors by defining event-independent and time and event-dependent behaviors found in the system.

Behaviors captured using activity diagrams

Although we have captured the behaviors, which may be exhibited in a best-case scenario, alternative behaviors may arise. Because the sequence diagram cannot fully represent the dynamics of a system the activity diagram is used as a supplement. Activity diagrams were used to capture alternative scenarios for influenza viral infection (**Figure 3**). Conditions that dominate the outcome of

viral infection were captured along with the viral infection process. For each condition, different execution paths were assembled. To simplify the modeling of complex activities, a layered approach was taken. Activities were assembled at a high level, and details were hidden (i.e. encapsulated) in each high-level activity. These encapsulated activities allow behaviors to be achieved without specifying how. Low-level activities can be expanded from the high-level activity diagram. As an example, **Figure 3** shows both the best-case scenario and alternative events that might occur during viral release. Neuraminidase-mediated cleavage of the virus from the host cell surface is required for viral release. If, for instance, neuraminidase is not present in the viral particle or some other interference prevents viral particle cleavage an alternative behavior needs to be designated. The decision diamond in the activity diagram (**Figure 3**) models this contingency. For instance, as indicated by the fork the virus might aggregate with other viruses or the host cell surface. This activity diagram demonstrates that process-level events of different biological system scenarios can be captured as dynamic models and further specifies the influenza A biological system behavior.

Behaviors captured using state transition diagrams

Other dynamic processes during viral infection were modeled as state changes of cellular and viral proteins. State transition is also very important to understand the influenza viral infection process. State changes define the internal behaviors of an object and influence its interaction with other objects. However, many components have multiple state transitions, and the relationships among these state transitions are not clear. We have modeled this transition in a parallel fashion. **Figure 4** shows an example of a state diagram for the hemagglutinin (HA) glycoprotein of influenza A. Since hemagglutinin may exist in multiple states at the same time these characteristics have been captured using concurrency within the state diagram. **Figure 4** captures some of the significant states that HA may progress through. Specifically, we have modeled the transition of HA between its monomerized and trimerized, glycosylated and unglycosylated, and sialylated and unsialylated states. Also, a significant event in the viral life cycle is the conformation change that HA undergoes due to a decrease in pH within the viral particle to allow it to fuse with the host membrane [3]. We have modeled these states as fusion incompetent and fusion competent. A guard condition was placed at the transition between these two states to indicate that the transition only occurs in the presence of a low pH. The state diagram provides a means to capture event-dependent and state behavior. Collectively, the sequence, activity and state diagrams capably captured the dynamics of the viral infection process and defined the behaviors of the influenza A system and its components.

4. CONCLUSIONS

Object-oriented modeling provides a powerful tool to capture the dynamic process in biological systems. By applying object-oriented modeling, we can begin to capture the behaviors of a biological system and its components in a time-dependent and time-independent fashion. However, UML is only one of the approaches to represent an object-oriented system and has certain limitations in capturing these behaviors. As described by Douglass [1], UML is a discrete modeling language that provides no direct means for modeling continuous system behavior. However, object-orientation is not limited to using only one tool and provides many mechanisms to represent a system. For example, an object-oriented language such as C++ or Java can incorporate much more detail to capture continuous system behaviors. Together, object-oriented modeling has great potential for integrating behavioral information associated with genomes, and representing biological knowledge as a system.

5. ACKNOWLEDGEMENTS

Daniel Shegogue is supported by NLM training grant 5-T15-LM007438-02. W. Jim Zheng is partly supported by a grant (DE-FG02-01ER63121) from the Department of Energy and a grant (GC-3609-04-43766CM) from the Department of Defense.

6. REFERENCES

- [1] Douglass, B.P., *Analysis: Defining Object Behavior*, in *Real Time UML: Advances in the UML for Real-Time Systems*, I.J.a.J.R. Grady Booch, Editor. 2004, Addison Wesley Professional: Boston.
- [2] Lakadamyali, M., Rust, M.J., and Zhuang, X., (2004) Endocytosis of influenza viruses, *Microbes Infect*, **6(10)**, 929-36.
- [3] White, J., Matlin, K., and Helenius, A., (1981) Cell fusion by Semliki Forest, influenza, and vesicular stomatitis viruses, *J Cell Biol*, **89(3)**, 674-9.
- [4] Hay, A.J., et al., (1977) Transcription of the influenza virus genome, *Virology*, **83(2)**, 337-55.
- [5] Seong, B.L., et al., (1992) Comparison of two reconstituted systems for in vitro transcription and replication of influenza virus, *J Biochem (Tokyo)*, **111(4)**, 496-9.
- [6] Skorko, R., Summers, D.F., and Galarza, J.M., (1991) Influenza A virus in vitro transcription: roles of NS1 and NP proteins in regulating RNA synthesis, *Virology*, **180(2)**, 668-77.
- [7] Zhirnov, O.P. and Klenk, H.D., (1997) Histones as a target for influenza virus matrix protein M1, *Virology*, **235(2)**, 302-10.

- [8] Palese, P., et al., (1974) Characterization of temperature sensitive influenza virus mutants defective in neuraminidase, *Virology*, **61(2)**, 397-410.
- [9] Rubin, K.S. and Goldberg, A., (1992) Object behavior analysis, *Communications of the ACM*, **35(9)**, 48 - 62.