BIOINFORMATICS ORIGINAL PAPER
Vol. 21 no. 10 2005, pages 2502–2509
doi:10.1093/bioinformatics/bti344

Databases and ontologies

Object-oriented biological system integration: a SARS coronavirus example

Daniel Shegogue[1] and W. Jim Zheng[1,2,*]

[1]Department of Biostatistics, Bioinformatics and Epidemiology and [2]Bioinformatics Core Facility,
Hollings Cancer Center, Medical University of South Carolina, 135 Cannon Street, PO Box 250835,
Charleston, SC 29425, USA

Received on September 15, 2004; revised on January 20, 2005; accepted on February 17, 2005
Advance Access publication February 24, 2005

ABSTRACT

Motivation: The importance of studying biology at the system level has been well recognized, yet there is no well-defined process or consistent methodology to integrate and represent biological information at this level. To overcome this hurdle, a blending of disciplines such as computer science and biology is necessary.

Results: By applying an adapted, sequential software engineering process, a complex biological system (severe acquired respiratory syndrome-coronavirus viral infection) has been reverse-engineered and represented as an object-oriented software system. The scalability of this object-oriented software engineering approach indicates that we can apply this technology for the integration of large complex biological systems.

Availability: A navigable web-based version of the system is freely available at http://people.musc.edu/∼zhengw/SARS/Software-Process.htm

Contact: zhengw@musc.edu

Supplementary information: Supplemental data: Table 1 and Figures 1–16.

INTRODUCTION

A paradigm shift is occurring with the study of individual cellular components progressing toward the study of the cell at the system level (Ideker *et al*., 2001). However, no well-defined process or effective integration methodology, in combination with a widely accepted system representation has been developed to provide a comprehensive view of a biological system. Biological systems are currently presented in a variety of forms, such as relational databases (Galperin, 2004), diagrams (Kohn, 1999; Peleg *et al*., 2002), ontology (Ashburner *et al*., 2000) and markup language (Hucka *et al*., 2003), but they are integrated through an ill-defined, informal process. This assortment of information integration methods can hamper biological studies, especially of large complex systems. A well-defined process and integration technology can orchestrate system-level integration efforts by biologists, computer scientists and engineers, and provide consistency and manageability during integration.

Current advanced engineering systems and biological systems have followed a convergent evolution (Csete and Doyle, 2002). This resemblance is particularly evident in a comparison between object-oriented software and typical biological systems (Table 1 and Supplemental Table 1). Both systems are based on components and protocols, and their layered control mechanisms allow them to efficiently handle complexity. This complexity is further managed by subsystems, both in biological and object-oriented software systems. Furthermore, these systems contain static time-independent and dynamic time-dependent relationships. This resemblance has led to a very similar architecture and system-level organization, making it possible to represent a biological system as an object-oriented software system.

To effectively construct a high-quality object-oriented software system, three components are necessary. First, a well-defined software engineering process, such as the waterfall model, Rational Unified Process, agile process, etc. (Graham, 2001), is needed to specify stepwise activities and define what should be accomplished during software development. Second, object-oriented technology, a paradigm of software development, including object-oriented programming, analysis, design and database centered on the object concept, is vital for object-oriented software development. This technology is used to dissect real-world problems and create an object-oriented software system to solve these problems. Third, a widely accepted object-oriented modeling language, such as the Unified Modeling Language (UML) (Rumbaugh *et al*., 1999), should be used to represent this object-oriented system. This visual representation can help to organize and communicate a complex software system as small, manageable pieces without losing system integrity. While these components were specifically designed for successful software development, they can be adapted for biological system information integration.

The similarity between biological and object-oriented software systems at the system organization level indicates that we can represent one by another. Therefore, the integration of biological information at the system level can be approached by reverse engineering the biological system into an object-oriented software architecture. Following a well-defined software development process to reverse engineer a biological system offers significant advantages. Although simple systems can be easily modeled by an indiscriminate approach, a formal approach is necessary for the creation of complex system models, such as those found in biology. Moreover, when this methodology is applied to biological system modeling, the distinct phases of software development allow tasks best suited for

[*]To whom correspondence should be addressed.

Table 1. Correspondence between biological system and object-oriented software concepts

Category	Concepts in object-oriented software system (Graham, 2001)	Concepts in biological system	Commonalities
Basic component	Objects	Cellular components (proteins, genes, etc.)	Both can be modeled as having intrinsic attributes and external functionalities or behavior
Component organization	Aggregation, composition	Protein complex formation/protein–protein interaction	Relationships among individual components
High-order components	Subsystems (also in the form of objects)	Organelles, individual pathways, cell, etc.	Both are complex modules built from basic components
Protocol for component interactions	Message passing	Interaction, signaling, modification	Information passing in both situations
Hierarchical relationship	Inheritance	is-a relationship (most enzymes are one kind of a protein)	Specialized components share common features
Multifunctionality	Polymorphism	Enzymes catalyze different substrates, etc.	One component can operate on different types of other components of the system with different processes and outcomes
Observed behavior without knowing underlining mechanism	Encapsulation/information hiding[a]	Experimental observations without known mechanism	Both situations define the observed behavior without exposing the underlying mechanism
Controls	Call-back functions, observer/visitor patterns (Gamma *et al.*, 1997)	Feedback controls	One part of the system originates a signal that affects the behavior of the other components of the system and this interaction in return affects the behavior of the subsystem where the signal originated
Component dynamics	Life cycle of individual objects	Life cycle of individual proteins	The birth and death of individual components

[a] See also Discussion section.

The concepts of biological system and object-oriented software systems are compared and the commonalities among these concepts are listed. We only used the cell system to represent a typical biological system in this table. This comparison focuses on the system organization and functionality. We provide correlations between the main object-oriented concepts (i.e. object, composition, inheritance, polymorphism and encapsulation). However, biological and object-oriented system concepts are not necessarily a one-to-one mapping. Modeling of some of the biological system components and a further explanation of object-oriented concepts can be found in the main text.

biologists or computer scientists to be divided and coordinated. In addition, applying a well-defined software engineering process and object-oriented methodology provide an effective means to capture specifications from experimental data and integrate the biological system information. Finally, this process provides a guideline for the development of an integrated biological system, represented as an object-oriented software architecture, in a widely accepted object-oriented modeling language (such as UML), which can facilitate communication about complex systems among software engineers, biologists and other users. Recently, biologists have also begun to develop and specify processes for biological data integration, such as the one used for the TRANSPATH database (Schacherer *et al.*, 2001), but this activity has been fairly limited and has not gained the community-wide acceptance comparable with popular software engineering processes.

To demonstrate the efficacy of a well-defined software engineering process in the translation of a biological system to a model grounded in object-oriented principles, we used UML in the development of a severe acquired respiratory syndrome-coronavirus (SARS-CoV) model. SARS-CoV, which killed nearly 800 people and infected 8000 worldwide, consists of 28 structural and non-structural proteins. Although the viral infection and replication process is not entirely resolved, some details have emerged. Briefly, the virus binds to the cellular ACE2 receptor via its spike (S) protein, then fuses with the host cell membrane (Li *et al.*, 2003). Genomic RNA is released, and gene one, encoding the non-structural replicase proteins is translated. The replicase proteins are cleaved by the papain-like proteinase (Rota *et al.*, 2003) and 3C-like protease (Fan *et al.*, 2004) also contained within gene one. The replicase produces the structural protein mRNAs from the remaining genome, which are subsequently translated. The structural proteins (Ying *et al.*, 2004) consist of the membrane (M) (Rottier *et al.*, 1986) and envelope (E) (Yu *et al.*, 1994) proteins, which form the viral envelope, the S protein that binds the cellular receptor (Xiao *et al.*, 2003) and nucleocapsid (N) protein, which associates with the viral RNA to form the ribonucleoprotein (Ying *et al.*, 2004). The recently identified transmembrane proteins U274 and U122 may also serve as structural proteins (Fielding *et al.*, 2004; Tan *et al.*, 2004). The virus assembles in an intermediate compartment between the Golgi and endoplasmic reticulum (Lai and Cavanagh, 1997), and is released via the constitutive exocytic pathway (Lai and Cavanagh, 1997). In this paper, we demonstrate that a complex biological system can be reverse-engineered, and the resulting software representation can capture the static and dynamic relationships contained with the system.

MATERIALS AND METHODS

Requirement-gathering phase

Information collection To define the requirements and collect the information necessary for model generation, two approaches were necessary. First, annotations of the SARS viral gene products were obtained through an automated pipeline, GeneAtlas, implemented by Accelrys, Inc. (Kitson *et al*., 2002). This pipeline can efficiently identify the function of a gene or a protein through homology searches and structural analysis (Yan *et al*., 2003). Second, an extensive literature review was manually conducted. Detailed information about the SARS-CoV and viral infection were collected. Because of the relatively recent appearance of SARS-CoV, when information was not available, functions and relationships were extrapolated from homologous genes in other viruses.

CRC card generation The attributes and the interactions of the viral proteins were captured using CRC cards as described by Shegogue and Zheng (2004) (for an example of a CRC card, see Supplemental Figure 1).

Use case development Based on the information gathered, a use case was developed to reflect the viral infection process (see Supplemental Figure 2). The use case also serves to define the boundary and scope of the SARS-CoV model. Therefore, although the infection process may involve other elements such as the immune system, this study was limited, as defined in the use case, to those events that directly involve the interaction between the virus and the cell.

Analysis phase

Use case diagram generation To define the high-level interaction between the SARS-CoV and the cell, a UML use case diagram was generated (see Supplemental Figure 3). This diagram was based on the boundaries defined in the use case. This and other UML diagrams in this study are generated using Microsoft Visio Pro. For a review on UML models see http://bdn.borland.com/article/0,1410,31863,00.html

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 (see Supplemental Figure 4). By applying object-oriented analysis, the SARS virus and the cellular components involved were decomposed into objects and the component relationships were realized. However, information regarding component properties is hidden. This intermediate artifact defines the organization of the biological system and provides an overview of the components and their relationships.

System sequence diagram generation To capture the sequence of events occurring between the SARS-CoV and cell at the system-level, a system sequence diagram was created (see Supplemental Figure 5).

Design phase

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. An example of a state diagram for the M protein is shown in Supplemental Figure 6. Additional state diagrams for the S, E and N proteins can be found at http://people.musc.edu/∼zhengw/SARS/State%20Diagrams/web%20page/Statechart_index.htm

Sequence, activity and class diagram generation Sequence, activity and class diagrams have been used as an example to demonstrate the feasibility of generating an object-oriented representation of the SARS-CoV and cell system. To generate these diagrams, objects representing corresponding bioentities are created and their essential attributes are captured. Interactions among objects are also identified. For each interaction, a corresponding method is generated. The nature of the interaction determines the method parameters. To generate sequence diagrams the sequence of events is captured. Scenarios are also generated for object interactions and used to generate activity diagrams. The information captured in the sequence diagram and activity diagrams are used, along with the bioentities attributes, to generate class diagrams.

Model validation Models are validated by referring back to the CRC cards and use case to determine if the requirements set forth in the requirement-gathering phase have been met.

Web-based implementation These diagrams, along with CRC cards for objects in the system, were incorporated into a web-based system to deliver our object-oriented representation to the users (http://people.musc.edu/∼zhengw/SARS/Software-Process.htm).

RESULTS

Software engineering process

To capture the static and dynamic relationships that occur between SARS-CoV and a cell, models were constructed by following an adapted, linear, sequential software process containing distinct phases of requirement gathering, analysis and design (Fig. 1). This approach is suitable because of the relative simplicity of the viral infection process. Since the major activities (such as requirement gathering, analysis and design) of this sequential software process serve as the foundation for other advanced software engineering processes, these advanced processes can be applied to modeling more complex biological systems. The requirement-gathering phase entails collecting all available information about the system, generating CRC cards (see Supplemental Figure 1) to capture biological entity (bioentity) functions and collaborators, and defining a use case (see Supplemental Figure 2). The analysis phase is used to dissect the interrelationships among components involved in viral infection and serves as a basis for the detailed models generated in the design phase. Here, the gathered information is used to create a use case diagram (see Supplemental Figure 3), conceptual model (see Supplemental Figure 4) and system sequence diagram (see Supplemental Figure 5). In the design phase, information from the analysis and requirement-gathering stages is incorporated to develop a detailed model, which represents the static (class diagram, see Supplemental Figures 14–16 and Figure 4) and dynamic (for state diagram see Supplemental Figure 6, for sequence diagram see Supplemental Figures 7–9 and Figure 2, and for activity diagrams see Supplemental Figures 10–13 and Figure 3) nature of the SARS-CoV genome and infection process. Together, these diagrams provide a comprehensive software representation of the biological system.

Sequence diagram generation

The major events of SARS-CoV infection include virus binding, membrane fusion, viral RNA replication, structural proteins translation, viral assembly and release. Sequence diagrams were created to capture the dynamic nature of these processes as an object-oriented software system. A high-level representation of the SARS viral infection is reflected chronologically in Figure 2. Simple (such as proteins) or complex (such as viral replicase or virus) bioentities, identified by literature searches, are modeled as objects, which are represented by rectangles with vertical lifelines. Object functions, which are implemented by the methods

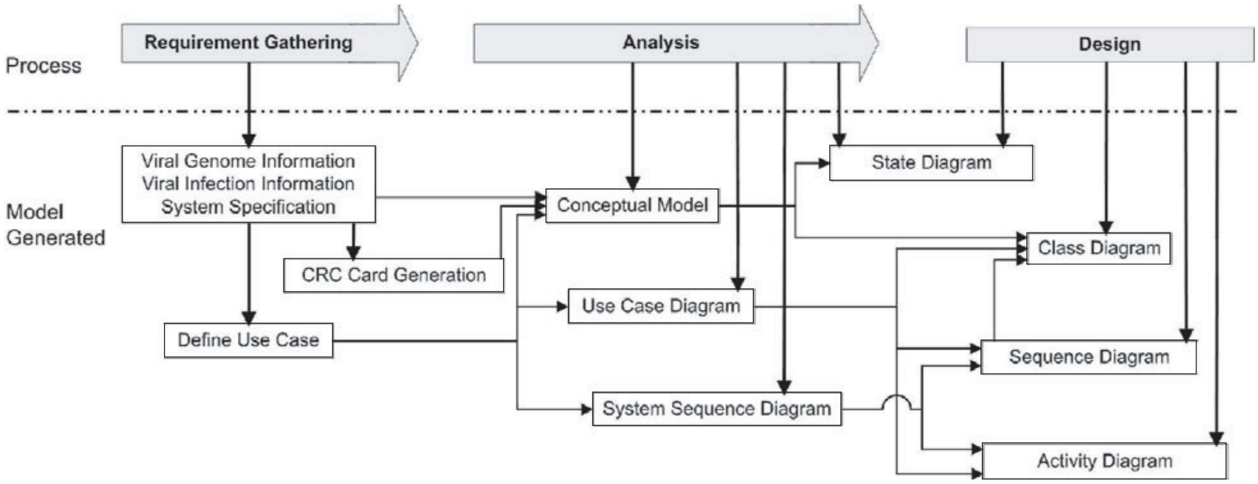

Fig. 1. A linear, sequential software process used for reverse engineering the SARS viral infection. Processes (at the top) are ordered in a time-dependent manner. Vertical lines extending from the processes indicate the process stage in which a model (at the bottom) is generated, and do not imply a time dependence. Major dependences among models are indicated.

contained within the objects, are used to represent the functions of these bioentities. For example, a viral replicase can take a viral RNA as a template and generates a new viral RNA through replication. Similarly, the Viral_Replicase object function, replicate(), takes a viral RNA as a template and generates new Viral_RNA.

To capture interactions between objects, one object can call a method of another object by connecting object lifelines in the sequence diagram (Fig. 2). This invocation of an object function by another is described as one object sending a message to another object. In this way, real-world processes may be captured using an object-oriented approach. For instance, the cell object can call the translation machinery to translate the viral replicase by sending a message translate(in: Positive_Genomic_RNA), where the value after the 'in' is an object, Positive_Genomic_RNA, that must first be passed into the method. To complete the process, the translation machinery will use this Positive_Genomic_RNA as a template to translate and create a 'viral replicase' object, which is sent back to the cell. The cell now contains a copy of the viral replicase for later use. Together, this message passing captures the cellular process of translation of a viral RNA into a viral replicase.

Alternatively, a message need not be passed between objects. A self-call may include self-checks or autoactivation signals. In the case of the assembleVirus() function call, the cell requires two components, structural protein and viral RNA, to generate the virus. However, a self-call might only indicate whether an event has occurred and not accept any parameters. For example, the bindVirus() method only indicates whether a virus has bound the cell. Together, the events leading to the creation of a virus can be chronologically ordered and can mimic the viral infection process. Additional details that reflect the RNA replication process, translation of the structural proteins and viral assembly are available in Supplemental Figures 7–9. Collectively, these diagrams demonstrate that the activities of a biological system can be represented as part of the dynamic processes of an object-oriented software system.

Activity diagram generation

To capture additional features of the dynamic architecture, activity diagrams were created to reflect the SARS-CoV and viral infection process. Unlike the sequence diagram, which captures best-case scenario events, the action sequence or flow of the activity diagram can portray alternative outcomes. For instance, as shown in Figure 3, a virus may bind a receptor if one is available, or else this action ends and the flow is directed to the final state. Once bound, if the S protein undergoes a conformational change that exposes the fusion peptide, then the virus can fuse with the cellular membrane. Alternatively, the virus may detach from the cell, or if a neighboring cell is available cell–cell fusion may be initiated or the flow may be directed to the final state, indicating that infection failed. If the virus is able to fuse with the membrane, release of the N protein will decoat the positive genomic RNA, or else this action will also end. Once decoated, the viral RNA is replicated after which the virus is assembled and released. These subactivities are presented in greater detail in Supplemental Figures 11–13. In addition, Supplemental Figure 10 illustrates a high-level view reflecting only the major stages of the viral infection process. Overall, these activity diagrams define the main success and alternative scenarios that may occur during viral infection.

Class diagram generation

Complex bioentities and their relationships are contained within the biological system encompassing the cell and SARS-CoV. These relationships include SARS-CoV binding to the cell receptor, viral protein interacting with the translation machinery and RNA interacting with the viral replicase. To capture this static architecture, class diagrams were generated that model the components, operations and interrelationships that occur between the SARS-CoV and cell. Specifically, Figure 4, a high-level class diagram, captures the major components of the SARS virus and viral infection as objects and their associations using an object-oriented representation. These objects, where applicable, were given attributes that describe important characteristics, or if changed, might alter the function of a

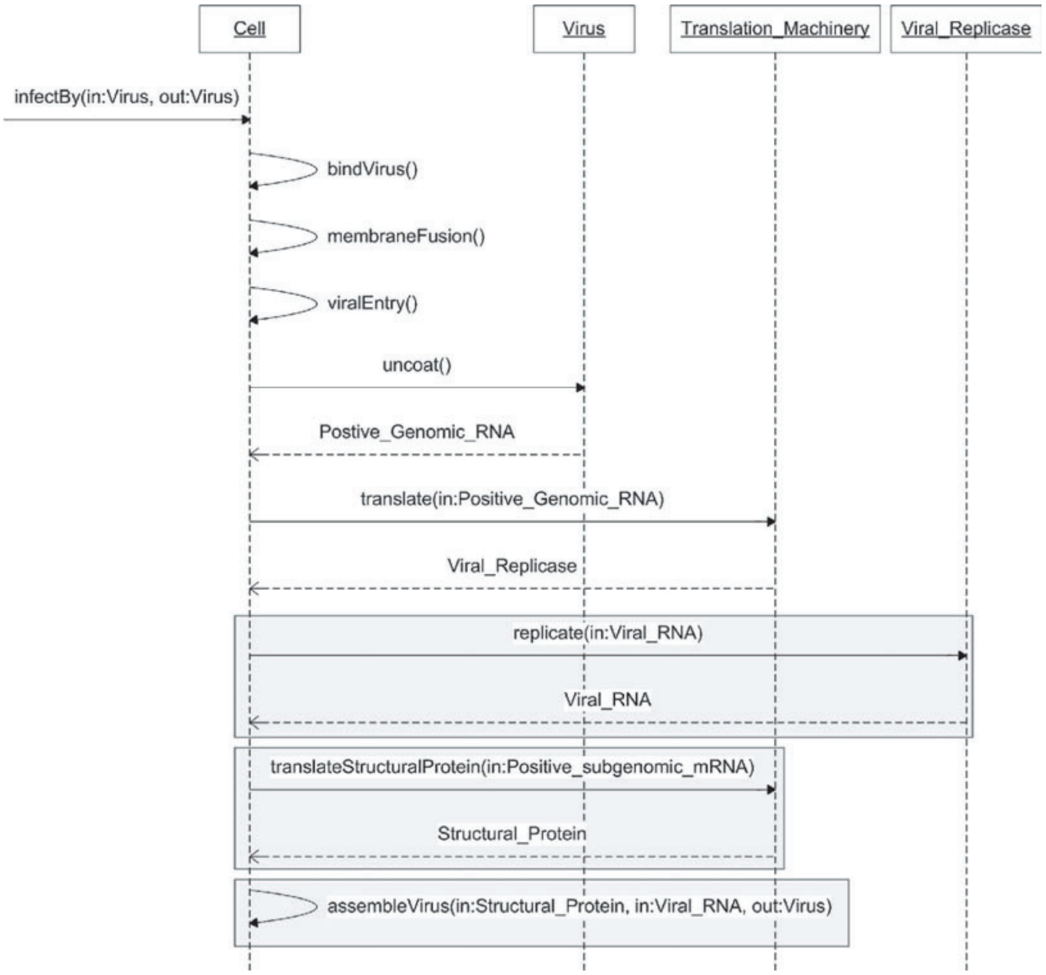
Cell
Virus
Translation_Machinery
Viral_Replicase
infectBy(in:Virus, out:Virus)
bindVirus()
membraneFusion()
viralEntry()
uncoat()
Postive_Genomic_RNA
translate(in:Positive_Genomic_RNA)
Viral_Replicase
replicate(in:Viral_RNA)
Viral_RNA
translateStructuralProtein(in:Positive_subgenomic_mRNA)
Structural_Protein
assembleVirus(in:Structural_Protein, in:Viral_RNA, out:Virus)

Fig. 2. A high-level sequence diagram for major SARS viral infection events. Bioentities (cell, virus, translation machinery and viral replicase) 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, 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, e.g. translation machinery executes its function of translating viral mRNA by receiving a translate message 'translate (in: Positive_Genomic_RNA)' from the cell. This message accepts a Positive_Genomic_RNA as a template, indicated by 'in: Positive_Genomic_RNA', then a 'Viral_Replicase' is created and returned. These return values are displayed below the message call as a named dotted arrow extending in the opposite direction as the original message call. Where return values are not explicitly indicated, as in an object calling itself (⊃), an 'out' followed by a return value is substituted. Events in gray boxes are shown in more detail in Supplemental Figures 7–9.

component. The object functions, which parallel bioentity functions, were generated from the sequence diagrams. These functions or operations are a declaration of the methods that an object may use. In addition, protein complexes can be represented, as in the case of the viral replicase, as complex objects, which are composed of many other objects (see Supplemental Figure 15). Similarly, utilizing composition allows the virus object to mimic a real-world virus, which contains structural proteins, positive genomic RNA and a membrane (Fig. 4). Larger systems such as organelles and pathways can also be contained within the object-oriented software system in the form of modules, subsystems or packages. In this way, class diagrams allow a biological system to be captured as a layered system, e.g. the cellular translation machinery may be represented as a single object, even though it has a very complex mechanism and regulation. This complexity could also be captured in a subsystem that specifically focuses on the cellular translation machinery. This layered structure indicates that the object-oriented technology can be used to represent the layered structure of complex biological systems.

In addition, the UML notation provides a mechanism to specify inheritance, which may be used to indicate an object that is the foundation for other objects. For instance, structural protein and Gene1 viral protein precursor inherit the properties of the viral protein object (Fig. 4). Binary associations are also used to capture associations between objects. These associations may contain cardinalities, which indicate how many of one object interacts with another. For instance, a virus can infect one cell and a cell can be infected by one to many viruses (Fig. 4). Additional class diagrams showing a higher level of detail and focusing on the interactions and components of the

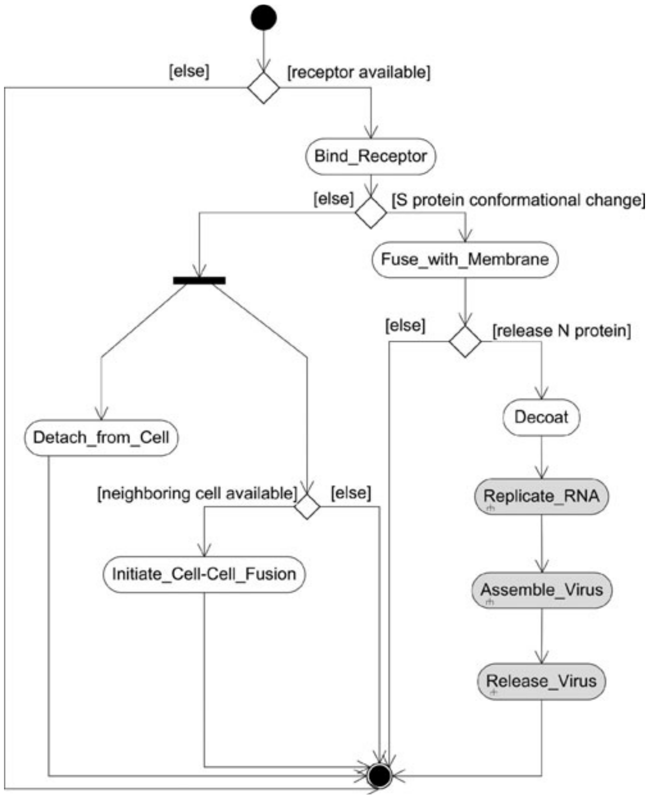

Fig. 3. Activity Diagram for SARS viral infection emphasizing the infection process. The infection process consists of three major events; bind cellular ACE2 receptor, fuse with the cell membrane and release its genomic RNA by uncoating. The process of RNA replication, assembly and viral release are detailed in Supplemental Figures 10–13. All activity diagrams have a pseudo-start (●) and pseudo-end (◉) states. Action states, represented by rounded rectangles (⬭) are captured during different time periods. This diagram can represent sequential as well as concurrent events. Concurrent events are represented by forks (⊥) and merges (⊤). This architecture signifies that multiple events are possible simultaneously and the occurrence of at least one of these events is enough to continue the flow of control. In contrast, the flow of control may also be conditional, e.g. 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.

structural proteins, replicase complex and viral replication process are available in Supplemental Figures 14–16. These class diagrams demonstrate that the static structure of a biological system can be represented as an object-oriented model. Together, the models generated using the described object-oriented methodology yield a software system representation of a biological system, capturing both static and dynamic relationships.

DISCUSSION

Here, we have demonstrated that a complex biological system can be modeled following a systematic, well-defined software engineering process. We have leveraged the power of the UML to construct our object-oriented software representation of the SARS-CoV and its infection process. Even though UML has been proposed as a tool to model biological systems (Bornberg-Bauer and Paton, 2002; Roux-Rouquie *et al*., 2004), and an overview of the modeling process used in the development of biological simulation software is available (Johnson *et al*., 2004), without a well-defined methodology and software-engineering process, such as object-oriented analysis and design, transforming biological information into UML becomes an *ad hoc* activity. However, by applying object-oriented analysis and design, biological entities and related information can be systematically transformed into an object-oriented system by this well-defined methodology. Furthermore, UML is created to represent object-oriented software systems. If a biological system itself is directly represented by UML, but not as an object-oriented software system, this representation may be confusing and the full capabilities of UML may not be achieved. Reverse engineering a biological system into an object-oriented system ensures that the object-oriented system can be accurately and completely represented by UML. Additional tools such as class-responsibility-collaboration cards (Wilkinson, 1998), which can be used to capture the attributes, collaborators and functions of the annotated object, may also be employed to capture data not fully represented by the UML notation.

One major problem for modeling and integrating biological information is the lack of information: experimental observations without known underlining mechanisms. This challenge can be met by applying object encapsulation: defining the behavior of an object through its operations that are accessible to other objects, but hiding the underlining mechanism of the operations and object attributes from other objects (Graham, 2001). For example, to communicate that the translation machinery translates a viral protein does not require that every component of the translation machinery be detailed. Specifically, to capture this information we gave the translation machinery a function called ‘translate’. This function will create a viral protein from the viral mRNA without defining the real process of how the viral protein is created. This approach can encapsulate the unknown cellular process inside a well-known function or behavior, and model the biological system without understanding every detail of the system.

Object-oriented models have additional applications in the construction of biological systems. The layered structure of the object-oriented model allows one to capture various levels of detail such as population, organism, organ, pathway or cell system. This layering can be used to create compartments, which define spatial relationships between cellular components. Webb and White (2003) have recently modeled these compartments as classes. However, the compartment issue may be addressed using packages or subsystems. The flexibility of the object-oriented model and UML tool also allow an assortment of biological information to be captured and subsequently translated into mathematical models. It has been shown by Priami *et al*. (2001) and Webb and White (2003) that rate equations can be generated using UML sequence and state diagrams. Together, an object-oriented approach is capable of modeling enzymatic reactions and metabolic pathways (Freier *et al*., 2003). However, challenges still exist for fully representing a biological system as an object-oriented system.

Significantly, because the breadth and depth of biological systems make a complete description of biological systems intractable, a collaborative effort is essential. To integrate new information into existing models, a system, such as a web portal, will have to be implemented that also allows experts to contribute to model maturation. The Alliance for Cellular Signaling

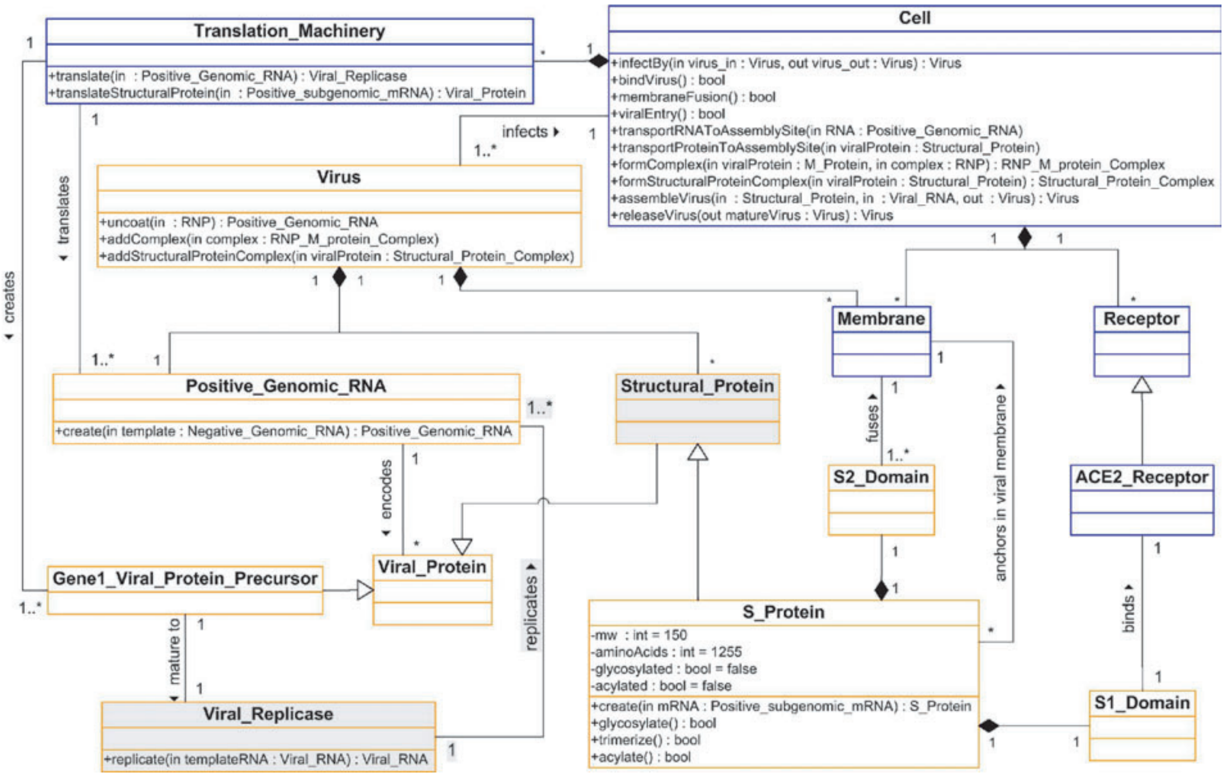
Translation_Machinery
+translate(in : Positive_Genomic_RNA) : Viral_Replicase
+translateStructuralProtein(in : Positive_subgenomic_mRNA) : Viral_Protein
Cell
+infectBy(in virus_in : Virus, out virus_out : Virus) : Virus
+bindVirus() : bool
+membraneFusion() : bool
+viralEntry() : bool
+transportRNAToAssemblySite(in RNA : Positive_Genomic_RNA)
+transportProteinToAssemblySite(in viralProtein : Structural_Protein)
+formComplex(in viralProtein : M_Protein, in complex : RNP) : RNP_M_protein_Complex
+formStructuralProteinComplex(in viralProtein : Structural_Protein) : Structural_Protein_Complex
+assembleVirus(in : Structural_Protein, in : Viral_RNA, out : Virus) : Virus
+releaseVirus(out matureVirus : Virus) : Virus
infects ▸
1..*
Virus
+uncoat(in : RNP) : Positive_Genomic_RNA
+addComplex(in complex : RNP_M_protein_Complex)
+addStructuralProteinComplex(in viralProtein : Structural_Protein_Complex)
translates
creates
Positive_Genomic_RNA
+create(in template : Negative_Genomic_RNA) : Positive_Genomic_RNA
Structural_Protein
Membrane
Receptor
fuses
S2_Domain
anchors in viral membrane
ACE2_Receptor
encodes
Viral_Protein
Gene1_Viral_Protein_Precursor
replicates
mature to
Viral_Replicase
+replicate(in templateRNA : Viral_RNA) : Viral_RNA
S_Protein
-mw : int = 150
-aminoAcids : int = 1255
-glycosylated : bool = false
-acylated : bool = false
+create(in mRNA : Positive_subgenomic_mRNA) : S_Protein
+glycosylate() : bool
+trimerize() : bool
+acylate() : bool
binds
S1_Domain

Fig. 4. High-level class diagram of SARS virus and viral infection. Major components of SARS virus and viral infection and their associations are modeled as objects. These objects are represented by rectangles within the diagram. Rectangles are divided into three parts. The first part contains the object name; the second part contains the attributes of the object; and the final part contains the functions an object can perform. Lines with solid diamonds (◆—) at the end indicate composition. These are read from the diamond end, e.g. as Virus contains Structural_Proteins. Lines with open triangles (◁—) represent generalizations. These are read from the triangle end, as Viral_Protein is a general type of Structural_Protein. Other interactions are represented as named, binary associations (⌐⎿). Interactions may contain cardinalities at their ends indicating the number of objects that 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. Grayed objects and interactions are shown in more detail in Supplemental Figures 14–16. Viral components (orange); cellular components (blue).

(http://www.signaling-gateway.org/) has already set a precedence for such a collaboration. This approach is especially suitable for object-oriented system development, since the modular nature of an object-oriented system allows the modification of individual components without affecting other parts of the system. Together, these approaches will distribute the burden of creating these models. Because portions of a model might be derived from information that contains varying degrees of confidence, in the future, UML models might benefit from the addition of a confidence indicator to indicate whether portions of a model are derived from experimental data or are based on hypothesis and speculation. While this study has focused on the construction of a web-based system for navigation of the layered structure of a biological system, future work will also strive to integrate information into a searchable database, and implement software systems to simulate biological processes. Finally, to extend this application to a more complex biological system, it will be necessary to adopt an advanced iterative process, such as the Rational Unified Process (Kruchten, 2003), or apply advanced techniques, such as Façade pattern (Gamma *et al*., 1997), in object-oriented design to reduce the complexity of the system. This will allow the development of more sophisticated methods to reverse engineer biological systems and significantly enhance the study of biological complexity at a system level.

We demonstrate that by applying an adapted, sequential software engineering process, a complex biological system (SARS viral infection) can be reverse-engineered and represented as an object-oriented software system. Similar to software engineering projects, the well-defined software process makes the biological system information integration repeatable, controllable, scalable and manageable. The resulting object-oriented system not only captures information about individual components but also displays the system-level architecture and interrelationships among system components. Furthermore, this object-oriented model captures system dynamics, in that the key events and the process of viral infection are also described. Our research demonstrates that object-oriented analysis and design can be employed as an effective methodology to integrate biological information, and the UML representation is a comprehensive approach to capturing the complexity of biological systems, integrating biological information and facilitating communication between biologists and computer scientists.

ACKNOWLEDGEMENTS

We thank E. Voit for critical reading of this manuscript. D.S. is supported by NLM training grant 5-T15-LM007438-02. W.J.Z is partly supported by a grant (DE-FG02-01ER121) from the Department of Energy.

REFERENCES

Ashburner,M. *et al*. (2000) Gene Ontology: tool for the unification of biology. The Gene Ontology Consortium. *Nat. Genet.*, **25**, 25–29.

Bornberg-Bauer,E. and Paton,N.W. (2002) Conceptual data modelling for bioinformatics. *Brief Bioinformatics*, **3**, 166–180.

Csete,M.E. and Doyle,J.C. (2002) Reverse engineering of biological complexity. *Science*, **295**, 1664–1669.

Fan,K. *et al*. (2004) Biosynthesis, purification, and substrate specificity of severe acute respiratory syndrome coronavirus 3C-like proteinase. *J. Biol. Chem.*, **279**, 1637–1642.

Fielding,B.C. *et al*. (2004) Characterization of a unique group-specific protein (U122) of the severe acute respiratory syndrome coronavirus. *J. Virol.*, **78**, 7311–7318.

Freier,A. *et al*. (2003) iUDB: an object-oriented system for modelling, integration and analysis of gene controlled metabolic networks. *In Silico Biol.*, **3**, 215–227.

Galperin,M.Y. (2004) The Molecular Biology Database Collection: 2004 update, *Nucleic Acids Res.*, **32** (Database issue), D3–D22.

Gamma,E., Helm,R., Johnson,R. and Vlissides,J. (1997) *Design Patterns*. Addison-Wesley Publishing Company, Reading, MA, pp. 185–193, 331–349.

Graham,I. (2001) Basic concepts. In *Object-oriented Methods, Principles & Practice*. Addison-Wesley, Harlow, UK, pp. 1–37, 461–494.

Hucka,M. *et al*. (2003) The Systems Biology Markup Language (SBML): a medium for representation and exchange of biochemical network models. *Bioinformatics*, **19**, 524–531.

Ideker,T. *et al*. (2001) A new approach to decoding life: systems biology. *Ann. Rev. Genom. Hum. Genet.*, **2**, 343–372.

Johnson,C.G. *et al*. (2004) Simulating complex intracellular processes using object-oriented computational modeling. *Prog. Biophys. Mol. Biol.*, **86**, 379–406.

Kitson,D.H. *et al*. (2002) Functional annotation of proteomic sequences based on consensus of sequence and structural analysis. *Brief Bioinformatics*, **3**, 32–44.

Kohn,K.W. (1999) Molecular interaction map of the mammalian cell cycle control and DNA repair systems. *Mol. Biol. Cell*, **10**, 2703–2734.

Kruchten,P. (2003) *The Rational Unified Process: An Introduction*, 3rd edn. In Grady Booch,I.J. and Rumbaugh,J. (eds). Addison-Wesley Harlow, UK, 336 p.

Lai,M. and Cavanagh,D. (1997) The molecular biology of coronaviruses. *Adv. Virus Res.*, **48**, 1–100.

Li,W. *et al*. (2003) Angiotensin-converting enzyme 2 is a functional receptor for the SARS coronavirus. *Nature*, **426**, 450–454.

Peleg,M. *et al*. (2002) Modelling biological processes using workflow and Petri Net models. *Bioinformatics*, **18**, 825–837.

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

Rota,P.A. *et al*. (2003) Characterization of a novel coronavirus associated with severe acute respiratory syndrome. *Science*, **300**, 1394–1399.

Rottier,P. *et al*. (1986) Predicted membrane topology of the coronavirus protein E1. *Biochemistry*, **25**, 1335–1339.

Roux-Rouquie,M. *et al*. (2004) Using the unified modelling language (UML) to guide the systemic description of biological processes and systems. *BioSystems*, **75**, 3–14.

Rumbaugh,J. *et al*. (1999) *The Unified Modeling Language Reference Manual*. Addison-Wesley, Reading, MA, chapter xvii, 550 p.

Schacherer,F. *et al*. (2001) The TRANSPATH signal transduction database: a knowledge base on signal transduction networks. *Bioinformatics*, **17**, 1053–1057.

Shegogue,D. and Zheng,W.J. (2004) Capturing biological information with class-responsibility-collaboration cards. *Bioinformatics*, doi:10.1093/bioinformatics/bti005.

Tan,Y.J. *et al*. (2004) A novel severe acute respiratory syndrome coronavirus protein, U274, is transported to the cell surface and undergoes endocytosis. *J. Virol.*, **78**, 6723–6734.

Webb,K. and White,T. (2003) UML as a cell and biochemistry modeling language. *Technical Report CUCSTR 2003-05*, Carleton University Cognitive Science.

Wilkinson,N.M. (1998) *Using CRC Cards: An Informal Approach to Object-Oriented Development.* 2nd edn. Cambridge University Press, Cambridge, UK, 252 p.

Xiao,X. *et al*. (2003) The SARS-CoV S glycoprotein: expression and functional characterization. *Biochem. Biophys. Res. Commun.*, **312**, 1159–1164.

Yan,L. *et al*. (2003) Assessment of putative protein targets derived from the SARS genome. *FEBS Lett.*, **554**, 257–263.

Ying,W. *et al*. (2004) Proteomic analysis on structural proteins of severe acute respiratory syndrome coronavirus. *Proteomics*, **4**, 492–504.

Yu,X. *et al*. (1994) Mouse hepatitis virus gene 5b protein is a new virion envelope protein. *Virology*, **202**, 1018–1023.