PERSONALIZED ANNOTATION AND INFORMATION SHARING IN PROTEIN SCIENCE WITH INFORMATION-SLIPS

Yujun Wang Tolga Can Yuan-Fang Wang Jianwen Su
Department of Computer Science
University of California
Santa Barbara, CA 93106-5110
{yjwang, tcan, yfwang, su}@cs.ucsb.edu

Abstract: In this paper, we describe a software tool called *Information-slips* (*i-slips*) that provides a convenient and customizable mechanism for remote collaboration and data sharing in protein science. I-slips are small 3D objects that coexist with and augment the host 3D objects (in our application, protein models). Our i-slip design makes two main contributions. Firstly, i-slip goes beyond simple passive annotation to also provide active interactivity. It can embed an action to perform a user-defined operation on-demand. The condition to perform the action is event-driven. So i-slips can monitor the environment and automatically handle some predefined events without the user intervention. Secondly, i-slip is a highly versatile and adaptable information container. The user can tailor i-slip templates for new domain-specific objects, and develop new actions to embed domain-dependent algorithms. Furthermore, the storage and transportation of i-slips use XML for the sake of interoperability. Two key implementation techniques about visualization and customization are discussed here. To the best of our knowledge, this is the first protein visualization tool that supports user-contributed information and embeddable actions/activities.

Keywords: Information-slips, protein visualization, collaborative system

1. Introduction

Molecular biology research has attracted a tremendous amount of interests. A large number of protein related databases exist, such as the Protein Data Bank (PDB) [1] that provides more than 22,000 macromolecular structures accessible on the Internet. They enable the user to retrieve protein data and information with ease. In return, users produce new results, experiences, and insights. Some of these are undoubtedly very valuable, thoughtful, and perhaps even profound.

While technologies in data archival and visualization in protein science have demonstrated their utility and have achieved great success, the important problems of personalized annotation and information sharing are still largely unsolved. Currently, visualizing 3D protein structure is done using a 3D visualization tool, and is separated from visualizing other information about the protein, which may exist in a multimedia format in web pages, research papers, emails, or other documents. It is arguably more elucidative to browse and display such auxiliary, or augmentative, information with a close association to the underlying 3D structure – instead of having the user to consult multiple sources. Furthermore, the user should be able to annotate protein structures with personal, putative information (e.g., research or class notes), and again, have the information displayed in close proximity to the underlying 3D structure. We consider below two specific examples that illustrate why a unified visualization paradigm is useful.

(1) The first example illustrates the need to associate auxiliary, augmentative information with the underlying 3D structure when visualizing protein data. Due to the limitation of instrumentation and experimental/recording errors, existing protein data files may contain erroneous or missing information. For example, experiments might not reveal the location of all the residues in a protein, and might report unexpected covalent bond angles. The PDB repository categorizes such augmentative information on protein structures as remarks, and assigns numbers and topics to those frequently occurring ones. For example, missing residues can be found in "REMARK 465," while erroneous bond angles can be found in "REMARK 500." Existing 3D visualization tools do not display such information. One technical issue is that missing residues do not have coordinates, making visualization difficult. There are additional difficulties in displaying bond angle errors. First, atoms that make an erroneous bond may be hidden in some visualization models. This is because a bond angle is defined by three atoms (N, CA and C) in a residue, and some visualization models, such as the ribbon model, do not show any atoms. Second, even when all atoms are displayed, it does not reveal the angular deviation (the detailed information) in a convenient, easily accessible way. The user is forced to consult elsewhere for such information. Clearly, the general methodologies for displaying derived and augmentative protein structural information in close association with the underlying structure are generally lacking.

(2) The second example illustrates the need to share user contributed or putative information on a protein. This is exemplified in a real-world case in the PDB Open Discussion Forum (http://www.rcsb.org/pdb/lists/pdb-l/). A group of geographically distributed users had a common interest in disulphide linkage in protein structures. User *A* noticed that a toxin protein (PDB ID: 1DL0) had two adjacent cysteines involved in a

disulphide linkage. She would like to share this discovery with other users. Following the report of user *A*'s finding, there were several more contributions from other users in the group on this topic. User *B* realized that user *A*'s discovery was related to the PDB keyword SSBOND. User *C* added that the protein with a PDB ID "4AAH" also had a vicinal S-S bond. User *D* pointed out a related paper in Nature Structural Biology in 2000. User *E* explained how to use the Sequence Retrieval System (http://srs.wehi.edu.au) to find all vicinal disulphide bonds as follows: First search the SwissProt for all instances of the feature "disulfide" yielding 69 entries. Then by linking the output to the PDB site one could find all the 16 related PDB entries. Further inspection of the headers indicated the following desired PDB files: 1DL0, 1EH5, 1EI9, 1FLG, 1OBR, and 4AAH.

This example shows that the users in the PDB community can help one another with valuable advices, timely discussions, and insightful contributions. At present, visualization of protein structures and organization of user-contributed information are separated. No protein visualization systems allow the users to contribute information and annotate it with the underlying structure for quick reference and browsing. Resorting to text comments on a monolithic bulletin board, where the user has to monitor and filter postings manually for relevant information, is hardly an adequate solution.

Our i-slip technology tries to fill in these gaps. It follows the time-honored tradition of the yellow sticky note in that it can be a parasitic object, whose existence is strongly tied to the host object in a model. The host objects can be any combination of chains, residues, and atoms. Each i-slip has a content object to carry additional information, and can choose among various visual forms according to its functions. Each i-slip also maintains an anchor point relative to the host object, and adapts its orientation to the camera's pose. So an i-slip moves along with the host object to identify the association relationship, and it can be viewed from multiple distances and vantage points.

Our i-slips do not blindly duplicate the traditional sticky notes. In particular, we move beyond a passive reminder service model and incorporate into i-slips actions and an automated activation mechanism. The action could be predefined by the system or user-defined. Actions can be performed on-demand, or be invoked when specific events happen. Such i-slips can support automated user intervention and function customization. These added features greatly enhance the usability of i-slips.

To illustrate the effectiveness of the concept, we have built a tool I-SLIPS, which represents a radical advance beyond the traditional 2D graphics and text annotation. Our system attaches i-slips directly to the 3D protein models. The users can interactively explore protein structures while calling up and reviewing additional information of the protein. To the best of our knowledge, it is the first tool to support user-contributed annotations and dynamic function invocation for 3D protein models. Our I-SLIPS system provides a 3D viewer for visualizing protein structures with i-slips, a 2D viewer for selecting host objects or submitting queries, a text i-slip editor for adding textual annotations and attaching actions. Together, these modules comprise a user-friendly environment for collecting user-contributed information and present such information in close association with the underlying structure for easy browsing.

Furthermore, we generalize our previous fast protein visualization techniques [3] in Java 3D to I-SLIPS. The new technique supports efficient construction of Java3D scene graphs, while allowing the users to highlight host objects and manipulate i-slips. We also present a simple Java programming interface to plug in new i-slip templates and user-defined actions.

The remainder of this paper is organized as follows: We introduce the concept of i-slips in Section 2. Details about the system I-SLIPS will be discussed Section 3. Two key implementation techniques are presented in Section 4. Finally we summarize our work and discuss future research direction in the conclusion section.

Related work

Our i-slips with text contents are similar to Post-it Notes [9] and TeleNotes [12]. A notable distinction is that they are only for 2D models. IRIS Annotator [7] does annotate 3D models; however, its annotations for 3D objects can only be accessed within the 3D models. It supports simple actions to execute external programs. It does not support automated monitoring services and situation-aware behaviors. Besides, our system can plug in new annotation data and actions.

Existing protein visualization tools such as Swiss-PdbViewer [5], Protein Explorer [8], RasMol [10] etc., and collaboration systems for protein structures such as BioCoRE [2], Chimera [6], MICE [11] etc., do not support user annotations. Cn3D [4] provides label and style annotation to 3D protein structures. Their annotations are mainly used for visualization purpose, that is, to distinguish the selected residues against the rest of the protein. Besides, the labels only show residue names that are authoritative information. We go beyond the above systems by providing i-slips for sharing additional user-contributed protein structure information. Their functionality and user interface will be greatly enhanced with the support of i-slips.

2. I-SLIP — A Customizable Portable Information Container

In this section we introduce the concept of an "information-slip (i-slip)" and its new features.

2.1 Concept of an i-slip

I-slips are small 3D objects that coexist with and augment the host 3D objects. An *i-slip* keeps information about the host object, maintains a suitable visual configuration for display, and records a content object. The host object is part of a 3D model. The visual configuration defines the way to visualize the i-slip in a 3D environment, and its position relative to the underlying 3D model. The content object stores additional information about the host object.

In our application scenario, the host objects can be any

combination of chains, residues and atoms in a protein structure, e.g., all the residues in a beta sheet, or the CA atom of a particular residue, etc. An example of an i-slip is shown in Figure 1, where the i-slip is shown as a hand icon and the host object is the residues in the first alpha helix in chain A of PDB ID: 1JPN.

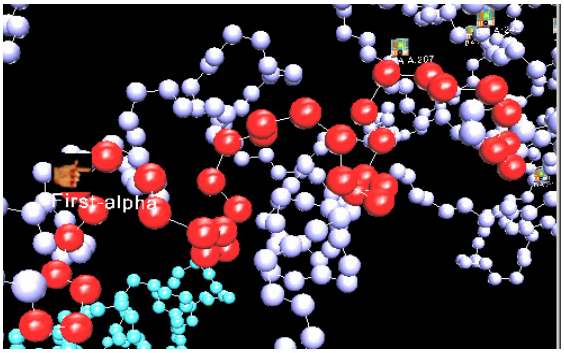

Figure 1: An i-Slip on the first alpha helix in 1JPN

An interesting visualization issue in a 3D world is that objects (and hence the attached i-slips) can be viewed from many different perspectives. While 3D geometry can be viewed as such, the same is not true for i-slips with 2D icons and text description. Hence, to be legible, texts and icons for i-slips should be displayed with some appropriate size and oriented toward the viewer. This turns out to be an interesting issue in visualizing. In our system, we design the visual form with a display behavior that automatically compensates for the rotation that skews the i-slip and brings the i-slip back to an orientation to face the viewer (camera).

The anchor point is the position for an i-slip in a 3D model. It is initially computed as a random position inside the bounding box of the underlying host object. The initial camera position is to make the center of the bounding box the center of the screen, and the viewing distance is twice of the longest side of the box. This viewing configuration allows the user to focus the view on the i-slip.

The content object may contain augmentative information from authoritative systems such as the PDB, or user-contributed information. For example, a text i-slip in our implementation stores free text annotation. Each text i-slip contains the following fields: author, time of last update, and body. The body contains free text. The time of the last update is automatically updated by the system. For the i-slip in Figure 1, it records a text description about the first alpha helix.

2.2 Domain specific I-slips

The content object of an i-slip can carry a variety of domain-specific information through customization. In our current design, we implemented a missing-residue i-slip and a bond-angle i-slip for visualizing predefined remarks in PDB data files together with the underlying protein model. I-slips for missing residues are extracted from REMARK 465 in a PDB data file. It contains the following fields: molecule name, chain number, residue name, and sequence number. I-slips for erroneous bond angles are extracted from REMARK 500. It contains the following fields: molecule name, chain number, residue name, sequence number, and deviation angle.

An i-slip is an open, customizable information container. New i-slip templates can be easily plugged in to accommodate domain specific information. The ability to adapt to new user data formats greatly increases the functionality, usability, and flexibility of our system.

2.3 I-slips with Actions

I-slips contain not only static data. A novel feature of our i-slips is that they can also store actions (or programs to be executed). The actions can be easily activated on-demand at some later time. This is convenient especially for executing certain repetitive tasks.

In the current design, we provide several predefined actions. One is to pop up a window and display a text message. The action is usually used for displaying warning messages, or instructions. Another predefined action is to open a related URL. The action can be used to go to other websites, and display web pages related to the current protein structure, e.g., to display the summary information from the PDB file, or to query a certain Internet archive for relevant information and models.

Our action mechanism also allows the users to easily plug in new actions. Such plug-ins allow the user to run their algorithms or programs on the selected host object. An abstract action, which is named ActionRoot in our implementation, defines an action name and an abstract method. The action name is used to identify an action. The abstract method, which we call actionPerformed, defines an interface for the program implementation when an action is performed. A user-defined action is required to be declared as a subclass of ActionRoot and implements the actionPerformed method. The arguments of actionPerformed allow user codes to access the molecule structure, protein viewer and the host object, and provide an input parameter. An example to add an action for protein structure comparison will be introduced in Section 4. We note here that the user-defined action behavior in our system was not extensively studied and implemented in the previous systems for annotations.

2.4 Event-driven activation mechanism

Actions are usually executed manually on-demand. In many cases, the user may register the action to be invoked automatically later when a specified event happens. For example, when a user first loads a protein structure, it will be helpful if a protein-structure-checking web service can be performed automatically. In the current implementation, we define events to reflect changes of the display environment. Each action can define an event, which is the condition to trigger the action. With the event-driven mechanism, an alert i-slip can monitor the environment for the specified condition, and then performs the actions if its condition becomes true.

Events supported in our current implementation are listed below: “Protein viewer is first shown”; “Protein view is changed”; “I-Slips are shown in protein viewer”; “I-Slips are hidden in protein viewer”; “New I-Slip is added”; “Existing I-Slip is modified” and “Existing I-Slip is deleted”. Many possible actions might be performed for

an event. For example, summary of best views, structure features, related protein structures, possible display options will be useful when a user first loads a protein data file for visualization. Detection of specific structure features to current camera's position will be very useful when the protein view is changed. In a distributed environment where multiple users participate in a collaborative session, notifications will be useful. Specifically, notifications of the events happened in the instructor's machine alert the students to pay attention to a certain protein feature or perform a certain action.

For example, we can use "Action for related URL" to go to What-IF (http://www.cmbi.kun.nl/gv/whatcheck/) and get a check report for the current protein structure. If the event is properly set, the window in **Figure 2** will be shown when user first opens the 3D structure viewer.

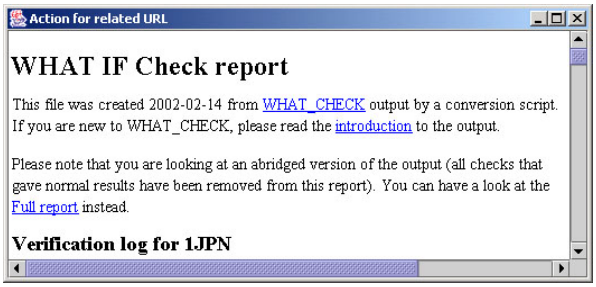

Figure 2: Get What-If check report

2.5 Storage and transportation of i-slips

In our system, i-slips are internally represented as serializable Java objects. They can be sent and received over a network. At the secondary storage level, i-slips are stored as XML files. This facilitates the sharing and transportation of i-slips for different discussion groups. It also provides an open interface for other systems to "read" i-slips.

3. Tools to Support I-Slips

We have developed a tool named I-SLIPS to demonstrate the concept of i-slips. I-SLIPS is a Java application running in a network environment. As shown in Figure 3, it needs to access external data source PDB.

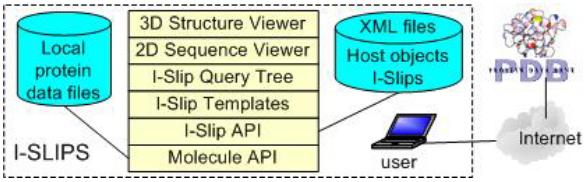

Figure 3: The System Configuration

In I-SLIPS, the 3D structure viewer is used to visualize 3D protein models with i-slips. The 2D sequence viewer provides a tree view for the protein sequence. In the 2D sequence viewer, the users can select objects to add i-slips, and query related i-slips. The query results are displayed in an i-slip query tree. Currently, our tool provides three i-slip templates: a text i-slip editor, a missing-residue template, and a bond-angle template. The I-Slip API is used to add, delete, modify, and query the repository for host objects and i-slips. The Molecule API is used to parse and build scene graphs, and get sequences or coordinates for the underlying protein model. Other utility tools, such as an FTP tool to automatically download protein data files and corresponding secondary structure information from the PDB, are also available but not shown in Figure 3.

Figure 3 also shows the architecture of the current version of I-SLIPS. The data for i-slips and protein structures are stored in a single machine. The advantage is that the users can work locally and annotate protein structures with i-slips. After it is done, the corresponding XML files can be electronically mailed to others for review. The client/server version to support synchronous distributed collaboration is currently under development.

More details about the components in I-SLIPS, including screen dumps, are presented below. Figure 1 shows the 3D structure viewer. It is based on the work on fast protein visualization [3]. The users can surf inside the 3D structure using mouse buttons to zoom in/out, rotate, and pan. The novel feature is the incorporation of the i-slips.

Figure 4 shows the 2D sequence viewer. It allows easy correlation of residues in the sequence with atoms in the 3D structure. This is accomplished by highlighting - just as with a text editor. Click-dragging the mouse across a region in the sequence window will cause the letters to become red (selected), and double clicking on a residue will select all the neighboring residues which are in the same secondary structure. To select a chain or an atom, simply double click it in the corresponding list. Once chains, residues or atoms are selected, they will be highlighted in red color in the 3D structure window. And the converse is true as well: double-clicking an atom in the structure window will cause it and the corresponding letter in the sequence window to light up. Furthermore, the users can switch between the sequence and secondary structure for a selection. This can be done by clicking on the button above the sequence of residues. The secondary structure helps the users to select relating residues.

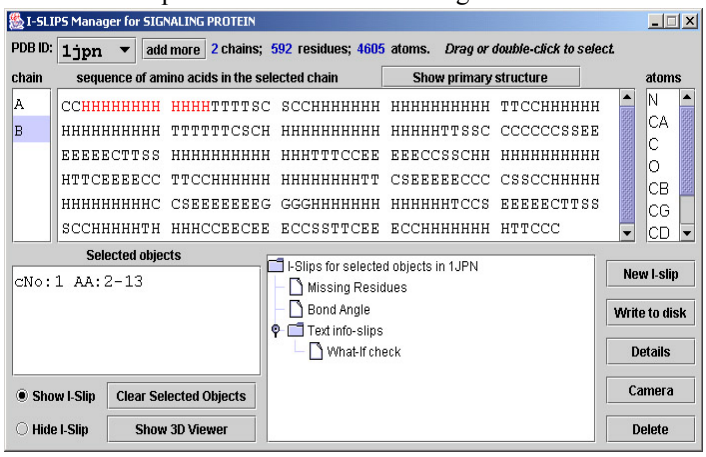

Figure 4: 2D sequence viewer

For example, let's walk through the procedure of selecting the first alpha-helix in chain B. A drop list for PDB ID allows the users to choose a protein structure that is locally available. A chain is selected in the chain list box. The sequence of protein is shown in the middle display panel. The user first clicks on the button above the

sequence to show the secondary structure in the sequence panel, and then drags the mouse over the first continuous H at chain B. Then a line will be added automatically to the host-object list box. To deselect the host object, select a line and double click it in host-object list box.

Once the host object is selected, a query request is sent to the i-slip repository for related i-slips. Those i-slips whose host objects overlap with the selected host object will be returned. If no host object is selected, all the stored i-slips will be returned. Returned i-slips are categorized and displayed in a tree as shown in the Figure 4.

A new i-slip can be added by clicking the button "New I-Slip." Figure 5 shows the editor window for a text i-slip. It has five panels. A host object information panel on the top left shows information about the host object of this i-slip. A text input panel on the upper left is for entering information on the subject, author, and message fields, and for selecting an icon to be used for visualization. On the lower left, an action panel lets the users select a predefined action, add a parameter, and set the event to trigger the action. On the bottom left, the command panel houses the control buttons (save, delete, and cancel). A preview panel on the right includes a graphical representation of the i-slip and the 3D hosts object it attached to. For the atoms not selected, only the bonds are displayed, so as to highlight the i-slip and its host objects.

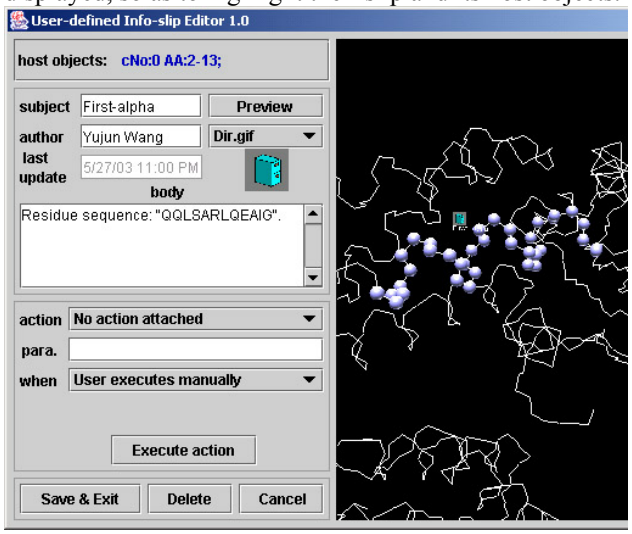

Figure 5: The Text i-slip Editor

Figure 6 shows two templates for domain-specific information, where the left one is for bond angles and the right one is for missing residues.

We now explain how the examples in Section 1 are supported in I-SLIPS. Information about missing residues and bond angles in protein structures can be extracted automatically from remarks in the PDB data files. The corresponding i-slips are organized in the query tree in Figure 4. The users can double click on the i-slips in the query tree or in the 3D viewer for detailed information. The templates are shown in Figure 6. For user-contributed information in the second example, they can be stored in a text i-slip. Augmentative information can be stored in the body field. Related URL link can be described as "Related URL Action."

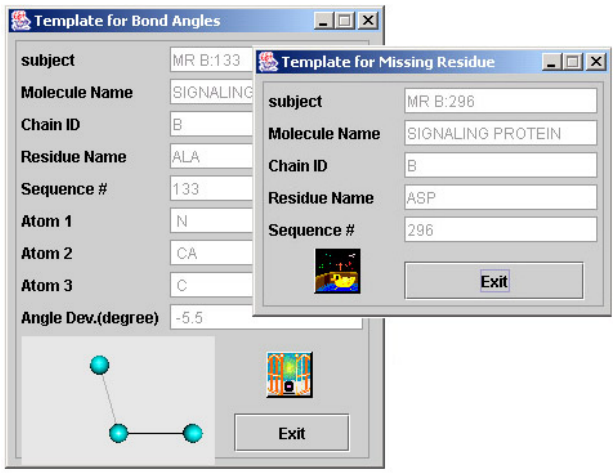

Figure 6: Domain-specific templates

4. Key Implementation Issues

In this section, we discuss two key techniques in the implementation: fast protein visualization technique with i-slips, and Java programming technique to support user-defined i-slip templates and actions. These techniques make our system flexible, adaptable, and efficient.

The visualization component of the I-SLIPS is implemented in Java 3D. Java 3D is a cross-platform API for developing 3D graphics applications in Java. Java 3D describes a 3D scene in a scene graph (basically, a tree structure). The interior nodes in the graph correspond to various grouping operations to collect simpler objects into larger, more complicated constructs (e.g., grouping atoms into a residue, and residues into a molecule). The exterior nodes are 3D objects and components to describe the shape, appearance, and behavior of these objects, and other entities such as lights, background, etc.

We previously have developed techniques to achieve fast protein visualization [3]. Unfortunately these techniques cannot be used with the introduction of i-slips. For example, if we combine shapes of the same appearance, the user cannot highlight the part of them as host objects.

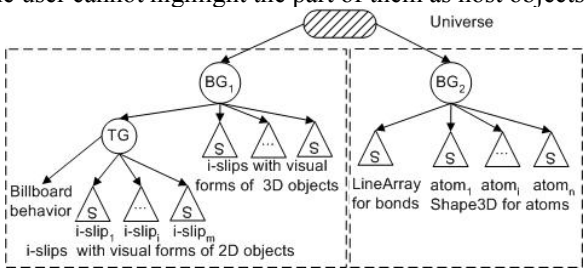

Figure 7: The scene graph of a visualization model with i-slips

In I-SLIPS, we represent each atom by a shape3D object. Each shape3D object is directly attached to the BranchGroup node for the whole molecule. That means that we reduce the number of Group nodes to zero. So the scene graph of a protein with i-slips has two parts as shown in Figure 7. The part for i-slips is shown in the left dotted rectangle, while the part for the molecule is shown in the right dotted one. Each part has a Java 3D BranchGroup object. So we can change the visualization models for molecules, and show or hide i-slips. As shown

in Figure 7, a shape3D is created at its coordinates for each atom. All the shape3D objects are added to BG_2 directly, as well as the LineArray for bonds. Similarly, Shape3D objects for i-slips with a visual form of 3D objects are added to BG_2 directly. For i-slips with 2D icons or text, we add a TransformGroup node TG and attach a billboard behavior to TG. Thus makes i-slip discernable from the underlying object rotation.

The flat structure of scene graph uses the least number of group nodes to reduce memory and calculation. It makes the visualization of protein structures with i-slips fast and efficient.

Now we discuss how user-defined templates and actions are implemented in I-SLIPS. The procedure to add a new i-slip template and action is similar, we only illustrate how to add an action. An abstract class called ActionRoot defines the Java programming interface for actions. As shown in Figure 8, each action has an actionName, and method named "actionPerformed". Method actionPerformed needs either no arguments, two arguments (the host object and an input string), or four arguments that include additional access to the molecule and viewer.

```
public abstract class ActionRoot{
  private String actionName;
  public ActionRoot(String actionName){
    this.actionName = actionName;
  }
  public void actionPerformed(){};
  public void actionPerformed(HostObjectVector hostObjV,
    String inputStr){};
  public void actionPerformed(PDBMol molecule,ProtViewer
    viewer,HostObjectVector hostObjV,String inputStr){};
}
```

Figure 8: Abstract class for all user-defined actions

To add a new action, a new Java class is to inherit ActionRoot. For example, to add the action for "Structure comparison", a class ActionStruCmp is defined as shown in the top of Figure 9. In the constructor, an action name is given. Through the arguments in actionPerformed, ActionStruCmp is applicable to chain objects. It compares the selected chain in the current molecule with a specified chain in another protein structure. Then the user can further code to display the comparison result.

```
public class ActionStruCmp extends ActionRoot{
  public ActionStruCmp(String actionName) {
    super(actionName);
    ...
  }
  public void actionPerformed(PDBMol molecule,ProtViewer
    viewer, HostObjectVector hostObjV, String inputStr){
    ...
  }
}
public class UserActionRegister{
  public UserActionRegister(ActionManager actionManager){
    ActionStruCmp actStruCmp =
      new ActionStruCmp("Structure Comparison");
    actionManager.registerAction(actStruCmp);
  }
}
```

Figure 9: A user-defined action and its registration

After the class for a new action is developed, it needs to be registered. In I-SLIPS, it can be simply done via a Java class UserActionRegister. The source code for UserActionRegister is shown in the bottom of Figure 9.

5. Conclusions

We have demonstrated the application of our i-slips system for visualization and collaboration in protein science. Our contribution lies in representing authoritative information and user-contributed information uniformly as i-slips and allow augmentative information to be associated with the underlying model. Our i-slips move beyond the passive reminder service model in traditional notes to provide action and interactivity. Our i-slips also support user-defined i-slips templates and actions, which simplify the task of adapting our i-slip tools to other application domains. Many application systems can benefit from the i-slip technology to enhance collaboration and user interface. The source code of the I-SLIPS tool and full version of this paper are available at: http://www.cs.ucsb.edu/~yjwang/i-slips/index.html.

Many useful features can be added to our system to further enhance its functionality, such as synchronous collaboration, advanced visualization techniques to visualize host objects, and query and search mechanism based on XML. I-slips can be further improved by adding these and other new functionality.

Acknowledgements

The work is supported in part by NSF grants IIS-9817432, IIS-9908441, and IIS-0101134.

References

1. H.M. Berman, J.Westbrook, Z.Feng, G.Gilliland, T.N. Bhat, H.Weissig, I.N. Shindyalov, P.E. Bourne. The Protein Data Bank. *Nucleic Acids Research*, 28, 2000, 235-242.
2. M. Bhandarkar, G. Budescu, W. F. Humphrey et al. BioCoRE: A collaboratory for structural biology. *Proc. of the SCS International Conference on Web-Based Modeling and Simulation, San Francisco,* 1999, 242-251.
3. T. Can, Y. Wang, Y. F. Wang, and J. Su. FPV: Fast Protein Visualization Using Java 3D™. *Bioinformatics* 2003 vol.19, 913-922.
4. Cn3D. http://www.ncbi.nlm.nih.gov/Structure/CN3D/cn3d.shtml.
5. N. Guex and M.C. Peitsh. SWISS-MODEL and Swiss-PdbViewer: an environment for comparative modeling. *Electrophoresis,* 18, 1997, 2714–2723.
6. C.C. Huang, G.S. Couch, E.F. Pettersen, and T.E. Ferrin. Chimera: An Extensible Molecular Modeling Application Constructed Using Standard Components. *Pacific Symposium on Biocomputing,* 1996, p.724.
7. IRIS Annotator. http://www.sgi.com/software/annotator/.
8. E. Martz. Protein Explorer: Easy Yet Powerful Macromolecular Visualization. *Trends in Biochemical Sciences*, 27 (2002.02). 107-109. (http://proteinexplorer.org)
9. Post-it Software Notes 2.0. 3M. http://www.3m.com/us/office/postit/.
10. R. A. Sayle and E. J. Milner-White. RASMOL: Biomolecular Graphics For All. *Trends in Biochemical Sciences,* 20, Sep. 1995, 374–376.
11. J. G. Tate, J. L. Moreland, and P. E. Bourne. Design and Implementation of a Collaborative Molecular Graphics Environment, *Journal of Molecular Graphics and Modelling,* 2001, 280-287.
12. S.Whittaker, J.Swanson, J. Kucan, and J. Sidner. TeleNotes managing lightweight interactions in the desktop. *ACM Trans. on Computer-Human Interaction*, 4(2), 1997, 137-168.