Plugging-in Visualization:

Experiences Integrating a Visualization Tool with Eclipse

Rob Lintern Jeff Michaud Margaret-Anne Storey Xiaomin Wu

Dept. of Computer Science
University of Victoria
Victoria, BC Canada
{rlintern, jmichaud, mstorey, xwu }@uvic.ca

Abstract

The Eclipse platform presents an opportunity to openly collaborate and share visualization tools amongst the research community and with developers. In this paper, we present our own experiences of "plugging-in" our visualization tool, SHriMP Views, into this environment. The Eclipse platform's Java Development Tools (JDT) and CVS plug-ins provide us with invaluable information on software artifacts relieving us from the burden of creating this functionality from scratch. This allows us to focus our efforts on the quality of our visualizations and, as our tool is now part of a full-featured Java IDE, gives us greater opportunities to evaluate our visualizations. The integration process required us to re-think some of our tool's architecture, strengthening its ability to be plugged into other environments. We step through a real-life scenario, using our newly integrated tool to aid us in merging of two branches of source code. Finally we detail some of the issues we have encountered in this integration and provide recommendations for other developers of visualization tools considering integration with the Eclipse platform.

Introduction

Many visualization tools that are developed in the research community are customized applications that are built from scratch. These visualization tools are dependent on having access to information sources about the software that are both rich and accurate. Research groups often have to write their own tools or even beg, borrow and steal parsers, and other information extractors to provide data for the visualization technique. These efforts are usually disjointed and many research groups have experienced frustration from reinvention of the wheel. Furthermore, since the visualization tools are stand-alone applications and do not integrate easily with the existing tools that developers use, it is difficult to evaluate their usefulness in real world contexts. Moreover, it is often impossible to combine features and tools from these stand-alone applications, or to compare them as each will offer many different features.

Over the past few years, we too initially focused on developing a stand-alone software visualization tool to assist in program understanding. Our tool is called SHriMP Views, which stands for Simple Hierarchical Multi-Perspective Views. SHriMP uses a nested graph view to display hierarchical structures in a Java program (see Fig. 1). Composite nodes in the graph represent key structures (for example, packages and classes) in the software. Leaf nodes correspond to entities in the software such as methods, and data types. Arcs in the graph show dependencies between these artifacts and may show inheritance, composition and association relationships. The *nested interchangeable view* feature in SHriMP allows a user to look at different presentations of information at any level of detail. A programmer can browse source code or documentation by following hyperlinks that result in animated panning and zooming motions over the nested graph.

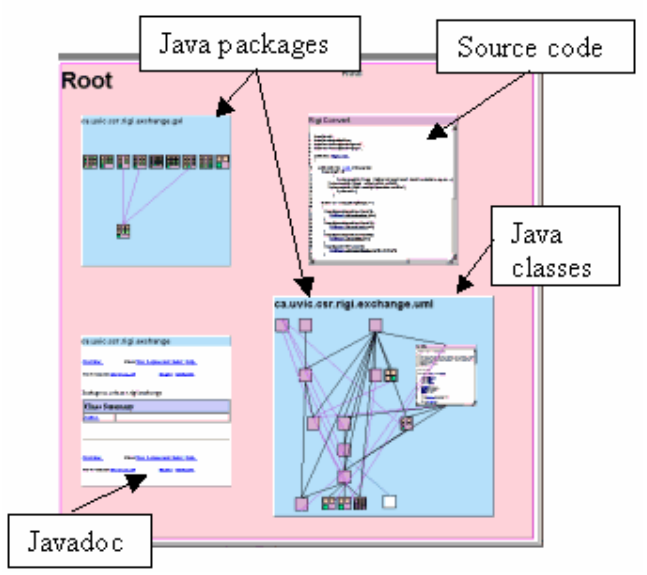

Figure 1. A SHriMP View of a Java program.

Integration

Over the past year we successfully integrated SHriMP with the open source Eclipse project (see Fig. 2). Eclipse (ww.eclipse.org) is a general purpose platform upon which other tools can be built as *plug-ins*. The JDT (Java Development Tools) are a suite of such plug-ins, comprising a full featured Java IDE, which comes bundled with the free download of the Eclipse platform. Many other commercial and research groups have developed further plug-ins for the Eclipse platform and the JDT– such as UML tools, version control tools, team support etc.

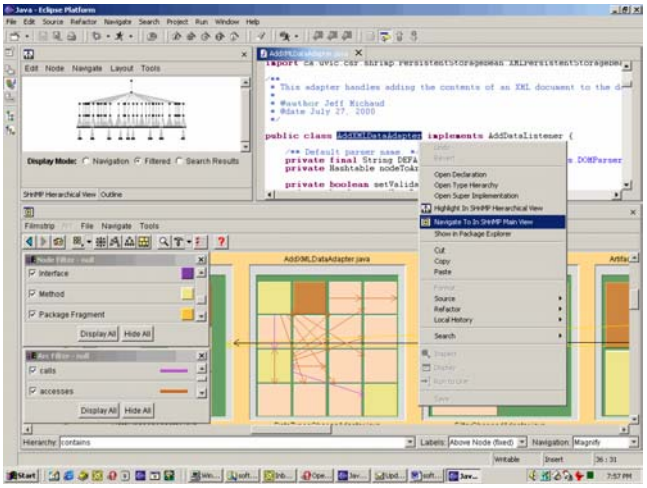

Figure 2: SHriMP plugged into the Eclipse platform. *SHriMP Hierarchical View* is shown in top left pane, *SHriMP Main View* shown is shown in the bottom pane and source code shown in top left pane, all of which are synchronized.

We refer to the integration of SHriMP with the Eclipse JDT as "**Creole**". Since Eclipse provides access to the program repository, we now can instead focus on visualization and how it can be further developed to provide support to the existing features in Eclipse.

Integrating SHriMP with Eclipse has also provided access to new information sources via existing plug-ins. Of particular interest is the CVS plug-in which is an integrated GUI front-end for the CVS version control system. We conjecture that visualization of team relevant information such as CVS histories could be of significant assistance in collaborative tasks. To explore this topic, we integrated SHriMP with the CVS plug-in giving us the ability to visualize information stored in the underlying CVS repositories. We refer to this integration as **"Xia"**.

Our most recent work has been spent creating a composite visualization of information from both the JDT repository and the CVS repository. We believe that such views could be used to reveal:

- Who is responsible for which parts of the system?
- Which parts of the system tend to change frequently?
- Which parts have been changed since a particular date?
- What are the relationships between these parts of interest and the rest of the system?
- … and any combination of the above

Discussion

We have found the integration of SHriMP with the JDT (i.e. Creole) to be of benefit when trying to **navigate** and **understand** code written by other groups. It is especially powerful when we are first exploring code and trying to get an overview of the scope and design of a program. With respect to providing support for **collaboration** and project **management**, we have found the visualizations of the CVS information to be very useful despite the fact that our tool is still at a prototype stage and is not very robust.

There are still, however, many issues that remain from this trial and many questions that have been raised. We are faced with much to explore. Much more empirical work is required before any conclusions can be drawn. We need to discover the specific tasks that our visualizations could help with. This leads us to the underlying question: *who* exactly is our user? Is it the team lead or software designer who needs a tool to support high-level decisions, or is it the programmer doing day-to-day programming tasks, or is it both? We need to empirically study Creole to determine whether or not it actually decreases cognitive load and increases performance on specified tasks.

Another issue we have come across is one that arises with any visualization. It is difficult to decide which view of the information is the most useful. Our visualization depends on the information we have at hand. In our case we have two sources of information: the JDT and the CVS plug-ins. Through these two plug-ins we now have easy access to reliable information, but, is it the right information for producing visualizations that help with the software task at hand?

Other future work will include using more animation in our visualizations to aid in refactoring code, comparing code, and synchronizing code with a repository, CVS repository we could animate the evolution of a project over time.

One major issue faced in our integration with Eclipse is that its GUI is built from a toolkit called SWT instead of using the more widely used AWT and Swing toolkits. The major advantage of SWT is that it uses native widgets wherever possible, increasing speed and guaranteeing the native platform's look and feel. This departure has been a major hurdle for our integration; SHriMP relies heavily on a zooming library based on AWT and Swing, making it difficult for us to create an SWT only version of our software. The approach taken to embed Swing and AWT widgets inside of SWT widgets is still problematic and results in some screen flickering, missing popup menus, and other GUI glitches. Furthermore, this UI integration currently only works on the Windows platform, and is not encouraged or officially supported by OTI (OTI are the primary developers of Eclipse).

Despite the work required to redesign aspects of our architecture, and issues integrating Swing and SWT widgets, the effort required to do the integration was not that arduous, especially when we consider what we have gained as a tool developer. The biggest issue that we have faced doing research in the largely non-validated area of software visualization, is trying to evaluate our own work. This has in part been hampered by not being able to evaluate how the visualization techniques work when they are used as part of the normal tools used by developers. By integrating with Eclipse, we can now continue with these evaluations, and furthermore, combine features from our tool with other visualization tools for further feedback and comparison.

More Information

http://shrimp.cs.uvic.ca/