Interactive Exploration of Compacted Visualizations for Understanding Behavior in Complex Software

Elizabeth L. Murnane
Architexa, emurnane@architexa.com

Vineet Sinha
Architexa, vineet@architexa.com

Abstract
In this poster we present Chrono, a tool that creates sequence diagram based visualizations. Since the diagrams produced by traditional sequence diagramming tools become large and unmanageable when dealing with complex code bases, Chrono focuses on removing less relevant information, condensing diagram components, and allowing for interactive exploration.

Categories and Subject Descriptors
D.2.2 [Design Tools and Techniques]: User Interfaces.
D.2.6 [Programming Environments]: Graphical Environments, Integrated Environments, Interactive Environments.

General Terms
Design, Human Factors, Management.

Keywords
Complex Software Systems; Diagram Compression; Exploration; Interaction; Reverse Engineering; Software Behavior; Software Visualization.

1. Introduction
Sequence diagrams are used to show the behavior of a system and allow developers to investigate how a system is used. A sequence diagram can be used to explore the life cycle of an element or the course that a task takes to complete. Problems arise, however, when processes get complicated since the diagrams produced by traditional tools can quickly become unmanageable. The growing abundance of software systems with complex codebases demands that developers are able to understand such processes, so Chrono aims to satisfy this need by using the following information management strategies to provide easy analysis of a developer’s logic and demonstrate overviews of behavior.

First, Chrono strives to make understanding as easy as possible by limiting the amount of information initially displayed, by providing users the ability to easily compress or hide components in the diagram, and by using unobtrusive visual components whenever possible to convey information. In these ways Chrono emphasizes the key parts of a system and the elements most important and relevant to each user.

Also, since Chrono presents users with this minimal, manageable amount of information and compacts representations in order to not overwhelm, it also gives users the ability to easily interact with and manipulate a diagram. Users can expand, collapse, add, or delete components in order to show more or less information, which allows for the incremental exploration of a large or complicated project.

2. Walkthrough
Let us consider using Chrono in order to understand the source code of Lapis [1], an open source web browser and text editor.

2.1 Exploration
A user can begin exploring by opening a class in Chrono or by dragging it from the package explorer to the Chrono editor.

Figure 1. The representation of a class in Chrono
When the class is selected, a button is shown that can be pushed to view the methods that class contains. When a method in the list is selected, it is added to the diagram, and the user can begin exploring how the system operates.

Figure 2. The list of methods declared inside the class
Once a figure that represents a method declaration has been added to the diagram, the user can select it and press a button to see a list of all of the methods that are invoked in the declaration. If no such button is displayed, the user immediately knows that the method invokes nothing.

Figure 3. The list of invocations made by a selected method declaration
As can be seen in Figure 3, Chrono permits a user to only add the most pertinent information to a diagram by showing all of the methods that are invoked but only allowing application calls to be displayed. (By application method we mean a method defined in the developer’s code and not in an external library). The user can choose to display all of the invoked application methods, or he can select just one of them and gradually progress through the life cycle of the method.

Copyright is held by the author/owner(s).
OOPSLA ’08, October 19–23, 2008, Nashville, Tennessee, USA.
Similarly, a user can not only explore what calls a method makes by using the right side button, but he can also see what code invokes the selected method by pressing its left side button. By selecting one of the methods in this list, the invocation call is added to the diagram, and a user can trace back through the code’s execution.

2.2 Automatic compacting and filtering
Because it is easy for sequence diagrams to get large fast, Chrono compacts and filters information by default and allows a user to expand, display, and explore components as desired in order to focus on the parts most relevant to him.

As seen in Figure 4, Chrono displays chained calls as a single message and allows a user to select one of the methods in the chain in order to expand and display it separately in the diagram.

![Figure 4. Chrono compacts chained calls and allows a user to expand them piece by piece](image)

Also, if a visualization contains instances that are ordered in such a way that an invocation message flows from right to left, Chrono assumes that this message is not the intention of the diagram since time increases from top to bottom and left to right in sequence diagrams, and it is hidden. The corresponding method invocation and declaration figures are highlighted in red, and only when the user hovers over one of them with the mouse is the backward call shown.

Finally, when the user scrolls below the bottom of an instance’s life line (the vertical line that descends from an instance and represents its life span), this instance is no longer participating in any of the processes that are in view. It is therefore currently extraneous, so Chrono hides it.

2.3 Direct compacting and filtering
In addition to condensing and hiding components automatically, Chrono also allows a user to do this directly. To illustrate, we discuss first Chrono’s representation of control flow and then its filtering of non-application methods.

In order to let a user visually observe the possible flow of control that a system could take depending on various conditions or tests, Chrono displays loops, iterations, and conditional statements. Of course, this could result in the same statement inside a loop being displayed multiple times or in a large number of paths being displayed when only one is actually executed. Chrono therefore eliminates an overabundance of information by displaying loop statements only once and by displaying only one of the possible execution paths and indicating that other paths exist but are not shown. For example, all possible branches in an if statement block are not displayed. Only the method calls that would execute if the initial if statement condition is true are shown, and the user can collapse or expand each portion of the block to reveal the statements in any else if and else if blocks (initially shown as simply “...”). Chrono also indicates that method calls are part of a control flow block by surrounding them with a simple highlight that is only displayed when the mouse is hovered over the block.

![Figure 5. Chrono only displays the if-then statements and hides all others by showing them as “...”](image)

![Figure 6. A user can collapse and expand the branches of a control flow block, which is highlighted on hover](image)

Chrono’s interface also contains a button that when pressed hides all non-application classes and methods in the diagram. The user can reveal these hidden components by pressing the button again. This de-clutters the diagram and emphasizes the parts most relevant to the user - those corresponding to elements in his own code.

By continuing to use Chrono, the user can build a more comprehensive diagram in order to further model and make sense of how his system behaves.

3. Implementation
Chrono is a plug-in for the Eclipse platform that is built on Relo [2] and is part of Architexa’s Source Explorer. It is tightly integrated with the development environment in order to minimize its learning curve and make using it intuitive for the user.

4. References