# A Methodological Note on Setting-up Logging and Replay Mechanisms in InfoVis Systems

Nathalie Henry
INRIA Saclay – Île-de-France
Université Paris-Sud &
University of Sydney
+33 1 69 15 34 86
Nathalie.Henry@lri.fr

Niklas Elmqvist
INRIA Saclay – Île-de-France
INRIA, Bât 490
Université Paris-Sud, Orsay, France
+33 1 69 15 61 97
Elm@lri.fr

Jean-Daniel Fekete
INRIA Saclay – Île-de-France
INRIA, Bât 490
Université Paris-Sud, Orsay, France
+33 1 69 15 64 94
Jean-Daniel.Fekete@inria.fr

## ABSTRACT
Information Visualization needs longitudinal studies to assess the usefulness, usability, and, more generally, the value of its techniques. However, most of the longitudinal studies conducted so far have involved human resources to collect and analyze evidences. Automatic logging and session replay mechanisms can help answering questions while limiting human collection and analysis of data. In this article, we describe how to set up these mechanisms in InfoVis systems with a minimal effort.

## Categories and Subject Descriptors
H.5.2 [User Interfaces]: User-Centered Design; H.5.m [Information Interfaces and Presentation e.g. HCI]: Computer Mediated Communication

## General Terms
Design, Experimentation, Human Factors, Verification.

## Keywords
Logging, information visualization, longitudinal user studies.

## 1. INTRODUCTION
The number of novel information visualization (InfoVis) techniques and visual exploration systems is constantly increasing. However, there is still no simple solution on how to evaluate them. Controlled experiments performed in laboratories with groups of recruited users can help validate a few aspects of a technique, but this remains only a partial validation, on small sets of simple tasks and datasets and in very specific "clean room" conditions.

The real challenge faced by our community is to understand how users manipulate InfoVis systems *in situ* to discover insights and understand their own data. Catherine Plaisant [1] underlined the challenge of InfoVis systems evaluation, explaining that discovery occurs over a long period of time and often answers questions you did not know you had. Following this line, several researchers proposed insight-based evaluation [2,3] performed through longitudinal studies with real users.

In the last BELIV workshop, Ben Shneiderman and Catherine Plaisant [4] proposed a methodology to perform Multiple In-depth Longitudinal Case Studies (MILCS). Conducting this kind of study is long and difficult since it is mainly done through human observation and analysis.

Automatic logging may be one way to avoid or reduce the need for human intervention, and has proved useful for several domains of computer science as well as in other sciences. Logging consists of recording information on "interesting events" that occur during the life of a running program so as to be able to analyze it later. High-level sensemaking concepts such as insights and their context of discovery are not feasible to collect directly because they require human interpretation. However, collecting the sequence of actions performed on the system and replaying the whole work session might help the experimenter understand how the discovery happened and what led to a successful exploration.

Techniques for logging concrete events produced by programs and signals are relatively straightforward. However, they often require the modification of a large quantity of code if the need for logging was not anticipated during the design of the system. Moreover, additional programming is often required to be able to replay a previous session recorded on the system. In this position paper, we present our solution to set up logging and replay mechanisms in InfoVis systems implemented in Java. Our goal is to minimize both the amount of code to modify and the implementation time.

## 2. INFOVIS SYSTEMS IN JAVA
We designed and implemented a number of visualization techniques with the InfoVis Toolkit [5], implemented in Java. Recently, we developed NodeTrix [6], a simple visualization technique to support community discovery in social networks.

### 2.1 Evaluating NodeTrix
NodeTrix is an interactive visualization technique for graphs. It merges the traditional node-link representation of graphs with visual adjacency matrices to represent communities. We selected this example as it contains only a small set of functionalities: modifying the visual layout of the representation (changing node and community positions), attributing visual variables to the graphical objects (nodes, links, communities) and editing community contents (by adding or removing nodes).

Even with this simple visualization technique, implementing the logging and replay mechanisms *a posteriori* implies the modification of a large quantity of code. A logging function has to be added for each action on a graphical object to change its position or its representation (color or size for example) or to edit communities. In addition to the time needed to integrate the logging mechanisms, they might also degrade the clarity of the code by adding variables and functions solely dedicated to

logging. To solve this problem, and implement logging with a minimal cost, we used aspect-oriented programming.

## 2.2 Logging with AspectJ

Aspect-oriented programming [5] is a paradigm for separating the different concerns of a program into sets of functions that overlap as little as possible. Thus, the program is decomposed into a maximum of "aspects", keeping each of them clear and easy to maintain. For example, a simple InfoVis technique could be divided into three aspects: one for the creation of graphical objects, one to manage user interaction and a third one to handle the logging. The particularity of logging is that it affects all other aspects, as it is required to log all actions of the system.

AspectJ is the name of the Java implementation of Aspect-oriented programming and is easy to integrate in programming environments [8]. Aspects can be added to any java programs and *creating and using a logging aspect does not require any modification of the code*. The principle is to write a separate file, in which each function of the system to log and the corresponding action to perform are described. The logging functions do not appear in the original program as the instructions are directly added to the executable version of the program.

In our case, we decided to generate a textual log for each user actions performed on NodeTrix. A simple example to log the name and content of a community follows:

```
pointcut createCommunity():
  call(int createCommunity(String,IntArrayList));

before(): createCommunity(){
  WriteLog("Community creation at"
  + System.currentTimeMillis();
}

after(String name, IntArraylist members)
returning (int cIndex):
  call(int createCommunity(String,IntArrayList))
  && args(name,members){
    WriteLog("Community #"+ cIndex +"("+name+")"
    +"contains "+ members.size() +"members.");
}
```

## 2.3 Replaying with Jython

To replay actions of a previous session, we use the scripting functionality of Jython. Jython is a Python interpreter written in Java [9]. The strength of Jython is to provide instructions directly executable by a running Java program. Thus, it offers the possibility to query the system interactively (already used in the Guess InfoVis tool [10]) and to replay previous logs if they use the Python syntax.

Using a Python interpreter inside an existing Java program allows for Java objects to be transferred and used exactly as they are in the original Java program. Therefore, replaying a log only requires writing instructions instead of a standard textual log. A simple example replaying how the user created a basic community follows:

```
//In the java program (visualization class)
PythonInterpreter pi = new PythonInterpreter();
// pass the object visualization
pi.set("visualization", this);
pi.execfile(scriptFile);
```

```
// In the scriptFile
// log of java operations performed on the system
visualization.circularLayout();
int cIndex = visualization.createCommunity("c1");
visualization.addMember(cIndex,2);
visualization.addMember(cIndex,3);
visualization.moveCommunity(cIndex,20,20);
```

# 3. CONCLUSION

Longitudinal studies help evaluating InfoVis systems. This position paper explains how to set-up logging and replay mechanisms in InfoVis systems written in Java using AspectJ for logging without modifying a line of code of an existing program and using Jython for replaying recorded sessions. These technologies greatly simplify the implementation of logging and sessions replay; they should encourage InfoVis system designers to undertake longitudinal evaluations.

# 4. REFERENCES


[1] C. Plaisant. The challenge of information visualization evaluation. In Proc. of the ACM conference on Advanced Visual Interfaces (AVI), pp 109–116, Gallipoli, Italy, 2004.

[2] C. North. Toward measuring visualization insight. IEEE Computer Graphics Applications, 26(3): 6–9, 2006.

[3] C. Plaisant, J-D. Fekete, and G. Grinstein. Promoting insight-based evaluation of visualizations: From contest to benchmark repository. IEEE TVCG, 14(1): 120–134, 2008.

[4] B. Shneiderman and C. Plaisant. Strategies for evaluating information visualization tools: multi-dimensional in-depth long-term case studies. In Proc. of the AVI workshop BELIV'06, pp 1–7, New York, NY, USA, 2006.

[5] J-D. Fekete. The InfoVis Toolkit. In Proc. of the IEEE Symposium on Information Visualization (InfoVis'04), pp 167–174, 2004.

[6] N. Henry, J-D. Fekete and M. J. McGuffin. NodeTrix: A Hybrid Visualization of Social Networks. In IEEE TVCG (Proc. of IEEE InfoVis'07), 13(6): 1302-1309, 2007.

[7] G. Kiczales J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J-M. Loingtier, and J. Irwin. Aspect-Oriented Programming. In Proc of the European Conference on Object-Oriented Programming, 1241, pp 220–242,1997.

[8] AspectJ. http://www.eclipse.org/aspectj/

[9] Jython. http://www.jython.org/Project/

[10] E. Adar. Guess: a language and interface for graph exploration. In Proc. of the SIGCHI conference on Human Factors in Computing Systems (CHI'06), pp 791–800, New York, NY, USA, 2006.