# Chapter 18
# A Programmer-Centric and Task-Optimized Object Graph Visualizer for Debuggers

**Anthony Savidis**
*Institute of Computer Science-FORTH, Greece & University of Crete, Greece*

**Nikos Koutsopoulos**
*Institute of Computer Science-FORTH, Greece*

## ABSTRACT

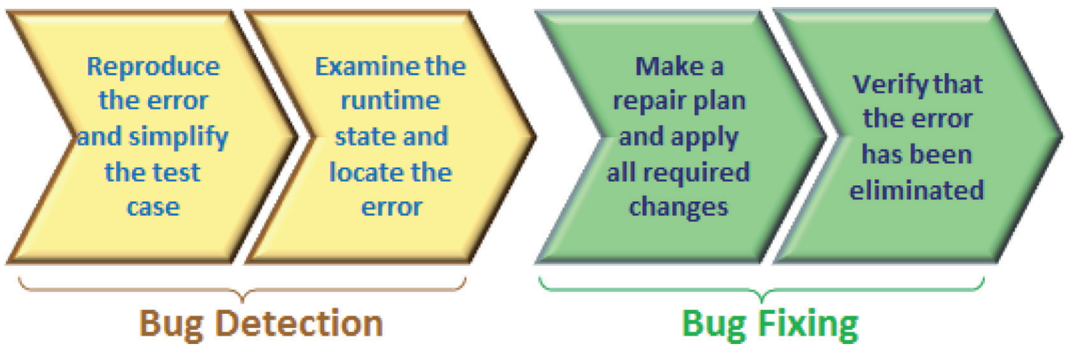
*Today, existing graph visualizers are not popular for debugging purposes because they are mostly visualization-oriented, rather than task-oriented, implementing general-purpose graph drawing algorithms. The latter explains why prominent integrated development environments still adopt traditional tree views. The authors introduce a debugging assistant with a visualization technique designed to better fit the actual task of defect detection in runtime object networks, while supporting advanced inspection and configuration features. Its design has been centered on the study of the actual programmer needs in the context of the debugging task, emphasizing: 1.) visualization style inspired by a social networking metaphor enabling easily identify who deploys objects (clients) and whom objects deploy (servers); 2.) inspection features to easily review object contents and associations and to search content patterns (currently regular expressions only); and 3.) interactively configurable levels of information detail, supporting off-line inspection and multiple concurrent views.*

## INTRODUCTION

Debugging is the systematic process of detecting and fixing bugs within computer programs and can be summarized (Zeller, 2005) by two main steps, bug detection and bug fixing, as outlined under Figurer 1. The examination of the runtime state requires inspection of object contents, associations and dependencies.

DOI: 10.4018/978-1-4666-4309-3.ch018

*Figure 1. The overall workflow of the debugging process*


Reproduce the error and simplify the test case
Examine the runtime state and locate the error
Make a repair plan and apply all required changes
Verify that the error has been eliminated
Bug Detection
Bug Fixing

The latter is a difficult task, even for small-scale applications, where traditional graph visualizations proved to be rather ineffective. The remark is justified by the fact that most popular commercial IDEs like Visual Studio (Microsoft) and IntelliJ IDEA (Jet Brains) do not provide them, while open source IDEs like Eclipse and NetBeans have a couple of relevant third-party plug-ins which are rarely used. But still, during debugging, objects remain the primary information unit for gaining insights (Yi et al., 2008) on how errors infect the runtime state. Interestingly, while the notion of visualization generally receives positive attention, relevant implementations failed to essentially improve the state examination process.

We argue that this is due to the primary focus of present tools on visualization, adopting general-purpose rendering algorithms being the outcome of graph drawing research, however, lacking other sophisticated interactive features.

More specifically, general graph drawing algorithms aim to better support supervision and pattern matching tasks through the display of alternative clustering layouts. This can be valuable when visualizing static features like class hierarchies, function dependencies and recursive data structures. However, during the debugging process the emphasis is shifted towards runtime state analysis involving primarily state exploration and comparison tasks, requiring detailed and advanced inspection facilities. We argue that the inability to systematically address this key issue explains the failure of object graph visualizers in the context of general-purpose debugging. As we discuss later, most existing tools are no more than mere implementations of graph drawing algorithms. Our primary goal is to provide interactive facilities which improve the runtime state examination process for defect detection. This goal is further elaborated into three primary design requirements:

- Visualization style inspired by a social networking metaphor enabling easily identify who deploys objects (clients) and whom objects deploy (servers)
- Inspection features to easily review object contents and associations and to search content patterns (currently regular expressions only)
- Interactively configurable levels of information detail, supporting off-line inspection and multiple concurrent views

The reported work (i-views) has been implemented as a debugger plug-in on top of the Sparrow IDE for the Delta programming language being publicly available (Savidis, 2010). This language has been chosen for ease of implementation, since it offers an XML-based protocol for extracting object contents during runtime (Savidis & Lilis, 2009). Overall, our results may be applied to any other debugging tool.