Bringing Objects to Life: Supporting Program Comprehension through Animated 2.5D Object Maps from Program Traces

Christoph Thiede, Willy Scheibel, and Jürgen Döllner

Hasso Plattner Institute, University of Potsdam, Germany

Abstract

Programmers who want to explore the architecture of software systems need appropriate visualizations such as software maps. However, existing software visualizations mainly display the static software structure, neglecting important dynamic runtime behavior. We propose *animated 2.5D object maps* that depict particular objects and their interac-tions from a program trace. From our experience of using our prototype with a couple of use cases, we conclude that animated 2.5D object maps can practically benefit program comprehension tasks, but further research is needed to improve scalability and usability.

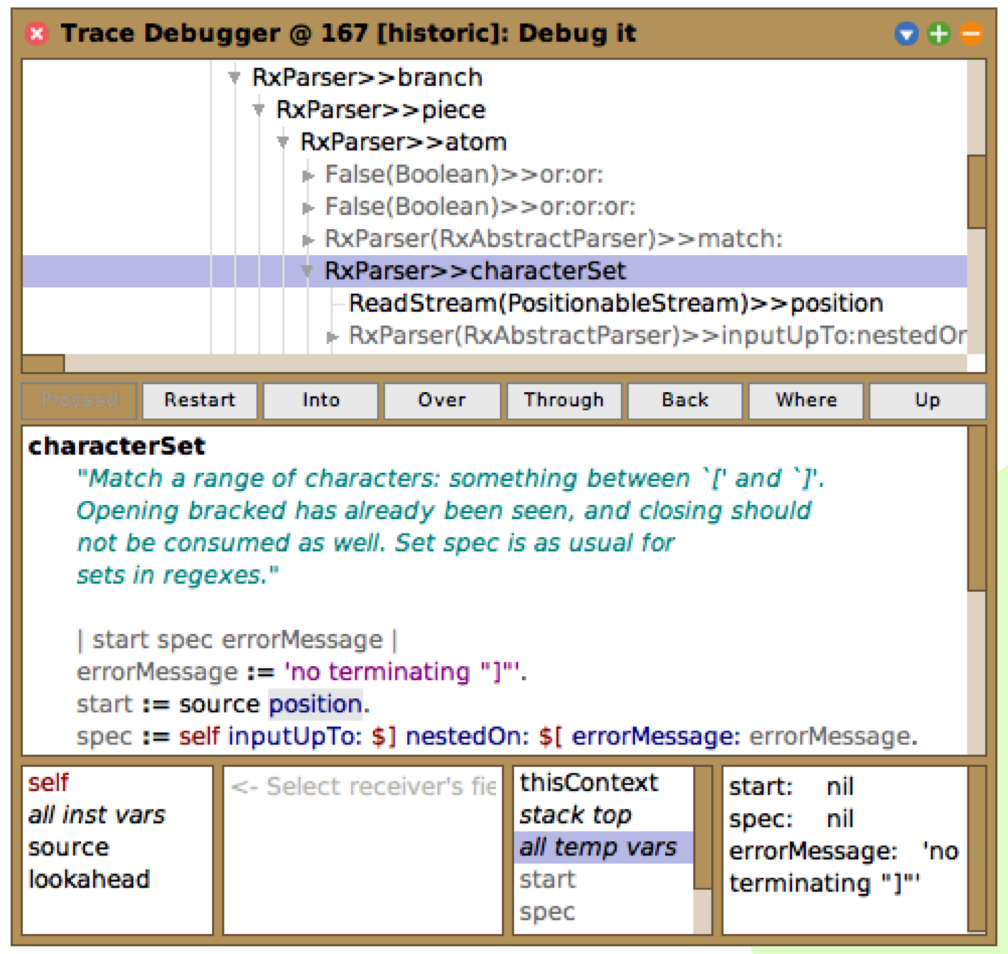

Figure 1: Using the TRACEDEBUGGER, an omniscient debugger for the Squeak/Smalltalk programming system, to explore the parsing of a regular expression pattern [Thiede, 2023].

Background

Programmers often encounter familiar or unfamiliar software systems that they want to repair, modify, or extend. To understand these systems, they build up a mental model that links observable behavior to architectural and implementation units. For this, programmers traditionally start by studying the source code of systems. Because this approach is often hampered by abstractions and a lack of examples, *behavior-centric exploration* has become more popular in which programmers invoke a system with concrete inputs or test cases and explore its execution in a debugger to identify relevant units and their interactions by example.

Since traditional debuggers can only move forward along the execution time of a program, *omniscient debuggers* (or *time-travel debuggers*) allow for exploration of a recorded *program trace* in a nonlinear fashion [Pothier, 2009] (fig. 1). However, the displays of omniscient debuggers are too fine-grained and text-heavy to provide an overview of large program traces with multiple subsystems and many interacting objects, posing a need for inter-active visualization techniques that can be used to explore systems' behavior at large.

Approach

We record a trace of an object-oriented program and display it in a novel interactive *animated 2.5D object map* visualization (fig. 2). The visualization comprises an object map and a timeline. The object map displays each object as a cuboid block with its name and fields and arranges object blocks based on their classes and mutual references and interactions. As the animation plays, colors and a curved trail indicate the activation of objects that execute methods, and users can track the control flow throughout the object graph. The timeline provides an overview of the execution time and the call tree. Through the 3D view, more details can be added to the visualization and users can take different perspectives on the object graph.

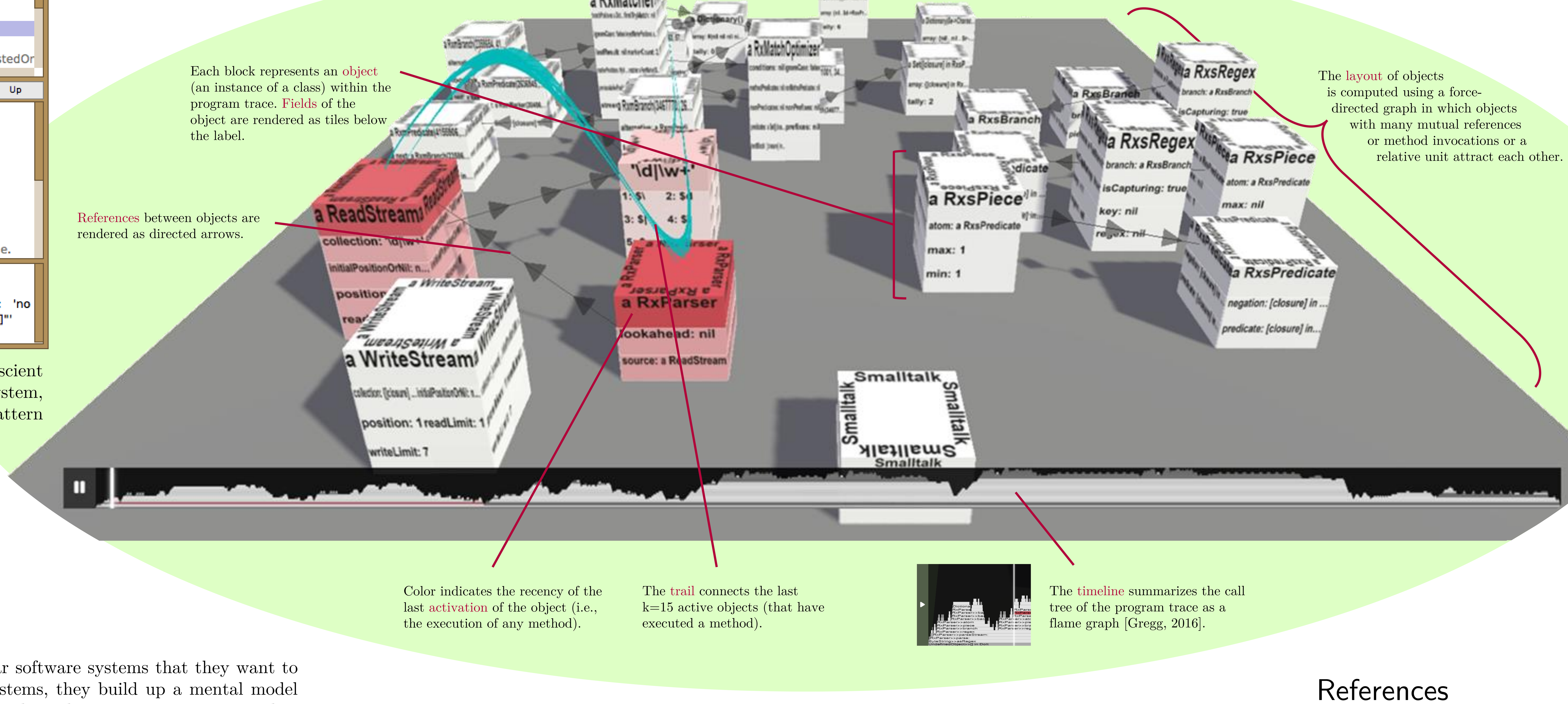

Figure 2: An animated object map in our TRACE4D prototype showing a program trace for the parsing of a regular expression in the Squeak/Smalltalk programming environment.

https://linqlover.github.io/trace4d/

Evaluation

We used our TRACE4D prototype to explore six different program traces from four different domains in the Squeak/Smalltalk programming system (tab. 1, fig. 3). Animated 2.5D object maps told us the most about program traces with a reasonably small number of relevant subsystems and objects as well as carefully designed systems that emphasize high class cohesion and extensive communication between related objects.

For suitable program traces, we could discover characteristic regions of the object graph (e.g., the input, the AST, and the NFA for the regular expression use case in fig. 1) as well as significant segments of program behavior (e.g., the parsing, compilation, and optimization stages). Based on these insights, we were able to develop and refine our mental model of the explored systems' functioning and link it to specific classes and objects in their implementation. Furthermore, the interactive visualization helped us to explore and analyze communication patterns, reflect on the system design, and share and discuss our mental models with other developers.

Program	Configuration effort	Clarity of objects	Object layout	Animation	Program comprehension
Regex *engine*					
• Construction	+	+	+	+	+
• Matching	+	+	+	∘	+
`Morphic` *UI framework*					
• Event handling	–	–	∘	∘	∘
• Layouting	∘	∘	+	∘	–
`Inspector` tool initialization	–	–	–	–	–
HTML parsing	∘	+	+	+	+

Table 1: Ratings of our experience with animated 2.5D object maps for program comprehension. We gained the most insights from smaller program traces that thoroughly model behavior through communication between objects and avoid many similar objects.

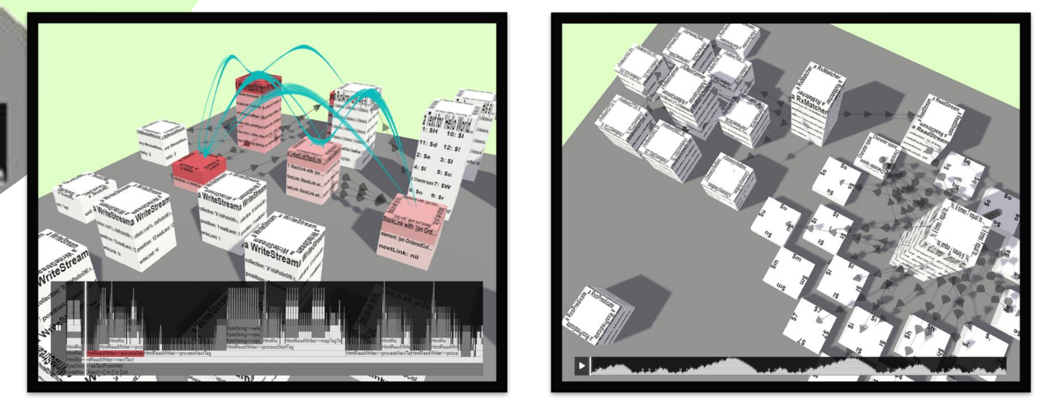

Figure 3: Animated object maps of an HTML parser (left) and a backtracking regular expression matcher (right).

References

Gregg, B. (2016). The flame graph. *Communications of the ACM*, 59(6):48–57.

Krause, A., Hansen, M., and Hasselbring, W. (2021). Live visualization of dynamic software cities with heat map overlays. In *Proc. VISSOFT*, pages 125–129. IEEE.

Limberger, D., Scheibel, W., Döllner, J., and Trapp, M. (2022). Visual variables and configuration of software maps. *Springer Journal of Visualization*, 26(1):249–274.

Pothier, G. and Tanter, É. (2009). Back to the future: Omniscient debugging. *IEEE Software*, 26(6):78–85.

Thiede, C., Taeumel, M., and Hirschfeld, R. (2023). Time-awareness in object exploration tools: Toward in situ omniscient debugging. In *Proc. SIGPLAN Onward!*, pp. 89–102. ACM.

Wettel, R. and Lanza, M. (2007). Visualizing software systems as cities. In *Proc. VISSOFT*, pages 92–99. IEEE.

Christoph Thiede
christoph.thiede@student.hpi.de
github.com/LinqLover

Willy Scheibel
willy.scheibel@hpi.de
hpi3d.de/people/current/scheibel.html

Jürgen Döllner
doellner@uni-potsdam.de
www.hpi3d.de

Computer Graphics Systems Group
Hasso Plattner Institute
Prof.-Dr.-Helmert-Str. 2–3
D-14482 Potsdam, Germany

Universität Potsdam

HPI Hasso Plattner Institut
Digital Engineering • Universität Potsdam