GTInspector: A Moldable Domain-Aware Object Inspector *

(Preprint †)

Andrei Chiş ‡ Oscar Nierstrasz
Aliaksei Syrel
Software Composition Group, University of Bern
Switzerland (scg.unibe.ch)

Tudor Gîrba
tudorgirba.com
Switzerland

Abstract

Understanding the run-time behaviour of object-oriented applications entails the comprehension of run-time objects. Traditional object inspectors favor generic views that focus on the low-level details of the state of single objects. While universally applicable, this generic approach does not take into account the varying needs of developers that could benefit from tailored views and exploration possibilities. *GTInspector* is a novel moldable object inspector that provides different high-level ways to visualize and explore objects, adapted to both the object and the current developer need. More information about the GTInspector can be found at: scg.unibe.ch/research/moldableinspector

Categories and Subject Descriptors D.2.6 [*Software Engineering*]: Programming Environments—integrated environments, interactive environments

Keywords Object inspector, Domain-specific tools, IDEs

1. Problem Description

Object-oriented programming makes use of objects to model application domains. There exists a wide diversity of objects from an even wider range of application domains, objects that depending on the actual task can interact in various ways. For example, objects like folders, files, parsers, HTML pages, remote resources, database connections, graphical components, *etc.*, a common presence in today's software systems, along with many more types of specialized objects.

Nevertheless, in spite of this diversity, object inspectors, an essential category of tools for comprehending run-time objects, display all objects, regardless of the domain or task, in a generic way. For example, when inspecting an object pointing to a remote resource or a graphical object, traditional inspectors show just object state, event if the developer wants to see, at that precise moment, the content of that resource or the visual representation of that graphical object. While mainstream IDEs (*e.g.*, Eclipse, Netbeans, VisualStudio) offer support for defining custom views, customization is usually limited to textual views (*e.g.*, tree or table) displaying object attributes. Traditional inspectors further focus on single objects, when it's rarely the case that a problem can be solved by just inspecting one single object in isolation.

2. GTInspector in a Nutshell

To address the aforementioned problems we propose GTInspector, an object inspector based on the Moldable Inspector model that enables developers to answer high-level, domain-specific questions by allowing them to adapt the whole inspection process to suit their immediate needs [2]. GTInspector builds on the following three operators:

Multiple Views Each object has multiple custom views; a view captures one representation of an object. For example, a file object has views that depending on the type of the file display its content in different ways (*i.e.*, a graphical view for a png file and a list view of files/folders for a zip archive – last two objects in Figure 1). Developers can easily add custom views for their own domain objects. At run time GTInspector selects views appropriate for the current objects and developer needs. As screen real estate is a scarce resource, in the current implementation, all selected views of an object are grouped using a tab widget; hence, for each object, only one view is displayed per tab. If the tab widget supports repositioning of tabs, multiple tabs can be made

* An extended version is published at Onward! 2015 [2]

† In Proceedings of the Companion Publication of the 2015 ACM SIGPLAN Conference on Systems, Programming, and Applications: Software for Humanity (SPLASH Companion 2015), October 25–30, 2015, Pittsburgh, PA, USA. DOI: 10.1145/2814189.2814194

‡ 3rd year PhD student, co-author of the GTInspector and member of the GT team (gt.moosetechnology.org), working towards development tools that can mold to both the development task and application domain (andreichis.com).

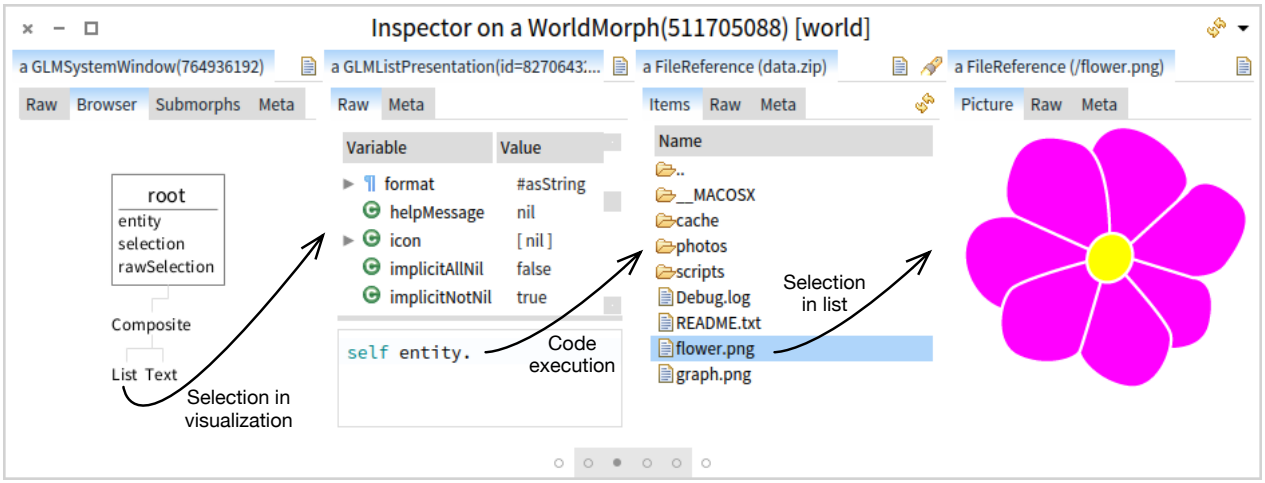

Figure 1: An inspection session consisting of six objects. Four objects are currently visible: (i) graphical window, (ii) model for a graphical component, (iii) file pointing to an archive, and (iv) file pointing to a png picture. From a dedicated view of an object in one tab of a Miller column, one can navigate to a view of another object in the next column. Navigating to each object involves a different mechanism: (a) selecting an object in a view, (b) executing code to locate the object, and (c) selecting an object in a list. The order in which these objects were inspected is given by their positioning from left to right.

available at the same time. The jGRASP IDE also allows developers to look at objects using specialized views [3]. However, objects are seen through unique views selected at run time based on the structure of an object. Self [4] allows each object to have a single unique representation through which users can also interact with the object.

Flexible Navigation Each view can specify a set of related objects, together with the mechanism for navigating to those objects. For example, a view showing the graphical representation of a widget can allow developers to navigate to any sub-widget by clicking it (navigating from the first to the second object in Figure 1). New objects can also be constructed/located using code snippets. GTInspector groups connected objects in an exploration session and displays them using an extension of Miller columns (`en.wikipedia.org/wiki/Miller_columns`) that provides an overview of the entire session, enables rapid navigation and makes it possible to control the number of visible objects. Code Bubbles also brings the idea of an exploration session to code understanding and debugging [1]. However, it relies on single representation for objects and requires developers to organize bubbles and manipulate a tree, rather than only a list. While this supports more elaborate scenarios it requires more spatial maintenance effort from developers.

Live Programming GTInspector features a simple extension mechanism where new views can be developed at run time using an internal DSL directly from within the inspector. Views are attached to objects using extension methods and located at run time using annotations. GTInspector is developed in Pharo (`pharo.org`), a modern Smalltalk dialect. We build, together with the developers of several applications, 131 views covering 84 distinct types of objects. Creating a view required on average of $9.2 \pm 6.6 (M \pm SD)$ lines of code. Combined with the ability of creating these views live directly from within the inspector, GTInspector provides a new workflow that makes custom inspection accessible. While developed in Pharo, there is no conceptual limitation that would impede an implementation in other object-oriented languages and IDEs.

3. Demonstration

In this tool demonstration we show how GTInspector improves the inspection process by applying the inspector to several use-cases and extending the inspector at run time with new views as new unanticipated problems arise. We focus on real-world use-cases such as interacting with the file system, exploring GUIs, inspecting results from a database query or mapping AST nodes to bytecode. Some of these features are usually addressed within IDEs using dedicated tools, without these tools being connected to the actual runtime objects. This leads to fragmented debugging activities as developers have to permanently look for these tools elsewhere and then bring the desired information back to the inspector/debugger. GTInspector removes this gap by providing the desired data right in the inspector at run time.

Acknowledgments

We gratefully acknowledge the financial support of the Swiss National Science Foundation for the project “Agile Software Assessment” (SNSF project Nr. 200020-144126/1, Jan 1, 2013 - Dec. 30, 2015).

References

[1] A. Bragdon, R. Zeleznik, S. P. Reiss, S. Karumuri, W. Cheung, J. Kaplan, C. Coleman, F. Adeputra, and J. J. LaViola, Jr. Code bubbles: a working set-based interface for code understanding and maintenance. In *CHI*, pages 2503–2512, 2010.

[2] A. Chiş, T. Gîrba, O. Nierstrasz, and A. Syrel. The Moldable Inspector. In *Onward!* to appear, 2015.

[3] J. H. Cross, II, T. D. Hendrix, D. A. Umphress, L. A. Barowski, J. Jain, and L. N. Montgomery. Robust generation of dynamic data structure visualizations with multiple interaction approaches. *Trans. Comput. Educ.*, 9(2):13:1–13:32, June 2009.

[4] R. B. Smith, J. Maloney, and D. Ungar. The self-4.0 user interface. In *OOPSLA*, pages 47–60, 1995.