Gaucho: Programming == Modeling

Fernando Olivero[1], Michele Lanza[1], Marco D'Ambros[1], and Romain Robbes[2]

[1] REVEAL @ Faculty of Informatics, University of Lugano, Switzerland
[2] PLEIAD @ DCC — University of Chile, Chile

Abstract. Rumor has it that programmers *write* software. Indeed, any modern integrated development environment (IDE) features numerous tools providing the means to write and browse textual representations of code artifacts. However, considering the complexity of current software systems, words other than writing are used, such as constructing, building, architecting, designing, etc. This "stepping away" from the notion that programming equals writing is supported by object-oriented languages, which promote reasoning around objects (mimicking real-world objects) sending each other messages.
Our goal is to depart from IDEs as fancy text editors, towards an environment that eases the interaction and the crafting of objects without creating a barrier between a developer and his creation. We present *Gaucho*, a direct manipulation environment which allows one to freely create, place, and interact with object-oriented concepts and abstractions in a consistent and unconstrained way.

Type of Demonstrated system: Open-source Research Tool

Audience/Track: Research Programme

Preferred Place of Demonstration: Classroom

1 Introduction

The vocabulary of modern object oriented programming developers has steadily evolved since its inception. Nowadays developers use words such as building, architecting, and designing [8], but in reality they still *write* programs as Weinberg [12] argued nearly 40 years ago. The current crop of IDEs provides numerous tools, but they are highly optimized for writing and browsing textual representations of code artifacts.

Do IDEs match the way programmers work? According to the study of Sillito et al. on developers habits [10], developers start development sessions by finding initial focus points, and then continue expanding those points by exploring relationships between the software artifacts forming interconnected graphs. Most IDEs comply with the former, allowing some form of concept location by means of a search widget. Regarding the latter, we argue that navigating the system is hampered by the text, list and tab-based nature of mainstream IDEs, termed by some as a "bento box[3]" philosophy.

For instance, relying on tab-based views depicting files of the system is inadequate since most tasks are not aligned with the structure of the IDE, and require navigating different parts of the system [6]. The visual constraints imposed by such views, make it difficult for developers to benefit from the use of secondary notations—such as persistent placement of visual objects [1,9]—that could come from a freer and customizable scheme for laying out the visual entities of the interface.

The tools included in most IDEs provide the means to effectively build and run programs. They are the intermediaries from which we manipulate objects, forcing them to be hidden in the user interface [2], failing to provide the proper level of abstraction, and increasing cognitive overhead when dealing with the artifacts of the system [11]. For example, to change the superclass of a class, or add methods, one must locate the proper source code fragment, and then edit the textual representation of the class definition, instead of performing changes with meaningful semantics.

[3] `http://en.wikipedia.org/wiki/Bento`

2 Gaucho

Gaucho is a direct manipulation environment; in Gaucho, the developer creates her own view of the system, by freely placing the software artifacts relevant to the current tasks. This allows her to focus on modeling the structure, behavior and collaborations between objects, instead of navigating and editing textual representations of dynamic software artifacts.

Gaucho's user interface is designed around a model-world metaphor [4] populated by directly manipulable representations of system elements. Figure 1 shows a screenshot of the user interface of Gaucho. Two pivotal concepts in Gaucho are the *pampas* and the *shapes*:

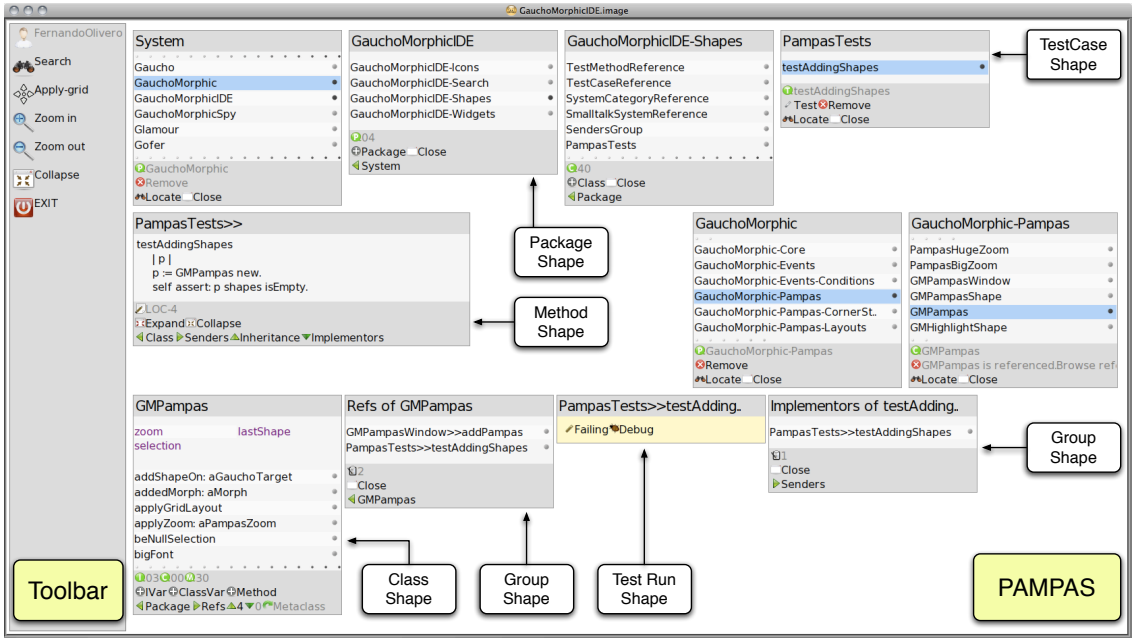

Fig. 1. The user interface of Gaucho

The pampas is a two-dimensional surface which hosts a software system's entities (e.g., packages, classes, methods, class references, recent changes, etc.) in the form of *shapes*. Shapes can be freely placed within the pampas, without the placement constraints imposed by the list-based nature of modern IDEs.

The shapes are the concrete visual entities that populate the pampas; there is one distinct shape representing each entity of the system. Shapes display every aspect of the entities being represented, and provide quick access—in the form of buttons—to the most important operations that can be performed on them. Shapes are always available to developers, ready for manipulation through an interaction based on a simple set of keyboard shortcuts and mouse commands.

Figure 1 shows several shapes: the class shape GMPampas presents the attributes and methods of the class, and provides buttons to add methods and attributes; the package shape GauchoMorphicIDE-Shapes shows the list of classes of the package, and provides a button to add classes.

The pampas and the shapes are the main assets in our quest for directness and abstraction, making Gaucho step away from the conversational metaphor in use by modern mainstream IDEs and move towards a model-world metaphor. Developers are empowered by directly manipulating the objects of interest, in a world reacting to their actions [4].

Locating concepts and behaviors. Gaucho provides a global search widget, located in a prominent location in the toolbar (see the top left corner of Figure 1). The search uses an auto-completed entry field that helps the developer to find the correct class, package or method that maps to a particular concept or behavior.

Exploring the system. Gaucho's shapes provide navigation operations, to explore the interconnected entities across the system; the actions are tailored to each kind of shape. For instance, class shapes present icons to open the group of class references, the class hierarchy, the group of subclasses and the package of the class (see the GMPampas shape in Figure 1). We chose to include navigation icons in every shape, because expanding initial focus points is a cornerstone operation that developers must undertake when performing development tasks [10]; direct tool support for rapid incremental exploration helps to manage context and supports program comprehension [11].

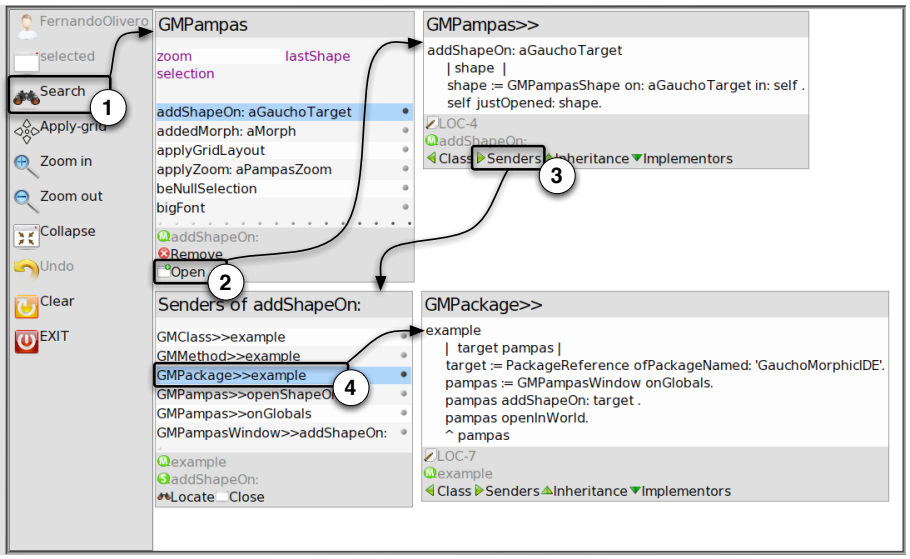

Fig. 2. Navigating through interconnected shapes

Gaucho in action. In Figure 2 we show a typical sequence of actions in Gaucho; the developer:

(1) uses the search widget in the toolbar to locate the GMPampas class;
(2) scrolls down the methods list and selects the method `addShapeOn:`, and presses the open button to open the method shape;
(3) presses the senders button of the method shape to open the senders group;
(4) selects the `example` method (of class GMPampas), pressing cmd-o to open a method shape on the selected method.

3 Tool information

Gaucho is written in Smalltalk [5], a highly dynamic and fully reflective language, which provides the ideal infrastructure for developing our next generation development environment [8].

Gaucho is implemented on top of Pharo[4], a modern open source IDE for Smalltalk, that includes a rich direct manipulation graphical interface framework called Morphic [7]. We enhanced Morphic by

[4] `http://pharo-project.org`

implementing a complete instrumentation of the Event-Command pattern [3], to keep track of every user interaction in the form of recorded events that trigger commands.

Gaucho is free, open source, and released under the MIT license. It can be downloaded from the Gaucho web site located at `gaucho.inf.usi.ch`, as a multi-platform one-click application to provide seamless installation. A screencast, demonstrating the main features of Gaucho, is available at `http://vimeo.com/17443946`

Acknowledgements. We gratefully acknowledge the financial support of the Swiss National Science Foundation (SNF Project No. 129496, "GSync").

A Demo Description

Our demonstration will consist in illustrating how Gaucho works by composing a small object-oriented system using Gaucho. As this is an improvised process, we do not provide a detailed description—which would probably not be describing what we will actually do—but prefer to provide a link to a movie which illustrates Gaucho in action: `http://vimeo.com/17443946`.

References

1. Blackwell, A.F., Whitley, K.N., Good, J., Petre, M.: Cognitive factors in programming with diagrams (2001)
2. Chang, B.W., Ungar, D., Smith, R.B.: Getting Close to Objects: Object-Focused Programming Environments. Prentice-Hall (1995)
3. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison Wesley (1995)
4. Hutchins, E., Hollan, J., Norman, D.: Direct manipulation interfaces. Human-Computer Interaction 1, 311–338 (1985)
5. Ingalls, D.H.: Design principles behind smalltalk. BYTE Magazine 6(8), 286–298 (1981)
6. Kersten, M., Murphy, G.C.: Mylar: a degree-of-interest model for ides. In: Proceedings of AOSD 2005 (4th international conference on Aspect-oriented software development). pp. 159–168. ACM (2005)
7. Maloney, J.H., Smith, R.B.: Directness and liveness in the morphic user interface construction environment. In: Proceedings of UIST 1995 (8th ACM symposium on User interface and software technology). pp. 21–28. ACM (1995)
8. Nierstrasz, O., Gîrba, T.: Lessons in software evolution learned by listening to smalltalk. In: Proceedings of SOFSEM 2010 (36th Conference on Current Trends in Theory and Practice of Computer Science). pp. 77–95. Springer (2010)
9. Petre, M.: Why looking isn't always seeing: Readership skills and graphical programming. Communications of the ACM 38, 33–44 (1995)
10. Sillito, J., Murphy, G.C., Volder, K.D.: Questions programmers ask during software evolution tasks. In: Proceedings of FSE-14 (14th International Symposium on Foundations of Software Engineering). pp. 23–34. ACM Press (2006)
11. Sinha, V., Karger, D., Miller, R.: Relo: Helping users manage context during interactive exploratory visualization of large codebases. In: Proceedings of OOPSLA workshop on Eclipse technology eXchange. pp. 21–25. ACM (2005)
12. Weinberg, G.: The Psychology of Computer Programming. Dorset House Publishing, silver anniversary edn. (1998)