Gaucho: From Integrated Development Environments to Direct Manipulation Environments

Fernando Olivero, Michele Lanza, Mircea Lungu
REVEAL @ Faculty Of Informatics - University Of Lugano, Switzerland

ABSTRACT

Object-oriented programming languages promote reasoning that revolves around objects that send each other messages. To practically make this happen, developers *write* programs, aided by integrated development environments (IDEs). A modern IDE offers a large number of tools that work on a textual representation of the program.

We argue that while modern IDEs –and their many tools– offer powerful means for manipulating source code, at the same time they introduce a barrier between the developer and the concepts under development because they still treat programs as files of text.

We present *Gaucho*, a lightweight development environment we are currently creating, which minimizes the presence of tools and allows the developer to directly manipulate objects. Drawing inspirations from both the Smalltalk and the SELF languages, our goal is to conceive a direct manipulation environment (DME) for software. We present the current status of our implementation, discuss its advantages and drawbacks, and delineate our next steps.

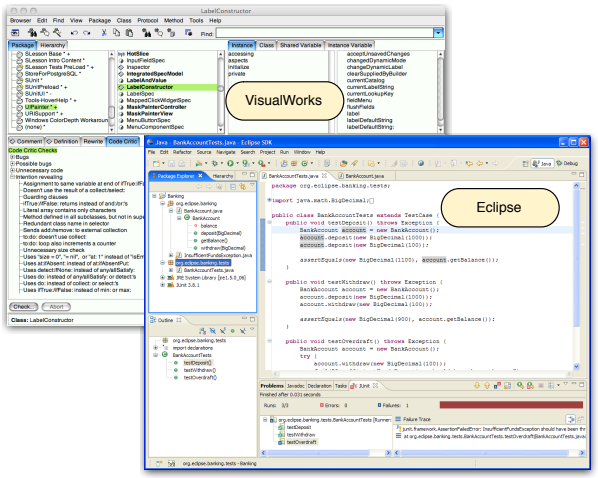

Figure 1: The VisualWorks and Eclipse IDEs

1. INTRODUCTION

Object-oriented programming languages promote objects as the fundamental building blocks of software systems. In class-based languages, programming revolves around using classes to define the relevant concepts of a system's domain, by providing a complete specification of their state (using attributes) and behavior (using methods).

Despite terms like software development, construction, composition, architecture, being present in the vocabulary of modern OOP developers[3], in reality programmers *write* programs as Weinberg [7] argued nearly 40 years ago, using integrated development environments (IDEs).

Modern IDEs (see Figure 1), such as Eclipse and VisualStudio include numerous tools that aid programmers to achieve all their development tasks. For example they provide advanced text editing facilities , a complete set of debuggers, refactoring engines, and a set of diverse browsers for navigating and accessing source code fragments for subsequent textual modifications.

We argue that the tool-centric stance of IDEs introduces a barrier between the developer and the concepts under development, because the objects are available only through the perspective of (admittedly powerful) tools, forcing the objects to be abstracted in the user interface.

More importantly, since IDEs grant access to source code fragments treated as text, programmers are led to believe that programming is writing, because the semantics of the modified language entities is lost when translated to the form of raw text. Even though the semantics are partly recovered, for example if a method name is selected the programmer can trigger a rename refactoring via a button or a menu, the language entity is still presented to the user as text.

In our work we want to remove the IDEs as intermediaries between the developer and the program, and liberate the individual programming concepts from the browser and become independent visual objects. Drawing inspiration from Smalltalk and SELF, two languages that embraced the *programming = modeling* philosophy[6], our goal is therefore to conceive a direct manipulation environment (DME) for software [1], which allows programmers to directly access and manipulate any entity using lightweight and intuitive depictions of the program entities. We named our DME *Gaucho*.

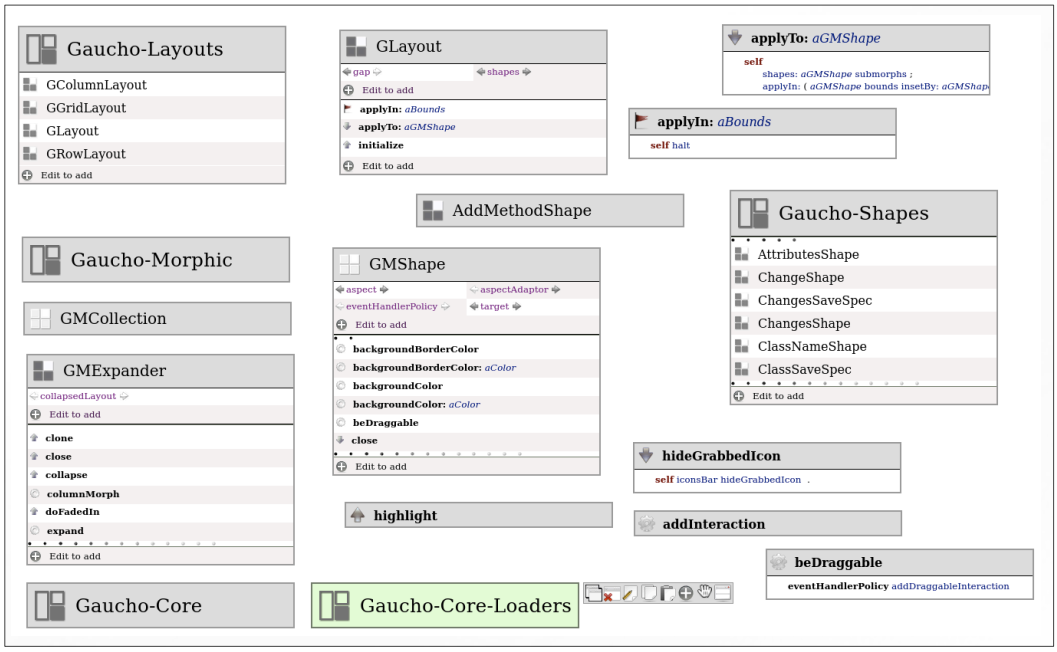

Figure 2: The Gaucho Direct Manipulation Environment

2. GAUCHO

2.1 Gaucho in a Nutshell

The Gaucho user interface is designed around a desktop metaphor populated by directly manipulable representations of system elements. We strive to present a modeless, customizable, persistent and simple experience to the developer using this new DME. Figure 2 depicts the Gaucho direct manipulation environment. Two of the key concepts in the architecture of Gaucho are the *pampas* and *shapes*:

The Pampas is a two-dimensional surface, which hosts the visual objects that represent entities that make up a software system (e.g., packages, classes, methods, developers, development sessions, recent changes, etc.) in the form of *shapes*. The pampas is where all programming and system management tasks are performed. Every visual element can be freely placed on the pampas according to the preferences of the programmer - thus departing from the list-based nature of mainstream IDEs.

The Shapes. Shapes are the concrete visual entities that populate the pampas. They are directly manipulable and provide higher-level interactions than the text based browsers. They visually represent system components such as packages, classes, methods, developers, and workspaces. Shapes bridge the gap between the programmer and the objects, removing the intermediaries (tools). Shapes embrace the concepts of liveness and availability [2], because they are composed of editable visual components, that can be expanded, edited, or deleted by using a simple set of keyboard and mouse commands. Shapes are always available to the programmer for interaction and modification. All shapes are denoted by distinctive icons, based on a consistent iconic language, to help developers to quickly grasp the type and status of any particular shape.

2.2 Detailing Gaucho

Gaucho is a term used to describe the "cowboys" of the pampas, the vast South American grasslands. We use the *gauchos* metaphor to denote the freedom that developers have with Gaucho: they can place elements arbitrarily and can directly manipulate them. Gaucho supports a uniform set of interactions, regardless of the type of visual elements in focus and the presented level of detail, such as:

- Creating a new workspace, package, class or method is achieved by pressing Cmd-N when the corresponding shape has focus.
- Every text presented in a shape is editable by simply pressing enter. For example renaming an method does not require bringing up context menus from which to trigger the refactoring like in modern IDEs- editing the name in Gaucho is equivalent to renaming it.
- Every change is recorded and can be undone.
- Expanding and contracting elements. Each shape can be contracted (only showing the name), expanded (revealing more information, such as the contained elements), or opened (becoming a pampas itself).

Gaucho is a zooming interface [4]: opening a shape is presented to the developer in the form of a "dive" action, where the package pampas takes up the whole screen. The pampas is thus a recursive concept, i.e., a developer pampas can have several workspace shapes, which, when opened, are a pampas themselves, which contain package, class, and method shapes. Each package and class shape can be opened to reveal yet another pampas (i.e., a package pampas contains class and method shapes; a class pampas contains method shapes).

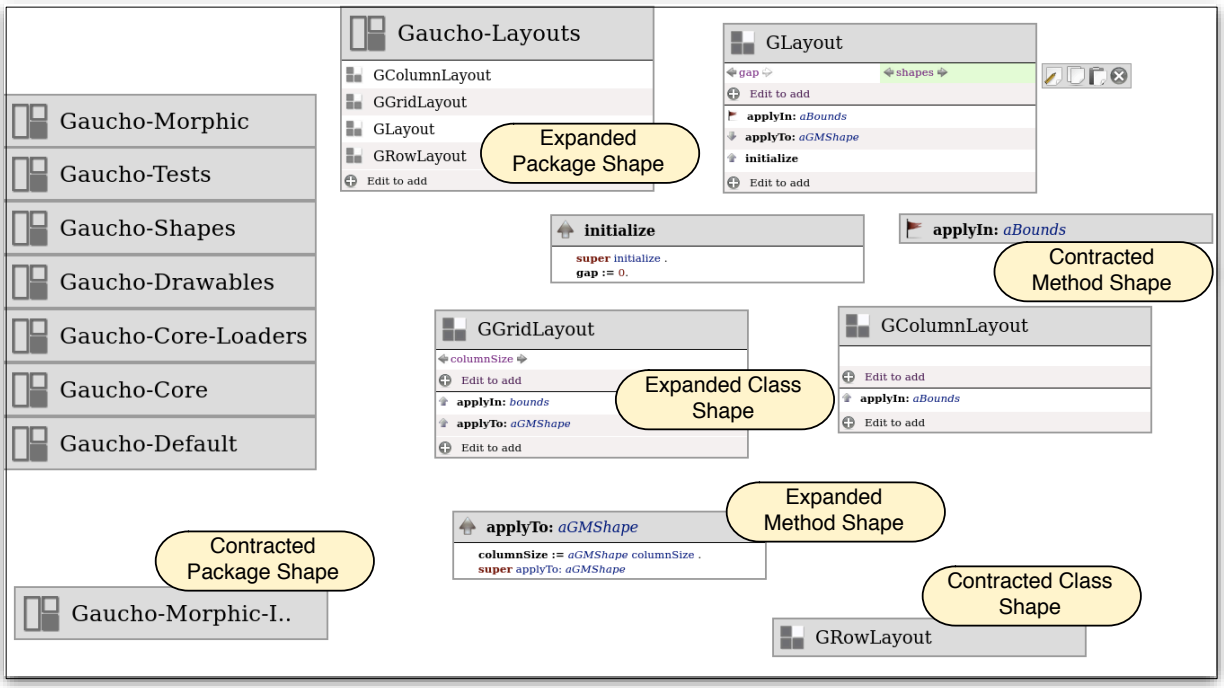

Figure 3: Workspace Pampas containing various shapes, both in expanded and contracted form.

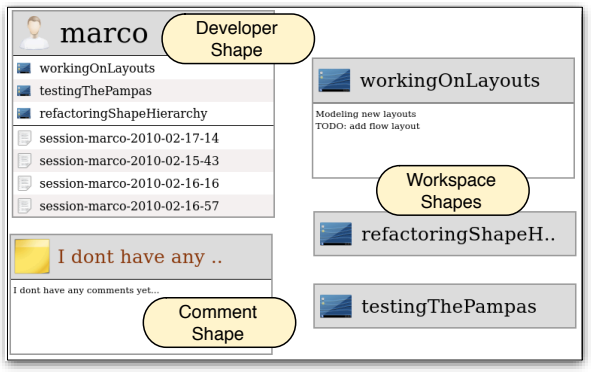

Figure 4: The Developer Pampas of Marco

The Developer Pampas. The Developer Pampas is where developers manage their workspaces, and choose the next task to work on. It is populated by one or more developer shapes, comment shapes (which allow to leave notes in the spirit of a bulletin board) and workspaces shapes. A developer shape is created and managed by interacting with the System pampas, this kind of shape presents the name, the workspaces and the sessions of a developer. Workspaces and sessions and are integral part of Gaucho:

A *workspace* is a perspective of the system composed of package, class, and method shapes. Workspaces allow programmers to group language components into named working sets, according to each development task they are currently undertaking. Workspace shapes have editable names and comments. The programmer can choose to commence working within a workspace by opening a workspace shape to reveal the workspace pampas.

A *session* tracks the work of a developer during a development task. Sessions remember every change made to the system, similar to Robbes' Spyware tool[5].

In Figure 4 we see the Developer Pampas of the developer named Marco, who currently owns three workspaces, and has started two sessions of work.

Workspace Pampas. The workspace pampas is where the actual development work is done. It contains package, class, and method shapes. Each shape can be freely placed to correspond to the developer's mental model. For instance, a developer may want to place the class shapes according to the model-view-controller pattern.

Figure 3 presents a workspace pampas containing package, class, and method shapes. The figure presents a number of contracted package shapes. They can be expanded to reveal more details, or opened to form their own pampas. We see an expanded package shape named *Gaucho-Layouts*, which presents a list of the classes contained in that package.

Class Shapes. Figure 3 presents several class shapes: *GRowLayout* is contracted; *GGridLayout* is expanded to show all the attributes and methods that it contains. New attributes and methods can be added and renamed in place. For the attributes the developer can define and access the getter and setters directly in the attribute list. The developer can access and edit a method's source code with a keyboard shortcut, which pops up an expanded method shape containing the source code. The source code in the expanded method shape is fully reified.

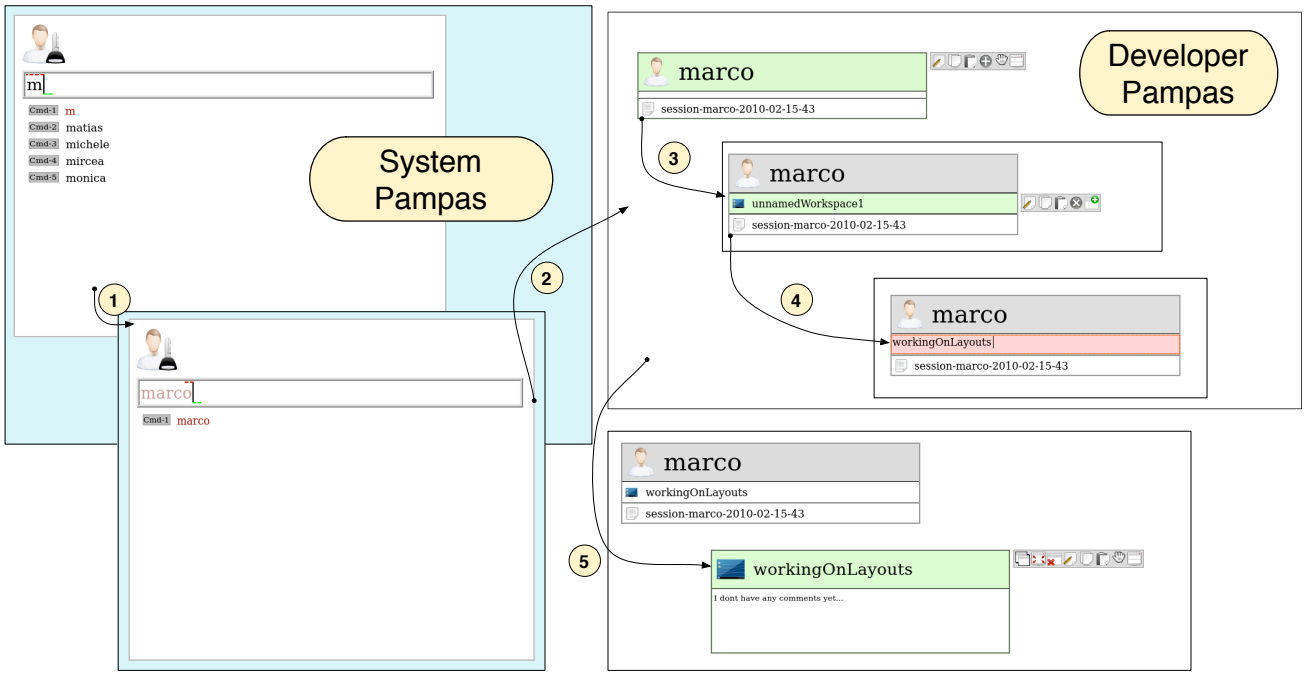

Figure 6: Gaucho interaction: Creating a workspace

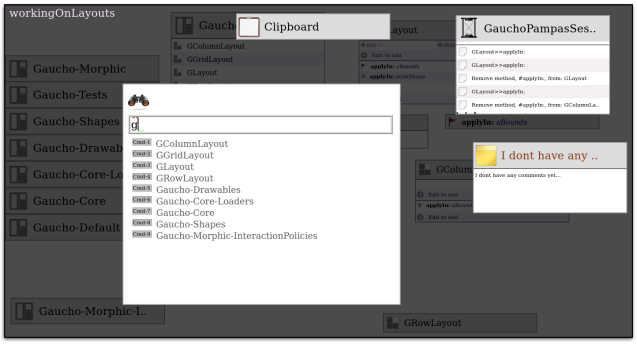

Figure 5: A Pampas dashboard

The Dashboard. Each pampas features an "augmented reality" tool called the Dashboard, accessible by pressing a special keyboard shortcut.

The dashboard (see Figure 5) is a transparent layer on top of the Pampas, populated by widgets that allow the programmer to perform tasks which are indirectly related to programming. The dashboard includes widgets for searching by name any shape in the pampas, to view the changes made to the system during the current session, and to view the clipboard contents.

Implementation. Gaucho is written in Pharo Smalltalk[1], a modern open source IDE for the Smalltalk language.

[1] `http://www.pharo-project.org/`

2.3 Gaucho In Action

In this section we present two examples of usage of Gaucho, the first example describes a developer and a workspace creation, and the second describes how a programming task is achieved interacting with the shapes of the environment.

2.3.1 Creating a Developer

The first example (see Figure 6) enumerates the steps that a programmer must follow to create a new developer called *marco*, and a new workspace called *working on layouts*.

1. Create Developer: type marco into the login widget (after every pressed character will list the the developer names that match the typed prefix).

2. Create and Open Developer Pampas: Press Cmd-1. The widget will present the conjunct option of creating and opening the new developer, conveying this information by displaying the name marco in red and associating a keyboard shortcut for opening a Developer Pampas on marco.

3. Create a workspace: Select the developer shape of the opened Pampas, and press Cmd-N .

4. Rename a workspace: Press enter in the only list entry, then type the new name in the list entry editor, and accept the contents by pressing enter once again.

After following the described sequence of steps the system includes a developer named marco, who owns a workspace called workingOnLayouts.

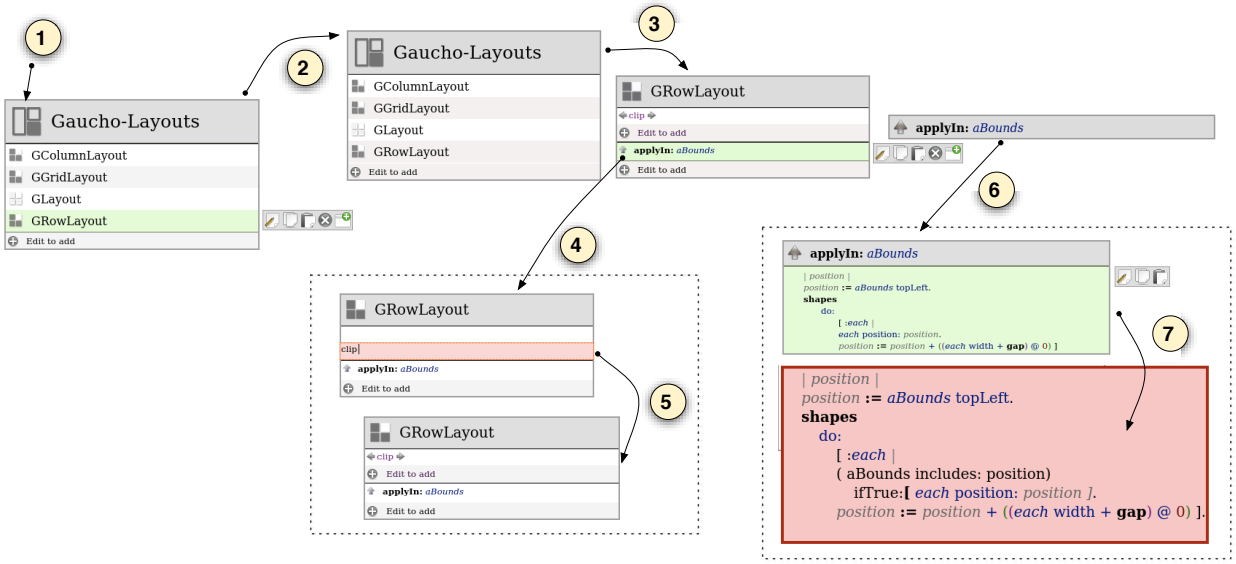

Figure 7: Gaucho interaction: Programming

2.3.2 *Editing a Class*

The second example (see Figure 7) describes how to add an attribute to a class, and how to modify the statements of a method. We will modify Gaucho from within the Gaucho environment, adding an attribute called **clip** to the Gaucho layout class called *GRowLayout* , and modify the method *applyIn: aBounds*.

1. Locate the package shape of Gaucho-Layouts in the Working On Layouts Pampas.
2. Open a class shape: select the list entry named GRowLayout, and press the right arrow.
3. Open a method shape: select the list entry named applyIn: aBounds, and press the right arrow.

4, 5. Add an attribute: edit the add attribute list item, type clip and accept the contents with enter. The class shape is updated, to reflect the attribute addition.

6. Expand a method shape: select the method shape and click.
7. Edit the method: select the code list item and press enter to open the code editor, type the modifications and accept contents pressing enter.

3. CONCLUSIONS

We presented Gaucho, a next generation IDE that provide means for experiencing development tasks as modeling, dealing with higher abstractions levels in the user interface, than files or source code fragments treated as text.

Gaucho uses an alternative desktop metaphor, departing from the tool-centric stance of modern IDEs, in order to bridge the gap between developers and the objects under construction, and more importantly to provide the building blocks for moving towards an environment that fully supports modeling and software composition as integral part of the software development task.

We are still in the initial stages of our research endeavours, have however already bridged what we call the *development singularity*: we have made Gaucho powerful enough to stop using the standard tools offered by Pharo to develop Gaucho, and are now developing Gaucho with Gaucho itself. We believe this is crucial for our future work.

Acknowledgments. We gratefully acknowledge the financial support of the Swiss National Science foundation for the project “GSync” (SNF Project No. 129496).

4. REFERENCES

[1] E. L. Hutchins, J. D. Hollan, and D. A. Norman. Direct manipulation interfaces. *Hum.-Comput. Interact.*, 1(4):311–338, 1985.

[2] J. H. Maloney and R. B. Smith. Directness and liveness in the morphic user interface construction environment. In *UIST ’95: Proceedings of the 8th annual ACM symposium on User interface and software technology*, pages 21–28, New York, NY, USA, 1995. ACM.

[3] O. Nierstrasz and T. GŤrba. *Lessons in Software Evolution Learned by Listening to Smalltalk*. Springer Berlin / Heidelberg, 2010.

[4] J. Raskin. *The humane interface: new directions for designing interactive systems*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 2000.

[5] R. Robbes and M. Lanza. Spyware: A change-aware development toolset. In *Proceedings of ICSE 2008 (30th ACM/IEEE International Conference in Software Engineering)*, pages 847–850. ACM Press, 2008.

[6] D. P. B. Smalltalk. Design principles behind smalltalk. *BYTE Magazine*, August 1981.

[7] G. Weinberg. *The Psychology of Computer Programming*. Dorset House Publishing, silver anniversary edition, 1998.