A graphical user interface for the Kenzo system, a program to compute in Algebraic Topology*

Jónathan Heras Vico Pascual Julio Rubio

Abstract

Kenzo is a symbolic computation system devoted to Algebraic Topology. It has been developed by F. Sergeraert mainly as a research tool. The challenge is now to increase the number of users and to improve its usability. The first task we have undertaken is to design simply a friendly front-end, and after that we have focussed on devising a complete framework which wraps the *pure* Common Lisp Kenzo system and makes the use of Kenzo easier to different clients.

Introduction

Kenzo [5] is a Common Lisp system, devoted to Symbolic Computation in Algebraic Topology. It was developed in 1997 under the direction of F. Sergeraert, and has been successful, in the sense that it has been capable of computing homology groups unreachable by any other means. Let us present a small sample of a session with the Kenzo system. The first command constructs the sphere of dimension 4, the second one constructs the loop space iterated two times of that sphere and the third one computes its fourth homology group.

```
> (sphere 4)
[K1 Simplicial-Set]
> (loop-space (sphere 4) 2)
[K18 Simplicial-Group]
> (homology (loop-space (sphere 4) 2) 4)
Component Z
-- done --
```

It can be seen that the original Kenzo user interface is Common Lisp itself, so the accessibility and usability are two weak points in Kenzo (implying difficulties in increasing the number of users of the system). With the aim of extending the use of the system, several lines have been explored (see [1] and [4], for instance) . In this work we have focussed on devising a graphical user interface for the system, using Common Lisp as implementation language.

*Partially supported by Comunidad Autónoma de La Rioja, project Colabora2007/16, and Ministerio de Educación y Ciencia, project MTM2006-06513.

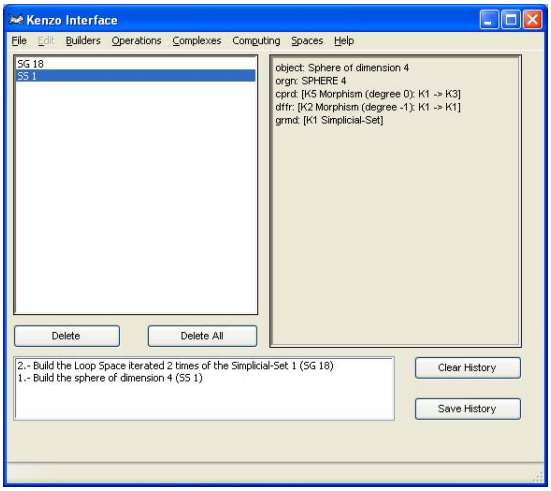

Figure 1: Screen-shot of Kenzo Interface.

1 A graphical user interface for Kenzo

As we have just commented, with the aim of increasing the number of users of Kenzo, we have developed a graphical user interface (GUI). A user request is transmitted to Kenzo from our GUI and when Kenzo finishes the calculation the result is returned to the GUI. Our GUI communicates with Kenzo itself by means of an XML format (it is well-known that XML gives us a universal tool to transmit information along the different components of a system). This XML format is based on XML-RPC [6].

In Figure 1, a screen-shot of our GUI is presented. The main toolbar is organized into 8 menus: *File*, *Edit*, *Builders*, *Operations*, *Complexes*, *Computing*, *Spaces* and *Help*. The rest of the screen is separated into three areas. On the left side, a list with the spaces already constructed during the current session is maintained. When a space is selected (the one denoted by `SS 1` in Figure 1), a description of it is displayed in the right area. At the bottom of the screen, one finds a *history* description of the current session, which can be cleared or saved into a file.

In the current version the *File* menu has just three options: *Exit*, *Save Session* and *Load Session*. When saving a session, a file is produced containing an XML description of the commands executed by the user in that session.

The constructors of spaces are collected by the menus *Builders, Operations* and *Complexes*. More specifically, the menu *Builders* includes the main ways of constructing new spaces from scratch in Kenzo as options: spheres, Moore spaces, Eilenberg-Mac Lane spaces, and so on. The menu *Operations* refers to the ways where Kenzo allows the construction of new simplicial spaces from other ones: loop spaces, classifying spaces, Cartesian products, suspensions, etc. The menu *Complexes* is similar, but related to chain complexes instead of simplicial objects.

The menus *Computing* and *Spaces* collect all the operations on concrete spaces (instead

of *constructing spaces*, as in the previous cases). In *Computing* we concentrate on calculations *over* a space. We offer to compute homology groups, to compute the same but with explicit generators and to compute homotopy groups. In menu *Spaces* currently we only offer the possibility of showing the structure of a simplicial object (this is only applicable to *effective*, finite type spaces).

As well as making the use of Kenzo easier, our GUI adds some extra functionalities. With respect to the constructors of spaces, remark that the control of their input specifications is naturally achieved in the GUI client. The XML format used provides the GUI with the mathematical knowledge needed for it. This is an enhancement of Kenzo, since in the kernel system a user could apply a constructor to an object without satisfying its input specification.

A second enhancement is related to minimizing the number of runtime errors. In order to do this, more intricate mathematical knowledge is needed. In this sense, with respect to the operations on concrete spaces, the items of the menus *Computing* and *Spaces* need more expert mathematical knowledge in order to avoid raising runtime errors (for instance, a user could ask Kenzo for a homology group which is not of finite type, producing a runtime error). This knowledge is provided by an intermediary component which has been introduced between Kenzo and the GUI. This component is communicated with Kenzo (via XML-RPC) and with the GUI by means of another more abstract XML format which is an extension of MathML [2].

Besides, the GUI provides the user with new tools (not directly available in Kenzo) by chaining some methods. For example, in Kenzo there is no final function allowing the user to compute homotopy groups. Instead, there is a number of complex algorithms, allowing a user to chain them to get some homotopy groups. The menu *Computing* has an option to compute homotopy groups. The intermediary component is in charge of chaining the different algorithms present in Kenzo to reach the final objective.

2 A framework wrapping Kenzo

Gathering all the components together, we obtain a software system whose architecture corresponds to a simplified version of a well-known architectural pattern, the *Microkernel* pattern [3]. (This pattern gives a global view as a *platform*, in terminology of [3], which implements a virtual machine with applications running on top of it, namely a *framework*, in the same terminology.)

Even more, inspired by this pattern, we are trying to wrap Kenzo in a framework in such a way that a global view of the system becomes a platform with different clients (other GUIs, web applications, web services, ...). A high level perspective of the system as a whole is shown in Figure 2. Kenzo itself, wrapped with an interface based on XML-RPC, is acting as *internal server*. The *microkernel* acting as intermediary component is based on an XML processor, allowing both a link with the standard XML-RPC, and the expert knowledge management. The view of the *external server* is again based on an XML processor, with a higher level of abstraction (since it is free of the implementation details related to Kenzo)

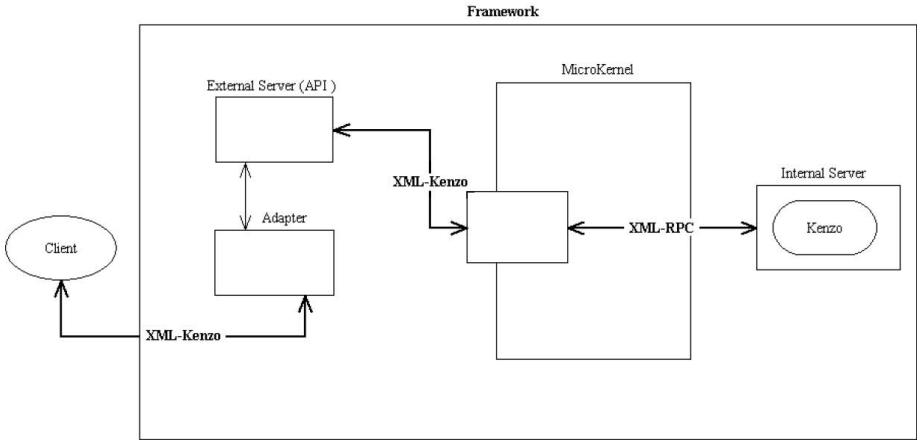

Figure 2: Microkernel architecture of the system.

which can map expressions from and to the microkernel, and which is decorated with an *adapter* (the *Proxy* pattern, [3], is used to implement the adapter), establishing the final connection with the client, a Graphical User Interface in our case. In this sense, our GUI is a client of our framework.

References

[1] Andrés M., Pascual V., Romero A., Rubio J., *Remote Access to a Symbolic Computation System for Algebraic Topology: A Client-Server Approach*, Lecture Notes in Computer Science 3516 (2005) 635–642.

[2] Ausbrooks R. et al., *Mathematical Markup Language (MathML)* Version 2.0 (second edition), 2003. http://www.w3.org/TR/2003/REC-MathML2-20031021/.

[3] Buschmann, F., Meunier, R., Rohnert H., Sommerland P., Stal M., *Pattern-oriented software architecture. A system of patterns, Volume 1*, Wiley, 1996.

[4] Coquand T., Spiwack A., *Towards Constructive Homological Algebra in Type Theory*, Lecture Notes in Artificial Intelligence 4573 (2007) 40–54.

[5] Dousson X., Sergeraert F., Siret Y., *The Kenzo program*, Institut Fourier, Grenoble, 1999. http://www-fourier.ujf-grenoble.fr/~sergerar/Kenzo/.

[6] Winer D., *Extensible Markup Language-Remote Procedure Call (XML-RPC).* http://www.xmlrpc.com.

Jónathan Heras, Vico Pascual, Julio Rubio
Departamento de Matemáticas y Computación, Universidad de La Rioja,
Edificio Vives, Luis de Ulloa s/n, E-26004 Logroño (La Rioja, Spain).
{jonathan.heras, vico.pascual, julio.rubio}@unirioja.es