A customizable GUI through an OMDoc documents repository*

Jónathan Heras, Vico Pascual, and 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

Abstract. Kenzo is a Symbolic Computation System devoted to Algebraic Topology. In some previous works we presented a framework wrapping Kenzo providing a mediated access to Kenzo making its use easier. In this work, a particular client of this framework is presented, namely a Graphical User Interface. By means of an OMDoc documents repository, this Graphical User Interface is totally customizable. Besides, it allows Kenzo to interoperate with other systems, namely with the ACL2 Theorem Prover. In this way, representation, computation and deduction are integrated in a same system.

1 Introduction

Kenzo [2] 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. Having detected the accessibility and usability as two weak points in Kenzo (implying difficulties in increasing the number of users of the system), several proposals have been studied to interoperate with Kenzo (since the original user interface is Common Lisp itself, the search for other ways of interaction seems mandatory to extend the use of the system).

In this work, a customizable Graphical User Interface (GUI) for Kenzo is introduced. This GUI is gathering in a single system *representation*, *computation* and *deduction*.

2 Antecedents

In [6] we introduced a framework wrapping Kenzo whose architecture is based on the *MicroKernel* pattern [3]. The MicroKernel design consists of two layers. The first one manages the input of requests and the output of results, converting OpenMath objects from/to the internal representation in the system by means of

* Partially supported by Universidad de La Rioja, project API08/08, and Ministerio de Educación y Ciencia, project MTM2006-06513.

a *Phrasebook*. The second one deals with the validation of requests. For instance, among other issues, the validation layer is in charge of controlling the input specifications of the constructors. Besides, it validates that the request, an XML data, is well-formed and it makes sure that the computations asked for Kenzo are mathematically correct, avoiding in this way some possible execution errors. This layer was called *Intelligent System* in [6], where more details about the functionality of this layer were described. A layer wrapping *Kenzo* executes the computations and returns the results to the input/output layer. A simplified diagram of this architecture appears in Figure 1.

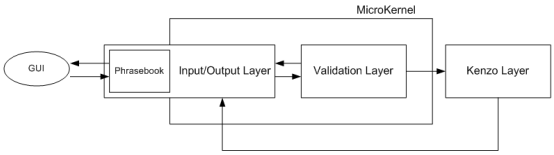

Fig. 1. Simplified diagram of the architecture.

Based on Figure 1, we can also see the execution flow of a calculation in our system. When the user asks for a computation the Phrasebook converts the OpenMath Object into the internal representation of the system. Then the intelligent system is in charge of validating the request. Finally, if the request is right, the Kenzo layer computes it and returns the result to the Phrasebook which converts it into an OpenMath object. In other case, that is, the request is not right, the Intelligent System informs to the Input/Output module the reasons why it is not possible to carry out the calculation asked for.

Moreover, in order to make the interaction with this framework easier, both a distinguished client, to be precise a Graphical User Interface (GUI) together an OMDoc documents repository (see [7]) have been developed. This repository provides different kinds of documents. These documents can be loaded as new modules and allow us to customize dynamically the GUI.

3 From an empty GUI to a meaningful GUI

We have based on [5] to define our GUI, where a proposal for the declarative programming of user interfaces with the aim of abstracting the ingredients for high-level UI programming was presented. To be precise, three constituents are distinguished: *structure*, *functionality* and *layout*.

The structure of our GUI is provided by XUL (XML User Interface [9], it is Mozilla's XML-based user interface language that lets us build feature rich cross-platform applications defining all the elements of a UI), functionality has been programmed in Allegro Common Lisp and the default layout has been used, although we could have used a style sheet to customize our application. Thus, we have all the ingredients to extend our empty GUI. A video demonstration of our application can be downloaded from `http://www.unirioja.es/cu/joheras/mathui-kenzo-interface.zip`.

In [7] an OMDoc documents repository to customize our GUI was presented. OMDoc [11] format is an open markup language for mathematical documents and the knowledge encapsulated in them. We have used different OMDoc sublanguages and modules in order to specify the different ingredients needed to customize the GUI. This repository contains OMDoc documents with different aims. Some of them are devoted to give an algebraic specification of the different mathematical structures which Kenzo works with. Others supply the functions to build these mathematical structures in our system (abstracting the ones of Kenzo), and, in this way, they make up a Kenzo wrapper. With respect to the GUI, some OMDoc documents contain the graphical elements and others contain their event handlers. And finally, on the one hand, the necessary code to interact with other systems is embedded in other OMDoc documents considered as interpreters from Kenzo to the specific system. On the other hand, the interface of the interaction with other systems has been defined in other OMDoc documents.

The first time the application is loaded, the main toolbar is organized into two menus: *File* and *Help*. The user can configure the interface using the OMDoc Repository. When the user exits the application, its configuration is saved for future sessions.

In this empty GUI, the *File* menu has the following options: *Add Module*, *Delete Module*, and *Exit*. Each module is associated to a kind of mathematical structures Kenzo works with. The goal of the Add Module option consists in loading the functionality related to that kind of mathematical structure.

So, the first user task consists in selecting the necessary modules in order to add to the GUI the required functionality. Each module has got associated one OMDoc document of the OMDoc Repository, which is included in the folder of the GUI and which links all the OMDoc documents needed. In an analogous way, the modules can be unloaded from the *Delete Module* option.

As can be seen in the GUI, it also has three tabs: *Main*, *Session* and *Computing* that are useful when a module is loaded.

In the following subsections how representation, deduction and calculation are integrated in a same system is explained.

3.1 Representation in the system

The Kenzo system encodes its objects by means of instances of CLOS (Common Lisp Object System) classes. As this internal representation can not be exported to other systems, we have provided Kenzo objects with an external representation, namely an OpenMath representation [1]. To this task several OMDoc Content Dictionaries, containing the semantics of the mathematical structures used, have been developed. In addition, a stylesheet has been provided to render the OpenMath Objects using mathematical notation in the system.

For instance, we are going to consider the next scenario. Firstly, we load the spaces the system works with, that are *Chain Complexes*, *Simplicial Sets*, *Simplicial Groups* and *Abelian Simplicial Groups*. This adds to the main toolbar

new menus, one for each kind of spaces (see Figure 2), and to the *File* menu two new options.

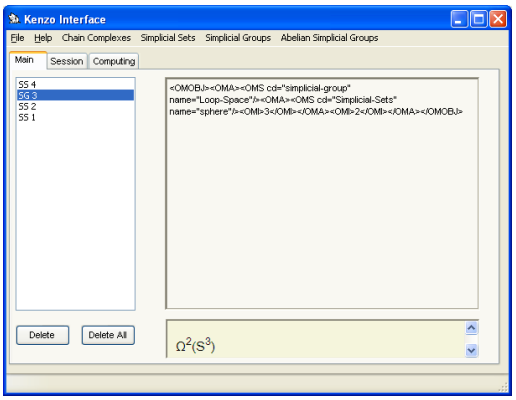

Fig. 2. The *Main* tab with an example of session.

Each option allows us to build spaces of its associated classes in two different ways. The first one from scratch (for instance, in the case of Simplicial Sets is possible to build spheres, Moore spaces and so on). The second one consists in constructing spaces from other ones (again in the Simplicial Sets, it is possible to build cartesian products, suspensions, and so on).

The *Main* tab 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 `SG 3` in Figure 2), its description is displayed in the right area using OpenMath, and just below the usual mathematical notation of the space can be seen.

When the *Session* tab is focused, a similar screen to Figure 3 is shown. The objects built during the current session, in an ordered way, are rendered. Note that the calculations are not rendered in this tab.

The options added to the *File* menu, *Save Session* and *Load Session*, are related to a particular session. When saving a session a file is produced containing the spaces built in that session. These session files are saved using the OpenMath format following the rules defined in the corresponding OMDoc Content Dictionaries and can be rendered in different browsers. These session files can be loaded, from the *Load Session* option, and allow to reproduce the saved sessions.

In addition, the control of the input specifications on the constructors of spaces, included in our framework in [6] as an "intelligent enhancement" with respect to the Kenzo system, is exploited by our GUI as client of our framework.

3.2 Computation through the Kenzo kernel in the system

Up to now, we have not explained how our system uses Kenzo to carry out computations. To this task, we must load the *Computing* OMDoc module and

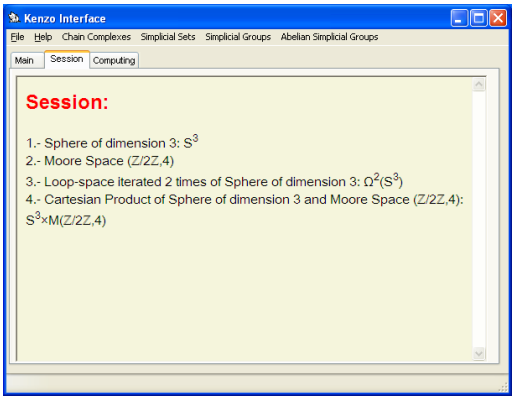

Fig. 3. The *Session* tab with an example of session.

so a new menu with the options to compute homology and homotopy groups and a new option, *Save Computing*, in the *File* menu are added to the GUI.

At this moment our system allows the user to compute homology and homotopy groups, the last option is not included in a direct way in the Kenzo system but it includes all the necessary machinery to compute some homotopy groups. The Kenzo Layer chains methods in order to provide the user with the computation of some homotopy groups, more details in [6].

The computations in the current session are displayed into the *Computing* tab, as can be seen in Figure 4.

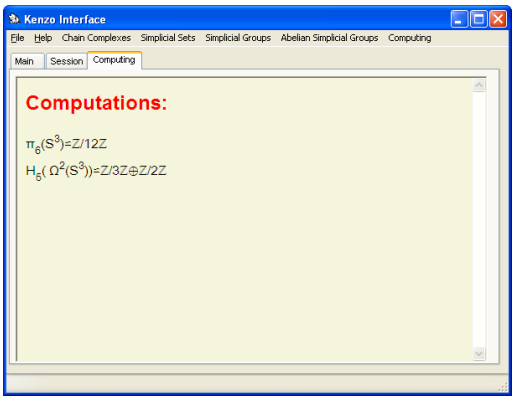

Fig. 4. The *Computing* tab with some computations.

The option *Save Computing* of the *File* menu works in a similar way to *Save Session* but instead of saving the spaces built during the current session, it saves the computations also using OpenMath format.

3.3 Deduction by means of ACL2 in the system

Finally, we have also tackled the task of interoperating with other systems. To be precise, we have focused on the interaction with the ACL2 Theorem Prover [10]. As we said in the Introduction, some Kenzo computations are unreachable by any other means so we are interested in integrating Kenzo with Theorem Provers in order to increase the reliability of the Kenzo system.

In this case, the needed OMDoc documents to customize the GUI by adding both a new tab page and a new menu to interoperate with ACL2, as can be seen in Figure 5, and the interpreter which is able to translate from the OMDoc content dictionary into an ACL2 encapsulate (an ACL2 tool to introduce new function symbols), have been developed. The new tab page contains two areas and one button: the ACL2 instructions will be written in the first area, the button will send the instructions to ACL2, and finally the second area will show the ACL2 result. In addition, a new menu in charge of converting from OMDoc Content Dictionaries to encapsulates has been added (based on the interpreter). Note that the encapsulate corresponding to simplicial sets is written (dynamically, from the corresponding content dictionary) into the left area and the ACL2 answer after evaluating it, appears in the right zone.

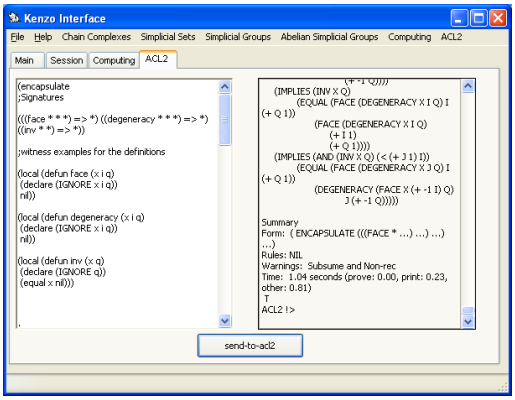

Fig. 5. The *ACL2* tab with a Simplicial Sets encapsulate.

3.4 Gathering all the pieces

Consider the following scenario where we want to customize our GUI in order to work with simplicial sets, including the interaction with the ACL2 system. For simplicial sets, an OMDoc content dictionary defining their mathematical structure (`SS-definition`), the logic to interact with Kenzo (`SS-Kenzo-functionality`) and the presentation for the GUI (`SS-GUI`) are available in separate OMDoc documents. With respect to ACL2, an interpreter (`ACL2-interpreter`) which is able to translate from an OMDoc content dictionary (in particular, simplicial sets content dictionary) into an ACL2 encapsulate

can be found. An OMDoc document to customize the GUI (`ACL2-GUI`) allowing the ACL2 system to interact with our system has been developed. The relations among the components and their role in the workflow of our system customization in this example is shown in Figure 6.

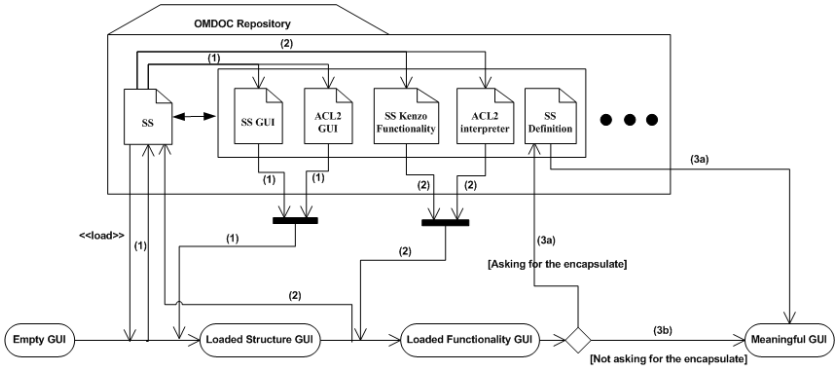

Fig. 6. Workflow diagram.

4 Conclusions

In this paper we have reported on a customizable GUI by means of an OMDoc documents repository for a framework wrapping the Kenzo system. This GUI, together with our framework, not only provides a friendly front-end but also a *mediated access* to Kenzo and the interaction with other systems, like the ACL2 Theorem Prover. This allows us, to a limited extent, to integrate, in a same system interaction (i.e. representation), computation (through the Kenzo kernel) and deduction (by means of ACL2).

Once the ACL2 and the Kenzo systems are integrated in a same GUI, much more work is needed to implement more interesting interactions. For instance, the encapsulates should be the basis for more complex theorem proving inside the system. As an example, let us consider the construction of a sphere in Kenzo. The GUI should generate an ACL2 script stating that this concrete (Common Lisp) object is a (functional) instance of the encapsulate `simplicial-set`, that is, the sphere is really a Simplicial Set. ACL2 very likely will not be able to prove those statements automatically, and some user interaction will be needed. Then, both the interface and the OMDoc documents should be enriched to cope with the user actions, allowing the system to recover, in further sessions, the full proof script, and then automating the verification of each construction generated in the system.

Other prototypes can be developed in order to integrate our system with different mathematical systems (for instance, GAP [4] has already been connected with Kenzo through OpenMath in [12]). In this sense, we have thought in customizing our GUI, in the same way as done for ACL2, with the aim of

integrating it with GAP. We claim that our architecture is enough modular to allow us to carry out this task in a non difficult way. In this sense, we think that general issues of this architecture can be applied to integrate any Symbolic Computation System with any Theorem Prover.

Moreover, in spite of working in a correct way the architecture presented in Figure 1 presents some drawbacks. In particular, if the system receives two requests, the second one must wait until the first one is finished. In addition, Kenzo computations used to be very time and space consuming (requiring, typically several days of CPU time on powerful dedicated computing servers); therefore to store these results in a persistent way would be useful to avoid recalculations. This possibility was not considered in our previous proposal. Now, we are focussing on solving these problems, but this only affects the MicroKernel part, not the Phrasebook nor the GUI, making transparent the changes to the users.

In a different line, a Web User Interface (WUI) for our framework can be developed. By using XUL again, its structure is already defined; developing its functionality is future work.

References

1. Buswell S., Caprotti O., Carlisle D.P., Dewar M.C., Gaëtano M., Kohlhase M. *OpenMath* Version 2.0, 2004. `http://www.openmath.org/`.
2. Dousson X., Sergeraert F., Siret Y., *The Kenzo program*, Institut Fourier, Grenoble, 1999. `http://www-fourier.ujf-grenoble.fr/~sergerar/Kenzo/`.
3. Gamma E., Helm R., Johnson R., Vlissides J., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1994.
4. GAP - Groups, Algorithms, Programming - a System for Computational Discrete Algebra. `http://www.gap-system.org/`.
5. Hanus M., Kluß C., *Declarative Programming of User Interfaces*, PADL 2009, Lectures Notes in Computer Science, 5418 (2009) 16–30.
6. Heras J., Pascual V., Rubio J., *Mediated Access to Symbolic Computation Systems*, MKM 2008, Lectures Notes in Artificial Intelligence, 5144 (2008) 446–461.
7. Heras J., Pascual V., Rubio J., *Using Open Mathematical Documents to interface Computer Algebra and Proof Assistant systems*. To appear in Proceedings MKM 2009, Lecture Notes in Artificial Intelligence 5625.
8. Hilton P. J., Wylie S., *Homology Theory*, Cambridge University Press, (1967).
9. Hyatt D. et al., *XML User Interface Language (XUL) 1.0* `http://www.mozilla.org/projects/xul/`.
10. Kaufmann M., Manolios P., Moore J., *Computer-Aided Reasoning: An Approach. Kluwer Academic Press*, Boston (2000).
11. Kohlhase M., *OMDoc – An open markup format for mathematical documents [Version 1.2]*, Springer Verlag (2006).
12. Romero A., Ellis G., Rubio J., *Interoperating between Computer Algebra systems: computing homology of groups with Kenzo and GAP*. To appear in Proceedings of ISSAC 2009.