CONTRIBUCIONES CIENTÍFICAS
EN HONOR DE MIRIAN ANDRÉS GÓMEZ
(Laureano Lambán, Ana Romero y Julio Rubio, editores),
Servicio de Publicaciones, Universidad de La Rioja,
Logroño, Spain, 2010.

MEDIATED ACCESS TO SYMBOLIC COMPUTATION SYSTEMS: AN OPENMATH APPROACH

JÓNATHAN HERAS Y VICO PASCUAL

Dedicado a la memoria de Mirian Andrés Gómez

Resumen. En este artículo, se describe una arquitectura para interactuar con el sistema de Cálculo Simbólico Kenzo (dedicado a la Topología Algebraica). El acceso que se proporciona esta mediado por medio de varios artefactos basados en XML, produciendo una capa intermedia *inteligente*. La parte más externa de la arquitectura ha sido creada usando Diccionarios de Contenido OpenMath y el correspondiente Phrasebook. Como consecuencia, varios Diccionarios de Contenido OpenMath para la Topología Algebraica Simplicial han sido desarrollados.

Abstract. In this paper, an architecture to interact with the Kenzo Computer Algebra system (devoted to Algebraic Topology) is described. The access is mediated by means of several XML-based artifacts, producing an *intelligent* intermediary layer. The most external part of the architecture is devised by using OpenMath Content Dictionaries and the corresponding PhraseBook. As a by-product, OpenMath Content Dictionaries for Algebraic and Simplicial Topology are developed for the first time.

1. Introduction

In [16] a first approach to a mediated access to the Kenzo Computer Algebra system was presented. The main objective was to increase the reliability and usability of the interaction with the Kenzo system. Kenzo was written by Sergeraert mainly as a research tool (see [11]), to implement his ideas on Constructive Algebraic Topology (see [27]). Due to this characteristic, most attention was paid to performance and to extend the scope of the system, but other aspects as the user interface were underrated. Indeed the user interface to Kenzo is Common Lisp itself, the language in which the system was programmed. This is a clear obstacle to increasing the number of users, likely researchers, teachers and students of Algebraic Topology, with no prior knowledge of Common Lisp.

The idea leading our previous paper [16] was that it is not possible to expose Kenzo in its full power to a non-trained user. Instead of it, we propose a mediated

2000 *Mathematics Subject Classification.* 68U99, 68N99.

Key words and phrases. Mathematical Knowledge Representation, Symbolic Computation Systems, OpenMath.

Jónathan Heras works with a grant of Comunidad de La Rioja. Partially supported by Universidad de La Rioja, project API08/08, and Ministerio de Educación y Ciencia, MTM2009-13842-C02-01.

access: an access through an intermediary layer that will control the user inputs and will stop his request if it is not allowed by Kenzo (that is to say, if Kenzo will raise an error when dealing with it). This implies to put more structure on top of Kenzo, and, as a consequence, to lose some kinds of *free* interactions with Kenzo (by means of ad-hoc Common Lips programs, for instance). Thus, we give to the user *less* computational power, but with *more* reliability and usability. Our bet is that this structured way to interact with Kenzo will be enough for most of the potential users. To this aim it is necessary that knowledge on Algebraic Topology is managed in an intermediary layer, in such a way that some *intelligent* behavior is obtained, guiding the user through the complexities of the Kenzo system.

In [16] emphasis was put in methodological and architectural issues. The ideas were made concrete by means of an implementation of the intermediary layer, and a Graphical User Interface (GUI) which takes profit from it. One of the comments raised by that paper was that the processes, even if organized by means of XML-based documents, were using mostly ad-hoc mechanisms, specified by us. OpenMath was suggested to us as a good alternative to make our proposal more standard, and to make easier the interaction with other systems. This enhancement is undertaken in this paper, adapting our general architecture to include OpenMath interfaces. As a proof of feasibility an *experimental* GUI is presented, where the interaction is guided by OpenMath Content Dictionaries (CDs) (the GUI presented in [16] was structured, as usual, by means of constructing objects and then operating on them). This GUI opens the possibility of integrating *dynamically* new modules, by loading some new OpenMath Content Dictionaries. As a by-product, some fresh OpenMath Content Dictionaries devoted to Algebraic and Simplicial Topology have been developed.

The rest of the paper is organized as follows. The next section presents antecedents of this work. Section 3 describes the architecture of our system, including in Subsection 3.1 the translation from Kenzo classes to OpenMath Content Dictionaries. Knowledge management in the intermediary layer is dealt with in Section 4. Then, in Section 5 our experimental GUI is presented. The paper ends with a section on Further work and Conclusions, and finally the bibliography.

2. Antecedents

Kenzo [11] 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 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). The most elaborated approach was reported in [2]. There, we devised a remote access to Kenzo, using CORBA [26] technology. An XML extension of MathML [3] played a role there too, but just to give genericity to the connection (avoiding the

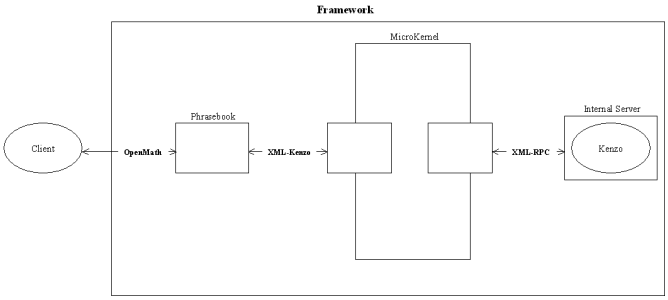

FIGURE 1. Microkernel architecture of the system.

definition in the CORBA Interface Description Language [26] of a different specification for each Kenzo class and datatype). There was no intention of taking profit from the semantics possibilities of MathML. Being useful, this approach ended in a prototype, and its enhancement and maintenance were difficult, due both to the low level characteristics of CORBA and to the pretentious aspiration of providing *full* access to Kenzo functionalities. We could classify the work of [2] in the same line as [6] or in the initiative IAMC and its corresponding workshop series (see, for instance, [8, 30]), where the emphasis is put into powerful and generic access to symbolic computation engines.

The first prototype for a mediated access to the Kenzo system was presented in [16]. That work is improved in several aspects in this paper. For instance, in our first prototype we used a MathML extension to represent mathematical concepts from Algebraic Topology but we were aware that it is not enough to express the complete meaning of these concepts. In MathML 3.0, one of the natural ways to represent content MathML expressions is to give a formal semantics in terms of OpenMath objects [5]. Instead of trying this rather indirect way, we have decided to move to an OpenMath approach. This implies several changes in our implementation of the proposed architecture (namely, including a Phrasebook), the introduction of new Content Dictionaries devoted to Topology concepts, and the possibility of having a GUI where OpenMath documents are explicitly managed, easing the interaction with other symbolic computation systems. These features are commented on the following sections.

3. Architecture of the system

In [16], we presented an architecture for the software system, inspired by the *Microkernel* architectural pattern [4]. This pattern gives a global view as a *platform*, in terminology of [4], which implements a virtual machine with applications running on top of it, namely a *framework* (in the same terminology). When introducing OpenMath machinery, the high level perspective of the system includes a Phrasebook node, as shown in Figure 1 (compare with [16] where the architecture was complicated by the occurrence of an external server and an adapter, roles that are now played, in an integrated manner, by our Phrasebook).

There are three different parts in our framework:

- an internal server,
- a Microkernel, and
- a Phrasebook.

Kenzo itself, wrapped with an interface based on XML-RPC [31], is acting as an *internal server*. The *Microkernel* plays the role of an intermediary layer and has two main objectives. On the one hand, it is responsible of the intelligent processing; see more details on it in Section 4. On the other hand, the Microkernel, which includes an XML processor, defines a link between the standard XML-RPC used by Allegro Common Lisp [13] and OpenMath. The conversion of an OpenMath object to/from the internal representation is performed by a *Phrasebook*, allowing the system to establish the final connection with clients.

A simplified version of this architecture (without the Phrasebook) would suffice if our objective was simply to build a GUI for Kenzo. Nevertheless, our intention is to build a framework capable of linking Kenzo with any possible client. This includes GUIs, but also web applications, web services, and, in addition, other symbolic computation systems (as GAP [15], for instance). This last aspect makes convenient to export the Kenzo mathematical knowledge in an standard format as OpenMath.

Each part of the framework has its own XML schema. In addition to the XML-RPC and OpenMath standards, we have defined another language called XML-Kenzo. This is a language expressing the concepts of Algebraic Topology, but free of the OpenMath verbose syntax. Furthermore, it gives more structure to the Kenzo primitives (distinguishing, for instance, between *constructors* and *operations*), defining something as a type system (see Section 4). Part of the XML schema for XML-Kenzo is described diagrammatically in Figure 2. However, the high level description provided by XML-Kenzo is not suitable to communicate with the internal server, because a low level register of each Kenzo session must be maintained (for instance, storing the unique identifier referring to each object, in order to avoid recalculations). Thus, a procedural language based on Kenzo conventions is needed; XML-RPC provides this functionality.

In the most external part inside our framework, all necessary mathematical concepts are included by means of OpenMath Content Dictionaries [5]. Figure 3 shows a fragment of the Simplicial Sets Content Dictionary. The structure of these new Content Dictionaries is dealt with in the following subsection.

To illustrate how information flows among the different layers, in Figure 4 we show how a Kenzo command (namely, the calculation of the third homology group of the sphere of dimension 3) will be transformed from a user command coming from a client (top part of the figure, expressed in OpenMath) to the final XML-RPC format (the conventional Lisp call is shown, too; however our internal server, Kenzo wrapped with an XML-RPC interface, will execute the command directly).

3.1. From Kenzo classes to OpenMath Content Dictionaries. The Kenzo system works with the main mathematical categories used in Combinatorial Algebraic Topology, [24]. In Figure 5 a fragment of the Kenzo class diagram is shown

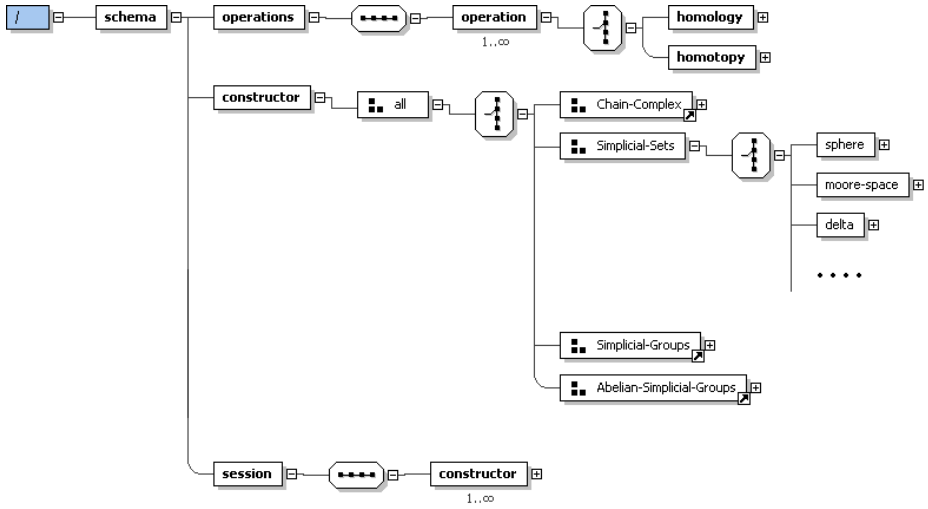

FIGURE 2. Description of the XML-Kenzo Schema.

```
<CD>
  <CDNAME>Simplicial-Sets</CDNAME>
  <CDURL/>
  <CDDATE>2008-08-21</CDDATE>
  <CDVERSION>1</CDVERSION>
  <CDSTATUS>Experimental</CDSTATUS>
  <CDUSES>
    <CDNAME>Simplicial-Groups</CDNAME>
    <CDNAME>Abelian-Simplicial-Groups</CDNAME>
  </CDUSES>
  <DESCRIPTION>This CD defines the simplicial sets used </DESCRIPTION>

  <CDDEFINITION>
    <NAME>Sphere</NAME>

    <DESCRIPTION>
      This symbol is a function with one argument, which should be a natural number
      n between 1 and 14. When applied to n it represents the sphere of dimension n.
    </DESCRIPTION>

    <EXAMPLE>
      The sphere of dimension 3
      <OMOBJ>
        <OMA>
          <OMS cd="Simplicial-Sets" name="sphere"/>
            <OMI>3</OMI>
        </OMA>
      </OMOBJ>
    </EXAMPLE>

  </CDDEFINITION>
```

FIGURE 3. Fragment of Simplicial Sets Content Dictionary.

(the full description can be found in [28]). From this part of the class diagram, that are the objects used in our system, the OpenMath Content Dictionaries have been defined.

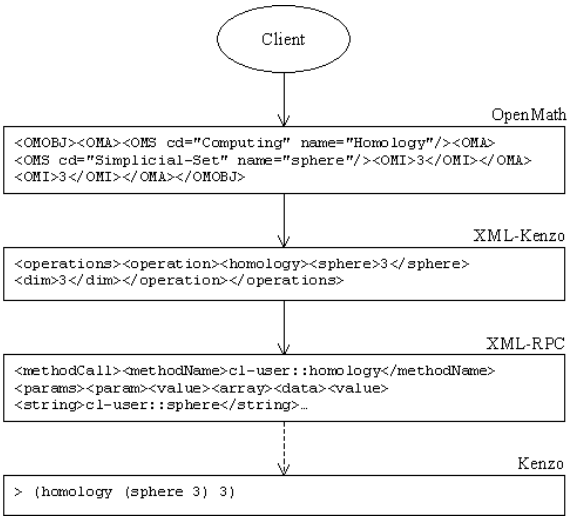

FIGURE 4. Transforming XML representations.

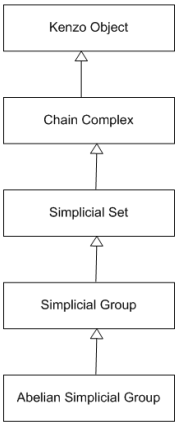

FIGURE 5. Fragment of the class diagram of the Kenzo system.

The starting point is to consider the Kenzo classes as the basic types of our system and some objects predefined in the Kenzo system (like spheres, Moore spaces and so on) as predefined objects of those types.

For each category represented in the diagram, an OpenMath Content Dictionary has been defined. In particular, four dictionaries have been written: *Chain-Complexes.ocd*, *Simplicial-Sets.ocd*, *Simplicial-Groups.ocd* and *Abelian-Simplicial-Groups.ocd*. The guideline used to define each Content Dictionary is to include in it the names of all the Kenzo primitives that *return* an object of the corresponding

```
<CDDEFINITION>
  <NAME>Sphere</NAME>

  <DESCRIPTION>
    This symbol is a function with one argument, which should be a natural number
    n between 1 and 14. When applied to n it represents the sphere of dimension n.
  </DESCRIPTION>

  <EXAMPLE>
    The sphere of dimension 3
    <OMOBJ>
      <OMA>
        <OMS cd="Simplicial-Sets" name="sphere"/>
          <OMI>3</OMI>
      </OMA>
    </OMOBJ>
  </EXAMPLE>

</CDDEFINITION>
```

FIGURE 6. Definition of the sphere.

class. For instance, in *Chain-Complexes.ocd* every operation which returns a chain complex is included.

To define the objects that are constructed by Kenzo from scratch, we use the OpenMath label "CDDefinition". These constructions must be included in the corresponding Content Dictionary. Such a definition contains: (1) the name of the object, (2) a description of its arguments (with the constraints fixed by Kenzo), and finally (3) an example of a particular instance of the construction. In Figure 6 (it is an excerpt of Figure 3, and so it is part of the *Simplicial-Sets.ocd*) the definition of simplicial sets modeling the *Sphere* is shown. There the description indicates a constraint: the dimension of a sphere should be between 1 and 14 (this restriction comes from the current Kenzo implementation).

Due to our decision of organizing the Content Dictionaries with respect to the return types, the class hierarchy showed in Figure 5 suffers, in some sense, an *inversion* in OpenMath. For instance, each operation producing a simplicial group also produces implicitly a simplicial set (since in Kenzo simplicial groups inherit from simplicial sets). Then, each operation in the Content Dictionary corresponding to a subclass (as *Simplicial-Group.ocd* in our example) must appear in the Content Dictionary of the superclass (as *Simplicial-Set.ocd*). This is got by means of the "CDUses" OpenMath label. This sort of *inverted* hierarchy is illustrated in Figure 7 in the case of the Chain Complex Content Dictionary. Let us note that we are not proposing this technique as a solution to the inheritance problem of Content Dictionaries (a difficult problem as recognized by the OpenMath community, see the end of the subsection *3.1.1 The Design Problem* of [18]). This is just a pragmatical trick, useful in our very concrete setting (another, more general solution, could be based on the notion of OpenMath *Document*; see [18] for details).

Finally, *Kenzo Object* is the superclass of the system grouping all the elements of other classes but this class does not represent any category, this class is the glue for the rest of classes. The corresponding OpenMath concept is that of Content

```
<CD>
  <CDNAME>Chain-Complex</CDNAME>
  <CDURL/>
  <CDDATE>2008-07-21</CDDATE>
  <CDVERSION>1</CDVERSION>
  <CDSTATUS>Experimental</CDSTATUS>
  <CDUSES>
    <CDNAME>Simplicial-Sets</CDNAME>
    <CDNAME>Simplicial-Groups</CDNAME>
    <CDNAME>Abelian-Simplicial-Groups</CDNAME>
  </CDUSES>
```

FIGURE 7. Content Dictionaries used by the Chain-Complex CD.

```
<CDGROUP>
  <CDGROUPNAME> Constructors </CDGROUPNAME>
  <CDGROUPVERSION> 1 </CDGROUPVERSION>
  <CDGROUPDESCRIPTION> Constructors. </CDGROUPDESCRIPTION>
  <CDGROUPMEMBER>
    <CDNAME> Chain-Complex </CDNAME>
    <CDCOMMENT> Constructors of Chain Complexes. </CDCOMMENT>
  </CDGROUPMEMBER>
  <CDGROUPMEMBER>
    <CDNAME> Simplicial-Sets </CDNAME>
    <CDCOMMENT> Constructors of Simplicial Sets. </CDCOMMENT>
  </CDGROUPMEMBER>
  <CDGROUPMEMBER>
    <CDNAME> Simplicial-Groups </CDNAME>
    <CDCOMMENT> Constructors of Simplicial Groups. </CDCOMMENT>
  </CDGROUPMEMBER>
  <CDGROUPMEMBER>
    <CDNAME> Abelian-Simplicial-Groups</CDNAME>
    <CDCOMMENT> Constructors of Abelian Simplicial Groups. </CDCOMMENT>
  </CDGROUPMEMBER>
</CDGROUP>
```

FIGURE 8. Constructors Content Dictionary Group.

Dictionary Group. Figure 8 shows the definition of this CD Group, that we have called *Constructors.cdg*.

4. Knowledge Management in the Intermediary Layer

The system as a whole will improve Kenzo including the following "intelligent" enhancements:

1. Controlling the input specifications on constructors.
2. Avoiding some operations on objects which will raise errors.
3. Chaining methods in order to provide the user with new tools.
4. Determining if a calculation can be done in a local computer or should be derived to a remote server.

In order to explain the differences between points 1 and 2, it is worth noting that in Kenzo there are two *kinds* of data. The first one is representing *spaces*

in Algebraic Topology (by *spaces* we mean here, any data structure having both behavior and elements belonging to it, such as a simplicial set, a simplicial group, a chain complex, and so on). The second kind of data is used to represent *elements* of the spaces. Thus, in a typical session with Kenzo, the users proceed in two steps: first, constructing some spaces, and second, applying some operators on the (elements of the) spaces previously built. This organization in two steps has been described by using Algebraic Specification methods in [19] and [10], for instance. Therefore, the first item in the enumeration refers to the inputs for the constructors of spaces, and the second item refers to some operations on *concrete* spaces.

Kenzo is, in its pure mode, an untyped system (or rather, a dynamically typed system), inheriting its power and its weakness from Common Lisp. Thus, for instance, in Kenzo a user could apply a constructor to an object without satisfying its input specification. For instance, the method constructing the classifying space of a simplicial group could be called on a simplicial set without a group structure over it. Then, at runtime, Common Lisp would raise an error informing the user of this restriction. This is shown in the following fragment of a Kenzo session:

```
> (loop-space (sphere 4)) ✠
[K6 Simplicial-Group]
> (classifying-space (loop-space (sphere 4))) ✠
[K18 Simplicial-Set]
> (sphere 4) ✠
[K1 Simplicial-Set]
> (classifying-space (sphere 4)) ✠
;; Error: No method in generic function CLASSIFYING-SPACE
;; is applicable to arguments: [K1 Simplicial-Set]
```

With the first command, namely `(loop-space (sphere 4))`, we construct a simplicial group. Then, in the next step we are verifying that a simplicial group has a classifying space (which is, in general, just a simplicial set). In the third command, we check that the sphere of dimension 4 is constructed in Kenzo as a simplicial set. Thus, when in the last command we try to construct the classifying space of a simplicial set, the Common Lisp Object System (CLOS) raises an error.

In the current version of our system this kind of error is controlled, because the inputs for the operations between spaces can be only selected among the spaces with suitable characteristics. This same idea could be used to improve the reliability of internal processes, by controlling the outputs of the intermediary computations. In Figure 9, we show the situation corresponding in our system to the example introduced before in pure Kenzo. There we use as illustration the experimental GUI we will describe in Section 5. In that figure, it can be seen that for the classifying operation just the spaces which are simplicial groups are candidates to be selected. This enriches Kenzo with something as a (semantical) type system which has been defined into our XML-Kenzo.

With respect to the second item in the previous enumeration, the most important example in the current version is the management of the *connection degree* of spaces. Kenzo allows the user to construct, for instance, the loop space of a non simply connected space (as the sphere of dimension 1). The result is a simplicial

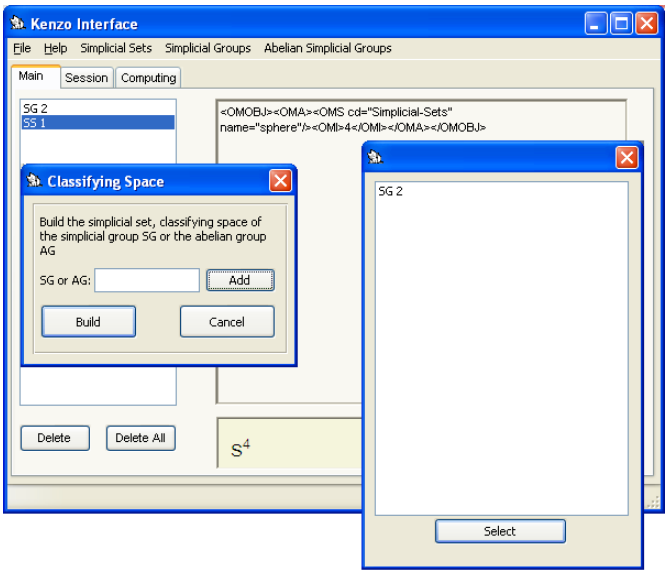

FIGURE 9. Screen-shot of Kenzo Interface with a session related to classifying spaces.

set on which some operations (for instance, to compute the set of faces of a simplex) can be achieved without any problems. On the contrary, theoretical results ensure that the homology groups are not of finite type, and then they cannot be computed. In pure Kenzo, the user could ask for a homology group of such a space, catching a runtime error.

In our current version of the system, the intermediary layer includes a small expert system, computing, in a symbolic way (that is to say, working with the *description* of the spaces, and not with the spaces themselves considered as Common Lisp objects), the connection degree of a space. The set of rules gives a connection degree to each space builder (for instance, a sphere of dimension n has connection degree $n-1$), and then a rule for each operation on spaces. For instance, loop space decreases the connection degree of its input in one unity, suspension increases it in one unity, a cartesian product has, as connection degree, the minimum of the connection degrees of its factors, and so on. From the design point of view, a *Decorator* pattern [14] was used, decorating each space with an annotation of its connection degree in the intermediary layer. Then, when a computation (of a homology group, for instance) is demanded by a user, the intermediary layer monitors if the connection degree allows the transferring of the command to the Kenzo kernel, or a warning must be sent to the user.

As for item three, the best example is that of the computation of *homotopy groups*. In pure Kenzo, there is no final function allowing the user to compute them. Instead, there is a number of complex algorithms, allowing a user to chain them to get some homotopy groups. Our current user interface has an option

to compute homotopy groups. The intermediary layer is in charge of chaining the different algorithms present in Kenzo to reach the final objective. In addition, Kenzo, in its current version, has limited capabilities to compute homotopy groups (depending on the homology of Eilenberg-Mac Lane spaces that are only partially implemented in Kenzo), so the *chaining* of algorithms cannot be *universal*. Thus, the intermediary layer should process the call for a homotopy group, making some consultations to the Kenzo kernel (computing some intermediary homology groups, for instance) before deciding if the computation is possible or not (this is still work in progress).

Regarding point four, our system can be distributed, at present, in two manners: (a) as a stand-alone application, with a heavy client containing the Kenzo kernel to be run in the local host computer; (b) as a light client, containing just the user interface, and every operation and computation is done in a remote server. The second mode has obvious drawbacks related to the reliability of Internet connections, to the overhead of management where several concurrent users are allowed, etc. But option (a) is not fully satisfactory since interesting Kenzo computations used to be very time and space consuming (requiring, typically, several days of CPU time on powerful computing servers). Thus a mixed strategy should be convenient: the intermediary layer should decide if a concrete calculation can be done in the local computer or it deserves to be sent to a specialized remote server. (In this second case, as it is not sensible to maintain open an Internet connection for several days waiting for the end of a computation, some reactive mechanism should be implemented, allowing the client to disconnect and to be subscribed in some way, to the process of computation in the remote server). The difficulties of this point have two sources: (1) the knowledge here is not based on well-known theorems (as was the case in our discussion on the *connection degree* in the second item of the enumeration), since it is context-dependent (for instance, it depends on the computational power of a local computer), and so it should be based on *heuristics*; (2) the technical problems to obtain an optimal performance are complicated, due, in particular, to the necessity of maintaining a *shared state* between two different computers. These technical aspects are briefly commented in the Further Work section.

With respect to the kind of heuristic knowledge to be managed into the intermediary level, there is some part of it that could be considered obvious: for instance, to ask for an homology group $H_n(X)$ where the degree n is big, should be considered harder than if n is small, and then one could wonder about a limit for n before sending the computation to a remote server. Nevertheless, this simplistic view is to be moderated by some expert knowledge: it is the case that in some kinds of spaces, difficulties decrease when the degree increases. The heuristics should consider each operation individually. For instance, it is true that in the computation of homology groups of iterated loop spaces, difficulties increase with the degree of iteration. Another measure of complexity is related to the number of times a computation needs to call the Eilenberg-Zilber algorithm (see [11]), where a double exponential complexity bound is reached. Further research is needed to

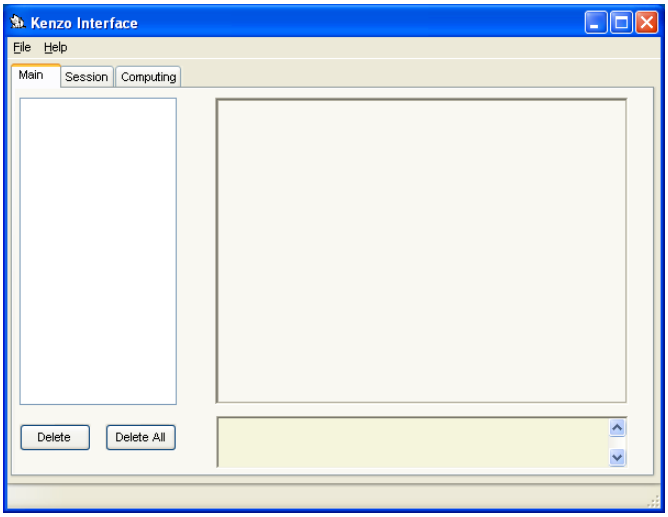

FIGURE 10. Screen-shot of the *Main* tab of Kenzo Interface.

exploit the expert knowledge in the area suitably, in order to devise a systematic heuristic approach to this problem.

5. A CLIENT FOR THE FRAMEWORK: AN EXPERIMENTAL GUI

In [16] we presented a GUI based on usual metaphors: objects are constructed, and then different operations can be applied on them. Here we are describing another GUI where OpenMath plays two important roles. On the one hand, OpenMath Content Dictionaries lead the evolution on the interface itself: when loading a Content Dictionary, the interface changes, appearing new options in the toolbar. On the other hand, the results of the operations, and the description of sessions, are encoded in OpenMath, which is explicitly showed to the user. Even if the GUI can be considered *experimental*, in the sense that usability cannot be considered fully satisfactory, at least it serves to us to illustrate in a very explicit way the role of OpenMath (and, in particular, it makes evident that the OpenMath descriptions could be used to interact with other systems offering an OpenMath interface, too).

In Figure 10, a screen-shot of our GUI is presented. 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 OpenMath Content Dictionaries and the OpenMath Content Dictionary Groups. When the user exits the application, his configuration is saved for future sessions.

In the current version the *File* menu has six options: *Add Module*, *Delete Module*, *Save Session*, *Load Session*, *Save Computing*, and *Exit*.

So, the first user task consists in selecting the necessary modules, each module is associated with one or more OpenMath Content Dictionaries (the OpenMath

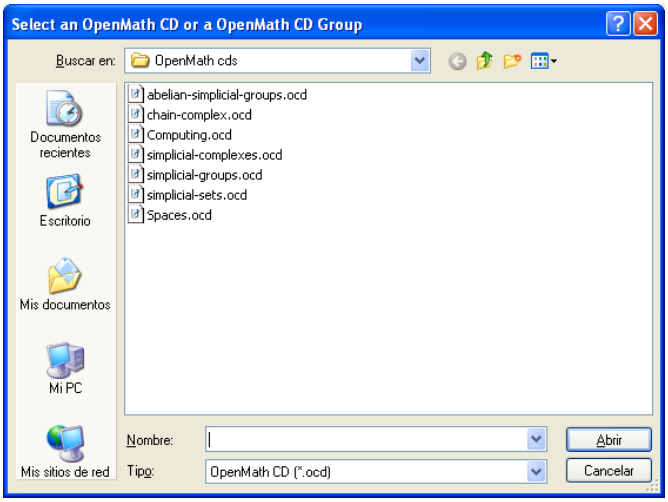

FIGURE 11. Selection of an OpenMath Content Dictionary.

Content Dictionaries of the system and also the necessary modules are included in the folder of the GUI). Figure 11 shows the dialog displayed when the *Add Module* option is selected.

If the user selects an OpenMath Content Dictionary, its corresponding module is loaded, but also all the Content Dictionaries used. For instance, in Figure 3 a fragment of the OpenMath Content Dictionary of the Simplicial-Sets is shown. In that case, this Content Dictionary uses the Simplicial-Groups and the Abelian-Simplicial-Groups Content Dictionaries, so if a user loads the Simplicial-Sets Content Dictionary, the Simplicial-Groups and the Abelian-Simplicial-Groups Content Dictionaries are also loaded. When an OpenMath Content Dictionary is loaded, the GUI main toolbar changes. For instance, when *Computing.ocd* (this OpenMath Content Dictionary defines the computations allowed in the system) is loaded the rest of the OpenMath Content Dictionaries are also loaded, and the GUI state can be seen in Figure 12.

If an OpenMath Content Dictionary Group is loaded, all the Content Dictionary members of this CD Group are loaded. For instance, the Constructor Content Dictionary Group, Figure 8, includes the Chain-Complexes, Simplicial-Sets, Simplicial-Groups and Abelian-Simplicial-Groups Content Dictionaries.

In an analogous way, the modules can be unloaded from the *Delete Module* option.

The main toolbar options when the GUI has been set with all its functionality are: *Computing*, *Chain Complexes*, *Simplicial Sets*, *Simplicial Groups* and *Abelian Simplicial Groups*.

The last four menus are related to the construction of spaces. Each menu contains all the possible ways of constructing spaces of its associated classes. Each option allows to build spaces of its associated classes in two different ways. The

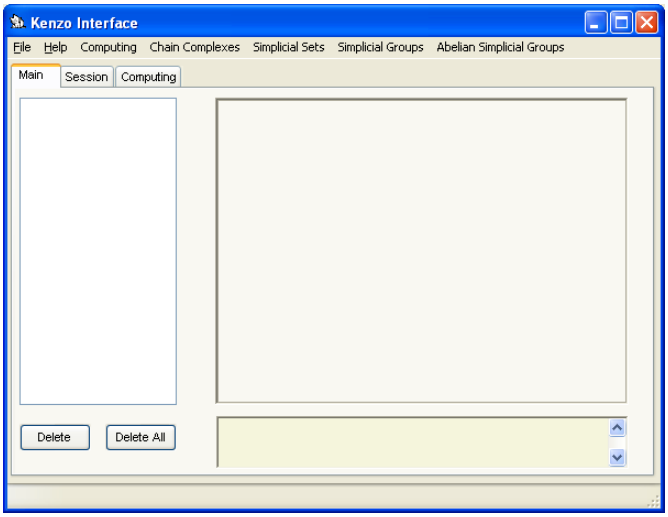

FIGURE 12. Screen-shot of the *Main* tab of Kenzo Interface with the Computing CD loaded.

first one from scratch in Kenzo as options (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, etc.).

The menu *Computing* collect all the operations on concrete spaces (instead of *constructing spaces*, as in the previous cases). In this menu we concentrate on calculations *over* a space. At this time, we offer to compute homology and homotopy groups.

The rest options of the *File* menu, *Save Session, Load Session* and *Save Computing*, are related to a particular session.

When saving a session a file is produced containing an XML description of the commands executed by the user in that session. These session files are saved using the OpenMath format following the rules defined in the corresponding Content Dictionaries. Figure 13 includes an example of an OpenMath session; this session file corresponds to the following Kenzo interaction:

```
> (sphere 3) ✠
[K1 Simplicial-Set]
> (moore 4 2) ✠
[K6 Simplicial-Set]
> (loop-space (sphere 3) 2) ✠
[K23 Simplicial-Group]
> (crts-prdc (sphere 3) (moore 4 2)) ✠
[K35 Simplicial-Set]
```

```
<OMOBJ>
  <OMA>
    <OMS cd="Simplicial-Sets" name="sphere"/>
    <OMI>3</OMI>
  </OMA>
</OMOBJ>

<OMOBJ>
  <OMA>
    <OMS cd="Simplicial-Sets" name="moore-space"/>
    <OMI>4</OMI>
    <OMI>2</OMI>
  </OMA>
</OMOBJ>

<OMOBJ>
  <OMA>
    <OMS cd="simplicial-group" name="Loop-Space"/>
    <OMA>
      <OMS cd="Simplicial-Set" name="sphere"/>
      <OMI>3</OMI>
    </OMA>
    <OMI>2</OMI>
  </OMA>
</OMOBJ>

<OMOBJ>
  <OMA>
    <OMS cd="Simplicial-Sets" name="Cartesian-Product"/>
    <OMA>
      <OMS cd="Simplicial-Sets" name="sphere"/>
      <OMI>3</OMI>
    </OMA>
    <OMA>
      <OMS cd="Simplicial-Sets" name="moore-space"/>
      <OMI>4</OMI>
      <OMI>2</OMI>
    </OMA>
  </OMA>
</OMOBJ>
```

FIGURE 13. Sample of an OpenMath session file.

This session file can be loaded from the *Load Session* option. Besides, the computations made during a session can be saved, in the same format, from the *Save Computing* 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.

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 14), its description is displayed in the right area using OpenMath, and just below the usual mathematical notation of the space can be found.

When the *Session* tab is focused, a similar screen to Figure 15 is shown. The objects built during the current session, in an ordered way, are rendered. In the same way, Figure 16 shows a screen with the computations in the current session, displayed into the *Computing* tab. Both sessions and computations could be expressed in the usual mathematical notation, as we have done in the bottom right area of the *Main* tab (see Figure 14). This was the option in the GUI described in [16]. Here we have preferred to keep explicitly the OpenMath notation to illustrate that this could be the right interface with other systems.

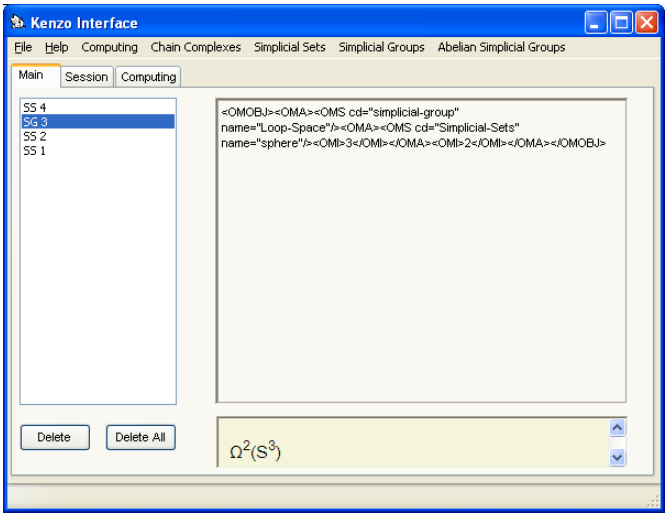

FIGURE 14. The *Main* tab with an example of session.

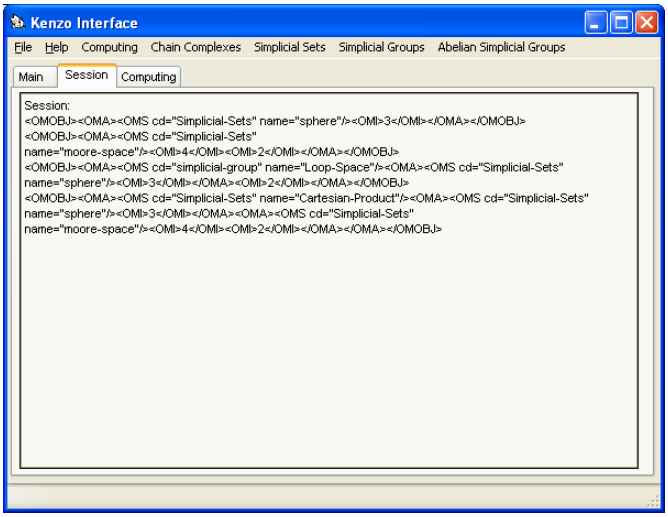

FIGURE 15. The *Session* tab with an example of session.

6. FURTHER WORK AND CONCLUSIONS

As a further demonstration of the adequacy of our framework, which pretends to be *neutral* with respect to the technology used in the client-side, two OpenMath *Web Services* have been developed. These web services allow to use the framework only from the OpenMath Content Dictionaries (no knowledge about Lisp or Kenzo is needed). So, a developer can build any other application in Java, .NET and so

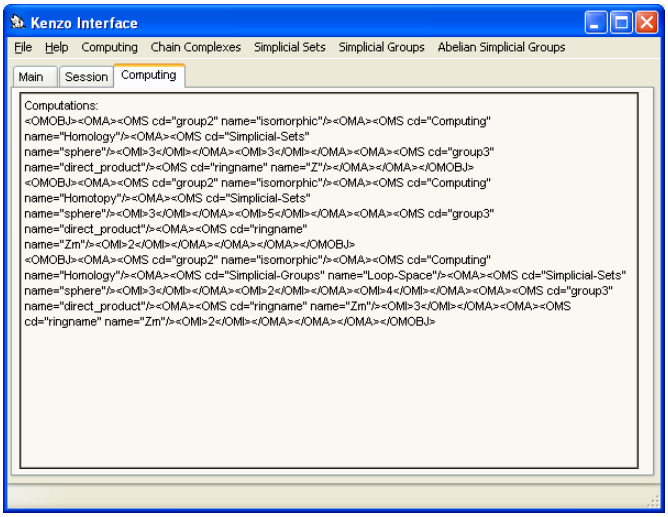

FIGURE 16. The *Computing* tab with some computations.

on, knowing only our OpenMath Content Dictionaries and, of course, the web services technology. The difference between both web services is the way in which the result is returned. The first web service makes a request and waits until the result is available. On the contrary, in the second web service the connection is not maintained, for this reason it needs not only the OpenMath request but also an e-mail direction where the result of the request is returned. Other different clients should be experienced (as complete web applications, for instance), in order to ensure the different layers are uncoupled enough in our framework. The *dynamical* style of our experimental GUI should be also explored further, by adding new modules, no necessarily present in Kenzo in its current version.

But the most important issue to be tackled in the next versions of the system is how organizing the decision on when a calculation should be derived to a remote server. To understand the nature of the problem it is necessary to consider that there are two kinds of *state* in our context. Starting from the most simple, the *state of a session* can be described by means of the spaces that have been constructed so far. Then, to encode (and recover) such a state, a session file as explained in the previous section would be enough: an OpenMath document containing a sequence of calls to different constructors and methods. In this case, when a calculation is considered too hard to be computed in a local computer, the whole session file could be transmitted to the remote server. There, executing step-by-step the session file, the program will re-find the same state of the local session, proceeding to compute the desired result and sending it to the client. Of course, as mentioned previously, some kind of subscription tool should be enabled, in such a way that the client could stop its running, and then to receive the result (or a notification

indicating the result is already available somewhere), after some time (perhaps some days or weeks of computation on the remote server).

Even if this approach can be considered reasonable as a first step, it has turned out to be too simplistic to deal with the richness of Kenzo. A space in Kenzo consists in a number of methods describing its behavior (explaining, for instance, how to compute the faces of its elements). Due to the high complexity of the algorithms involved in Kenzo, a strategy of *memoization* has been systematically implemented. As a consequence, the *state of a space* evolves after it has been used in a computation (of a homology group, for instance). Thus, the time needed to compute, let us say, a face, depends on the concrete states of every space involved in the calculation (in the more explicit case, to re-calculate a face on a space could be negligible in time, even if in the first occasion this was very time consuming). This notion of *state of a space* is transmitted to the notion of *state of a session*. We could speak of two *states of a session*: the one *shallow* evoked before, that is essentially *static* and can be recovered by simply re-executing the top-level constructor calls; and the other *deep state* which is dynamic and depends on the computations performed on the spaces.

To analyse the consequences of this Kenzo organization, we should play with some scenarios. Imagine during a local session a very time consuming calculation appears; then we could simply send the *shallow state of the session* to the remote server, because even if some intermediary calculations have been stored in local memory, they can be re-computed in the remote server (finally, if they are cheap enough to be computed on the local computer, the price of re-computing them in the powerful remote server would be low). Once the calculation is remotely finished, there is no possibility of sending back the *deep state* of the remote session to the local computer because, usually, the memory used will exhaust the space in the local computer. Thus, it could seem that to transmit the *shallow state* would be enough. But, in this picture, we are losing the very reason why Kenzo uses the memoization (dynamic programming) style. Indeed, if after obtaining a difficult result (by means of the remote server) we resume the local session and ask for another related difficult calculation, then the remote server will initialize a new session from scratch, being obligated to re-calculate every previous difficult result, perhaps making the continuation of the session impossible. Therefore, in order to take advantages of all the possibilities Kenzo is offering now on powerful scientific servers, we are faced with some kind of *state sharing* among different computers (the local computers and the server), a problem known as difficult in the field of distributed object-oriented programming.

In short, even if our initial goal was not related to distributed computing, we found that in order to enable our intermediary layer as an *intelligent assistant* with respect to the classification of calculations as simple (runnable on a standard local computer) or complicated (to be sent to a remote server), we should solve problems of distributed systems. Thus, a larger perspective is necessary, and we are working with the Broker architectural pattern, see [4], in order to find a natural organization of our intermediary layer. From the symbolic computation literature, we will look for inspiration in different projects and frameworks such as

the MathWeb software bus [23], its successor the MathServe Framework [22], the MoNET project [25, 7, 9] or the MathBroker [20] and MathBroker II [21] projects, as well as in other works as [29], [32] or [12].

As a final direction of research, we could try to take more advantage of the semantical possibilities of OpenMath. Concretely, we could include the *axioms* which the categories encoded in our Content Dictionaries really must satisfy. This will open a way to integrate some deductive capabilities in our system. To this aim, it could be interesting to interface it in some manner with the Common Lisp theorem prover ACL2 [17], which has been already used to formalize some aspects of Simplicial Topology [1].

As a conclusion, we claim that OpenMath can be smoothly integrated in our Framework, acting as an external layer. The intermediary layer converts OpenMath descriptions into low level XML-RPC documents and, in addition, is capable of an intelligent processing of the user commands. This intelligent mediated access can be achieved without using traditional Artificial Intelligence techniques, but managing some expert Mathematical Knowledge (on Algebraic Topology, in our case) by means of our XML-Kenzo schema, specially designed to work with spaces in a symbolic manner (that it to say, to work with their XML descriptions, instead of with their Common Lisp Kenzo representation). This put more structure on top of Kenzo (namely, something as a semantical type system), giving *less* computational power, but interactions *more* usable and reliable.

Referencias

[1] M. Andrés, L. Lambán, J. Rubio, J. L. Ruiz Reina. Formalizing simplicial topology in ACL2. En *Proceedings Workshop ACL2 2007*, pp. 34 – 39. Austin University, 2007.

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

[3] R. Ausbrooks et al. *Mathematical Markup Language (MathML)*. Version 3.0 (second edition). 2008. `http://www.w3.org/TR/2008/WD-MathML3-20080409/`.

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

[5] S. Buswell, O. Caprotti, D.P. Carlisle, M.C. Dewar, M. Gaëtano, M. Kohlhase. *OpenMath* Version 2.0. 2004. `http://www.openmath.org/`.

[6] J. Calmet, K. Homann. Towards the Mathematics Software Bus. *Theoretical Computer Science* **187**, 221–230, 1997.

[7] O. Caprotti, J.H. Davenport, M. Dewar, J. Padget. Mathematics on the (Semantic) NET. *Lecture Notes in Computer Science* **3053**, 213–224, 2004.

[8] D. Chiu, Y. Zhou, X. Zou, P.S. Wang. Design, implementation, and processing support of MeML. En *Proceedings Internet Accessible Mathematical Computation 2003*. `http://www.symbolicnet.org/conferences/iamc03/meml.pdf`

[9] M. Dewar, E. Smirnova, S.M. Watt. XML in Mathematical Web Services. En *Proceedings of XML 2005 Conference – Syntax to Semantics (XML 2005)*. IDEAlliance, 2005.

[10] C. Domínguez, L. Lambán, J. Rubio. Object oriented institutions to specify symbolic computation systems. *Rairo - Theoretical Informatics and Applications* **41**, 191–214, 2007.

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

[12] A. Duscher. *Interaction patterns of Mathematical Services*. Technical report, Research Institute for Symbolic Computation (RISC), Johannes Kepler Univeristy, Linz, 2006.

[13] Franz Inc. *Allegro Common Lisp*. http://www.franz.com/.
[14] E. Gamma, R. Helm, R. Johnson, J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.
[15] *GAP - Groups, Algorithms, Programming - a System for Computational Discrete Algebra.* http://www.gap-system.org/
[16] J. Heras, V. Pascual, J. Rubio. Mediated Access to Symbolic Computation Systems. *Lectures Notes in Artificial Intelligence* **5144**, 446–461, 2008.
[17] M. Kaufmann, P. Manolios, J. Moore. *Computer-Aided Reasoning: An Approach. Kluwer Academic Press*. Boston, 2000.
[18] M. Kohlhase. *OMDoc – An open markup format for mathematical documents [Version 1.2]*. Springer Verlag, 2006.
[19] L. Lambán, V. Pascual, J. Rubio. An object-oriented interpretation of the EAT system. *Applicable Algebra in Engineering, Communication and Computing* **14**, 187–215, 2003.
[20] *MathBroker: A Framework for Brokering Distributed Mathematical Services*. http://www.risc.uni-linz.ac.at/projects/basic/mathbroker/.
[21] *MathBroker II: Brokering Distributed Mathematical Services*. http://www.risc.uni-linz.ac.at/projects/mathbroker2/.
[22] *MathServe Framework*. http://www.ags.uni-sb.de/~jzimmer/mathserve.html.
[23] *MATHWEB-SB: A Software Bus for MathWeb*. http://www.ags.uni-sb.de/~jzimmer/mathweb-sb/.
[24] J.P. May. *Simplicial objects in Algebraic Topology*. Van Nostrand Mathematical Studies 11, 1967.
[25] *MoNET: Mathematics on the Net*. http://monet.nag.co.uk/cocoon/monet/index.html.
[26] Object Management Group. *Common Object Request Broker Architecture (CORBA)*. http://www.omg.org.
[27] J. Rubio, F. Sergeraert. Constructive Algebraic Topology. *Bulletin des Sciences Mathématiques* **126**(5), 389–412, 2002.
[28] F. Sergeraert. *Common Lisp, Typing and Mathematics*. Talk of the EACA Meeting at Ezcaray (Spain), 2001.
[29] E. Smirnova, C.M. So, S.M. Watt. An architecture for distributed mathematical web services. *Lecture Notes in Computer Science* **3119**, 363–377, 2004.
[30] P.S. Wang, N. Kajler, Y. Zhou, X. Zou. Initial design of a Web-Based Mathematics Education Framework. En *Proceedings Internet Accessible Mathematical Computation 2002*. http://www.symbolicnet.org/conferences/iamc02/wme.pdf
[31] D. Winer. *Extensible Markup Language-Remote Procedure Call (XML-RPC)*. http://www.xmlrpc.com.
[32] J. Zhu, Y. Wu, F. Xie, G. Yang, Q. Wang, J. Mao, M. Shen. A model for Distributed Computation over the Internet. En *Proceedings Internet Accessible Mathematical Computation 2003*. http://www.symbolicnet.org/conferences/iamc03/zhu.pdf.

Departamento Matemáticas y Computación, Universidad de La Rioja, Spain
Correo electrónico: jonathan.heras@unirioja.es

Departamento Matemáticas y Computación, Universidad de La Rioja, Spain
Correo electrónico: vico.pascual@unirioja.es