Remote Access to a Symbolic Computation System for Algebraic Topology: A Client-Server Approach*

Mirian Andrés, Vico Pascual, Ana Romero, and Julio Rubio

Departamento de Matemáticas y Computación, Universidad de La Rioja
{miriam.andres, vico.pascual, ana.romero, julio.rubio}@dmc.unirioja.es

Abstract. Kenzo is a Symbolic Computation system created by Sergeraert for computing in Algebraic Topology. It is programmed in Common Lisp and this programming language also acts as user interface. In this paper, a prototype to provide remote access for Kenzo is presented. This has been accomplished by using Corba technology: clients have been developed both in Java and Common Lisp (the server is always in Common Lisp, being a wrapper of the original Kenzo program). Instead of using one CORBA IDL to encode each data structure, our approach incorporates a generic way of transfering every data structure through XML strings; specifically, by means of an XML extension of MathML. This research should be understood as a first step towards building a distributed computation system for Algebraic Topology.

Introduction

Nowadays, Internet appears as a suitable tool for performing scientific computations in a distributed collaborative way. One of the fields where this idea can be applied is that of Symbolic Computation. Computer Algebra packages are being extended to interconnect them. In fact, some Computer Algebra systems, such as Distributed Maple, are already capable of performing distributed computing.

It is clear that this trend could be explored in any other Symbolic Computation system, in particular in systems devoted to algorithmic Algebraic Topology. The leader systems in this field are EAT [5] and Kenzo [6]. Both systems are written in the Common Lisp programming language and have obtained some results (specifically, homology groups) which had never been determined before using either theoretical or computational methods. Since the programs are very time (and space) consuming, it would be interesting to begin a study about the task of making them distributed. As a first step, we have considered the possibility of providing a remote access for Kenzo.

The organization of this paper is as follows. The next section presents some preliminary ideas which are needed for our task: some basic concepts in Algebraic

* Partially supported by SEUI-MEC, project TIC2002-01626.

V.S. Sunderam et al. (Eds.): ICCS 2005, LNCS 3516, pp. 635–642, 2005.

Topology, some aspects of Symbolic Computation systems (especially Kenzo and EAT), and an explanation of the previous work. In Section 2 we describe the prototype of remote access to Kenzo that we have developed, which makes use of CORBA and XML. Section 3 presents some examples of the use of our programs. The paper ends with a conclusions section and the bibliography.

1 Preliminaries

1.1 Notions of Algebraic Topology

In this section, some basic concepts of Algebraic Topology are introduced.

Definition 1. *A* chain complex $C = (C_p, d_p)$ *is a family of R-modules* $\{C_p\}_{p\in\mathbb{Z}}$ *(R is a ring), with R-modules homomorphisms* $\{d_p\}_{p\in\mathbb{Z}}$ *(the* differential application*)* $d_p : C_p \to C_{p-1}$ *such that* $d_{p-1} \circ d_p = 0$.

We consider chain complexes formed by free $\mathbb{Z}$-modules. The fact that C_p is a free $\mathbb{Z}$-module implies that it is generated, and therefore each element of C_p can be expressed as a linear combination of the set of its generators: $\sum \lambda_i \sigma_i, \quad \lambda_i \in \mathbb{Z}$. An element σ_i of a basis is called a *generator*. A product $\lambda_i \sigma_i$ is called a *term* or *monomial*, and a sum of these terms is called a *combination*.

1.2 Symbolic Computation in Algebraic Topology

There are some Symbolic Computation systems for Algebraic Topology, EAT and Kenzo being the most important ones. Created by Sergeraert and coworkers, and written in Common Lisp, they can be used to compute homology groups of infinite topological spaces, namely loop spaces.

We present here a simple example of computation with Kenzo, which consists on computing the differential of the chain complex (delta 3) applied to the combination $3 * 4 + 2 * 6$ of degree 2. The solution obtained is the combination (of degree 1) $3 * 0 - 2 * 2 + 2 * 4$.

```
> (dffr (delta 3) (cmbn 2 3 4 2 6))
(:cmbn 1 (3 . 0) (-2 . 2) (2 . 4))
```

The system allows the computation of more complicated results, for instance homology groups of chain complexes, one of the fundamental problems of Algebraic Topology. It is possible to obtain easily calculations that an expert on topology would need a lot of time and effort to get.

Some important features of these systems are:

- Intensive use of functional programming to deal with infinite data.
- Organization in two layers of data structures. The first one corresponds to the algebraic structures they work with, such as a chain complex. The second one corresponds to the elements of these structures, as for example a combination inside a chain complex.

- They have obtained some results which had never been determined before using either theoretical or computational methods.
- However, the usability of Kenzo and EAT presents some barriers. Since their user interface is Common Lisp itself (a not very extended programming language), they do not provide a friendly front end.

This last aspect of Kenzo was one of the reasons why we tried to provide a remote access for it, allowing in this way the use of other (better known) languages to construct the user interface. Since our goal is distributed computing, we try to do this in a scalable way, that is, in a way that would allow us in the future to extend our remote access system in order to perform collaborative computations among different computers.

1.3 Previous Work

In a previous work, we considered reconstructing (part of) the system in Java, a programming language very extended nowadays, especially in Internet. There are big differences between this language and Common Lisp, so we considered an intermediate step: we rebuilt (some fragments of) EAT in the programming language ML, that is not so far from Common Lisp as Java. So, we have implemented two prototypes to perform computations in Algebraic Topology in ML and Java respectively.

The next step was to interoperate between them and the Common Lisp version. To build our exchange language, we used XML [9], a metalanguage that allows a flexible representation of the information. We considered MathML [7] and OpenMath [8], two standards that give XML representations for mathematical objects (the second one also used to exchange mathematical objects between symbolic computation systems). We represented simple data with MathML, and for more complicated structures we defined our own DTD as an extension of that of MathML. OpenMath was not used because content dictionaries are strictly organised and we found it difficult to apply them in our case. With this exchange format the three systems were able to interchange XML documents among them, but only in a local non-distributed way (see [2]).

This was the starting point of this work, consisting on developing a remote access for Kenzo, with a client-server architecture. We would have the Kenzo program in a (server) machine, and several clients (with their own languages) could access it from different computers. A client would invoke some calculations in Algebraic Topology, the server would then process the computation and finally send the result to the client.

2 Development of the Prototype

As mentioned before, our goal was to construct a prototype that allows an access to the Kenzo program from different machines with other programming languages. Our first idea was to try to reuse the system developed previously

and make it work with different computers. We had the XML exchange language but this was an incomplete picture because an infrastructure that would allow the interchange between the client and the server was necessary. It could be CORBA [4], a middleware that allows pieces of programs, called objects, to communicate with one another regardless of programming languages and operating systems, especially used in a local net. The objects of CORBA have an interface written in IDL (an interface description language described in the CORBA specification). For each object, this interface is the same for client and server but the implementations in both sides can be written in different languages.

In a previous work (included as a poster in ISSAC 04 and presented in [3]) we have examined some of our first trials towards this remote access, and some problems found. The main problem was to find a way to exchange the data in Kenzo between client and server. In a first attempt, we tried to define an IDL file for each of the structures we wanted to work with. For instance, an interface for a momomial (one of the simplest data that appear in the program) could be as presented below.

```
module kenzo
interface Monomial
//Attributes
readonly attribute long cffc;
readonly attribute string gnrt;
//Methods
void setMnm(in long cffc, in string gnrt);
long getCffc();
string getGnrt(); ; ;
```

It would be clearly a tedious task to construct an IDL for each type in Kenzo (the program defines 15 classes and about 40 types; see [6]). If we worked in this way, we would have an IDL explosion and the running of CORBA would be complicated. Our prototype solved this problem with the combination of CORBA and XML in a natural way, inserting two string attributes in the object interface which will contain the XML representation of the call to the Kenzo function and the result. We obtain in this way a generic IDL for every Kenzo elements.

The IDL file we use to interoperate between client and server is showed below. The two attributes `obj` and `result` will contain the XML representation of the call to the Kenzo function we want to compute and its result. With `setObject` the client is able to set the operation and its arguments (in XML). When the client calls `computeXML`, the server obtains the XML representation of the operation with `getObject`, computes it and inserts in `result` the XML representation of the solution. Finally, `getResult` allows the client to recapture the result of the operation.

```
module kenzo
interface XMLObject
//Attributes
readonly attribute string obj;
readonly attribute string result;
//Methods
void setObject(in string obj);
string getObject();
string getResult();
void computeXML(); ; ;
```

The use of this IDL file presents some advantages. It can be easily implemented in both client and server, and the same interface can be used to exchange all types of data (XML plays the leading-role to represent each structure). The use of only one interface makes easier the running of CORBA, because for each object the IDL file must be in both client and server sides. Besides, if we work with Java or ML in the client, we can reuse the code we had written in the previous work for the local system.

The exchange format used is an XML dialect, a proper extension of MathML [7]. Some labels of MathML are used, such as `<cn>` for integers. However, the data of MathML are simpler than those of Kenzo, so some other labels have been created for other algebraic structures (for combinations we use `<cmb>`, for monomials `<mnm>`...). For instance, the combination (of degree 2) $4 * 5 + 2 * 6$ is represented as follows:

```
<cmb>                    </mnm>
<dgr>                    <mnm>
<cn> 2 </cn>             <coef>
</dgr>                   <cn> 2 </cn>
<list>                   </coef>
<mnm>                    <gnr>
<coef>                   <cn> 6 </cn>
<cn> 4 </cn>             </gnr>
</coef>                  </mnm>
<gnr>                    </list>
<cn> 5 </cn>             </cmb>
</gnr>
```

With the showed interface, it is not difficult to implement the IDL methods in the server. The main task consists on writting some "parsers", functions that translate XML strings to Kenzo objects and vice versa (some of them written previously for the local system). With the IDL implementation built, the server program is an easy sequence of orders to create the CORBA object and set it ready to be required by the clients.

For the clients' side, we also need to implement the parsers. It has been done for Common Lisp and Java, reusing again our code. The client programs

use CORBA to connect with the server and invoke its operations. First, the client builds the XML format of the operation wanted and assigns it to `obj`. Next, it invokes `computeXML` of the server and the server gets the representation, translates it to Lisp, computes it and assigns the XML of the solution to `result`. Finally, with `getResult` the client gets the result, and then translates it to a Kenzo object.

3 Examples

We consider now some examples with a Java client. It can be called in the command line, and to facilitate the communication with the user we have also written a Java applet.

As a first example, we consider the multiplication of a combination by an integer, which is invoked as it is showed below. Once computed, the wanted combination is showed on the screen.

```
C:\>java JavaClient "n-cmbn" "nil" 3 "cmbn" 2 4 5 2 6
Result: (Cmbn 2 (12 5) (6 6))
```

More complicated operations are those that require the ambient space where the computation is carried out (first layer of data structures). In the previous example, the operation is independent of the chain complex, so we define the ambient as "nil". However, the differential of a combination is defined in the chain complex, so we must specify the ambient. To encode this space, Kenzo uses functional programming, so this was a difficulty we found when representing this ambient in XML. To solve it, we encoded the ambient space with the tree of

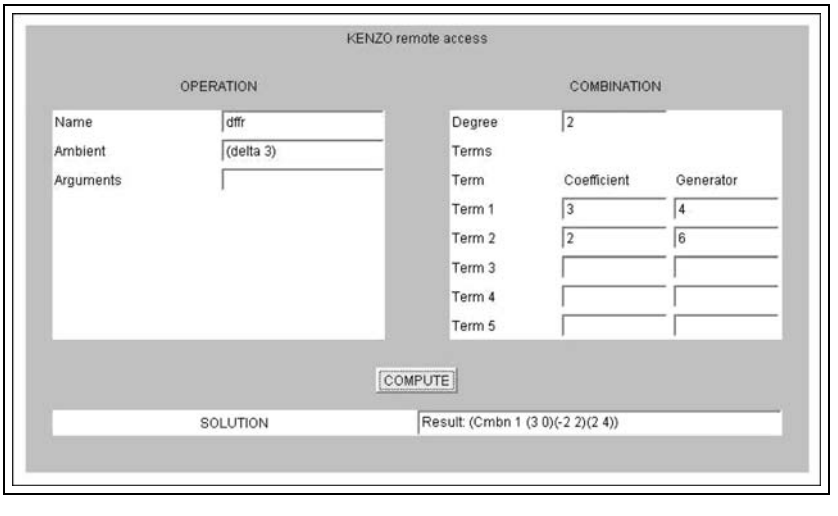

Fig. 1. Example: differential of a combination

calls that generates it. In this case, we compute the differential of a combination in the chain complex (delta 3).

```
C:\>java JavaClient "dffr" "(delta 3)" "cmbn" 2 3 4 2 6
Result: (Cmbn 1 (3 0) (-2 2) (2 4))
```

We can also use the Java Applet to introduce the parameters in several text boxes. In Figure 1 we include an image of the applet that corresponds to the differential of a combination presented before.

4 Conclusions and Further Work

In this paper we have presented a prototype based on CORBA that provides a remote access for Kenzo. The leading-role in the data exchange is played by an XML format which is a proper extension of MathML. The inclusion of XML inside a CORBA object allows the use of only one IDL for every structure in Kenzo, which makes easier the implementation and the running of CORBA. Moreover, with our IDL interface we can use the server program without any change, with only a little work for the wrapper of the system that allows making use of it from a CORBA object.

It is a small prototype that works only with few of all the algebraic structures that appear in Kenzo. Besides, the programs are totally based in CORBA, a technology designed for computers connected by a local net, so our system can not be used directly to interoperate in Internet.

Obviously, one of our short-term goals is to complete the prototype, to offer the total functionality of Kenzo. For this task, we only will have to design the XML representation for each structure and the parsers to translate them, but in the part of CORBA and IDL interfaces the work is already done, our XML format is generic. Another short-term goal is to try other technologies that allow us to work outside a local net.

The prototype built can be understood as a first step towards a distributed computation system, where some other technologies such as Web Services [1] (using for instance WSDL for the description of the object instead of IDL, and SOAP for the exchange of messages) or Grid Computing could be considered. The infrastructure could consist for example on several services cooperating in the same calculation, coordinated by a specific Web Service. Even if this work shows a first step in this direction, many difficult research questions should be addressed to reach this goal.

References

1. Alonso, G., Casati, F., Kuno, H., Machiraju, V.: Web Services. Concepts, Architectures and Applications. Springer, 2004.
2. Andrés, M., García, F. J., Pascual, V., Rubio, J.: XML-Based interoperability among symbolic computation systems. In Proceedings WWW/Internet 2003, Vol. II, Iadis Press (2003) 925-929.

3. Andrés, M., Pascual, V., Romero, A., Rubio, J.: Distributed computing in Algebraic Topology: first trials and errors, first programs. In e-proceedings IAMC 2004. http://www.orcca.on.ca/conferences/iamc2004/abstracts/04002.html
4. Object Management Group: Common Object Request Broker Architecture (CORBA). http://www.omg.org.
5. Rubio, J., Sergeraert, F., Siret, Y.: EAT: Symbolic Software for Effective Homology Computation. Institut Fourier, Grenoble, 1997. ftp://ftp-fourier.ujf-grenoble.fr/pub/EAT.
6. Dousson, X., Sergeraert, F., Siret, Y.: The Kenzo program. Institut Fourier, Grenoble, 1999. http://www-fourier.ujf-grenoble.fr/~sergerar/Kenzo/.
7. Ausbrooks, R. et al.: Mathematical Markup Language (MathML) Version 2.0. 2003. http://www.w3.org/TR/2001/REC-MathML2-20010221/.
8. Caprotti,O., et al (eds): The OpenMath standard. 2000. http://www.openmath.org/standard
9. Bray, T. et al. (eds): Extensible Markup Language (XML) 1.0. 2003. http://www.w3.org/TR/REC-xml/.