Using Open Mathematical Documents to interface Computer Algebra and Proof Assistant systems*

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. Mathematical Knowledge can be encoded by means of Open Mathematical Documents (OMDoc) to interface both Computer Algebra and Proof Assistant systems. In this paper, we show how a unique OMDoc structure can be used to dynamically generate, both a Graphical User Interface for a Computer Algebra system and a script for a Proof Assistant. So, the OMDoc format can be used for representing different aspects. This generic approach has been made concrete through a first prototype interfacing the Kenzo Computer Algebra system and the ACL2 Theorem Prover, both based on the Common Lisp programming language. An OMDoc repository has been developed allowing the user to customize the application in an easy way.

1 Introduction

OpenMath [2] is an XML standard widely adopted to express mathematical knowledge. In [9] we presented an architecture based on OpenMath and allowing, in principle, to dynamically load new modules in a Graphical User Interface (GUI) for the Kenzo system [3] (a Common Lisp system devoted to Symbolic Computation in Algebraic Topology). The main obstacle to plug-in dynamically new modules, was that OpenMath content dictionaries (the OpenMath technology used in that paper) are not designed to store code parts. So, in our new development we have moved from OpenMath content dictionaries to Open Mathematical Documents, OMDoc [11], which allow us to encode information on both the user interface specification and the code, carrying out the functionality of the GUI. This fulfills our objective of dynamically loading new modules into the GUI.

In addition, we realized that it was possible to take advantage of the OMDoc technology to represent different kinds of information allowing us to extend our system by including deduction capabilities. In order to do this we have exploited the content dictionaries' capacities for representing richer mathematical

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

knowledge. To be precise, each mathematical structure used in Kenzo has been represented by means of an algebraic specification which has been embedded in OMDoc documents. These OMDoc documents are the basis to construct some *encapsulates* in the ACL2 theorem prover [10]. ACL2 is a system for proving properties of programs written in (a subset of) Common Lisp.

Thus, from some OMDoc documents all the pieces needed to dynamically customize a GUI and to integrate a Computer Algebra system, Kenzo [3], with a Proof Assistant, namely ACL2 [10], can be generated. Therefore, the definitions and examples included in the OMDoc documents can be formally validated.

2 Specifying with Open Mathematical Documents

The Kenzo system works with the main mathematical categories used in Combinatorial Algebraic Topology, [12]. In [9], a framework wrapping Kenzo with a Phrasebook as external interface was developed, so the unique possible interaction with it is by means of OpenMath. In addition, an OpenMath content dictionary [2] was defined for each mathematical category which the Kenzo system works with.

In order to make the development of clients of our framework easier, we have intended to supplement it with information related to both the user interaction and functionality of the user interface. For this task, we have used OMDoc.

The OMDoc [11] format is an open markup language for mathematical documents and the knowledge encapsulated in them which allows for the representation of three levels of information in mathematical knowledge: formulæ, mathematical statements and mathematical theories. Different sub-languages including only part of the OMDoc functionality have been specified, and the respective modules developed. We have focused on three of them, namely the *Basic OMDoc*, *OMDoc content dictionaries* and *MathWeb OMDoc* sub-languages; the complete list of sub-languages and their descriptions can be found in [11].

The first task we have had to deal with has been to specify with OMDoc documents the basic mathematical structures used in Kenzo as well as the Kenzo functionality itself. On the one hand, the OpenMath content dictionaries developed for the Kenzo system provide not only the different objects used in the Kenzo system (spheres, Moore spaces, loop spaces and so on) but also a specification of the mathematical structures that they represent. So, the signature (which consists of the headers of the functions) and their formal properties (this will be useful to interact with theorem provers) are included in the content dictionaries. The sub-language for OMDoc content dictionaries allows us to specify the meaning of basic mathematical objects (symbols) by axioms and definitions; and grouping them, it is possible to refer to the symbols defined via their theory. In general, OMDoc content dictionaries can add some functionalities with respect to the OpenMath content dictionaries, so our previous OpenMath content dictionaries can be embedded into OMDoc content dictionaries. On the other hand, an OMDoc document including the functionality of our framework (related to that mathematical structure) has been written. This functionality

has been included into the OMDoc document by using the `<code>` tag of the EXT OMDoc Module, which is aimed at embedding pieces of program code into an OMDoc document. This OMDoc document can be interpreted as a Kenzo wrapper.

Starting from these OMDoc documents, we have specified both the interaction and functionality of a user interface for Kenzo and the integration with the ACL2 Theorem Prover.

2.1 Generating dynamically a GUI for the Kenzo system

Thinking about increasing the usability of Kenzo, a GUI has been developed to interact with the system, making the interaction with the user easier. The first GUI presented, detailed in [8], allowed the user to build different spaces (like spheres, Moore spaces, loop spaces and so on) and compute homology and some homotopy groups, using the Kenzo system as its kernel. Although it worked in a correct way, it had an extensibility problem. At this moment the Kenzo system keeps on growing. So it would be desirable that our GUI could evolve with Kenzo. To add new functionality to the Kenzo system, a file with the new functions must be included; the Kenzo main code is not modified. It is worthwhile having an extensibility system for the GUI that consists in including only a file with the new functionality, as it is done in Kenzo itself.

The first approach consisted in extracting all the functionality included in the first version of the GUI, changing from a static to a dynamic GUI. So the starting point was a meaningless GUI, and OpenMath content dictionaries dealt with the evolution of the interface itself: when loading a content dictionary, the interface changed, with new options appearing in the toolbar. Each content dictionary had a module associated with it, including the extension of the system, both the GUI and the functionality. Even if this extensibility way worked in a correct way, it had a drawback: adding the necessary modules to our GUI in order to extend its functionality had to be programmed in Allegro Common Lisp [4].

In [6], a proposal for the declarative programming of user interfaces (UI) was presented with the aim of abstracting the ingredients for high-level UI programming. To be precise, three constituents are distinguished: *structure*, *functionality* and *layout*. To be based on the previous proposal, the structure of our GUI is provided by XUL (XUL, [7], 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 meaningless GUI.

Some OMDoc documents being based on the MathWeb sub-language and containing all the information needed to dynamically add the Kenzo functionality to the GUI have been defined. For each Kenzo mathematical structure, an OMDoc document including the definition of the GUI corresponding to the specific mathematical structure has been written. This OMDoc document contains the structure, functionality and layout of our GUI. In order to include

the XUL containing the structure and the layout, an OpenMath Foreign object (`<OMForeign>`), which allows us to associate NON-OpenMath objects with OpenMath objects, has been used. With respect to the functionality, that is, the event handlers, it has been included into a `<code>` tag again.

Note that the `<code>` tag would be able to include code for different applications allowing us to build several UIs. This last aspect is related to transform our desktop application into a Web User Interface.

2.2 An interpreter from OpenMath to the ACL2 Theorem Prover

With the aim of including some deductive capabilities in our system, we have added, in our content dictionaries, the properties which the mathematical structures encoded in our OMDoc documents must really satisfy. This opens the possibility of interfacing our system with the Common Lisp theorem prover ACL2 (some aspects of Simplicial Topology has already been formalized in [1]).

ACL2 [10] supports the constrained introduction of new function symbols by means of the *encapsulate* notion. An *encapsulate* in ACL2 is composed of a set of function signatures, a set of properties of these functions and a "witness" for each one of the functions, where a witness is an existing function that can be proved to have the required properties.

We have included signatures in OpenMath object definitions. In addition, we have specified their properties in two different ways (by means of `<FMP>` and `<CMP>` tags) and we have associated an instance example with them. Gathering together all the previous aspects, it is possible to include in the content dictionaries all the needed information to generate an ACL2 encapsulate from an OMDoc content dictionary.

By using the OMDoc content dictionaries sub-language to define the objects, an interpreter which transforms each OMDoc content dictionary into an executable encapsulate in ACL2 has been developed.

The necessary functions to transform the OMDoc content dictionaries into the respective ACL2 encapsulates are stored in an OMDoc document which is based on the MathWeb sub-language. By collecting all OMDoc documents of this kind a specific purpose interpreter from OpenMath to ACL2 is obtained.

3 Integrating all the pieces

Finally, the Basic OMDoc sub-language is sufficient for mathematical documents that do not introduce new symbols or concepts. In our system, a Basic OMDoc document glues the different documents associated with a specific mathematical structure together. To be precise, for a specific mathematical structure three different OMDoc documents are provided: the first one gives an algebraic specification of the mathematical structure using the OMDoc content dictionaries, the second one supplies the functions to build these mathematical structures in our system (abstracting the ones of Kenzo), and the last one defines the GUI that can be loaded as a new module of our main GUI. These documents try to

make the interaction with Kenzo easier. In addition, some OMDoc documents represent the integration with other systems. These documents can be considered as interpreters from Kenzo (by means of OMDoc) to the specific system. To sum up, both mathematical knowledge and different kinds of interactions have been specified by means of OMDoc documents.

From these different kinds of documents, *templates* customizing the system can be generated. If a user wants to work with a specific structure of the Kenzo system, he must create a document that links to the documents that provide the corresponding content dictionaries, the necessary Kenzo functionality and the part of the GUI which must be added to the meaningless GUI. Besides, if he wanted to interact with another system, he must supply an interpreter from Kenzo OMDoc documents to the system and a client to interact (for instance, a GUI, a web service, and so on). If some of the OMDoc documents is not available, the user can develop them and in this way the system grows up. In addition the different templates, that is, an OMDoc grouping all the OMDoc documents needed for an specific interaction, can also be added to the repository to be used by any other clients.

To define these templates the Basic OMDoc sub-language is used. Namely, the *DOC* module provides the document infrastructure (in particular, the `<omgroup>` tag allows us to group the references to other documents) and the *DC* module supplies the metadata.

Now, we can present a concrete example which integrates all the pieces explained in the previous sections. On the one hand, we want to be able to work with the simplicial sets using the Kenzo functionalities, for instance, compute their homology and homotopy groups. On the other hand, we want to use ACL2 to prove that the simplicial sets built by Kenzo (spheres, Moore Space, cartesian product and so on), and used in our system, are really simplicial sets. In this way, representation, computation and deduction will be integrated in the same system.

In our OMDoc repository, we can find almost all the ingredients to customize our application to achieve our objective. The relations among the components and their role in the workflow of our system customization in this example is shown in Figure 1. For the 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. 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.

As can be seen in Figure 1, the only interaction made by the user consists in loading the Basic OMDoc document. The numbers in the diagram indicate the execution workflow.

Figure 2 shows a customized Kenzo GUI for simplicial sets integrating and ACL2 GUI.

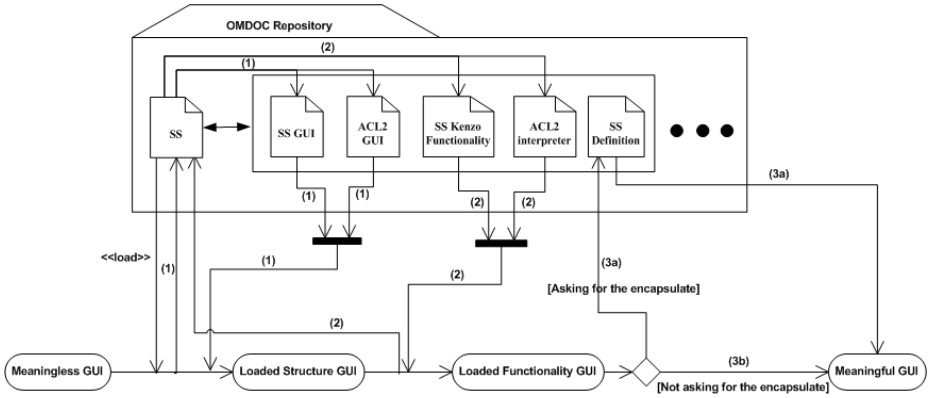

Fig. 1. Workflow diagram.

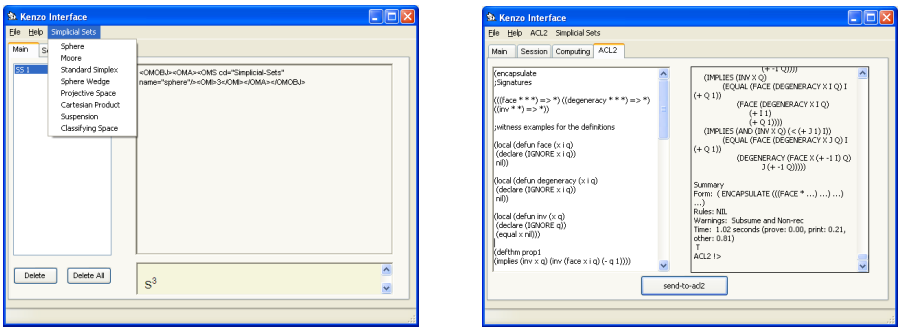

Fig. 2. Screen-shots of our customized Kenzo GUI.

All this work is made without any changes in the code of our previous framework. The only thing that must be done consists in loading an OMDoc document, containing all the necessary information to customize the application adding the new functionality.

4 Conclusions and Further Work

In this paper we have reported on an OMDoc documents repository. This repository is composed of several OMDoc documents which have been defined using different OMDoc sub-languages to reach different goals. On the one hand, some OMDoc documents, based on the OMDoc content dictionaries sub-language, supply the mathematical structures of our system. On the other hand, OMDoc documents in the MathWeb sub-language provide us with the necessary tools to specify user interfaces, the functionality of these interfaces, the functionality of the system itself and also the interaction with other systems.

As an example of the use of this repository we have described a first prototype that allows an integration of the Kenzo computer algebra system and the

ACL2 theorem proving system. The interaction part of our OMDoc documents generates new modules in the GUI, and the axiomatic part generates an *encapsulate* in ACL2, allowing us to check, in an automated way, that the properties are consistent. 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). Other prototypes can be developed in order to integrate our system with different mathematical systems (for instance, GAP [5] has already been connected with Kenzo through OpenMath in [13]).

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 prepare an ACL2 script stating that this concrete (Common Lisp) object is a (functional) instance of the encapsulate `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.

References

1. Andrés M., Lambán L., Rubio J., Ruiz Reina J. L., *Formalizing simplicial topology in ACL2*, In *Proceedings Workshop ACL2* 2007, Austin University, 34 – 39.
2. Buswell S., Caprotti O., Carlisle D.P., Dewar M.C., Gaëtano M., Kohlhase M. *OpenMath* Version 2.0, 2004. `http://www.openmath.org/`.
3. Dousson X., Sergeraert F., Siret Y., *The Kenzo program*, Institut Fourier, Grenoble, 1999. `http://www-fourier.ujf-grenoble.fr/~sergerar/Kenzo/`.
4. Franz Inc. *Allegro Common Lisp*. `http://www.franz.com`.
5. GAP - Groups, Algorithms, Programming - a System for Computational Discrete Algebra. `http://www.gap-system.org/`.
6. Hanus M., Kluß C., *Declarative Programming of User Interfaces*, PADL 2009, Lectures Notes in Computer Science, 5418 (2009) 16–30.
7. Hyatt D. et al., *XML User Interface Language (XUL) 1.0* `http://www.mozilla.org/projects/xul/`.
8. Heras J., Pascual V., Rubio J., *Mediated Access to Symbolic Computation Systems*, MKM 2008, Lectures Notes in Artificial Intelligence, 5144 (2008) 446–461.
9. Heras J., Pascual V., Rubio J., *Mediated Access to Symbolic Computation Systems: An OpenMath Approach*. Preprint.
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. May J.P., *Simplicial objects in Algebraic Topology*, Van Nostrand Mathematical Studies (11), (1967).
13. 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.