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 generate dynamically, both a Graphical User Interface for a Computer Algebra system and a script for a Proof Assistant. So, the OMDoc format can be used to represent different aspects. This generic approach has been made concrete through a first prototype interfacing of 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 is an XML standard widely adopted to express mathematical knowledge. Nevertheless, up to now, its possibilities seem underestimated. It is mostly used as a partner of MathML, [2], providing a mechanism for describing the semantics of the MathML presentation format for mathematical objects. Besides it has been successfully employed to interoperate among Computer Algebra systems [4]. Being important, this use of OpenMath looses the reasoning possibilities given by content dictionaries, where axioms and properties of structures can be encoded.

As an example of this *weak* use of OpenMath we could show our own work, about giving an intelligent mediated access to the Kenzo [7] symbolic computation system. In [13] 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. 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

* 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, project MTM2006-06513.

Mathematical Documents, OMDoc [16], 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 loading dynamically 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 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 dealing with simplicial and algebraic structures. These OMDoc documents are then used to construct some *encapsulates* in the ACL2 theorem prover [14]. 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 [7], with a Proof Assistant, namely ACL2 [14], can be generated. So, the definitions and examples included in the OMDoc documents can be formally validated.

So, providing OMDoc documents deal with specifying the mathematical structures used in a system together with the ones that specifying interfaces and the ones that define the interaction, it is possible to interoperate with Computer Algebra or Proof Assistant systems by means of OMDoc. In particular, we have enhanced our Kenzo GUI [13] to include an experimental ACL2 interface. 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 proving).

The organization of the paper is as follows. In the next section, we explain what our OMDoc documents are specifying. In Section 3, the way of representing a dynamical GUI by means of some OMDoc documents is explained. The same task with respect to the deductive part and the ACL2 system is undertaken in Section 4. The way in which all the pieces are integrated is presented in Section 5. The paper ends with Conclusions, Further Work and the Bibliography.

2 Specifying with Open Mathematical Documents

The Kenzo system works with the main mathematical categories used in Combinatorial Algebraic Topology, [17]. In [13], a framework wrapping Kenzo with a Phrasebook as external interface, was developed, so the unique possible interaction with it is by means of OpenMath. This framework is shown in Figure 1. In addition, an OpenMath content dictionary [3] was defined for each mathematical category which Kenzo system works with.

As we already commented at the beginning of this paper, it is well-known that OpenMath is aimed at expressing mathematical knowledge but not any other kind of interaction.

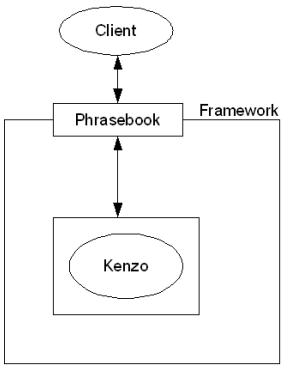


Fig. 1. An skeleton of the framework wrapping Kenzo.

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.

OMDoc [16] format is an open markup language for mathematical documents and the knowledge encapsulated in them. One of its goals is aimed at representing mathematical documents on the web. For this, OMDoc format allows for the representing of the three levels of information in mathematical knowledge: formulæ, mathematical statements and mathematical theories. OMDoc format is made up of several modules, each one devoted to some concrete tasks. Different sub-languages, related to them, including only part of the OMDoc functionality have been specified.

In order to specify the interaction and functionality of a user interface for Kenzo, 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 [16].

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 concrete 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, as we will show in Section 4) 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.

With respect to the user interaction and interface functionality specification, we have used the MathWeb OMDoc sub-language. It has been specified as a content-oriented basis for web publishing of mathematics. Our use of this sub-

language is not related to this topic, but we want to take advantage of the infrastructure for images, applets, code fragments, and other data provided by this sub-language. To be precise we have used the `<code>` tag to embed Common Lisp code in our OMDoc documents. Thus, several OMDoc documents have been generated for different tasks, each one of them having different proposals and utilities but sharing the same structure and format.

An OMDoc document includes the functionality of our framework so it can be interpreted as a Kenzo wrapper. With the same format, other OMDoc documents embed the necessary code to interact with other systems, as we will see in Section 4.

With respect to the GUI, several OMDoc documents have been defined. On the one hand, one OMDoc document contains the graphical elements and another one contains their event handlers. On the other hand, the interface of the interaction with other systems has been defined in another OMDoc, as we will explain in Section 5.

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 mathematical structure together. For instance, in the case of simplicial sets, which will be used in the following sections to illustrate the ideas, three different OMDoc documents are provided: the first one gives an algebraic specification of the simplicial sets using the OMDoc content dictionaries, the second one supplies the functions to build simplicial sets in our system (abstracting the ones of Kenzo), and the last one is used to define the GUI that can be loaded as a new module of our main GUI. The Simplicial Set Basic OMDoc Document contains the links to these documents, clustering all the mathematical information about simplicial sets that appear in our system.

To sum up, OMDoc has been used to specify both mathematical knowledge and different kinds of interactions.

3 Generating a GUI for the Kenzo system dynamically

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 [12], 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. There are several researchers that are developing new functionalities for it; for instance, spectral sequences [19], resolutions [18], Koszul complexes [20], and so on. 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. However, to increase the functionality of our GUI, its code must be modified, and obviously, this is a problem. 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, which looked like that of Figure 2, 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 must be programmed in Allegro Common Lisp [8].

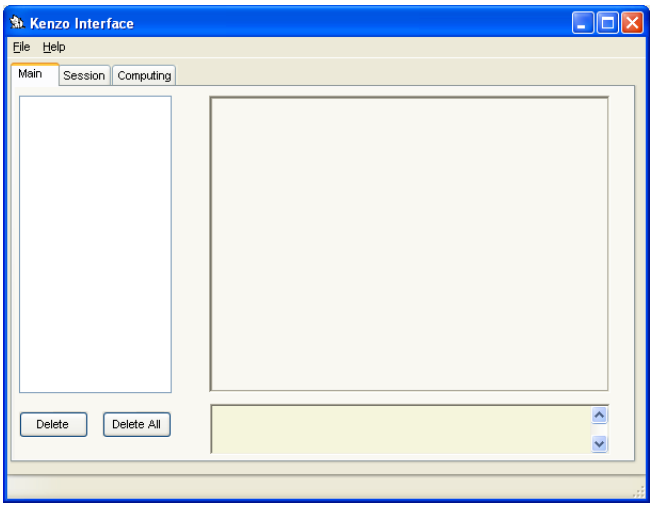


Fig. 2. Screen-shot of a meaningless Kenzo Interface.

In [10], Hanus and Kluß presented a proposal for the declarative programming of user interfaces (UI) with the aim of abstracting the ingredients for high-level UI programming. To be precise, three constituents are distinguished: *structure*, *functionality* and *layout*. Each UI has a specific hierarchical *structure* which typically consists in basic elements (like text input fields or selection boxes) and composed elements (like dialogs); when a user interacts with a UI, some events are produced and the UI must respond to them. The event handlers must include the necessary *functionality*. These elements are put in a *layout* to achieve a visually appealing appearance of the UI. The Curry language [9] is used to declare all the ingredients in that work. Instead of doing a similar work, but using the Lisp language, we have preferred to define the structure of our GUI which is based on XUL (XML User Interface) [11]. XUL is Mozilla's XML-based

user interface language that lets us build feature rich cross-platform applications defining all the elements of a UI.

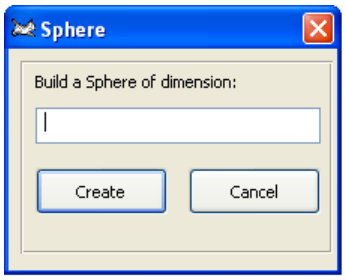
To be based on the previous ones, the structure of our GUI structure is provided by XUL, functionality has been programmed in Allegro Common Lisp and the default layout has been used, although we could have used a style sheet to customize our application. Thus, we have all the ingredients to extend our 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, two OMDoc documents have been written. The first one can be seen as a wrapper of the Kenzo functionality for that mathematical structure. 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. For instance, the next fragment of code is an example of the `<code>` tag use.

```
<code id="create-sphere">
    <metadata>
        <description>the function create-sphere has one
            argument as input, the dimension of the sphere,
            and builds the sphere of that dimension in the
            system.
        </description>
    </metadata>
    <data format="application/Kenzo-wrapper">
        <![CDATA[... << the create-sphere code goes here >>...]]>
    </data>
    <input>
        <CMP>A integer with the dimension of the sphere.</CMP>
    </input>
    <output>
        <CMP>A integer that identifies the sphere</CMP>
    </output>
    <effect>
        <CMP>None.</CMP>
    </effect>
</code>
```

The second one includes the definition of the GUI corresponding to the specific mathematical structure. 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 to associate NON-OpenMath objects with OpenMath objects, has been used. Figure 3 shows an example of the definition of the sphere dialog with its XUL code. With respect to the functionality, which is the event handlers, it has

been included into a <code> tag again. Figure 4 shows our GUI with the simplicial sets functionality loaded; a new menu has come out allowing the user to build simplicial sets, the interaction of the interface can be seen in [13].




```
<OMForeign>
  <window name="Sphere">
    <groupbox>
      <label value="Build a Sphere of dimension:"/>
      <textbox id="n" type="number" min="1" max="14"/>
      <hbox>
        <button label="Create" onclick="create-sphere-on-click"/>
        <button label="Cancel" onclick="cancel-sphere-on-click"/>
      </hbox>
    </groupbox>
  </window>
</OMForeign>
```

Fig. 3. The sphere dialog with its XUL description.

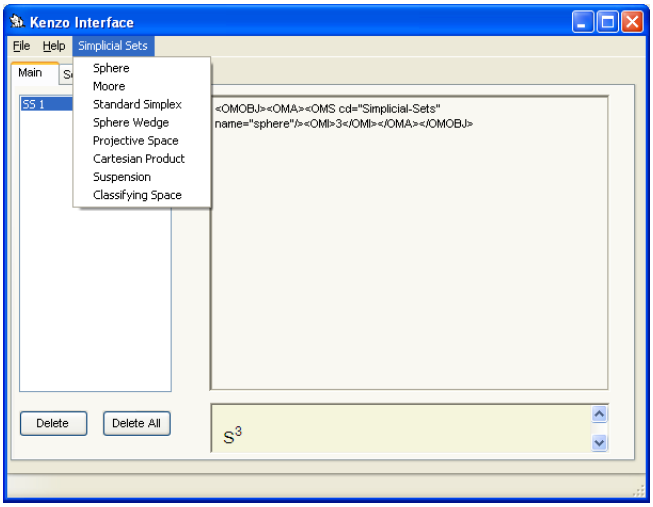


Fig. 4. Screen-shot of Kenzo Interface for simplicial sets.

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. Usually the applications developed using XUL have got associated Javascript code and can be executed in different operating systems and in the web. So, by including this Javascript code in our OMDoc document, a simple program would be able to extract the

structure definition and the Javascript code to obtain an application running in the web and in different operating systems.

Thus, thanks to these two OMDoc documents, new functionality can be added to our meaningless GUI in an easy way. Defining another OMDoc document that links both of them is the only thing that must be done. In this way, knowing the XUL schema for any UI and how to interact with the system, an extensibility level similar to that of the Kenzo system is reached; that is, new functionality can be included by means of a unique file. A concrete example of the topics explained in this Section will be shown in Section 5.

4 An interpreter from OpenMath to the ACL2 Theorem Prover

As a different direction of research, we are trying to take more advantage of the semantical possibilities of OpenMath. Concretely, 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 [14], which has already been used to formalize some aspects of Simplicial Topology [1]. A similar approach, but using the proof checkers Lego and Coq instead of the ACL2 theorem prover, was proposed by Caprotti and Cohen in [5].

ACL2 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 based our work on the Small Type System formalism, see [6] for details, which has been designed to give semi-formal signatures to OpenMath symbols. By using this mechanism 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.

4.1 A case study: simplicial sets

Definition 1. A *simplicial set* K, [7], is a disjoint union $K = \bigcup_{q \geq 0} K^q$, *where the* K^q are sets, together with functions

$$\partial_i^q : K^q \to K^{q-1},\ q > 0,\ i = 0, \ldots, q,$$
$$\eta_i^q : K^q \to K^{q+1},\ q \geq 0,\ i = 0, \ldots, q,$$

subject to relations

$$\begin{array}{ll}
\partial_i^{q-1}\partial_j^q = \partial_{j-1}^{q-1}\partial_i^q, & i<j \\
\eta_i^{q+1}\eta_j^q = \eta_j^{q+1}\eta_{i-1}^q, & i>j \\
\partial_i^{q+1}\eta_j^q = \eta_{j-1}^{q-1}\partial_i^q, & i<j \\
\partial_i^{q+1}\eta_i^q = \partial_{i+1}^{q+1}\eta_i^q, & \textit{identity} \\
\partial_i^{q+1}\eta_j^q = \eta_j^{q-1}\partial_{i-1}^q, & i>j+1
\end{array}$$

The functions ∂ and η are called the *face operators* and the *degeneracy operators* respectively.

In a first step, a content dictionary for simplicial sets has been developed. To define a simplicial set, we must provide the disjoint sets $\{K^q\}_{q\geq 0}$ and the face and degeneracy operators. With respect to the underlying set of a simplicial set K, the union $\bigcup_{q\geq 0} K^q$ can be seen as a graded set, and it is possible to associate to it, a characteristic function which, from an element x and a degree g, determines if the element belongs to the set K^g. This is the Kenzo way of representing the underlying set of a simplicial set. To be precise, an *invariant* function encodes the characteristic function of its underlying set K.

Being based on the previous way of representation, the following OpenMath signature has been defined for simplicial sets.

```
<Signature name="simplicial-set">
    <OMOBJ>
        <OMA>
            <OMS name="mapsto" cd="sts"/>
            <OMA id="face">
                <OMS cd="sts" name="mapsto"/>
                <OMS cd="sts" name="Object"/>
                <OMS cd="sts" name="PositiveInteger"/>
                <OMS cd="sts" name="PositiveInteger"/>
                <OMS cd="sts" name="Object"/>
            </OMA>
            <OMA id="degeneracy">
                <OMS cd="sts" name="mapsto"/>
                <OMS cd="sts" name="Object"/>
                <OMS cd="sts" name="PositiveInteger"/>
                <OMS cd="sts" name="PositiveInteger"/>
                <OMS cd="sts" name="Object"/>
            </OMA>
            <OMA id="inv">
                <OMS cd="sts" name="mapsto"/>
                <OMS cd="sts" name="Object"/>
                <OMS cd="sts" name="PositiveInteger"/>
                <OMS cd="setname2" name="boolean"/>
            </OMA>
            <OMV name="Simplicial-Set"/>
```

```
        </OMA>
    </OMOBJ>
</Signature>
```

The formal mathematical properties of the simplicial sets are given in the <FMP> tag of the simplicial-set definition. In this case the <FMP> element states the properties of invariance of face and degeneracy operators and the relations among them. We have also included the mathematical properties in natural language using the <CMP> tags. For instance, the face operator invariance has been represented as follows.

```
<CMP> The face operator is invariant </CMP>
<FMP>
...
<OMA>
    <OMS cd="logic1" name="implies"/>
    <OMA>
        <OMS  name="inv"/>
        <OMV name="x"/>
        <OMV name="q"/>
    </OMA>
    <OMA>
        <OMS  name="inv"/>
        <OMA>
            <OMV  name="face"/>
            <OMV name="x"/>
            <OMV name="i"/>
            <OMV name="q"/>
        </OMA>
        <OMA>
            <OMS cd="arith2" name="minus"/>
            <OMV name="q"/>
            <OMI>1</OMI>
        </OMA>
    </OMA>
</OMA>
...
</FMP>
```

Finally, an instance example of simplicial set has been included. In this case we have considered the simplicial set whose sets only have an element, `nil`, and each face operations of degree q returns the element of degree $q-1$.

```
<example>
...
    <OMBIND>
        <OMS name="face"/>
```

```
        <OMBVAR>
            <OMV name="x"/>
            <OMV name="i"/>
            <OMV name="q"/>
        </OMBVAR>
        <OMS cd="list" name="nil"/>
    </OMBIND>
...
</example>
```

From this content dictionary, an ACL2 encapsulate can be generated. From now on, we are going to explain the transformation from our OMDoc content dictionary to an ACL2 encapsulate; both of them in the simplicial sets case.

Each application `OMA` inside the main `mapsto` of the simplicial set signature is translated into a function of the encapsulate, in the following way. The `id` of the application tag will be the name of the function, the `mapsto` is converted to `=>` in ACL2. The `mapsto` symbol is applied to n variables and/or symbols, the first $n-1$ will be the inputs and the last one the output. Note that the ACL2 system is a system without explicit typing, so, although the type of the objects has been included in the OMDoc content dictionary they will be translated into asterisks in ACL2. Adding the necessary brackets, the following ACL2 signature is obtained.

```
(((face * * *) => *)
 ((degeneracy * * *) => *)
 ((inv * *) => *))
```

The following step is to transform the mathematical properties, specified into the content dictionary, into ACL2 lemmas. In order to do this, we proceed in the following way. First of all, the `<CMP>` tags will be translated into comments in ACL2, expressed with "`;`". To each `<FMP>` tag, a new lemma, `defthm` in ACL2 syntax, with the name `prop-n` where `n` is a variable that indicates the number of the property, must be defined. The `implies`, `and`, `eq` and `minus` OpenMath symbols are translated respectively into the `implies`, `and`, `equal` and `-` ACL2 functions. For instance, the following ACL2 lemma about the face operator invariance is obtained from the respective one in the content dictionary.

```
; The face operator is invariant
(defthm prop1
(implies (inv x q) (inv (face x i q) (- q 1))))
```

And, finally, from the `<example>` tags, the witnesses is obtained. Each example in a content dictionary will be a local definition in an ACL2 encapsulate. The `OMBIND` symbol indicates the beginning of the definition, in ACL2 `defun`, its first argument is a symbol indicating the name of the function, the second one is an `OMBVAR` representing the name of the parameters and the third one is the body. If some of the arguments of the function do not appear in its body, they

will be ignored to obtain a correct ACL2 function, as we show in the translation of the previous example:

```
(local (defun face (x i q)
 (declare (IGNORE x i q))
 nil))
```

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 in the same way that we explained in Section 3. By collecting all OMDoc documents of this kind an interpreter from OpenMath to ACL2 is obtained.

5 Integrating all the pieces

Up to now, we have developed different OMDoc documents that can be used to achieve different goals. The following task was to use these documents to improve the functionality of our framework, adapting it to the needs of different users without modifying the main system.

Gathering together all the OMDoc documents developed, an *OMDoc repository* is obtained. OMDoc documents with two different intentions have been included. On the one hand, 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. On the other hand, other OMDoc documents try to make the interaction with Kenzo easier. There are OMDoc documents representing some of the mathematical structures presented in Kenzo, for instance the simplicial sets, using the content dictionary format. These are the basic OMDoc documents for the system. Another kind of OMDoc document provides the Kenzo functionality, so, in a sense they can be seen as a wrapper of the phrasebook of our framework (see Figure 1). And finally, from our meaningless GUI, different OMDoc documents allow us to dynamically generate different GUIs, each one of them containing, a part of the Kenzo functionality.

From these different kinds of documents *templates* that customize 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 (see Figure 2). 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 are 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 consider the following scenario 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. For the simplicial sets, (1) an OMDoc content dictionary that defines their mathematical structure (explained in Section 4), (2) the logic to interact with Kenzo and (3) the presentation for the GUI (both explained in Section 3) are available. With respect to ACL2, (4) an interpreter which is able to translate from the OMDoc content dictionary for simplicial sets into an ACL2 encapsulate can be found. The only thing still missing is the present user interface, allowing the ACL2 system to interact with our system. In this case, the needed OMDoc document (5) 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, have been developed. The new tab page contains two areas and one button: the first area will be used to write the ACL2 instructions, the button will send the instructions to ACL2, and finally the second area will show the ACL2 result. Note that, the encapsulate related to simplicial sets is written (dynamically, from the corresponding content dictionary) into the left area and the answer of ACL2 after evaluating it, appears in the right zone.

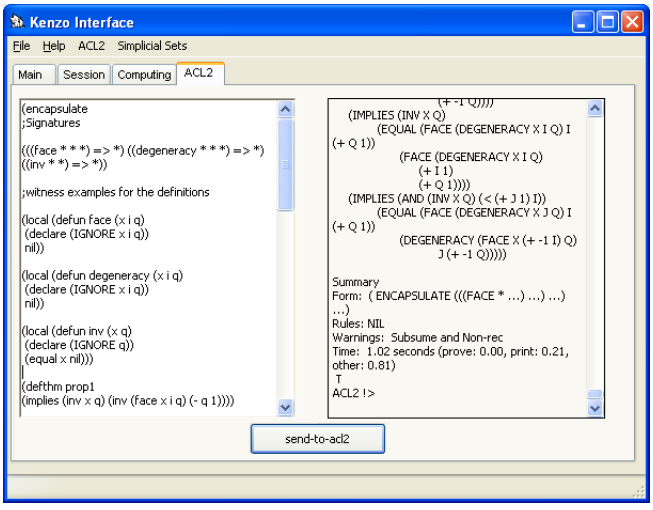


Fig. 5. Customized GUI.

To sum up, the *head* document used to customize the application refers to the previous five documents, the documents containing the mathematical definition, functionality and a way of presentation of the simplicial sets, and the ACL2 interpreter and a way of presenting the results. In this case, to build this OMDoc, we must provide the metadata (authorship, title and so on) and the reference to the different documents;

```
<omgroup type="sequence">
    <ref xref="simplicial-sets-conceptual-model"/>
    <ref xref="simplicial-sets-logic"/>
    <ref xref="simplicial-sets-presentation"/>
    <ref xref="acl2-logic"/>
    <ref xref="acl2-presentation"/>
</omgroup>
```

When a user exits the application, his configuration is saved for future sessions.

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, containing all the necessary information to customize the application adding the new functionality.

6 Conclusions and Further Work

In this paper we have reported on an OMDoc repository. This repository is composed of several OMDoc documents which have been defined using different OMDoc sub-languages and have also been used to reach different goals. On the one hand, some OMDoc documents based on OMDoc content dictionaries sub-language, supply the mathematical structure of our system. On the other hand OMDoc documents in 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.

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, encapsulates should be generated for the rest of the mathematical structures appearing in Kenzo (algebras, reductions, and so on). More important, 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 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. Ausbrooks R. et al., *Mathematical Markup Language (MathML)* Version 3.0 (third edition), 2008. `http://www.w3.org/TR/MathML3/`.
3. Buswell S., Caprotti O., Carlisle D.P., Dewar M.C., Gaëtano M., Kohlhase M. *OpenMath* Version 2.0, 2004. `http://www.openmath.org/`.
4. Caprotti O. et al., *Using OpenMath Servers for Distributing Mathematical Computations*, ATCM 2000: Proceedings of the Fifth Asian Technology Conference in Mathematics (2000) 325–336.
5. Caprotti O., Cohen A., *Connecting proof checkers and computer algebra using OpenMath*, THPOLs 1999, Lectures Notes in Computer Science (1999) 109–112.
6. Davenport J., *A Small OpenMath Type System*, April 1999, `http://www.openmath.org/standard/sts.pdf`
7. Dousson X., Sergeraert F., Siret Y., *The Kenzo program*, Institut Fourier, Grenoble, 1999. `http://www-fourier.ujf-grenoble.fr/~sergerar/Kenzo/`.
8. Franz Inc. *Allegro Common Lisp*. `http://www.franz.com`.
9. Hanus M., *Curry: An Integrated Functional Logic Language (Vers. 0.8.2)* (2006), `http://www.curry-language.org`.
10. Hanus M., Kluß C., *Declarative Programming of User Interfaces*, PADL 2009, Lectures Notes in Computer Science, 5418 (2009) 16–30.
11. Hyatt D. et al., *XML User Interface Language (XUL) 1.0* `http://www.mozilla.org/projects/xul/`.
12. Heras J., Pascual V., Rubio J., *Mediated Access to Symbolic Computation Systems*, MKM 2008, Lectures Notes in Artificial Intelligence, 5144 (2008) 446–461.
13. Heras J., Pascual V., Rubio J., *Mediated Access to Symbolic Computation Systems: An OpenMath Approach.*. Preprint.
14. Kaufmann, M., Manolios P., Moore, J., *Computer-Aided Reasoning: An Approach. Kluwer Academic Press*, Boston (2000).
15. Kaufmann M., Moore J., *Structured theory development for a mechanized logic*, Journal of Automated Reasoning, 26(2) (2001) 161-203.
16. Kohlhase M., *OMDoc – An open markup format for mathematical documents [Version 1.2]*, Springer Verlag (2006).
17. May J.P., *Simplicial objects in Algebraic Topology*, Van Nostrand Mathematical Studies (11), (1967).
18. Romero A., Ellis G., Rubio J., *Interoperating between Computer Algebra systems: computing homology of groups with Kenzo and GAP*. Preprint.
19. Romero A., Rubio J., Sergeraert F., *Computing Spectral Sequences*. Journal of Symbolic Computation 41(10) (2006) 1059-1079.
20. Rubio J., Sergeraert F., *Constructive Homological Algebra and Applications* Genova MAP Summer School (2006).