FIGUE: MATHEMATICAL FORMULA LAYOUT WITH INTERACTION AND MATHML SUPPORT

H. NACIRI AND L. RIDEAU

INRIA Sophia Antipolis, BP. 93, 06902 Sophia Antipolis Cedex, France.
E-mail:{Hanane.Naciri,Laurence.Rideau}@sophia.inria.fr

Tools for mathematics often need to display formulas, to interact with them, and also make them easily accessible on the Internet. In this paper, we present FIGUE, an interactive two dimensional layout engine specialised for building WYSIWYG environments for symbolic systems and scientific document editors. This component is currently used for developing a mathematical proofs interface. In order to communicate mathematical data on the Internet, FIGUE also supports MathML standard: it generates, processes, and renders MathML documents.

1 Introduction

The development of WYSIWYG (*What You See Is What You Get*) mathematical interfaces requires significant effort due to the sophisticated data structures needed for representing mathematical objects. These interfaces should ensure some important functionalities such as providing ways to edit and display formulas and also providing a simple and natural way to interact with them. They become more interesting if they make mathematical data easily accessible on the Internet, enabling to share and exchange these data with other users and between different applications. In this perspective, we have designed FIGUE [1,2], a software component dedicated to the formatting and the interactive displaying of structured objects in a document containing mathematical formulas. FIGUE is an independent module, offering the possibility of presenting graphically structured objects edited by the user according to a natural and non-ambiguous syntax, or formulas produced by symbolic systems. FIGUE's layout engine offers an effective selection mechanism and allows the direct manipulation of the underlying structured objects (computation, formula simplification, automatic command generation, ...) using mouse events.

Presently, our aim is to integrate our graphical tool in the Internet environment to profit from the current, and future, developments to browsers, and to reach a wider population of users. For that goal, two approaches are considered. First, FIGUE is written in Java, this allows an easy integration with the majority of existing web browsers through a Java Applet. Second, FIGUE contains a module for processing the mathematical expressions using the MathML standard (XML dialect for mathematical formulas). This module allows us to render and to handle MathML elements using FIGUE and also to generate MathML expressions starting from FIGUE objects. FIGUE is already in use at INRIA for building interactive tools and structured editors in order to manipulate objects like programs, mathematical formula or proof commands. As an example of application, it is used for developing the PCOQ [3] system, a graphical interactive interface for the proof system COQ [4].

The next section details FIGUE functionalities such as mathematical display and interaction mechanism. Section 3 shows how FIGUE supports MathML by processing and generating MathML code, and the last section describes a graphical interface using FIGUE for developing mathematical proofs.

2 Display and Manipulation of Mathematical Formulas

2.1 Graphical Constructors

Like the majority of layout tools, FIGUE is based on the idea of layout boxes [5] for representing structured objects of the document. It manipulates a box-tree which describes how to layout the objects on a page. Each node of the tree is a box (graphical constructor) including all its child boxes and the leaves are atoms (basic units like strings). Figure 1 shows a simple example of box representation of a mathematical formula.

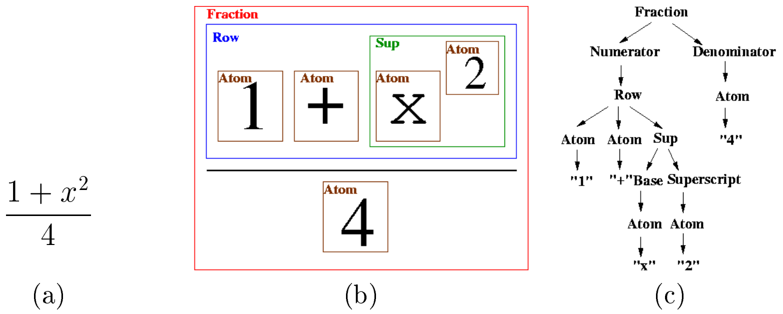

Figure 1. (a) Mathematical formula. (b) Corresponding boxes. (c) Box-tree representation.

FIGUE lays out the boxes in a two-dimensional space (see figure 2), taking into account the stretching properties of the mathematic constructors depending on their contents (root, integral, fraction, tables, ...). FIGUE directly manipulates a box-tree which memorises the dependence relation (inclusion) between graphical objects. Each constructor has its own formatting algorithm to specify the relative position of its sons and to determine the size and the alignment of its bounding box (the whole box). This algorithm updates the box's graphical properties (or attributes specific to a box): origin, size, alignment by taking into account the display area parameters like the page width and the different parameters related to the graphic context of each box (fonts, color, background,coordinates).

$$\begin{bmatrix} x & \cdots & y+2 & \cdots & 0 \\ \vdots & & \vdots & & \vdots \\ \frac{z}{4} & \cdots & 5 & \cdots & w \\ \vdots & & \vdots & & \vdots \\ 1 & \cdots & 3w & \cdots & 0 \end{bmatrix}$$

(a)

$$\Phi = \frac{1+\sqrt{5}}{2}$$

$$\Phi = 1 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{1 + \cdots}}}$$

$$\Phi = 1 + \sqrt{1 + \sqrt{1 + \sqrt{1 + \cdots}}}$$

(b)

Figure 2. Examples of 2-D formatted objects. (a) A matrix (b) Fractions and Roots

Once the formatting task is done, FIGUE displays the formatted objects according to their graphical contexts and draws the needed symbols or typographical characters (for example, the mathematic root symbol - see figure 2b).

When displaying formulas, efficiency and incrementality are two important issues. Efficiency is important to handle potentially large expressions while incrementality allows us to minimise the cost of reformatting after any modification of one or more formulas in an interactive environment. Each box modification (context, size, origin, ...) must be propagated in the whole tree in an optimal way in order to update only the concerned boxes.

2.2 Layout and Interaction

Typesetting systems, network browsers, and other user interfaces often deal with the presentation markup (formatting structure) as their primary representation, while symbolic systems use languages that more closely reflect the structure of the content (syntax tree). The presentation markup is adapted to offer a powerful display while the content markup is necessary to develop some computational program (simplification, code generation, ...). Our approach separates the content from the presentation while keeping a natural relationship between them. This separation allows for a powerful display as well as communication with computational systems.

FIGUE manipulates only the box-tree (presentation) to display the document objects. Moreover it gives the user the possibility to select these displayed objects owing to FIGUE's selection mechanism which is directly related to the box-tree structure. FIGUE automatically translates the coordinates of the mouse on the screen into a place in the box-tree structure and thus locates the subtree corresponding to the selected objects (see figure 3).

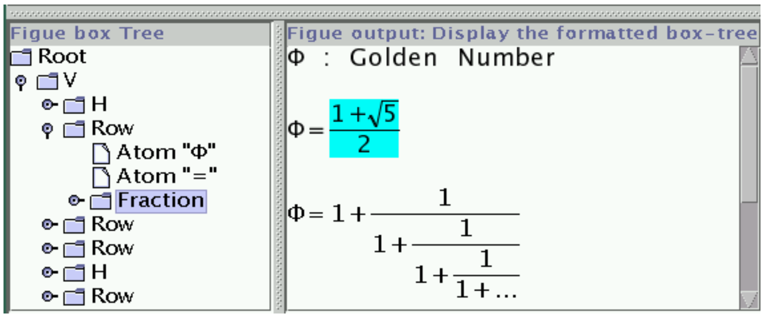

Figure 3. Selection of an object by the user (right), and its place in the box-tree (left).

When FIGUE is used to develop graphical interfaces for symbolic systems, the FIGUE's box-tree is produced by a description given in an abstract language, called PPML (*Pretty Printing Meta Language*) [6]. A PPML description is a succession of rules where each rule associates a syntactic structure with the box structure describing the corresponding formatting. The PPML engine transforms the objects represented by an abstract syntax tree (according to a given formalism) into a FIGUE box-tree as shown in Figure 4.
As an example, let us consider the following simple PPML rule:

plus(*x, *y) $\rightarrow$ [<Row> *x "+" *y]

We associate a *Row* graphic constructor having three elements *x, + and *y to a syntax tree whose root is a "plus" having two children *x and *y. In the right part of this rule *x and *y are the results of recursive calls of the display function on the children of the *plus* operator. They will be separated by "+" character which is an atom (leaf of the box-tree). A *Row* constructor formats horizontally its children and applies rules of line cutting when the width of its bounding box is greater than the display area width.

The PPML compiler maintains the correspondence between a box-tree structure and a syntax tree. Associated to the FIGUE's selection mechanism, this powerful interaction mechanism allows more *intelligent* manipulations than a simple copy-paste, such as the development, the simplification, and the modification of the underlying expressions. The interaction is a very significant and useful functionality in user interfaces. For example, the PCOQ

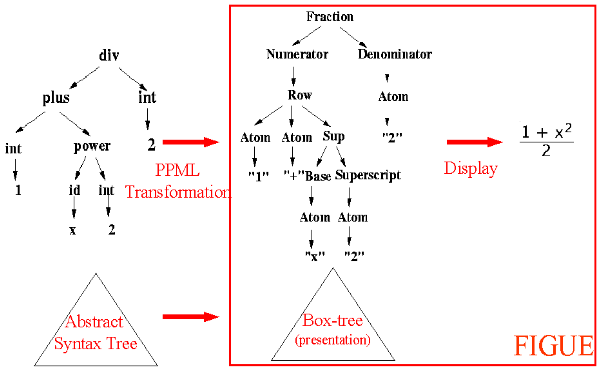

Figure 4. Translation of a formula syntax tree into FIGUE's box-tree using PPML rules.

system uses FIGUE and exploits this powerful interaction mechanism to develop a proof by selection and to manipulate algebraic formulas [8,7].

3 FIGUE and MathML

3.1 MathML

One of the major interests for doing mathematics on the computer is the possibility to share the data with other users, by exchanging data between different applications or by diffusing them on the Internet. For this goal, many exchange protocols for mathematics exist like OpenMath [9], MathLink [10] for Mathematica, and more recently MathML [11]for mathematics on the Internet.

MathML has been proposed by the W3C as a standard format for mathematics in order to unify today existing formats and facilitate the use of mathematical and scientific content on the Web. Up to now, mathematics on the Internet were displayed as images without structure. MathML is based on two types of formula description: mathematical presentation for notation and mathematical content. The presentation encoding is used for visual display (bidimensional description of the expressions), while mathematical content is used for applications where the semantics (close to the abstract syntax) is important, such as in scientific software. Figure 5 illustrates the MathML-*content* and MathML-*presentation* encoding for the formula: $x^2 + 1 = 0$.

$$x^2 + 1 = 0$$

(a)

```
<math>
<reln>
  <eq/>
  <apply>
    <plus/>
    <apply>
      <power/>
      <ci>x</ci>
      <cn>2</cn>
    </apply>
    <cn>1</cn>
  </apply>
  <cn>0</cn>
</reln>
</math>
```

(b)

```
<math>
<mrow>
  <mrow>
    <msup>
      <mi> x </mi>
      <mn> 2 </mn>
    </msup>
    <mo> + </mo>
    <mn> 1 </mn>
  </mrow>
  <mo> = </mo>
  <mn> 0 </mn>
</mrow>
</math>
```

(c)

Figure 5. MathML: (a) The formula. (b)(c) Content and Presentation Markup.

3.2 Generating, Processing, and Rendering MathML in FIGUE

FIGUE is able to interpret XML documents describing its box-tree structure according to a grammar specified by the FIGUE DTD (*Document Type Definition*). Each FIGUE box is associated with an XML element. We follow a DOM (*Document Object Model*) development to manipulate XML documents: an XML document describing the box-tree structure is analysed by a generic XERCES parser[14] and validated using the FIGUE DTD to obtain a DOM tree corresponding to the structure of the XML document. Next, the DOM tree is translated into a FIGUE box-tree which could then be displayed by FIGUE.

Conversely, FIGUE generates XML format from the displayed graphical objects and creates an XHTML document including MathML-*presentation* code to encode the mathematical formulas. This XHTML document can then be displayed by any browsers supporting MathML (Mozilla, Amaya, etc.). To generate the MathML elements from the FIGUE mathematic objects, we associate each FIGUE graphical constructor to the corresponding MathML-*presentation* element. This correspondence is done naturally because FIGUE and MathML-*presentation* use a similar structured notation for representing graphically the objects. So, XML can be used by FIGUE as an import and export format and also as an exchange format with other programs.

More particularly, we have developed in FIGUE a module to interpret directly MathML presentations and contents. This module takes a MathML input, interprets it and then displays it using FIGUE. First of all, this module analyses and validates the MathML document by using the MathML DTD. It produces a DOM tree representing a structure of a MathML document. Then, this DOM tree is translated into a box-tree which can then be displayed. This module ensures the link between the box structure and the MathML element. It is then possible to obtain the MathML source corresponding to all displayed objects. Our approach is not restricted to a rendering of MathML, but also offers an interaction with the MathML elements. In order to interpret the MathML-*presentation*, we adopt two methods: (1) Translate directly the MathML elements by Java code into FIGUE structured objects; (2) Translate the MathML elements using parameterised rules of transformation into FIGUE structured objects. The first method makes a simple correspondence between the MathML elements and the FIGUE boxes. The second method applies the transformation rules of a formalism called XPPML[a] in order to transform each MathML element into an XML element describing the corresponding FIGUE box. This element will be then transformed into a FIGUE box to be displayed.

XPPML rules describe, in an XML document, the transforming rules from a source tree into another result tree. The XPPML compiler is based on the PPML compiler ideas (see section 2.2) and offers the same functionalities using an XML syntax. It is also modular and offers the user the possibility to add his transformation rules depending on his own style or on the mathematical theory that he is developing. An XPPML description is a succession of rules where each rule associates a MathML element with its FIGUE box. These rules describe how the MathML elements will be displayed by FIGUE. The XPPML compiler maintains the correspondence between the two structures.

An XPPML rule is made up of two parts: the first part is a *pattern* used to recognise the XML element and the second part is a *format* which describes its layout in terms of a box language. As a simple example, let us consider a XPPML rule describing how the *reln* MathML element will be displayed:

```
<rule>
  <pattern> <template match="reln(*op,*x,*y)"/> </pattern>
  <elt name="Row">
    <variable name="x"/>
    <extFun name="getNamepp"> <arg value="op" type="var"/> </extFun>
    <variable name="y"/>
  </elt>
</rule>
```

We associate a tree whose root is the *reln* MathML element having three children $*op$, $*x$ and $*y$, with a *Row* graphic constructor having three elements, the first and last elements are the results of recursive calls of the display function on $*x$ and $*y$ while the second element is an *Atom* box whose value is the result of an external function call, *getNamepp*, written in Java and

[a] XPPML is a form of XSLT for transforming XML documents into other XML documents

which returns the unicode character corresponding to *op operator (in the case of <in/> operator, this function returns the unicode character \u2208).

Our MathML interpretation module is flexible thanks to the power of the XPPML transformation rules. It can be used to render both MathML *content* and *presentation* codes.

4 Using FIGUE to develop a user interface

Some symbolic systems provide a graphical interface to help the users do symbolic manipulation (like proof of theorem) in an interactive environment. These user-friendly and intuitive interfaces can make mathematical computations much easier to do and can make the symbolic system directly accessible to beginners, while also increasing the richness of functionalities for expert users. In this aim, we develop PCOQ a graphical user interface for the COQ theorem prover using FIGUE as interactive display module. PCOQ has been developed in JAVA following a general approach for building user-interfaces for theorem provers [13]. This approach separates the interface and the prover. The principal modules participating in the PCOQ development are: (1) an exchange protocol between the graphical interface and COQ; (2) structured editing and presentation mechanisms of formulas and commands; (3) efficient display of formulas and commands.

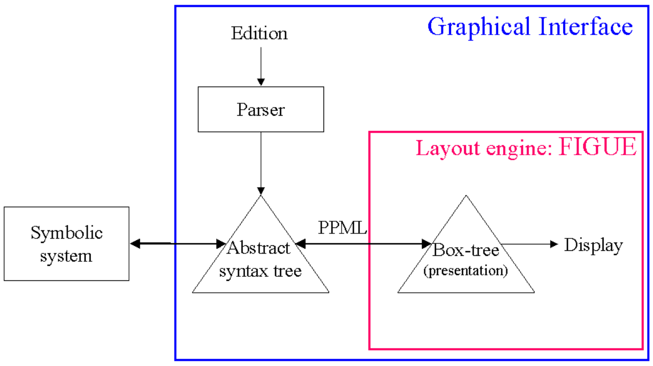

Figure 6. A general architecture for symbolic systems user-interfaces using FIGUE

Figure 6 illustrates our general architecture for building user-interfaces using FIGUE. The graphical interface and COQ run as two separate processes which communicate via a protocol. This protocol [15] allows the structured data to be exchanged between processes. The graphical interface uses a structured editing module providing ways to structurally edit formulas and commands following a formalism (definition by abstract types of a language). The structured formulas and commands can be stored in files, communicated to symbolic system to do the computations or transformed into a box structure to be displayed by FIGUE. The PPML transformation rules (see section 2.1) maintain the link between the syntax tree and the box-tree. This link allows a powerful mechanism of selection of objects in the interface. PCOQ environment uses the structure of logical formulas to help systematically the user direct the proof by simple clicks of the mouse [8].The figure 7 illustrates an example of PCOQ session, where different objects are manipulated: text, mathematic formulas, and images, and where some commands have been generated automatically.

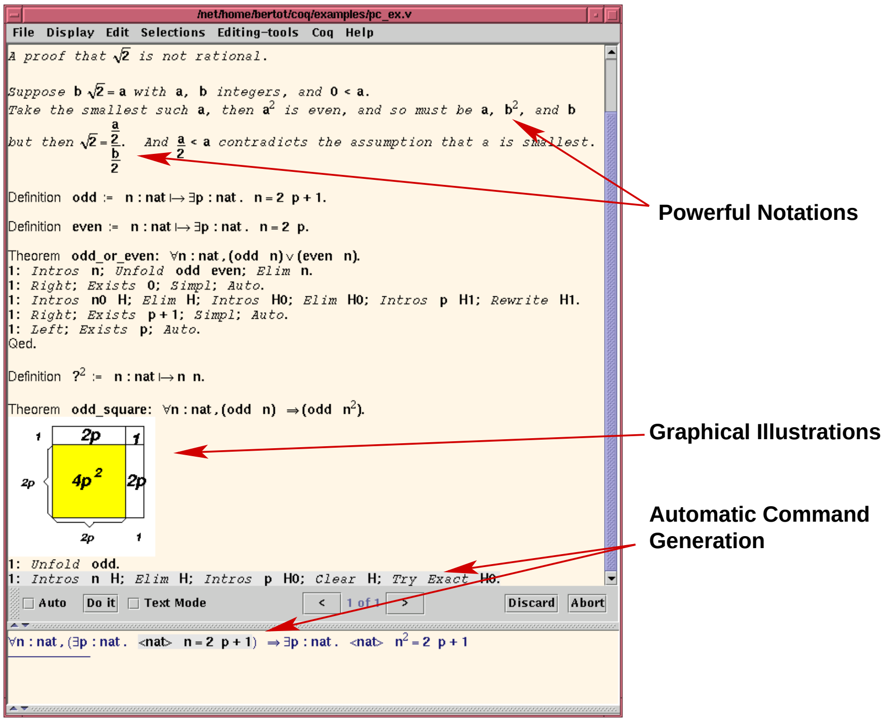

Figure 7. A PCOQ session.

5 Conclusion

By developing FIGUE, we are interested in two important functionalities for WYSIWYG mathematical environments: *display and interaction*. At the same time we are equally interested in the *exchange and communication* of mathematical formulas. With FIGUE, we have a specialised display component for building WYSIWYG environments for symbolic systems and scientific document editors, providing a simple and natural way to interact with mathematical formulas, and allowing exchange of XML data with other symbolic systems. In addition, the module of MathML interpretation in FIGUE enables us to render MathML (*presentation* and *content* markup). With this development, FIGUE becomes a specialised component for MathML rendering with a powerful interaction mechanism. FIGUE is more than a simple prototype, it is used in PCOQ development and it provides a powerful display with interaction easily adaptable to the user's needs.

References

1. Figue, `http://www-sop.inria.fr/croap/figue`.
2. H. Naciri and L. Rideau. Affichage interactif, bidimensionnel et incrémental de formules mathématiques. *Proceedings of the 5th African conference on research in computer science (CARI)*, Oct. 2000, Madagascar.
3. Pcoq, `http://www-sop.inria.fr/croap/pcoq`.
4. Coq, `http://coq.inria.fr/`.
5. D. Knuth. *The TeX-Book (revised)*, Addison Wesley, 1990.
6. I. Jacobs , J. Bertot, ed.. *Centaur 1.2*, The PPML Manual. INRIA, 1993.
7. Y. Bertot. Direct manipulation of Algebraic Formulae in Interactive Proof Systems. *UITP'97*, Sophia Antipolis, Sep. 1997.
8. Y. Bertot, G. Kahn, and L. Théry. Proof by Pointing. *Lecture Notes in Computer Science*, vol. 789, 1994, p. 141.
9. OpemMath, `http://www.openmath.org/`.
10. MathLink reference guide. 1992, Wolfram Research.
11. MathML, `http://www.w3.org/Math/`.
12. N. Kajler and N. Soiffer. A Survey of User Interfaces for Computer Algebra Systems. *Journal of Symbolic Computation*, vol. 25, n. 2, 1998.
13. L. Théry, Y. Bertot, and G. Kahn. Real Theorem Provers Deserve Real User-Interfaces. *ACM SSDE'1992*, Washington D.C, dec 1992.
14. Xerces, `http://xml.apache.org/xerces-j/`.
15. A. Dery and L. Rideau. Distributed programming environments: an example of a message protocol. Technical Report, RT-0165, 1994, Inria.