An Investigation on the Dynamics of Direct-Manipulation Editors for Mathematics*

Luca Padovani and Riccardo Solmi

University of Bologna, Department of Computer Science
Mura Anteo Zamboni 7, 40127 Bologna, Italy
{lpadovan, solmi}@cs.unibo.it

Abstract. Mathematical expressions are pieces of structured information that could benefit from direct-manipulation approaches for document authoring. Yet, not only there is disagreement on the behaviors of authoring tools, but also these behaviors are often ill-designed and poorly implemented. This situation leads to dissatisfaction amid users who prefer more classical editing approaches.

In this paper we compare the behaviors of several state-of-the-art editors for mathematical content and we try to synthesize a set of rules and principles to make the authoring experience pleasant and effective.

1 Introduction

Direct-manipulation editors for mathematical content allow an author to edit *in place* a mathematical formula as this is formatted and displayed on the screen in its traditional notation. Editing and displaying occur simultaneously and the formula is reformatted at every modification. These editors are usually characterized by the fact that they work on a structured representation of the document, hence they fall in the category of *model-based* editors.

Despite their aim of being "friendlier", direct-manipulation editors turn out to be rather unattractive to use for both unexperienced and advanced users since they suffer from severe usability problems. In fact, they are more challenging to design: the use of a structured model requires the editor to implement some kind of incremental parsing meaning that user actions are mapped on non-trivial operations on the model. At the same time, part of the information about the model structure is not displayed in order to reduce the user's awareness of the model and to provide a familiar, lightweight presentation. We claim that these are not sufficient reasons that prevent the design of a direct-manipulation editor with good, effective usability [9, 1].

While we do not aim to describe the behavior of the *perfect* model-based editor for mathematical content, we can at least try to highlight the deficiencies in the existing tools. The goal is to synthesize a set of principles inspired by successful text-based editors and common usability guidelines and to provide a qualitative evaluation of the examined math editors on these bases.

* This work was supported by the European Project IST-2001-33562 MoWGLI.

A. Asperti et al. (Eds.): MKM 2004, LNCS 3119, pp. 302–316, 2004.

The structure of the paper is as follows: in Section 2 we describe the dynamics of a number of direct-manipulation editors for mathematics. We enrich the prose of the descriptions with a graphical notation whose purpose is to capture the dynamic behavior of the editors on a static medium like the paper. In Section 3 we do a little step back to the world of text editors, for which a tighter and more standardized set of rules and behaviors have emerged over the years. The analysis of these rules will be the starting point for our proposal in Section 4, where we try to classify distinct and possibly orthogonal aspects of model-based editors, in particular editors for mathematics, and list some intuitive guidelines for their development. We conclude in Section 5 with a comparison of the tested editors.

2 Behaviors

We began our analysis by trying out a number of currently available editors to understand how behaviors were implemented and what rationales were behind the choices. The following is the list of the products that we have tried. Some of them are just software components whose only purpose is to display and edit mathematical formulas, others are more complex applications for which editing mathematics is only an auxiliary (sometimes indispensable) functionality:

1. **Amaya** version 8.2. *Web page:* `http://www.w3.org/Amaya/`
2. **FrameMaker** by Adobe, version 7.
   Web page: `http://www.adobe.com/products/framemaker/main.html`
3. **MathType** by Design Science, version 5.2.
   Webpage:`http://www.mathtype.com/en/products/mathtype/`, see also[13].
4. **Scientific WorkPlace** by MacKichan Software, version 5
   Web page: `http://www.mackichan.com/products/swp.html`
5. **LyX** version 1.3.4. *Web page:* `http://www.lyx.org/`, see also [6].
6. **Mathematica** by Wolfram, version 5.
   Web page: `http://www.wolfram.com/products/mathematica/index.html`
7. **TEXmacs** version 1.0.1.23. *Web page:* `http://www.texmacs.org/`

The products have been chosen to represent the current state-of-the-art in both commercial and freely available software.

2.1 Models and Edit Points

One of the characteristics of these editors is that they are *model-oriented* rather than *text-oriented*. By this we mean that the internal representation of the edited document is structured and the editing commands work directly on the internal representation. This is the main source of problems because editing operations (including movements) are performed on a non-linear data structure that, once displayed, may convey only partially or approximately the overall information it represents. For example, if the model represents the sum of two entities as a binary node, because of the associativity law of addition a sum like $x + y + z$

may be displayed with no parentheses, thus concealing the actual structure which may be either one of $(x + y) + z$ or $x + (y + z)$.

Because of the structured nature of the internal application model, the information that is necessary for unambiguously specifying the point where the next editing operation will occur is made of:

- the *node* of the model pointed to;
- the *index*, also called insertion point, indicating the sub-part of the node where something has to be inserted. Terminal (or leaf) model nodes, which usually represent identifiers, numbers, and, more generally, sequences of characters, typically have as many indexes as the number of characters plus one. Non-terminal (or internal) model nodes have a number of valid indexes which may vary depending on the structure of the model.

We call the combination of these two entities *edit point*.[1] Most editors give a visual feedback for both entities. The index is typically presented by a vertical bar called *caret*. The node presentation, called *focus*, ranges from a horizontal line spanning the node's horizontal extent on the view, a prolongated caret spanning the node's vertical extent on the view, a solid or dashed rectangle surrounding the node, and so on. Some editors like amaya have no visual feedback for the focus at all, others like lyx and texmacs emphasize *all* the elements from the root of the internal application model to the focus. amaya and texmacs give additional feedback at the bottom of the editing window by providing the stack of node names from the root of the document down to the edit point.

In this paper we represent the caret with $\Uparrow$ or $\Downarrow$ symbols and we underline the focused node. For example, s$\underline{\Uparrow n}$ represents a focused token node whose content are the two letters 's' and 'n', with the caret sitting in between. An insertion of the character 'i' would change the node to s$\underline{i \Uparrow n}$.

2.2 Presentation of Missing Information

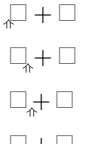
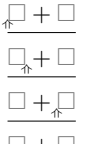
Model-based editors are usually constrained by the structure of the model. For example, a node of the model representing a binary operator may be required to have two child nodes, one for each operand. However the sequential nature of the editing process prevents the document to be well-formed at all times. Missing parts of the document that are expected to be filled in are called *slots*. The visual feedback of slots may vary, ranging from question marks on a reverse background in framemaker, to solid or dashed rectangles in mathtype, mathematica, and lyx to nothing at all in texmacs.

2.3 Basic Moves

Since the purpose of editing is to change a document and since the operations that change the document are performed at edit points, one of the main concerns

[1] The use of the term "point" dates back to the TECO editor as documented in [12].

Table 1. Traversal of a horizontal group of elements

	amaya	framemaker, mathtype, mathematica, lyx, scientific workplace, texmacs
⋮ RIGHT		
LEFT *	reverse	reverse

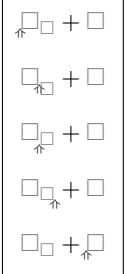
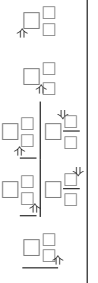
is *how to reach* edit points. We will devote this section to a comparison of the various strategies adopted by the editors. In order to do so, we present a series of tables, each of them devoted to a common mathematical construct. The tables show the movements of the caret and possibly of the focus as the user requests a sequence of actions. Actions are triggered by keystrokes: LEFT, RIGHT, UP, DOWN represent the basic cursor movement keys. In the tables time flows from top to bottom, the keystrokes are shown in the leftmost column of the diagram whereas the cells in the other columns show the state of the various editors *after* the action associated with that keystroke has been executed. We denote with the word "reverse" sequences of states that mirror the corresponding states for the opposite action. We denote arbitrary subexpressions with the symbol □ and assume that they are traversed with a single action.

Rows. We start with a simple list of identifiers separated by operators (Table 1). Even for this simple formula editors may have different behaviors. The amaya editor advances the caret by a little step between identifiers and operators, thus moving from the end of a node in the model to the beginning of the next one. This gives visual feedback about the focused node, which is assumed to be the one the caret is closest to, but the user has the impression of a slowed-down traversal. Conversely, the other editors provide a natural traversal behavior where one action corresponds to one step.

Scripts. Next we examine the scripting construct (Table 2), which is used for exponents, indices, integration limits, and so on. From a geometrical point of view this is the construct where bi-dimensional layout starts playing an important role since scripts are vertically shifted with respect to the main baseline.

The observed behavioral variants include: full traversal of the scripts (amaya and mathematica), deterministic traversal of a subset of the scripts (mathtype and scientific workplace), skipping of the scripts unless an explicit action is re-

Table 2. Traversal of scripts

	amaya, mathematica	framemaker		lyx	mathtype	scientific workplace	texmacs
⟨RIGHT⟩ ⋮							
⟨LEFT⟩*	reverse	reverse	reverse, but not always (see text)	reverse	reverse	reverse	reverse

quested (framemaker and lyx, note however that lyx traverses one more edit point). A particularly original behavior is observed in the texmacs editor, in which only one of the two scripts is traversed: the script that was visited last in the previous traversal is the one to be visited during the next traversal. framemaker is also bizarre: the traversal is reversible if the state reached by ⟨RIGHT⟩ moves is $\square_{\square} + \square$, but it is not reversible if the reached state is $\square_{\square} + \square$.

The ⟨UP⟩ and ⟨DOWN⟩ keys have very different associated behaviors: from no-ops to switching between subscripts and superscripts, to jumping to the beginning or the end of the whole construct.

Radicals. Roots (Table 3) can be thought as a variant of the script construct, from both semantical and notational points of view. Still they present very different behaviors if compared to scripts. Again the traversals range from partial to full navigation of the subparts, and again framemaker presents an odd behavior that is not reversible in some cases. The mathtype editor skips the index and provides no way of accessing it except by moving the caret on a different line and finding an edit point such that a vertical movement in the opposite direction causes the caret to hit the index.

Fractions. In Table 4 we traverse fractions. This construct is characterized by a clear vertical layout which is completely orthogonal to the baseline. Here too the behaviors are very different, although slightly less bizarre, probably because there is no horizontal component in the displacement of the subparts. Again we

Table 3. Traversal of roots

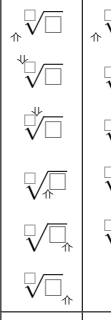
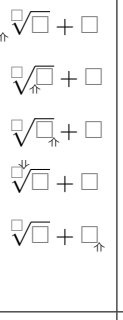
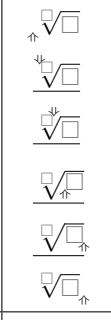
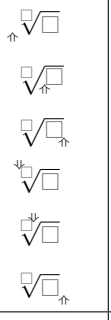
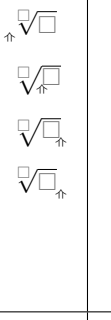
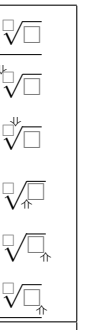
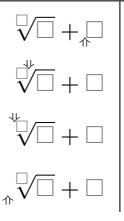
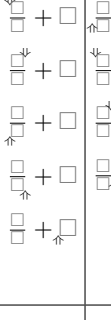
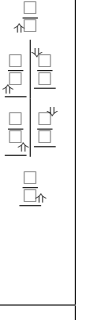
	amaya	framemaker	lyx, texmacs	mathematica	mathtype	scientific workplace
RIGHT ⋮						
LEFT ⋮	reverse		reverse	reverse	reverse	reverse

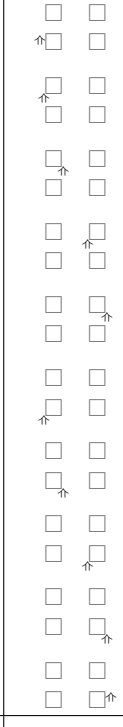
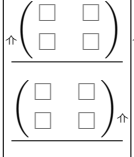
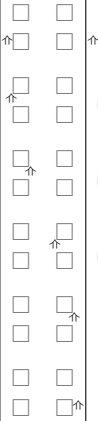
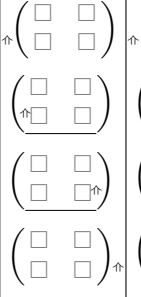
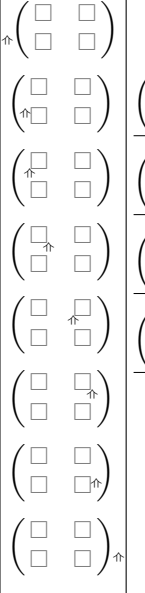
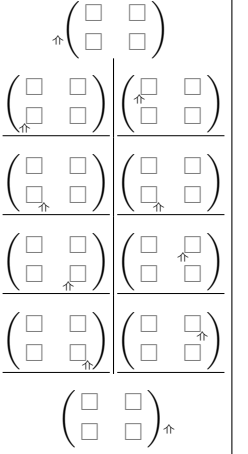
Table 4. Traversal of fractions

	amaya, mathematica	framemaker		lyx	mathtype	scientific workplace	texmacs
RIGHT ⋮							
LEFT ⋮	reverse	reverse	reverse		reverse	reverse	reverse

recognize full traversals in amaya and mathematica, partial deterministic traversal in lyx, mathtype and scientific workplace (lyx has a different traversal in the opposite direction), inner traversal caused by explicit user action in framemaker, and par-

tial, visited-last-dependent traversals in texmacs. In all cases ⟨UP⟩ and ⟨DOWN⟩ cause the caret to be moved from the numerator to the denominator or vice versa.

Table 5. Traversal of matrices

	amaya, mathematica	framemaker	lyx	mathtype	scientific workplace	texmacs
⟨RIGHT⟩ ⋮						
⟨LEFT⟩*	reverse	reverse	reverse	reverse	reverse	reverse

Matrices. Finally we examine a truly bidimensional mathematical construct in Table 5, which shows the traversal of possibly fenced matrices. In framemaker the construct is skipped unless the user triggers the ⟨DOWN⟩ action explicitly, in which case a full traversal is performed.

2.4 Special Moves

Aside basic moves, most editors provide a behavior associated with the ⟨TAB⟩ key that causes the edit point to jump across parts of the model following varying strategies. Among the observed ones there are: cycling the empty slots of the

whole formula (framemaker, mathematica and mathtype); moving to the next slot (lyx, there is no inverse action); cycling the child nodes of the focused node (scientific workplace). amaya and texmacs have no behavior associated with the ⟨TAB⟩ key.

Regarding ⟨UP⟩ and ⟨DOWN⟩ moves, they do not always behave geometrically. For example, in framemaker the ⟨DOWN⟩ key has the effect of moving the edit point down the structure to the first child of the node being edited and the ⟨UP⟩ key has the effect of moving the the edit point up the structure to the parent of the node being edited. In mathematica ⟨UP⟩ and ⟨DOWN⟩ have a context-sensitive behavior which is not always geometrical. In lyx the behavior is partially geometrical but constrained by the model structure. For example, moving from a superscript down towards the subscript (or from a subscript up towards the superscript) causes the edit point to be placed just after the base element.

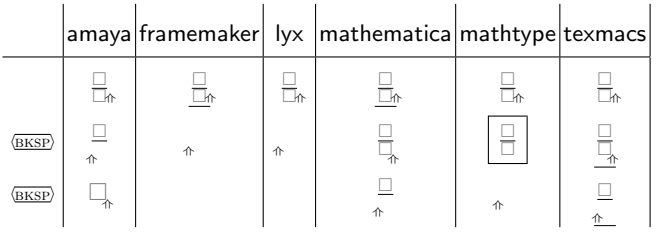
2.5 Editing Actions

Constrained Versus Unconstrained Editing. Different editors have different concepts of a well-formed formula and consequently they constrain editing operations in very different ways. On one end is framemaker which tries to keep the formula semantically meaningful. So, for example, the user is prevented from entering a formula like $a++b$ and parentheses must always be balanced. mathematica sometimes provides different variants of a construct, one *algebraic* and one *typographical*, that have different constraints. Other editors like amaya and lyx have a looser concept of well-formed formula and, apart from the constraints imposed by the typographical structure of the formula, they allow for much more freedom in the editing process. In mathtype editing is almost totally unconstrained, for instance it is possible to have scripts not associated with a base element.

Templates and Overlays. These concepts represent two similar ways of assisting the author in changing the structure of the document. Templates are partial formulas with empty slots. The user can insert a template in a certain point of the document, typically at the current edit point or, if the editor allows it, in place of a previously selected part of the document. Overlays are special templates in which one designated slot is filled with the document part that is referenced by the edit point or that is selected at the time the overlay is inserted. All of the tested editors implement one or the other or, more frequently, both templates and overlays.

Delete Commands. Delete commands for model-based editors are particularly delicate because the edit point may refer to internal nodes of the model. In the particular case of fractions, Table 6 shows some of the observed behaviors associated with the ⟨BKSP⟩ key (delete to the left of the caret): entering the node, similar to a ⟨LEFT⟩ move (mathematica and texmacs); entering the node and deleting recursively (amaya); deleting the whole node in one shot (framemaker and lyx); selecting the node and deleting it only if the user repeats the action (mathtype, in the table the selected fraction is shown inside a box).

Table 6. Different behaviors associated with the deletion of fractions



3 Learning from Text Editors

Although a plain text document can be seen as a monodimensional entity (a stream of characters), word processors usually introduce some structure by allowing parts of the text to be annotated with style information. So, albeit to a limited extent, even text editors are based on a somehow structured model. In their case, however, there happens to be a much stronger convergence of behaviors associated with user actions. We can characterize such behaviors, at least with respect to the kind of actions we have been examining so far, as follows:

- basic steps on the document view can be achieved by means of basic user actions (basic = one keystroke). The emphasis is on the view rather than on the model. For example the behaviors associated with the ⟨UP⟩ and ⟨DOWN⟩ keys move the edit point to another one (on a different line) that is not adjacent with respect to model structure.
- basic operations like insert or delete actions act on the smallest entity in the proximity of the edit point;
- when provided, movements and operations on the document model (movement from a word to the following or preceding ones, the deletion of a whole line, and so on) require the user to perform dedicated actions;
- there are no fake moves: each user action clearly corresponds to a definite movement of the caret in the view. The user is never required to perform extra movements to get across different elements on the model (e.g. entering a part of text which has a different style requires no extra moves if compared to continuing moving on a paragraph with the same style);
- movements are geometrically reversible: in any position except for the document border, one ⟨LEFT⟩ nullifies exactly one ⟨RIGHT⟩, one ⟨UP⟩ nullifies exactly one ⟨DOWN⟩. Modern editors have often preferred reversibility of actions over simpler geometrical movements: for example moving from the end of a line to a shorter one causes a horizontal displacement of the caret, however a reverse move restores the caret location.

Editors providing different styles have to face the problem of edit point clashing. When the caret is placed between two characters having different associated styles, which style is taken for the next inserted character? Some editors try to give the caret a different appearance (like drawing a slanted vertical bar instead

of a straight one, or drawing a caret which is as tall as the current font size) but this is not always enough to disambiguate the focus in general. Most editors implement a deterministic rule like "the style of the character preceding the caret is taken" which works well in all cases but a few rare exceptions. In editors for mathematical content this solution might be impractical because the model structure is more complex and plays a much more important role.

4 Analysis and Proposal

Although the two kinds of editors, for structured mathematical content and for plain text, are conceptually very different, we believe that the set of behaviors they implement could and should share a large common ground. In this respect, the current state-of-the-art math editors are certainly unsatisfactory and this claim is supported by the following observations:

- the application model is exposed to the user: while working at the model level simplifies the implementation of the editor, it also forces a view of the document which often does not match the user's mental image of the mathematical formula being edited. Geometric movements, editing operations, selections should not be constrained by the model unless the user explicitly requests so;
- model-oriented and geometric navigation modes are mixed: for example, the ⟨RIGHT⟩ keystroke sometimes triggers a geometric move and sometimes it triggers a movement on the model. In other cases, see for example the framemaker editor, ⟨RIGHT⟩/⟨LEFT⟩ always behave geometrically, but ⟨UP⟩/⟨DOWN⟩ correspond to movements on the model;
- important feedback, like the placement of empty slots in amaya and texmacs, is sometimes missing;
- there is excess of useless feedback. There is no point in showing a focus if it serves no purposes in terms of feedback (moreover the focus is very model dependent by definition). Even less useful, and actually confusing, is showing the structure of the document in places that are not directly related to the edit point (see the texmacs editor);
- unexpected or ambiguous behaviors lack suitable feedback: operations that are uncommon in text editors (like deletion of a complex model node) should be carefully implemented (see deletion in framemaker and lyx);
- some actions are non-deterministic: they depend on a state of the document which is not made explicit by any form of feedback;
- simple movements are not always reversible;
- there is lack of dedicated views for model-oriented navigation and editing.

What follows is our proposal for a more uniform editing behavior.

Edit Points. It is fundamental for the application to provide effective visual feedback of the caret. The caret should give the impression of actual movement, while the focus should be displayed only if it helps disambiguating the context

where actions take place. In fact the focus may be misleading when the editor implements incremental parsing rules that override the focus position depending on the requested actions. For example, upon insertion of the digit 2 from either one of the states $\underset{\uparrow}{1}+$ and $\underset{\uparrow}{1}+$ the editor might go into the state $\underset{\uparrow}{12}+$. In this case the focus prior to the insertion does not necessarily indicates the model node affected by the operation.

Slots. While an empty slot occupied by the caret might need no visual feedback, parts of the document that are missing should be clearly identifiable and easily reachable.

It is often convenient to provide a quick way for moving across the slots within the document (or within a smaller scope such as a single formula). This functionality is typically accessed by the ⟨TAB⟩ key (and ⟨SHIFT⟩ + ⟨TAB⟩ for moving in the opposite direction). As there is quite general agreement on this behavior among the current editors, there are no particular principles to be listed except for reversibility. The order in which the slots are visited is also of secondary importance as the user expects the caret to jump anyway and the number of slots, which is typically very limited, makes them all reachable very quickly.

Geometric Navigation. This should be the default navigation mode as it is consistent with the "what-you-see-is-what-you-get" principle that the user edits what she sees. More precisely, the user should be allowed to traverse the document in such a way that basic move actions cause an actual, yet basic, movement of the caret in the requested direction. As formulas often occur inside text, where geometric movements are commonly accepted, these rules should apply within the formulas as well in order to guarantee adequate consistency. More specifically:

- the caret is moved to the geometrically closest and deepmost edit point in the direction of the movement. The notion of "direction" we are referring to here is necessarily blurred, as in mathematical notation related parts (like a base and its associated script) are often placed with both horizontal and vertical displacements;
- in case of ambiguities (edit points at different locations that are equally distant and equally deep) the editor should behave in a deterministic way. The model structure can be used for resolving such ambiguities;
- the movement should be perceived by the user as worthwhile, the user should not be required to move across entities she has not inserted explicitly;
- movements of the caret on the view should be reversible whenever possible.

Determinism is important for avoiding the introduction of a *state* in the user's mental image of the document ("which node have I visited last?").

The geometric navigation mode does not guarantee in general that all the available edit points are traversed. The principle that prefers deeper edit points ensures that the caret is moved on a position where operations have the less disruptive effect on the structure of the model.

Content Navigation. As text editors normally allow the navigation at some larger granularity (like the granularity of words and paragraphs) we may expect

a similar functionality for editors of mathematical formulas. An analogous concept of higher-level entity for mathematics may be that of *subexpression*, or *subformula*. Unfortunately, this concept is so broad (and often fuzzy) that it cannot be used for the implementation of a generally useful navigation mode.

It is however possible to provide simple navigation modes that are based on a smaller higher granularity, like that of tokens, with the purpose of speeding up document navigation. The principles of geometric navigation should hold at this level as well.

Model Navigation. This navigation mode is indispensable for reaching edit points that are not accessible using the geometric navigation mode alone. However, we regard this as a far less frequent navigation mode, especially because it requires the user to have a fairly clear understanding of the model structure used by the application. For these reasons we strongly advocate the use of one or more *dedicated views* that enable model-oriented navigation and editing functionalities. This approach is followed by so called two-view editors, such as [5, 7, 4, 10] and also by some modern Integrated Development Environments[2] where a special view is provided to show a structured outline of the edited documents.

Selection. The basic selection mode should allow for the maximum flexibility. In particular, it should be possible to select parts of the edited document that have no semantic meaning or that form incomplete subexpressions, the same way as movements and editing should be unconstrained as much as possible. Because of the complex bidimensional layout of mathematical formulas, it should be possible to refine a selection in multiple, discrete steps, rather than allowing only a one-shot action of the user.

Notwithstanding this, it is not excluded the possibility of using the application model for enabling constrained forms of selection even on the classical view. For instance, it may be important to notify the user that the selection she has made does not entirely cover a valid[3] model subpart and that subsequent paste operations might fail for this reason.

Editing. Depending on the application and on the model structure, the range of editing operations available at one specific edit point may vary significantly. In the tested editors operations mostly work at the bottommost level (the leaves of the model), while operations on the model structure are limited to selection, cut-and-paste, deletion. However, as we have already discussed in the paragraph regarding edit points in this section, editors may implement incremental parsing rules such that even simple user actions cause deep rearrangements in the structure of the document (similar approaches are presented in [15, 14, 8]). This way, the editing environment would provide for improved flexibility thus reducing training time and discomfort while keeping the actual structure of the model hidden from the user.

[2] See for instance Eclipse, `http://www.eclipse.org`

[3] The notion of validity is obviously application-dependent.

5 Concluding Remarks

It was during the implementation of a model-based editor for mathematical documents that we gradually realized the countless and subtle difficulties that this kind of application underlies. When we turned our attention to existing editing tool, hoping to grab some knowledge about the implemented behaviors and about the user needs and expectations, it was surprising to find out that no two editors shared exactly the same behavior on every aspect, and that several choices made by implementors seemed supported by no solid motivations. The amazing range of possibilities cannot be justified merely on the basis that different editors have different underlying document models. Although such differences do exist, the editors should still strive for providing a comfortable and familiar environment.

Precise descriptions of the dynamics of model-based editors are rare in the bibliography. Incidentally, none of the tested editors provides comprehensive documentation about their behavior, probably because the behavior cannot be easily described on a static medium like the paper. A few informal attempts in this direction can be found in [15], where a token-oriented model is proposed, and in [8]. Barfield [2] has tried to classify tree-editing operations and some of his ideas influenced the navigation modes in Section 4.

Comparisons of document editors have usually the aim of measuring their effectiveness in terms of rate of accomplished tasks, average number of errors, and so on. In the case of text editors, most of the work was carried out during the eighties. Of particular relevance are Roberts et al. [11] and Borenstein [3]. Some interesting statistics about document editors can also be found in Whiteside et al. [16], where the importance of movement actions is highlighted. This is one of the reasons why we have devoted so much effort in understanding and analyzing geometric moves in the examined editors. It is reasonable to assume

Table 7. Scores of the tested editors

	amaya	framemaker	lyx	mathematica	mathtype	scientific workplace	texmacs
Edit point feedback	●	●	○	●	●	●	
Edit point accessibility	○		●	○	●	○	●
Geometric moves			○	○	○	●	○
Reversibility of moves	○		●	●	●	●	●
Deterministic moves	●	○	●	●	●	●	○
Model view	●					○	○
Slot navigation		●	○	●	●	○	
Selection flexibility	●	○	○	○	○	○	○
Model structure	○	●	○			○	○

that a similar suite of tests can be prepared for editors for mathematical content, but to the best of our knowledge no formal comparison has been developed so far.

At last, we could not refrain from summarizing the usability of the tested editors. For each of the features investigated in this paper we have given a measure of usability of its implementation. In Table 7 an empty cell means "not implemented" or "poor support", a ∘ symbol means "partial support" and a • symbol means "good support". We have also given a rough measure of the complexity of the model used by the editors. Intuitively, the more complex the model the more difficult it is to implement behaviors that respect our proposals. The results are clearly subjective and approximate. In fact, in many cases we could only guess about the model structure adopted by the editor. However, the table provides us with a rather strong feeling that direct-manipulation editors can be significantly improved, and that this might justify their unpopularity among both unexperienced and expert users.

References

1. Apple Computer, Inc. "Apple Human Interface Guidelines", March 2004, `http://developer.apple.com/documentation/UserExperience/Conceptual/OSXHIGuidelines/`
2. L.G. Barfield, Editing Tree Structures, Technical Report CS-R9264, Amsterdam, 1992.
3. N.S. Borenstein, The evaluation of text editors: a critical review of the Roberts and Morgan methodology based on new experiments, Proceedings of the SIGCHI conference on Human factors in computing systems, pp. 99–105, San Francisco, California, 1985.
4. K.P. Brooks, A Two-view Document Editor with User-definable Document Structure, Digital Systems Research Center, Palo Alto, CA, November 1988.
5. J. Fine, Instant Preview and the TEX daemon, TUGboat, 22(4), pp. 292-298, December 2001.
6. L.E. Jackson, H. Voß, LyX – An Open Source document processor, TUGboat, Vol. 22, Number 1/2, pp. 32-41, March 2001.
7. D. Kastrup, Revisiting WYSIWYG Paradigms for Authoring LATEX, Proceedings of the 2002 Annual Meeting, TUGboat, Volume 23, No. 1, 2002.
8. J.-F. Nicaud, D. Bouhineau, T. Huguet, The Aplusix-Editor: A New Kind of Software for the Learning of Algebra, LNCS 2363, pp. 178–187, Springer-Verlag, Berlin, 2002.
9. D.A. Normal, The Psychology of Everyday Things, Basic Books, Inc., Publishers, New York, 1988.
10. L. Padovani, Interactive Editing of MathML Markup Using TEX Syntax, to appear in the Proceedings of the International Conference on TEX, XML and Digital Typography, 2004.
11. T.L. Roberts, T.P. Moran, The evaluation of text editors: methodology and empirical results, Communications of the ACM archive, Volume 26 , Issue 4, New York, NY, USA, April 1983.
12. R. Stallman, GNU Emacs Manual, for Version 20.1, Thirteenth Edition, Free Software Foundation, Cambridge, MA, USA, 1997.

13. P. Topping, Using MathType to Create TEX and MathML Equations, Proceedings of the 1999 TEX Annual Meeting, TUGBoat, Volume 20, No. 3, 1999.
14. M.L. Van De Vanter, Practical Language-Based Editing for Software Engineers, in Proceedings of Software Engineering and Human-Computer Interaction: ICSE '94 Workshop on SE-HCI: Joint Research Issues, LNCS 896, pp. 251–267, Springer-Verlag, Berlin, 1995.
15. M.L. Van De Vanter, M. Boshernistan, Displaying and Editing Source Code in Software Engineering Environments, Proceedings of the Second International Symposium on Constructing Software Engineering Tools, CoSET'2000.
16. J. Whiteside, N. Archer, D. Wixon, M. Good, How do people really use text editors?, Proceedings of the SIGOA conference on Office information systems, pp. 29–40, New York, NY, USA, 1982.