CHAPTER 3

Proxima Edit Model

*** *Version: 20th June 2003* ***

In this chapter, we explain the basic edit model of the Proxima editor. The main focus is how edit gestures from the user are translated to edit operations on the data levels of Proxima. Because Proxima is a generic editor, it is not possible to give a very specific description of the edit model, since different instances have different edit behavior. Much of the edit functionality for an actual instance of the editor will be described in the sheets that specifiy the instance. Here, we present only the general concepts.

In Proxima, a user only sees a presentation of the document. Or, more precisely, the user only sees a rendering of the document. However, because the mapping between the presentation and the rendering is subject to little variation (contrary to the evaluation and presentation mappings), the lower levels are often identified with the presentation. Hence we use the term presentation instead of rendering.

A structural view on the document in Proxima is a presentation, just as any other presentation. The only difference is that in a structural presentation each element is presented with its type name, and that the layout of the presentation reflects the structure. The exact details may vary for different structural presentations. A default structural presentation is available for each element type, but other than that no built-in primitive support for structural presentations is present or necessary.

Although the user only sees a presentation, it is possible to target edit operations at other levels besides the presentation. One way to do this is by switching to a more structural presentation before performing the edit operation. However, in practice it turns out that higher level edit operations are useful also in a non-structural presentation. For example, in the word processor it is useful to be able to move sections around without having to switch views. Similarly, in a source editor it is desirable to be able to select entire declarations, sub-expressions, etc. in the regular source text view. Therefore, Proxima

offers editing on all editable levels at any time. We explicitly mention the term editable, because not every level has sensible edit operations. ***

* * * mention here which ones don't?

3.1 Editing at different levels

Edit operations may be targeted at different levels in the Proxima architecture. When an edit gesture is received from the user, it is translated to an edit operation on one of the data levels, and from there to updates on all higher levels up to the document. Subsequently, the document is presented again and also the lower levels, including the rendering are updated. The higher layers may be skipped when the edit operation takes place on local state that is not present in the higher levels. Furthermore, it is possible that certain layers are invoked not on every edit operation, but only after a sequence of edit operations or after an explicit request from the user. Section 4.4, explains the process of bypassing layers in more detail.

* * * explain cycles from architecture paper here?

***

Although each level may be targeted, edit operations can be divided roughly in two categories: *presentation-oriented edit operations* and *structural edit operations*. Presentation-oriented edit operations are targeted at the structure of the presentation, whereas structural edit operations are tree based edit operations on the structure of the document. ***

* * * examples?

Presentation-oriented editing includes all textual editing. However, because a presentation is not necessarily purely textual, also non-textual presentation oriented-edit operations exist.

In Proxima, structural edit operations are performed at the document and the enriched document levels, and presentation-oriented edit operations are performed at the layout level. Contrary to what the name might be suggested, presentation-oriented editing is not done at the presentation level itself, since we want to be able to textually edit the layout as well.

The fact that Proxima has several levels has consequences for the focus model as well. *** Editing may take place at several levels, and therefore each level has its own representation of the focus and the clipboard. ***

* * * repeat definition for focus here?

* * * also rendering and presentation?

The following subsections provide a short description of the edit functionality at each level.

3.1.1 Rendering

The rendering level is not edited directly and therefore has no focus or edit functionality. A direct edit operation would consist of moving around bitmaps, which would subsequently be translated to updates on the arrangement. The semantics of such edit behavior are unclear and supporting it will probably be difficult, but since direct rendering editing is not required by any of the use cases that drive the design of Proxima (Section **??**), it is

not supported.

3.1.2 Arrangement

The arrangement level is structurally very similar to the presentation level, except that line breaking has been done and elements have absolute positions. The edit operations targeted at the arrangement are edit operations concerned with lines and absolute positions, such as navigating to the end of the line that contains the focus, or selecting all elements in a rectangular area.

Currently, Proxima does not support direct editing on the arrangement level. It is not possible to actually update the arrangement level and compute an edit operation on the presentation level. Edit operations targeted at the arrangement may only involve the focus. A consequence is that although rectangular areas may be selected and deleted (the deletion takes place at a higher level), it is not possible to insert a rectangular area. Such edit behavior, often referred to as column editing in text editors, is not a big requirement for Proxima. In text editors, column editing is used to edit a text that is organized in columns, but since Proxima is a structure editor, a document whose data needs to be presented in columns can use a presentation level matrix, whose elements can be selected columnwise at presentation level.

3.1.3 Layout

The layout level is where all textual editing takes place. This includes the typing in program text in a source editor, but also text entry in the tax form editor or the word processor. Besides text insertion and deletion, also cut, copy and paste operations are supported. The focus model at the layout level is a set of contiguous areas in the layout tree.

3.1.4 Presentation

As mentioned before, the presentation level is not directly edited in Proxima. All information in the presentation level is also present in the layout level, and can therefore be edited at that level. The only reason for directly editing the presentation level would be when a user wishes to manipulate tokens without paying attention to their whitespace. Such behavior seems appropriate only in an editor with automatic whitespace handling, but in that case editing may just as well be performed at the layout level, since the whitespace will be ignored.

3.1.5 Enriched document and document

Because the enriched document is a tree, similar to the document, the edit functionality on both levels is the same. The only difference between the two is that if the enriched

document is edited, a subsequent translation takes place to compute the document update. Because of the similarities, both levels are discussed together.

The edit operations at the document level are basic tree operations, such as cut, copy and paste on subtrees. If an list element has a parent that is also a list (eg. a list of subsections in a list of sections), split and join operations can be applied. Besides the standard edit operations, an editor designer may specify other generic transformations, or domain specific transformations. ***

*** implemented only single subtree focus, is set of paths correct and feasible? or should it be from-to

Child elements that are not of optional or list type are required and hence cannot be left out of the parent element. In order to still be able to manipulate an element without its children, Proxima employs the concept of a placeholder. A placeholder of a type T is a dummy value that can be used in any place where an element of type T is required. A built-in presentation is available for placeholders, but it can be overridden in a presentation sheet. If a document contains a placeholder it is not valid. The placeholders are typically present only during the construction of a document (or part of it), or as an intermediate situation during a document modification that consists of several steps. Lists and optional types do not require placeholders: for a list type, the empty list is the placeholder, and for an optional it is the nothing alternative ***.

*** how to call this thing?

3.2 Modeless editing

One of the requirements of the Proxima editor is the possibility of modeless editing; it must be possible to freely mix edit gestures targeted at different levels without having to switch to a different editor mode. Because the structure of the presentation is often closely related to the structure of the document, many structural edit operations are also presentation edit operations, and therefore it does not matter at what level the edit operation is performed. However, in some cases it makes a difference whether an edit operation is interpreted as a structural operation, or as a presentation edit operation.

One example concerns a date element that appears in several places in a document, but with different presentations. In one place, it is presented in the US format (month/day/year), whereas somewhere else it is presented in the European style (day/month/year). However, at document level, both are represented by the same type of date element. Now, if for example the date January the twelfth in the year 1900 is copied from a US date field (”1/12/1900”) to a European one, two possible interpretations exist. Interpreted as a presentation copy, the resulting date looks similar to the source date (ie. ”1/12/1900”), but it now refers to the first of December. Interpreted as a document edit operation, on the other hand, the copied date gets a different appearance (”12/1/1900”), but it refers to the same date as its source.

A second example is if the element `1` is deleted from the Haskell list `[1,2]`. This can be interpreted as a presentation deletion of the character `1`, leading to `[,2]`. *** On the other hand, it can also be interpreted as the structural deletion of the first element of the list, leading to `[2]`.

*** explain that this one fails during parse?

Ambiguity may arise when an element's presentation parses to a different element (as in the date example), or when the presentation of a parent element depends on its children, as is the case in the Haskell example since the commas in the list belong to the list rather than the children. ***

*** don't know exactly when. Should we try to say this?

In case of ambiguity, an edit operation is performed by default on the highest possible layer. For the date example, the appearance of the date changes but the meaning is preserved. In the Haskell example, the delete operation is on document level and the comma is removed as well. The preference for a higher level seems to make sense, but will have to be confirmed by practice. ***

*** not exavtly clear

The choice of the target level for an ambiguous edit operation may be subject to variation, because optimal edit behavior is not only determined by the consistency of the underlying edit model, but also by the habits of a user. It is possible that a user finds it strange when a comma disappears in a source editor. There are several alternative ways to handle ambiguities. Firstly, backspace and delete operations, which are mainly used for deleting characters, could be always interpreted as presentation edit operations, while cut, copy, and paste may be document operations that automatically insert and remove commas. Secondly, a document delete operation may cause the creation of a volatile placeholder, which disappears when the focus is changed. When a list element is deleted, a comma remains, but changes color. If the user starts to type, the comma changes back to the text color, whereas if the user changes the focus, the comma disappears completely. Finally, a source editor may simply give priority to presentation editing over document editing. Practical use has to decide what the best choice is. In any case, a user is always be able to influence the automatic choice made by the editor, possibly after first performing an undo.

3.3 Focus model

***

*** add pictures to explanation?

Because Proxima is a modeless editor, the concept of focus exists simultaneously on several levels. Similar to the presentation and translation mappings for levels, the editor also has presentation and translation mappings for the focus on each level. Whenever the focus changes on one of the levels, the focus on the other levels is updated as well. This section explains the issues that come into play when maintaining the focus on different levels. In the Proxima editor, only a prototype focus model has been implemented. The focus model is therefore preliminary and requires further research as well as practical experience from implementation.

Between each pair of adjacent levels, Proxima keeps track of exactly which element is mapped onto which during presentation and translation. One upper level element may be mapped onto several lower level elements, or none if the element is part of the upper level's local state. On the other hand, a lower level element may be mapped onto at most one upper level element (or zero if it is part of the lower level's local state). A

consequence from the difference between the presentation and the translation mappings is that even when a lower level element depends on several upper level elements, only one is responsible for its presentation. *** The information on the mappings is used when translating the focus between levels.

*** where does this become important

The focus is modeled as a boolean property of each element in the level. If an element is in focus, then its children are also in focus, which means that an element cannot be in focus if one of its children is not. As a result, if the presentation of a document element contains the presentation of a document element that is not a descendent, then the presentation cannot have presentation focus. In such cases, the editor designer needs to specify how to handle the focus. Perhaps a more relaxed focus model is possible, but this requires further research.

For list elements, the focus is not a boolean property, but rather a set of ranges. The whole list is in focus if the set contains one range that spans ell elements, and the list element is not in focus if the set of ranges is empty. The set may also contain empty ranges that correspond to insertion points in the list.

The focus is a distributed property of the level, but it can also be viewed as a set of subtrees and ranges of subtrees. One of the subtrees or ranges is the main focus, which is used for insertion and navigation, since these operations are not clearly defined on a multiple focus.

3.3.1 Navigation

***

*** add examples and pictures?

Just as any other edit operation, a navigation operation may be targeted at various levels in Proxima. However, unlike the other edit operations, a user explicitly specifies the level on which navigation takes place. Pressing arrow keys and clicking with the mouse results in navigation on the layout level, whereas a modifier keys can be used to navigate on the document and enriched document levels. More exotic navigation operations such as selecting rectangular areas in the arrangement navigation are available through menus. Furthermore, a presentation element may bind a mouse click to an edit operation, and hence also to a navigation operation. Clicking in a presentation may therefore result in a change of focus.

When the focus is not a single subtree or (possibly empty) range in a list element, but a set of subtrees and ranges, one of the elements of the focus set is the main focus. Navigation commands operate on the main focus. ***

*** not entirely right if focus is two ranges "...¡..."¨...¿..."

Because of the different characteristics of each level, different navigation operations are used. On the document and enriched document levels, navigation is mainly tree oriented: moving up and down to parent and child elements, or left and right to sibling elements. On the layout level, most elements are in lists, and the focus consist mainly of ranges. Moving the layout focus means changing it to an adjacent empty range. The direction of the elements in the arrangement tree (horizontal or vertical) is taken into account when the

new focus position is computed. Document navigation can also include list navigation, but subtree navigation is not a common operation on the layout level.

***

Formatters in the presentation form a rather special case in focus handling. In a formatted paragraph, horizontal navigation is targeted at the presentation level; a focus move to the left or the right always goes to an adjacent element in the layout level, whereas on the layout level, it may jump to a different line. On the other hand, vertical navigation is interpreted at arrangement level; the focus moves to an adjacent line and the horizontal position is maintained as close as possible. This operation cannot be performed at the layout level due to absence of position and line information. ***

* * * mention widening of focus? (shift-arrows & shift-click)

* * * mention that nested formatters make it trickier, but still possible?

3.3.2 Presenting and translating the focus

For the presentation direction, a lower level element is in focus if its children are in lower level focus and the upper level element that presented it is in the upper level focus. For the focus translation (also known as the *pointing problem*), the situation is somewhat more complicated, since one upper level element may be presented on several lower level elements. Hence, an upper level element is in focus if all its children are in upper level focus, and also all lower level elements it is presented on are in lower level focus.

From the focus mappings it follows that focus on one level does not always result in focus on the other level. For example, if an enriched document element `Plus (Int 1) (Int 2)` is presented as the tokens `["1","+","2"]` and only the first two tokens are in presentation focus, then the sum is not in enriched document focus, but only the first integer (`Int 1`). However, if the document is presented, the presentation focus should not be lost, even though the originating enriched document element is not in focus. This can be achieved by regarding the focus as extra state, which may be set on translation or presentation, but if it is not, the previous value is reused. If an explicit navigation command was issued on one of the other levels, then all old focus information is cleared before the new focus is computed. ***

* * * is this really completely symmetrical?

Extra state elements on both upper and lower level may also have focus, although if the upper level focus is in upper level extra state, then it this not visible at the lower level and hence also not in the rendering. Focus in extra state elements is handled by the same mechanisms that handle extra state in general. Similar to the non-extra state elements, the extra state focus is cleared in case of an explicit navigation action on one of the other levels. ***

* * * mention that just like other ES, es focus may get lost?

The proposed focus translation mapping is not ideal for handling focus in extra state. Because extra state in a lower level represents non-essential information, it can be useful when focus on the extra state is not considered when computing the higher level focus. As an example, consider the ernriched document element `Int 2`, with a presentation `IntToken (0,2) 2` and corresponding arrangement "␣␣2". If the `'2'` and only part of the preceding whitespace is in arrangement focus (␣ ␣2), then the presentation focus will include the `2` child of the token, but not the whitespace child `(0,2)`, since its arrangement

is not entirely in focus. Therefore, the `IntToken` parent cannot be in focus either, which in turn means that the `Int` element is not in document focus.

***** A remedy to the situation is not to use the condition that all lower level elements that an upper level element is presented on are in focus, but rather that all lower level elements are in non-extra state focus. The non-extra state focus can be computed for each lower level element by checking whether each of its children is either in the non-extra state focus or it is in the extra state (with regard to the upper level that is considered). If an element is in focus, then it is also in the non-extra state focus. *** ***

* * * this is not correct, we need a self focus as well

* * * on presentation is there a similar problem?

List elements form a special case in the focus presentation and translation process. If a list element is mapped onto an isomorphic list ***, then the focus ranges can simply be copied. If the the list is mapped onto a list that has different lenght or is reordered, then the editor designer must provide a function that maps ranges in one list onto ranges in the other. If a list is mapped onto a structure that is not a list, or vice versa, then focus handling is left to the editor designer. If no mapping is provided, then the focus cannot be translated to the other level. For example when the presentation focus is on a ”then” token in an if expression, there is no corresponding enriched document focus.

* * * right term: isomorphic?

3.3.3 Focus ambiguity

Besides ambiguity concerning the level at which an edit operation is targeted, there is also ambiguity regarding the focused element on a single level. If a parent element does not have any presentation elements of its own, then there is no difference at the lower level between the selection of its children alone or the selection of the children together with the parent. An example is the following fictitious element (`Italic (Word "bla")`) of a word processor editor. It is presented as ıbla. If the focus is in front of the first letter, it is not clear whether it is inside the italic region or not.

Focus ambiguities can be solved can be partially solved by letting the focus depend on the direction from which the focus was navigated. For the example, this means that if the focus came from the right (between the ’b’ and the ’l’ characters), it is inside the italic region, whereas if it came from the left, it is outside. However, as soon as more than two levels of ambiguity exist, for example with (`Bold (Italic (Word "bla"))`), this strategy is no longer sufficient. In such a case, the user needs to use a more structural view if the desired navigation is not possible in the regular view.

A related case of ambiguity arises when the enriched document type

```
data Exp_Enr = IdentExp_Enr Ident_Enr | ...
data Ident_Enr = Ident_Enr String
```

has such a presentation that the expression `IdentExp (Ident "x")` is presented only as the lower case token `"x"`. If the token is in presentation focus, it is unclear whether the enriched document focus is on the string, the identifier, or the expression. The problem can be approached by either merging the possible edit operations for each of the possi-

bilities, or by marking one possibility as the main one that should receive the focus on selection. The Cornell Synthesizer Generator (see Section **??**) employs the latter solution and denotes the element that receives the focus as a *resting place*. For Proxima, no final choice has been made, since the merging solution requires further research.

Until now, we considered that an edit operation works on the element that is in focus. However, in some cases, it is useful if ancestors of the currently focused element can be edited. For example when the focus is somewhere in a chapter, but not on the entire chapter, it is still useful to be able to perform chapter specific edit commands and transformations for that chapter. Similarly, we want to be able to select or move a declaration by clicking somewhere in its presentation. Therefore, apart from the edit operations on the currently focused element, Proxima also gives access to the edit operations on its ancestors.

3.4 Bypassing higher layers

During editing, it is possible to temporarily skip the updates on the higher levels. The main reason for this is efficiency, but in some cases it also increases the usability of the editor. An example of the efficiency reason is that when a user is typing program text, the invocation of the parser can be postponed until the user inserts a whitespace character or performs a navigation. However, even when a higher layer is not computationally expensive, it may be desirable not to invoke it on every edit action because the constantly changing presentation may be confusing for a user. For example when a field in a tax form editor is changed by typing a number, it may distracting to see the computated fields updated on every key press.

When one or more higher layers have been skipped, the data levels may not be consistent with regard to the presentation and translation mappings. The level of consistency of a presentation is shown to the user. However, this does not mean that the editor has different modes, because it is always possible to enforce that all layers are invoked. If a higher level edit operation is issued, the intermediate layers are first invoked in order to guarantee consistency. Because it must be possible to complete the entire translation process, no layer may halt on error. Therefore, an error correcting parser is used at the presentation level. Even if the layout contains a parse error, the parser still returns an enriched document in which incorrect parts are marked. The parser can be tuned to help keep an error as local as possible in the tree.

*** ***

*** Should we say more about: ***

- Multiple document views/presentation?
- Generalized Paste?
- Local/extra state and the structure watching behavior following from it?

* * * explain better why this is not a mode switch?

* * * also talk about scanner and evaluator?