

Conclusions and further research

In this thesis, we have introduced the presentation-oriented structure editor Proxima. Proxima provides an intuitive and user-friendly way of editing for complex document presentations which may contain derived structures.

In Chapter 2 we investigated a number of use cases for a generic editor, and formulated a set of functional requirements based on these use cases. We have provided an evaluation of existing generic editors with regard to the requirements. It turns out that none of the existing systems is able to handle all of the use cases. In our opinion, the reason for this is that these editors either lack the power to support the complex presentations of the use cases, or have an edit model that is overly restrictive. The chapter ended with a brief description of the Proxima editor, which has been designed to meet the requirements and will be able to handle all of the use cases.

Proxima has a layered architecture that makes it possible to support both presentation-oriented and document-oriented editing. An overview of the layers and data levels of the architecture has been provided in Chapter 3, followed by a specification in chapters 4 and 5.

In Chapter 6 we introduced the declarative presentation formalism XPRES. The language is a Haskell combinator library for creating graphical presentations with an advanced alignment model. Using Haskell's abstraction facilities, complex presentations may be defined in a concise way.

A prototype for Proxima has been implemented and was discussed Chapter 7. To instantiate an editor, a basic Haskell document type definition must be provided together with a presentation sheet and a parsing sheet. The presentation sheet is an attribute grammar,

and the parsing sheet is a combinator parser. A number of example editors have been instantiated with the prototype.

8.1 Further research

The formal specification has provided a wildcard representation for extra-state equivalence classes. However, the specified reuse function is rather basic and only allows reusing extra state after simple edit operations. The specification should be improved with a more advanced reuse function, a sketch of which was provided. Furthermore, the specification for an editor supporting duplicates in the presentation was only informal. A more formal definition of the notion of duplication as well as of the mechanism of ignoring duplicates is needed to establish a formal specification for the editor that supports duplicates.

In Proxima, functions for both the presentation and the interpretation direction need to be specified. The inverse of the presentation sheet is the parsing sheet, and the inverse of the evaluation sheet is the reduction sheet. Because specifying these inverses by hand is a possible source of errors, a bidirectional presentation formalism, which automatically generates an inverse function, is desirable.

The evaluation layer of Proxima is an ideal place for such a bidirectional presentation (or rather evaluation) formalism. Thus, the reduction sheet will be automatically generated from the evaluation sheet. For the presentation layer the situation is somewhat more complicated because it does not seem realistic to assume that an efficient parser can always be generated automatically from the presentation sheet. On the other hand, many parts of the presentation are straightforward and could be inverted automatically. In the prototype, for example, it is already clear how the structure recognizers in the parsing sheet can be derived from the presentation sheet. For the more difficult parts of the presentation, the presentation sheet could contain special directives for parsing, or even an explicit specification of the appropriate part of the parser.

Chapter 7 already provided an overview of the future research concerning the Proxima prototype. We recapitulate a few main issues here. The most important area of research is support for incrementality, consisting both of built-in incremental behavior for the lower layers, as well as the application of techniques for incremental attribute evaluation to the attribute-grammar compiler. A second important area concerns the evaluation layer for which we need to establish the formalisms for the sheets, as well as provide an implementation. Furthermore, we need extensions to the attribute grammar formalism and the XPRES presentation language. And finally, libraries of useful functions must be compiled, to facilitate the specification of common editor behavior.

8.2 Final remarks

The approach taken for Proxima is different from most other generic editing projects. Most of these projects take as the starting point a specification formalism that guarantees a correct and efficient editor for a limited range of applications. Proxima, on the other hand, provides a general architecture with presentation and computation formalisms that are powerful enough to build serious editor applications. In the initial stages of the Proxima project, it is left to the editor designer to guarantee safety and efficiency of the implemented editor. Support for automatic interpretation and incrementality will be added at a later stage.

Because both the presentation and the interpretation need to be specified, it is possible to specify an editor for which the parser does not match the presentation. Furthermore, because the presentation formalism allows arbitrary computations, it is possible to specify a presentation that is too slow for editing, or even crashes. Nevertheless, the safety of an editor built with Proxima is already much easier to guarantee than if a similar editor had been built by hand. Further, in practice, it turns out to be rather straightforward to avoid inconsistencies between the presentation and interpretation functions.

An advantage of our approach is that even in an early stage, complex editors can be specified (albeit with a little more effort). And, moreover, it is possible to specify editors for which automatic interpretation is not yet an option. A related advantage is that by building and experimenting with editors, it becomes clear which parts of the generic system should get the highest development or research priorities.

Even without the language support and libraries for common presentation patterns, it is already straightforward to specify a complex editor in Proxima. Thus, the Proxima prototype shows that it is possible to combine a powerful presentation formalism with a modeless integration of document-oriented and presentation-oriented editing. The resulting editors are powerful, yet easy to use.

