A Text Model for the Java-Based Model-View Multieditor *

J. Kiho, V. Hagu, M. Sutt

Institute of Computer Science, University of Tartu
Liivi str.2, Tartu, Estonia
e-mail:{kiho veljo msutt}@cs.ut.ee

Abstract. In this paper further developments of the project *AMADEUS*, aiming at implementing diagram-style multieditors are considered. A structured text model for the new *Java*-based model-view version is outlined. The text model is suitable for arbitrary source texts which consist of (nested) components, (possibly) having caption(s) and one or more branches. Each branch has members, such as primitive (plain) text, (sub)component et al. The model is independent of views i.e. display/edit modes, so a number of different user-oriented views can be developed on its basis.

1 Previous Work

The idea of developing a program multiviewer [3] was proposed and corresponding project *AMADEUS*, as a students' group work, was started in 1995 at the University of Tartu. *AMADEUS* aims at combining both textual and graphic editors for handling source programs. A given program can be viewed and edited both as a text and as a two-dimensional diagram. For the latter, the sketchy representation technique [6] has been chosen. It has certain advantages compared to other graphical notations [8][9]. A sketchy program describes program structure by means of simple (nested) graphical blocks, called the sketches, containing textual elements – elementary declarations, comments, statements, procedure calls and conditions from a textual host language, and determining control flow by graphical means instead of keywords.

AMADEUS is supposed to support a wide spectrum of host programming languages. At present stage it includes *Pascal* and *C*-family (incl. *Java*) programming languages. Beside usual text/graphics processing facilities and some others [3], the most specific features of the program multiviewer are the transformations [2] between textual and graphic (sketchy) views, and structure reduction [4] of sketchy programs.

Up to now, *AMADEUS* is ported to *Windows NT* platform and some new features, including TEX/LATEX processing and sketch shrink(iconify)/unshrink tools, are added.

Developing such multi-purpose and semi-graphic editors relates closely to general problems of development heterogeneous systems [7][10].

* Partly supported by Estonian Science Foundation, grant no 3343.

2 Generalization

At present, a new Java-based version of *AMADEUS* (for Java 1.1 [1]) is under development. Beside preserving all previous source program editing features (sketchifying, reducing, normalizing, textualizing), some essential generalization steps are taken.

So the new version is designed to support more advanced *undo* methods and free choice of fonts together with image inclusion (similarly to Oberon [5]). But even more important is the new approach, based on the universal nature of the sketchy diagramming techniques. Most of the texts considered in the software developing process are, in fact, represented as two-dimensional (commented) diagrams (with captions/heads), consisting of some (commented) text lines or, in turn, such (sub)texts. Common text editors force to use some special characters (such as curly brackets or `BEGIN/END` keywords) to indicate borders of such texts and sub-texts. It is obviously rather inappropriate in the case of vertically positioned diagrammic blocks. Sketchy diagramming techniques suggests alternative and more convenient graphic tools (prolonged vertical brackets, for instance) for the block indicators. In particular, the new version of *AMADEUS* is based on the model-view approach, enabling to use several different graphic views for the same text model. At present, beside the sketchy view, also box view, tree-like view and hypertext view are implemented. This spectrum may be easily extended by any kind of other views, for instance, *Schemacode* [11], *Pancode/Boxchart* [12] or whatever.

3 Sketchy Text Model

In this section, a brief explanation of the basic text model for *AMADEUS*, called the *sketchy text model*, is given. Corresponding *Java*-classes are described in the next section.

A text diagram or block is called the *sketch* and described as class `Sketch`. The latter inherits common attributes from its superclass `SketchInfo`, where, among others, a reference to class `SketchView` (defining the current view and user interface methods) is given. Since the model is separated from the view, here we do not describe the class `SketchView` (nor any other of the view-oriented classes).

Every instance of sketch has its type (`BRANCH`, `SIMPLE`, `LOOP`, etc.).

A sketch instance of type `BRANCH` is called the *branch*, otherwise it is called the *non-branch*. Every non-branch is a collection (body-vector) of branches. Every branch is a collection (body-vector) of *sketch members* - primitive texts (non-sketchified texts of type Text), conditions , arrows and non-branches. In addition, every sketch has a comment and a collection (head-vector) of head items. A source (structured) text in this model is a non-branch.

For instance, the text

```
switch(s) // choose on s {
 case 1: <Part1>
```

```
  case 2: <Part2>
  case 3: <Part3>
  {<Common part>}
 }// switch
```

may be represented as a non-branch of type `SWITCH` (with comment `switch`), which head has one item (`switch(s)`) with a comment (`choose on s`), and four branches, each consisting of one non-branch (three of them of type `CASE`, the last one of type `SIMPLE`). The non-branch `CASE` serves as an example of a sketch with a single head item and a single branch. The head of a non-branch of type `SIMPLE` is empty in this case.

The following text

```
for(i=0; i<m; i++)// for each row
 for(j=0; j<n; j++)// for each column
  a[i,j] = 0;
```

may be considered as a non-branch (of type `LOOP`), head of which has two items (commented texts). It has one branch, containing only one member (primitive text `a[i,j] = 0;`).

However, in this sketchy text model some structuring of primitive texts as well as sketch heads is imposed: an instance of class Text consists of a number of rows and of a comment (to this text). Corresponding view methods (in the current version) display rows of a text line by line, appending the comment at the end of the last row. Notice also, that all comments are restricted to the type Row.

4 Classes of the Model

```
class SketchInfo {
    // sketch types:
    static final byte BRANCH = 0;
    static final byte SIMPLE = 1;
    static final byte LOOP = 2;
    static final byte SWITCH = 3;
    static final byte CASE = 4;
    static final byte DEFAULT = 5;
    static final byte TRY = 6;
    static final byte CATCH = 7;
    static final byte SYNCHRONIZED = 8;
    static final byte CLASS = 9;
    static final byte INTERFACE = 10;
    //sketch attributes:
    byte type;
    Row comment;
    Sketch parent;
    SketchView view;
   }
```

```
class Sketch extends SketchInfo {
 Vector head; // elements (head items) of type Text
 Vector body;
 // body element types:
  // if this sketch is branch
  // then Text/Condition/Arrow/non-branch
  // else branch
}

class Row {
 Vector rowElements;
 // elements are of type Symbol or Image
 RowView view;
}

class Text {
 Vector rows; // elements of type Row
 Row comment;
 TextView view;
}

class Symbol {
 char ch;
 Font font;
 Color color;
}

class Condition {
 Text expression;
 ConditionView view;
}

class Arrow {
 boolean strong;
 short endpointLevel;
 ArrowView view;
}
```

5 Conclusion

Due to its universal nature, developed in this work sketchy text model, together with corresponding methods for handling its components, may serve as a basis for implementing (several) multieditor(s) with a big variety of views. Since the model is separated from the view, adding a new view involves no changes in the model. Given an appropriate sketchifier, which maps source texts in given language onto the model, the multieditor can provide different comprehensive (graphic and edit-able) views of the texts. Such multieditors can be used not only for more effective developing program source texts, but also in the case of many other textual sources (data description languages, TEX/LATEX, *HTML* et al.).

References

1. J. Lewis, W. Loftus, *Java Software Solutions - Foundation of Program Design* Addison-Wesley, 1997.
2. J. Kiho, D. Brodski, A. Podlipski, "Control Structure Transformations in the Program Multiviewer" *Proc. 5th Symposium on Programming Languages and Software Tools, June 7-8, 1997, Jyväskylä*, Report C-1997-37, University of Helsinki, pp. 61-71, 1997.
3. J. Kiho, A. Oja, "Designing a Program Multiviewer" *Proc. 7th Nordic Workshop on Programming Theory, Nov. 1-3, 1995, Göteborg*, Programming Methodology Group Report 86, Göteborg University and Chalmers University of Technology, pp. 165-184, 1996.
4. J. Kiho, "Self-reducing control structures in the sketchy programming environment" *Proc. Nordic Workshop on Programming Environment Research, June 1-3, 1994, Lund*, LU-CS-TR:94-127, Lund University, pp. 333-334, 1994.
5. N. Wirth, J. Gutknecht, *Project Oberon - The Design of an Operating System and Compiler* Addison-Wesley, 1992.
6. J. Kiho, "Diagramming techniques and sketchy programming" *Proc. 2nd Symposium on Programming Languages and Software Tools, August 21-23, 1991, Pirkkala, Finland*, Report A-1991-5, University of Tampere, pp. 78–82, 1991.
7. S. Meyers, "Difficulties in integrating multiview development systems" *IEEE Software*, **8** 1, pp. 49-57, 1991
8. L. L. Tripp, "A bibliography on graphical program notations" *ACM SIGSOFT Software Engineering Notes*, **14** 6, pp. 56-57, 1989.
9. L. L. Tripp, "A survey of graphical notations for program design, an update" *ACM SIGSOFT Software Engineering Notes*, **13** 4, pp. 39-44, 1988.
10. M. S. Feather, "Language support for the specification and development of composite systems" *ACM Transactions on Programming Languages and Systems*, **9** 2, pp. 198–234, 1987
11. Schemacode: Knowledge editor for program implementation, http://www.rgl.polymtl.ca/schema/tool.htm
12. The Pancode/Boxchart Home Page, http://www.algonet.se/ djon/panbox/