Automatic Construction of Intelligent Diagrammatic Environments

Bernd Meyer¹, Hubert Zweckstetter¹, Luis Mandel², Zoltan Gassmann¹

¹Univ. of Munich, Dept. of Computer Science, Oettingenstr. 67, 80538 Munich
²FAST e.V., Arabellastr. 17, 81925 Munich, Germany
contact email: bernd.meyer@acm.org

1 Introduction

Graphical user interfaces have become an integral part of almost every modern application type and it can be claimed that they are among the driving forces that have made the computer accessible to non-expert users. However, comparing the use of graphics in existent user interfaces with that in non-computer-based work, the inadequacy of standard GUIs for complex visual communication is revealed: Most GUIs are still WIMP interfaces centered around such simple interaction devices like icons, buttons, menus or image maps. On the contrary, in non-computer-based work rich and highly structured graphical notations prevail. There are diagrammatic languages in almost every technical discipline, for example circuit diagrams, architectural floor plans or chemical formulas, and modern software engineering is embracing all kinds of diagrammatic specification methods. Likewise, non-technical fields use their own well-established diagrammatic systems, for example choreography notation. But even in such application domains where diagrammatic notations are a natural element of discourse and where their meaning is well-understood, they are still rarely utilized to their full extent in the human-computer interface. It is hardly ever possible for the user to communicate with a computer just by sketching an annotated diagram like two human experts would do in their discourse. This poor integration of richly structured visual communication into graphical user interfaces is mainly due to a lack of method and tool support for the implementation of the required interfaces. While countless specification mechanisms and interface builders for WIMP interfaces exist, there are almost no tools and only few specification formalisms for building intelligent graphical interfaces for such richly structured diagrammatic notations.
If we want to unleash the full power of diagram languages, we have to turn to a syntax and semantics-based high-level approach for the implementation of diagrammatic front-ends.

In the following we present Recopla, a meta-environment for the automatic construction of diagram environments from declarative high-level specifications.

From a specification of appearance, syntax and semantics of a diagram language Recopla generates a specialized interface for the specified diagram type. These automatically constructed front-ends are capable of checking the correctness of a given diagram and can interpret the defined diagram types and translate them into a representation suitable for processing by the back-end application. The generated front-ends are self-contained Java applications that can flexibly communicate with arbitrary back-end applications via dedicated bidirectional Internet communication channels. Due to Recopla’s non-monotonic incremental interpretation capabilities, these interfaces do not limit the user to syntax-directed interaction but allow natural interaction with the diagram. In contrast to virtually all other systems, Recopla’s interpretation component incrementally maintains a consistent interpretation even under object deletion and other arbitrary modifications of the diagram.

2 Building Diagrammatic Interfaces from Declarative Specifications with Recopla

Our meta-environment for the automatic construction of intelligent diagrammatic interfaces from declarative formal specifications consists of a syntax-based high-level interpretation component (Meyer and Zweckstetter 1998) on top of the generic editor-builder Recopla (Gassmann 1998).

The syntax and semantics specification of the diagrammatic language is achieved by using attributed grammars in the form of conditional relational grammars (CRG, Meyer and Zweckstetter 1998). CRGs are a multidimensional grammar formalism that inherits from both relational grammars and constraint multiset grammars (Marriott, Meyer and Wittenburg 1998). In addition to specifying the syntax and semantics (i.e. the interpretation) of diagrammatic notations, conditional relational grammars also provide mechanisms to specify interpretation-based modal interaction, so that immediate semantic feedback can be given to the user without requiring the intervention of the back-end.

The appearance of the basic graphical entities, i.e. the visual vocabulary (such as icons, line types, etc.), that may be used in an editor instance is specified interactively on the meta-level by drawing prototype entities with Recopla’s object-oriented graphics editor. The syntax, i.e. the permissible ways to arrange these graphical entities spatially and topologically into valid diagrams, is defined with a CRG. The interpretation and translation of diagrams is specified with the same
formalism by using attributed productions and/or by attaching actions to productions. As a toy example consider the following production which is taken from a simplified grammar for state transition diagrams:

```
Transition --> Arrow, Label ; % where
  exists(State_1, State_2), % such that
  starts_at(State_1, Arrow),
  ends_at(State_2, Arrow),
  close_to(Label, Arrow) ; % and set
  Transition.condition = Label.string,
  Transition.from = State_1.name,
  Transition.to = State_2.name.
```

CRGs allow the usage of fully context-sensitive productions and of arbitrary user-defined predicates that are evaluated as conditions on the objects' attributes. A CRG is essentially processed in bottom-up fashion by simultaneously rewriting a set of graphical objects and a set of spatial relations between these objects.

In interactive systems it is highly desirable to support immediate semantic feedback from the interpretation without restricting the admissible user-interactions. For systems working on the basis of a formal syntax definition this is usually a difficult problem since they have to address the issue of incremental parsing and, more importantly, of non-monotonic incremental interpretation if they want to support unrestricted interaction.

In Recopla's interpretation component this is solved by viewing parsing as logical deduction: The grammar execution can essentially be understood as a bottom-up derivation of definite clauses with set-valued attributes. This perspective allows us to employ deduction techniques developed in the area of Truth Maintenance Systems (Doyle 1979, 1981; DeKleer 1985, 1989). In this way a consistent interpretation can incrementally be maintained when the diagram is changing without that a complete re-interpretation has to be performed from scratch.

Despite the rigorously specified syntax this schema frees the user from the restraints of syntax-directed editing. If required, immediate feedback during editing as well as modal interaction situations can be specified in a flexible manner using one of two mechanisms: (a) In a declarative manner by grouping the productions according to editing modes and activating/deactivating groups depending on the current context, (b) in a procedural manner by attaching arbitrary user-defined actions to the productions.

Since a vast number of diagrammatic notations used in practice essentially have a (hierarchical) graph as their underlying backbone structure, Recopla is designed to offer special support for the definition of and interaction with graph structures. However, the system also supports editing and interpretation of more general, non-graph-like diagrammatic notations.
3 Evaluation and Future Plans

Recopla was successfully used to build editors for a number of standard notations, among these animated Petri nets, object-oriented class diagrams, timing diagrams, digital circuit diagrams, extended flow charts and even interactive board games.

Few other editor construction toolkits exist that support declarative high-level specifications. Notable exceptions are Penguins (Chok and Marriott 1998), DiaGen (Minas 1995) and GenEd (Haarslev 1998a). The features distinguishing Recopla from these systems are:

1. an expressive and fully context-sensitive specification formalism,
2. the capability to handle non-monotonic interaction with a fully incremental interpretation method that supports instantaneous semantic feedback,
3. support for flexible specification of modal interaction,
4. built-in support for hierarchical graph structures,
5. generation of portable, monolithic Java-based stand-alone editors.

While all of the above mentioned systems support some of these features, none of them supports all. Penguins additionally performs a form of error correction, a capability not yet offered by Recopla’s interpretation component. While the formalism underlying GenEd could in principle support error correction, the system does not yet utilize these capabilities (Haarslev 1998b).

Current development plans for future versions of Recopla include automatic layout mechanisms, an animation specification language such as Villon (Meyer 1997), and the tighter integration into web-based settings by generating signed applets instead of applications.
4 References


