Ein graphischer Editor für Generische Zeichnungen

Eike Schwindt
schwindt@uni-paderborn.de
Universität Paderborn
Institut für Informatik

Betreuer der Arbeit: Prof. Dr. Uwe Kastens
Art der Arbeit: Studienarbeit
GI-Fachbereich: Graphische Datenverarbeitung

Zusammenfassung

Der in dieser Arbeit entwickelte Editor ermöglicht die interaktive Spezifikation graphischer Darstellungen von Sprachelementen visueller Sprachen. Aus der graphischen Repräsentation erzeugt der Editor eine entsprechende textuelle Spezifikation für *VL-Eli*, ein Werkzeugsystem zur Implementierung visueller Sprachen. Dadurch wird die Entwicklung anwendungsspezifischer visueller Sprachen wesentlich erleichtert und beschleunigt.

1. VL-Eli und Generische Zeichnungen

Die Implementierung visueller Sprachen erfordert umfangreiche konzeptionelle und technische Kenntnisse aus vielen unterschiedlichen Bereichen: von der Gestaltung der Benutzungsschnittstelle über die graphische Implementierung bis hin zu allgemeinen Aspekten der Sprachanalyse und -weiterverarbeitung. Das Werkzeugsystem *VL-Eli* [1, 2] kapselt das Wissen, das zur Implementierung visueller Sprachen notwendig ist, und generiert aus Spezifikationen mit hohem Abstraktionsgrad Strukturediteren als Frontend einer visuellen Sprache.

Die graphischen Details einzelner Sprachelemente sind dabei durch sog. „Generische Zeichnungen“ festlegbar. Im Rahmen dieser Arbeit wurde ein Editor entwickelt, mit dessen Hilfe solche Generische Zeichnungen interaktiv auf graphischem Weg spezifiziert werden können.

Abbildung 1 zeigt eine generische Zeichnung, die das Aussehen des If-Konstruktes in Nassi-Shneiderman-Diagrammen [3] beschreibt. Strukturell werden durch dieses Programmkonstrukt drei Dinge in Beziehung gesetzt: eine Bedingung und zwei Anweisungsfolgen. Die Generische Zeichnung beschreibt deren relative Anordnung zueinander, zusätzliche vektorgraphische Linien und Symbole, die zur Abgrenzung oder zur Verdeutlichung von Zusammenhängen dienen, und schließlich das dynamische Verhalten im Fall von Größenänderungen.

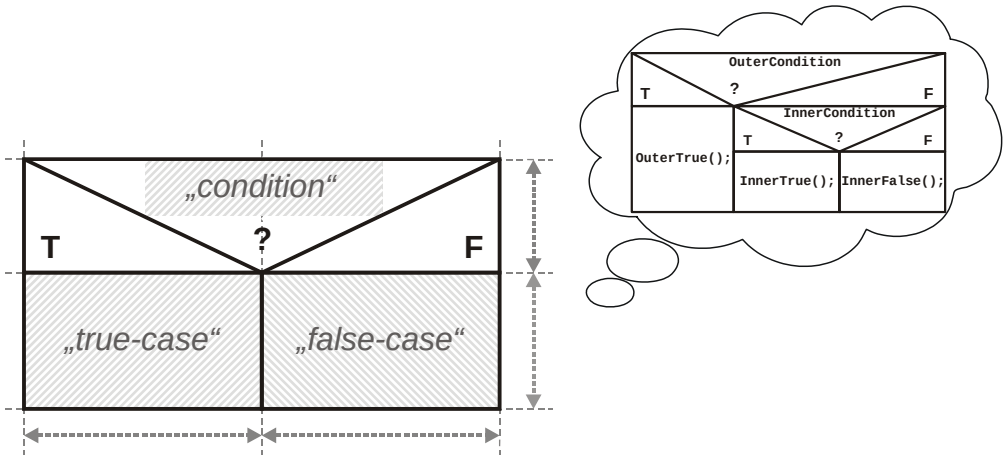

Abb. 1: Generische Zeichnung des NSD If-Konstruktes

Das prinzipielle Erscheinungsbild wird durch vektorgraphische Elemente bestimmt, im Beispiel sind das die schwarzen Linien und Symbole. Außerdem ist festgelegt, an welchen Positionen der Zeichnung Detailinformationen von Unterobjekten dargestellt werden sollen. Dazu dienen Platzhalter, sog. Container, die in der obigen Abbildung durch benannte schraffierte Rechtecke gekennzeichnet sind. Für den Fall, dass die Darstellung eines Unterobjektes mehr Raum einnimmt als durch den Container reserviert wird, muss schließlich definiert werden, wie dieser Platz durch entsprechendes Dehnen der Zeichnung bereitzustellen ist. Dies geschieht durch Dehnungsintervalle, die im Beispiel als gestrichelte Pfeile dargestellt sind. Falls erforderlich, werden die Koordinaten innerhalb eines solchen Intervalls linear gedehnt, bis der Container die erforderliche Größe zur Aufnahme des Unterobjektes erhält. Die Wirkungsweise der Dehnungsintervalle ist im rechten Teil der obigen Abbildung ebenfalls zu ersehen.

Das graphische Frontend der von *VL-Eli* erzeugten Struktureditoren ist mit der Tool Command Language/Tool Kit (Tcl/Tk) [4] realisiert, daher werden Generischen Zeichnungen in Anlehnung an die dort verwendete Notation über eine Angabe von Vektorelementen und -koordinaten spezifiziert. Die Spezifikationsnotation ist dabei in die Wirtssprache des Systems eingebettet, d.h. es handelt sich um Methodenaufrufe einer speziellen C++-Klasse. In Abbildung 2 wird beispielhaft die *VL-Eli* Spezifikation des If-Konstruktes aufgelistet.

```
figure(2,"rect","-fill white"); point(0,0); point(100,60);
figure(2,"line",""); point(0,30); point(100,30);
figure(3,"line",""); point(0,0); point(50,30);  point(100,0);
figure(2,"line",""); point(50,30); point(50,60);
figure(1,"text","-text T -anchor c"); point(15,22);
figure(1,"text","-text F -anchor c"); point(85,22);
figure(1,"text","-text ? -anchor c"); point(50,24);

container("true-case", 0,30,50,60,VisAlignScale, VisAlignScale);
container("false-case", 50,30,100,60,VisAlignScale, VisAlignScale);
container("condition", 25,0,75,15,VisAlignCenter, VisAlignCenter);

stretchX(0,50);
stretchX(50,100);
stretchY(0,30);
stretchY(30,60);
```

Abbildung 2: Textuelle Spezifikation des If-Konstruktes in *VL-Eli*

Obwohl die textuelle Notation zur Spezifikation Generischer Zeichnungen in *VL-Eli* sehr deklarativ und leicht verständlich ist, bietet sich eine visuelle Spezifikation an. Bei der ausschließlichen Nutzung der textuellen Beschreibung lässt sich das Aussehen einer Zeichnung nur kontrollieren, indem wiederholt aus der textuellen Spezifikation ein ausführbarer Editor erzeugt und gestartet, die visuelle Darstellung überprüft und – falls notwendig – die Spezifikation verändert wird. Dieser Zyklus aus »Spezifizieren-Generieren-Kontrollieren-Spezifizieren« kann sehr zeitaufwendig sein und lenkt die Konzentration eines Anwenders auf einen eher untergeordneten Aspekt der Sprachentwicklung. Zudem ist für einen ungeübten Anwender nicht immer sofort ersichtlich, welche Änderungen der textuellen Beschreibung vorzunehmen sind, um die erwünschte Darstellung zu erzielen.

2. Ein Graphischer Editor als visuelle Entwicklungsumgebung

Im Unterschied dazu sind in einer visuellen Entwicklungsumgebung zur Spezifikation Generischer Zeichnungen, wie sie der in dieser Arbeit konzipierte Editor darstellt, die Auswirkungen von Änderungen sofort ersichtlich und noch im Entwicklungsprozess korrigierbar. Ein Anwender muss daher lediglich das gewünschte Erscheinungsbild eines visuellen Sprachelementes zusammenstellen. Der Editor generiert daraus den entsprechenden textuellen Code für *VL-Eli* und speichert diesen in einer Spezifikationsdatei. Um eine eventuelle manuelle Nachbearbeitung des erzeugten Codes zu erleichtern, werden bei der Generierung Aspekte des Pretty Printing unterstützt und gleichartige Elemente einer Spezifikation zusammengefasst .

Einen Schwerpunkt der Arbeit bildet die intuitive und komfortable Bedienbarkeit des Editors. Dazu wurden geeignete Darstellungsformen für die unterschiedlichen Zeichnungselemente entwickelt, um einem Anwender auf übersichtliche Art und Weise möglichst viele Informationen zu vermitteln, vor allem auch solche, die nicht nur die rein graphischen Ausdrucksmittel einer Generischen Zeichnung betreffen. So wird beispielsweise die Ausrichtung von Containerinhalten durch die Positionierung des Namens innerhalb des jeweiligen Containers veranschaulicht.

Für die vektorgraphischen Elemente bietet der Editor die aus anderen Graphik- und Zeichenprogrammen bekannten Interaktionsoperationen und Editiermöglichkeiten. So lassen sich die verschiedenen Elemente in einer Werkzeugleiste auswählen und beliebig auf der Zeichenfläche platzieren, verschieben und wieder löschen. Durch die Bereitstellung entsprechender Dialoge können zulässige Attribute der einzelnen Zeichnungselemente, wie z.B. Farben oder Linienbreiten, festgelegt und in eine korrekte textuelle Beschreibung umgesetzt werden, was das „Durchforsten“ von Benutzerhandbüchern für Tcl/Tk überflüssig macht.

Ein Zeichnungsgitter unterstützt die einfache Positionierung, Skalierung und Ausrichtung von Zeichnungselementen, wobei die erforderlichen Koordinatenberechnungen, die bei der ausschließlich textuellen Spezifikation einen erheblichen Zeitaufwand erfordern, für den Anwender vollständig transparent sind.

Die graphische Darstellung einer visuellen Sprachen benötigt im Allgemeinen mehrere unterschiedliche Generische Zeichnungen, die aber sinnvollerweise in einer gemeinsamen Datei spezifiziert werden. Daher kann ein Sprachentwickler mit dem Editor eine Liste von Zeichnungen anlegen und diese jeweils einzeln bearbeiten. Die Code-Generierung fasst die Zeichnungen in dieser Liste dann in einer gemeinsamen Spezifikationsdatei zusammen.

Abbildung 3 zeigt einen Überblick des Editors anhand eines Screenshots, wobei im linken oberen Fenster die Liste von Zeichnungen, im rechten oberen Fenster die Detailansicht einer Zeichnung und im unteren Fenster ein Ausschnitt des generierten Codes zu sehen ist.

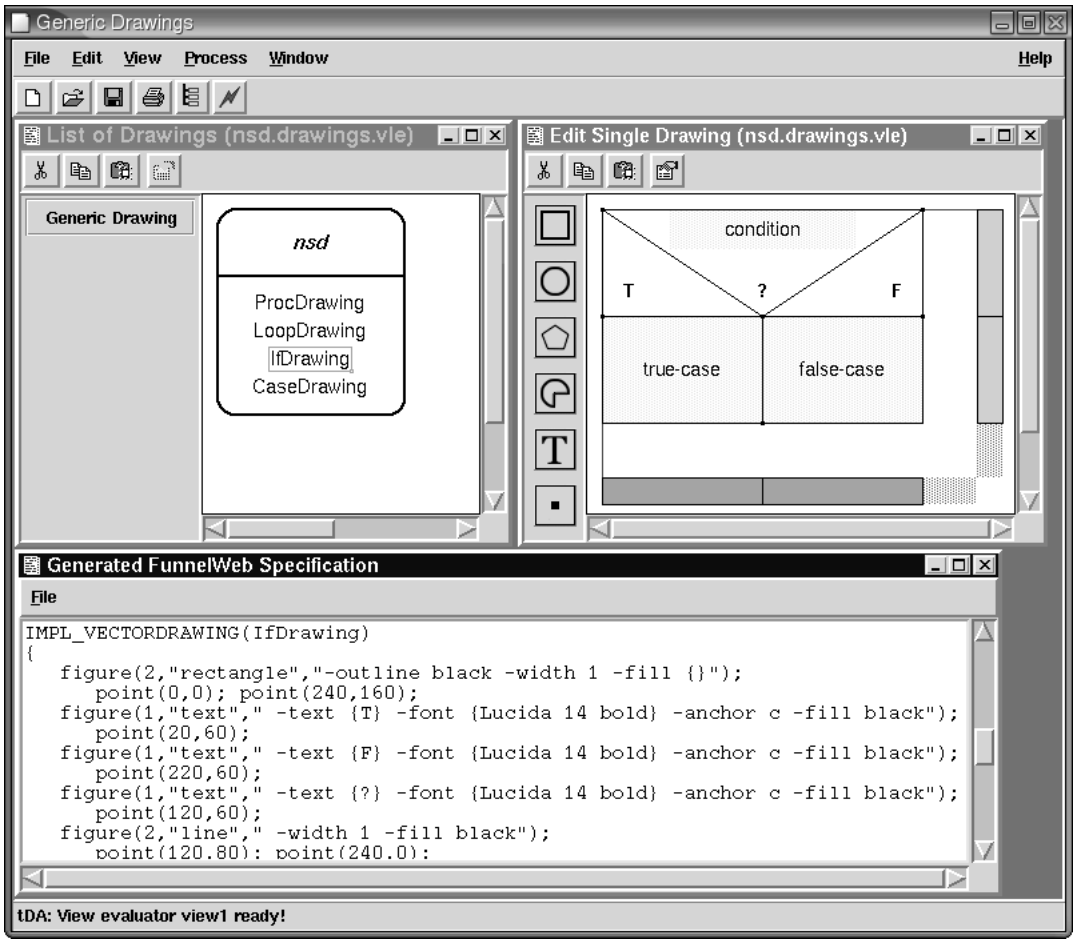

Abb. 3: Übersicht des entwickelten Editors

Ein weiterer großer Vorteil des Editors ist, dass während der Konstruktion einer Generischen Zeichnung Konsistenzprüfungen vorgenommen werden können. Fehlerhafte bzw. unvollständige Benutzereingaben führen zu entsprechenden Warnmeldungen und der Markierung desjenigen Programmkonstruktes, das den Fehler verursacht hat. Ein Anwender kann zudem keinen Code generieren, solange noch Unregelmäßigkeiten festgestellt werden. Dadurch wird die Einbindung mangelhafter Generischer Zeichnungen in die Spezifikation einer visuellen Sprache unterbunden und eine mögliche Fehlerquelle von vorneherein ausgeschlossen.

Durch den in dieser Arbeit entwickelten Editor lassen sich also die graphischen Details einer visuellen Sprache deutlich schneller und sicherer entwickeln. Auch Änderungen bestehender Darstellungen sind einfacher realisierbar, wodurch der Sprachentwurf mit Hilfe des *VL-Eli* Systems insgesamt noch weiter erleichtert wird.

Literatur:

[1] Matthias Jung. Ein Generator zur Entwicklung visueller Sprachen. Dissertation, Universität Paderborn, November 2000.

[2] Matthias Jung, Uwe Kastens, Christian Schindler, and Carsten Schmidt. Visual Pattern in the VLEli System. In Reinhard Wilhelm, editor, Proceedings 7th International Conference on Compiler Construction CC'2001, Nummer 20027 in Lecture Notes in Computer Science, Seiten 361–364. Springer Verlag, April 2001.

[3] Isaac Nassi and Ben Shneiderman. Flowchart Techniques for Structured Programming. ACM SIGPLAN Notices, 8(8):12–26, August 1973

[4] Tcl/Tk Website.
URL: http://www.tcl.tk