Visuelle Sprachen

Martin Mittring

14.12.2000

Dieser Text wurde angefertigt für ein Referat im FWP-Fach: ,,Konzepte moderner Programmiersprachen". Das Fach belegte ich während meines Informatikstudiums an der FH München. Wegen der geforderten Kürze der Darstellung ist der Text leider nicht so exakt, wie er sein könnte.

Inhaltsverzeichnis

1 Einleitung

Sicher kennt jeder den Ausspruch:

,,Ein Bild sagt mehr als 1000 Worte!"

Stimmt die Aussage? Da sich Beispiele für und gegen die These finden lassen, ist sie scheinbar wertlos. Der Satz ist ohne Zusammenhang falsch, je nach konkretem Zusammenhang kann er jedoch richtig sein. Interessant ist der Ausspruch deshalb, weil sich fast alle Eigenschaften von Visuellen Sprachen hier bereits finden lassen.

Man kann sich z.B. fragen: *,,Kann ein Bild nicht zu viel sagen?"*

In der Tat enthält ein Bild einen großen Grad an Redundanz, der auch mißinterpretiert werden kann.

oder: *,,Kann ein Wort nicht mehr als 1000 Bilder sagen?"*

Da sowohl Bilder, als auch Worte eine Form der Informationskodierung sind, die je nach kodierter Information mehr oder weniger Komplexität erzeugen, muß man die Frage mit ja beantworten.

oder: *,,Kann ein Bild unterschiedlichen Betrachtern nicht Unterschiedliches sagen?"*

Das Gleiche kann man für Worte auch behaupten. Im Rahmen der Vorlesung ist es sinnvoll beim Thema ,,Visuelle Sprachen" nur auf die Unterbereiche Programmiersprachen und Mensch-Maschine Kommunikation einzugehen. Gerade hier soll die Information betrachterunabhänig sein. Es muß also eine exakte Interpretationsvorschrift für die Bilder oder Worte geben.

Ich beziehe mein Wissen zum Thema hauptsächlich aus zwei Büchern und einigen Artikeln in Fachzeitschriften. Werder sind dort die Informationen exakt definiert, noch sind sich die Autoren ganz einig. Ich werde versuchen das ganze objektiv zu vermitteln, trotzdem sollte man dem Ganzen kritisch gegenüberstehen. Man lange darüber streiten, wo die Grenze zwischen visuellen und nicht visuellen Sprachen liegt. Ich kann die Frage nicht beantworten, will es aber auch nicht, weil ich persönlich den Erfolg in einem hybriden System sehe.

2 Die Kommunikation

In der direkten Kommunikation unterscheidet man 3 Bereiche, die beiden Kommunikationspartner und das Kommunikationsmedium. Bei einzelnen Informationspaketen ist ein Kommunikationspartner der Sender und der andere der Empfänger.

Es gibt zwei Kommunikationsarten, die einseitige und die zweiseitige Kommunikation. Die einseitige Kommunikation findet z.B. statt, wenn wir einen Computerausdruck lesen. Hier ist der Computer der Sender, das Papier das Medium und der Mensch der Empfänger. Selten in der einseitigen Mensch Maschine Kommunikation ist der Mensch der Sender. Da die Informationen vom Menschen sehr fehlerbehaftet sind, braucht dieser eine Möglichkeit um die Fehler zu erkennen, eine zweiseitige Kommunikation ist nötig. Bei der zweiseitigen Kommunikation bei Benutzung eines Texteditors ist diese Möglichkeit gegeben. Dennoch ist hier der Mensch der klar dominierende Sender. Bei intelligenteren Reaktionen auf Benutzeraktionen spricht man dann von Interaktivität.

2.1 Der Kommunikationspartner Mensch

Auf den ersten Blick scheint der Mensch eine fixe Größe zu sein. Das Kommunikationsmedium wird ständig technisch weiterentwickelt und der Kommunikationspartner Maschine kann mit komplexer Software und Hardware jede Art von Kommunikation bieten. Dennoch findet auch beim Mensch eine Entwicklung statt. Was noch wenigen Jahren belächelt und nicht ernst genommen wurde, ist heute Standard auf quasi jedem System. Als Beispiel kann man hier z.B. Maus und graphische Benutzeroberfläche nennen.

Eine Sprache wie C/C++ hätte es ohne den etablierten Vorgänger sehr schwer gehabt. Nicht allein die Qualität der Sache entscheidet, sondern auch die Akzeptanz dieser. Unter Umständen muß beim Menschen die Akzeptanz für neue Programmierkonzepte erst gebildet werden. Gerade die Abkehr von rein textbasierten Sprachen stellt hier ein psychologisches Hindernis dar. Dieses läßt sich aber auch technisch begründen. So lassen sich keine der vielen textbasierten Tools anwenden und die Plattformunabhängigkeit des Compilers wird ein Problem. Neben dem Mißtrauen gegenüber Neuem kommt der Verlust des Vertrauten hinzu. Ein Texteditor vermittelt Vertrauen, Elemente werden nicht plötzlich umarrangiert, anders dargestellt oder versteckt. Allein das Verstecken von Information könnte dem professionellen Benutzer die Möglichkeit nehmen Inkonsistenzen im Code zu beseitigen. Das ist nicht zu unterschätzen. Inkonsistenzen können gerade bei neuen Sprachen auftreten.

2.1.1 Darstellungsdimension

1-Dimensional: Der klassische Compilerbau geht von einer eindimensionalen Sprache aus. Läßt man die optionalen Trennsymbole wie Return, Space und Tab weg, erhält man aus dem Quellcode einen eindimensionalen Zeichenstrom. Darauf beruhen die meisten Compiler- oder Interpretersprachen. Für den Entwickler ist die Darstellung mit den Trennsymbolen zweckmäßiger. Man könnte sagen die Darstellung der Sprache hat eine Dimension zwischen 1 und 2. (In der Chaostheorie spricht benutzt man den Dimensionsbegriff auch für reelle Zahlen. Man spricht hier von der fraktalen Dimension, die aber auf einem nichtdiskreten Raum definiert ist.) Die Benutzung einer eindimensionalen Darstellung wäre für sehende Menschen ein klarer Nachteil gegenüber der Darstellung mit Trennsymbolen. Wegen der technischen Einfachheit gab es aber ein populäres visuelles eindimensionales Medium, das Telex. Bei Audioübertragungen ist dagegen die Eindimensionalität nicht umgehbar. Deshalb wäre eine Programmiersprache, die auf einer Audiokommunikation zwischen Mensch und Maschine beruht eine eindimensionale Sprache.

Vorteil einer eindimensionalen Darstellung ist, daß sequentielle Abläufe wesentlich direkter modelliert werden können.

2-Dimensional: Der Mensch kann mit einem Blick einen zweidimensionalen Bereich erfassen. Forschungen haben gezeigt, wenn ein Mensch sich eine dreidimensionale Karte eines komplizierten Schachtsystems im Kopf machen soll, wird diese im Gehirn immer zweidimensional abgebildet. Die Forscher führen das auf die Dreidimensionalität des Gehirns zurück, das eine Dimension zur Kommunikation und Versorgung benötigt.

Vorteil einer zweidimensionalen Darstellung einer Programmiersprache ist, daß parallele Vorgänge oder Bedingungszweige wesentlich direkter modelliert werden können. Außerdem ist die zweidimensionale Darstellung dem menschlichen Denken näher. Ohne klar festgelegte Konventionen lassen sich sequentielle Abläufe nicht abbilden. Für die meisten Programmiersprachen ist aber gerade das sehr wichtig. Ein Pfeildarstellung kann hier eine mögliche Lösung sein. Sofern die Sprache parallele Vorgänge ermöglicht, ist eine weitere Konvention nötig, um eine Verzweigung von einem parallelen Vorgang zu unterscheiden.

3-Dimensional: Unsere Welt ist dreidimensional. Mit virtueller Realität könnten wir direkt in unserer natürlichen, uns vertrauten Welt arbeiten. Ein Objekt wäre anfaßbar, mit virtuellen Werkzeugen könnten man dieses manipulieren.

Problem dabei ist, daß man nicht in ein Objekt sehen kann, was aber evtl. für die Datenkapselung geschickt genutzt werden kann. Um einen Vorteil gegenüber der echten Welt zu haben, würden einige Naturgesetze nicht gelten, andere würde man biegen und nur unter bestimmten Bedingungen verwenden. Für den Benutzer müßte aber klar sein, welche Aktionen von ihm welche Reaktionen erzeugen.

Da der Mensch die Welt nur zweidimensional erfassen kann, ist der Eindruck beim Menschen je nach Betrachterstandort und Winkel ein anderer. Das dreidimensionale Bild baut sich erst indirekt auf. Das kann bei komplexen Zusammenhängen sehr schwierig werden.

Es empfiehlt sich also eine zweidimensionale Darstellung mit klaren Konventionen, wie diese zu lesen ist. Hier ist auch die bewährte strukturierte Programmierung einzuordnen.

2.1.2 Assoziation

Durch die konkrete Darstellung der visuellen Elemente hat man eine Möglichkeit beim Menschen bestimmte Assoziationen zu bewirken. Assoziationen mit realen Dingen machen es Einsteigern leicht sich zurechtzufinden. Assoziationen mit ähnlichen Elementen im gleichen System lassen Zusammenhänge vermuten und beschleunigen den Einarbeitungsprozeß. Die Informationen können in können in Form, Farbe und Struktur kodiert werden. Konkret kann z.B. ein rautenförmiges Symbol eine Bedingung ausdrücken, ein Rechteck eine Anweisung als Teil einer Sequenz. Man könnte nun mit Hilfe einer Farbkodierung besondere Bedingungen oder Anweisungen kodieren, um z.B. MMX oder ISSE Befehle zu ermöglichen. Der Benutzer kennt die Symbole bereits, nun jedoch erlernt er schnell die ähnlichen Zusammenhänge. Die Ausgänge könnten die gleiche Farbe tragen und somit ist auch leicht ersichtlich, daß die Befehle nur untereinander kombiniert werden können. Hier wurde die Typinformation als Farbwert kodiert. Um die Elemente mit dem restlichen System zu verbinden, bieten sich Kopplerelemente an, die an Ein-/ und Ausgängen beide Farben tragen. Naheliegend ist auch, bestimmte Formen der Kodierung dem Benutzer zu frei zu überlassen. Damit kann das erstellte Programm übersichtlicher gestaltet werden und ermöglicht anderen Benutzern somit eine leichtere Einarbeitung.

2.1.3 Verbindung zwischen Sehen und Hören

Zwischen Menschen ist die gesprochene Sprache das wichtigste Medium. Damit sich Menschen über etwas unterhalten können, müssen aussprechbare Begriffe existieren. Die meisten haben gelernt ohne lange zu überlegen vom geschriebenen Wort zum gesprochenen Wort zu kommen. Das Gleiche gilt für die Gegenrichtung. Wichtig für den Erfolg einer visuellen Sprache ist also, daß der Mensch ohne lange zu überlegen die visuellen Elemente benennen kann. Für die Kommunikation ist dabei Eindeutigkeit in den Begriffen von Vorteil. Wenn z.B. User ein Element Kasten, Box, Block, Rechteck, Fenster oder Rahmen nennen, ist die Verwirrung groß. Der Programmiersprachenentwickler hat also für eine konsistente Namensgebung zu sorgen.

In den Anfangszeiten der visuellen Programmierung hat man versucht völlig ohne Text auszukommen, um sich von menschlichen Sprachkonflikten und den entstehenden Mehrdeutigkeiten zu befreien. Davon ist man aber abgekommen. Kommentare und Namensgebung haben auch in visuellen Sprachen ihre Berechtigung.

2.2 Das Kommunikationsmedium

Das Kommunikationsmedium muß beiden Kommunikationspartnern angepaßt sein. Einerseits muß es mit dem Computer steuerbar sein, andererseits die Eigenschaften der menschlichen Kommunikatiosorgane und des menschlichen Gehirns berücksichtigen.

2.2.1 nicht interaktive Darstellung

Alle visuellen Ausdrucksmittel, die bereits in gedruckten Werken Verwendung fanden, fallen unter diesen Punkt. Warum hat der Autor dieser Werke zu dem Mittel gegriffen, wenn eine textuelle Darstellung auch möglich wäre?

2.2.2 interaktive Darstellung

Lange Zeit war technisch keine interaktive Darstellung möglich. Erst die Verbreitung von Grafikbildschirmen ermöglichte das. Liest man ältere Kommentare zu visuellen Systemen, findet man dort das Nr. 1 Gegenargument zur damaligen Zeit, die „fehlende und auch in Zukunft nicht zu erwartende Verbreitung von hochauflösenden Grafikbildschirmen". Der Mensch braucht Zeit um die Darstellung zu erfassen und er soll den Inhalt wiedererkennen. Deshalb hat eine starre Darstellung ihre Vorzüge. Nach erfassen des einfachen Grobinhaltes kann der Benutzer mit Interaktionen weitere, detailliertere Inhalte aufdecken.

2.2.3 analoge Eingabegeräte

Typische analoge Eingabegeräte sind z.B. die Maus oder ein Grafiktablet. Diese lassen sich gut als Zeigeinstrumente auf einer zweidimensionalen Oberfläche nutzen. Damit sind sie der zweidimensionalen Darstellung des Bildschirms angepaßt. Der Mensch kann diese sehr schnell und intuitiv benutzen, das ermöglicht schnelle und einfache Interaktionen.

2.2.4 digitale Eingabegeräte

Ein typisches digitales Eingabegerät ist z.B. die Tastatur. Die vier Hauptarten der Nutzung sind:

- Texteingabe
- Funktionstasten oder Hotkeys
- Modifikation anderer Eingaben (z.B. Shift oder Ctrl in Verbindung mit Maus oder Tastatur)
- Steuerung des Cursors (ähnlich Zeigerinstrument)

Alle Nutzungsarten lassen sich auch für visuelle Sprachen gewinnbringend einsetzen, auch wenn in visuellen Systemen das Zeigeinstrument mehr Gewicht bekommt.

2.3 Der Kommunikationspartner Computer

Leider haben unsere Computer auch Schwächen. Einige menschliche Fähigkeiten die für leicht sind, bedeuten für den Computer eine nahezu unlösbares Problem. Beispiele sind das Verständnis natürlicher Sprache und eine gute akustische Spracherkennung. Wir tun der Kommunikation aber auch keinen Gefallen, wenn wir in diese Richtung gehen. Mißinterpretation und Unvollständigkeit liegen bereits in den Wurzeln dieser Kommunikationsformen. Gerade bei komplexeren Inhalten zeigen sich hier Unzulänglichkeiten. Nur mit klaren Konventionen lassen sich diese Probleme beseitigen, was automatisch dem streng logischen Kommunikationsparter Computer entgegenkommt.

2.3.1 Turing-Maschine

Die Darstellungsdimension einer Sprache hat nichts mit Ihrer Mächtigkeit zu tun, da sich alle mehrdimensionalen Daten bei endlicher Genauigkeit auf eine Dimension abbilden lassen. Die Mehrdimensionalität ist nur eine andere Darstellung, im Speicher liegen die Daten ja auch linear vor. Visuelle Sprachen sind also gleichmächtig mit nichtvisuellen Sprachen.

In dem Zusammenhang kann man auf die Church'sche These eingehen. Diese besagt: ,,Die Klasse der Turing-berechenbaren Funktionen ist gleich der Klasse der intuitiv berechenbaren Funktionen". Klassische Programmiersprachen ermöglichen maximal Turing-berechenbare Funktionen abzubilden. Für visuelle Sprachen gilt das gleiche, womit die Mächtigkeit dieser gleich der intuitiv berechenbaren Funktionen ist. Das gilt für alle bisher gefundenen Berechnungskonzepte. Alle lassen sich auf einer Turing-Maschine umsetzen.

2.3.2 Parallelverarbeitung

Die meisten Programmiersprachen unterstützen leider nur ungenügend die parallelen Einheiten moderner Prozessoren. Prozessor und Compiler versuchen ihr Möglichstes um parallelisierbare Vorgänge zu erkennen und zu nutzen. Die Benutzung von Threads ist meist die einzige Möglichkeit des Programmierers hier zu helfen. Parallele Einheiten innerhalb eines Prozessors werden damit aber nicht besser genutzt. Die Kenntnis des Programmierers um parallelisierbare Vorgänge kann dieser selten im Sprachkonstrukten ausdrücken. Die eindimensionale Darstellung klassischer Programmiersprachen hat hier eine Schwäche, die eine Sprache mit höherer Darstellungsdimension nicht hat.

3 Anwendungen

Es wäre dumm das bisher etablierte Wissen um nichtvisuelle Anwendungen nicht auch bei modernen visuell orientierten Anwendungen zu verwenden. So hätte wohl eine rein auf visuellen Elementen beruhende Sprache keinen Erfolg.

3.1 Softwareentwicklung

Softwareentwicklung findet vereinfacht in drei Phasen statt: Design, Implementierung und Praxistest. Wenn nötig, kann der Zyklus wiederholt werden. Das Modell kann man auf das ganze Projekt, aber auch auf Teilkomponenten beziehen. Wichtig ist bei professioneller Softwareentwicklung, daß in jeder Phase eine dort spezifische Dokumentation nötig ist.

3.1.1 Design

Im professionellen Bereich bietet die Sprache UML eine einheitliche Notation für den Software Entwurfsprozeß. UML ist keine visuelle Programmiersprache, sondern eine visuelle Beschreibungssprache. Die vielen Diagrammtypen die es in UML gibt, sind der Problemstellung des zu beschreibenden Gebietes angepaßt. UML kann nichtinteraktiv, z.B. auf dem Papier benutzt werden, aber auch interaktiv mit Softwareunterstützung. Die interaktive Nutzung hat den Vorteil, daß Quellcode und Dokumentation parallel gewartet werden können. Bei Quellcodeänderungen lassen sich Klassendiagramme anpassen, ebenso können Diagrammänderungen Quellcodeänderungen nach sich ziehen.

3.1.2 Implementierung

Ein modernes Schlagwort im IT Bereich ist ,,komponentenbasierte Softwareentwicklung". Üblicherweise implementieren professionelle Softwareentwickler die Komponenten. Diese werden in einer Applikation benutzt. Dort ist weiterer Code zu finden, der die Komponenten ansteuert und somit miteinander vernetzt. Bei geschickter Planung des Systems ist dieser Code relativ einfach strukturiert. Damit eignet er sich relativ gut für eine visuelle Darstellung. Die komplizierten Implementierungsdetails können hier vom Benutzer

versteckt werden. Der Endanwender kann damit das System nutzen, und es bleibt nicht mehr dem professionellen Entwickler vorbehalten. Dieser kann die Komponentenbibliothek aufbauen. Der Endanwender profitiert von mehr Funktionalität, ohne zu große Komplexität.

3.1.3 Praxistest

Im Praxistest werden Bugtracking Tools, Debugger und Stress Tools eingesetzt. Es werden Dokumentationen über die auftretenden Bugs erstellt und Workaround beschrieben. Zur Fehlerfreiheit soll aber auch die Tauglichkeit der Anwendung getestet werden.

Die ganze Problematik ist derart vielfältig, daß sie sich schlecht in ein System zwingen läßt und damit eine visuelle Notation, zumindest für größere Bereiche, unmöglich macht. Der Anwender versucht sein Problem objektiv und wenn nötig auch subjektiv zu beschreiben. Oft fehlt dem Anwender das Fachwissen um das Problem objektiv zu beschreiben, oder das Problem ist subjektiver Natur. Subjekte Inhalte lassen sich nur mit natürlicher Sprache ausdrücken.

3.2 Shadetree

Im 3D Rendering Bereich gibt es den Begriff Shader. Ein Shader ist eine Approximationsfunktion für das Lichtverhalten eines Materials in der wirklichen Welt für einen Punkt in der Szene. Als Eingang kann die Funktion viele Parameter, wie z.B. Lichtquellenpositionen, Augposition, Punktposition, Normalenvektor, UV Koordinaten für 2D Texturemapping und vieles mehr erfragen. Mit einer kleinen Bibliothek von Funktionen wird schließlich ein Farbwert errechnet und zurückgeliefert. Ein Shader wird klassisch in C implementiert. 3D Renderingsoftware ohne eine shaderähnliche Sprache können nicht den im professionellen Bereich erforderlichen Funktionsumfang bieten. In moderneren Softwarepaketen lassen sich Shader in einer visuellen Sprache zusammenbauen. Man zieht sich Eingänge wie z.B. die Lichtquellenposition auf sein Arbeitblatt, Operatoren wie z.B. eine Farbmultiplikation und verbindet diese in einem Netzwerk das schließlich zu Ausgang, dem Farbwert führt. Das Ganze zeigt den Datenfluß sehr anschaulich. Operationen wie z.B. Fresnel Term lassen sich sehr leicht durch ihr Icon dem Verwendungszweck zuordnen.

3.3 Videocompositing

Im Film versteht man unter dem Begriff Compositing die Verschmelzung vieler einzelner Bildelemente zu einem Gesamtbild oder Film. Das berühmte Bild der Titanic im gleichnamigen Film bestand z.B. aus 30 Bildelementen. Die Kunst des Compositings besteht nun darin, das Feintuning dieser Bildelemente zu beherrschen, um beim Zuschauer den gewünschten Effekt zu erzeugen. Die Bildelemente können mit verschiedenen Farboperationen miteinander verknüpft werden. Das Ergebnis ist wieder ein Bildelement. Mit einigen Operationen lassen sich lokale oder globale Farbkorrekturen an den Bildelementen vornehmen. Ähnlich dem Shadetree läßt sich hier mit einfachen Elementen ein Datenfluß modellieren. Der Künstler kann Zwischenergebnisse begutachten und interaktiv Parameter variieren.

3.4 Hyperbolic Tree

Wie visualisiert man einem beliebig komplexen Graphen zweidimensional? Das ist ein nichttriviales Problem. Bei einer nichtinteraktiven Darstellung ist dafür einiges an Rechenzeit nötig. Die Darstellung wird große freie Bereiche enthalten und benachbarte Knoten können weit auseinandergerissen werden. Erlaubt man jedoch eine interaktive Darstellung läßt sich das Problem eleganter, ohne die genannten Schwierigkeiten lösen. Man kann hier einen Knoten im direkten Fokus behalten. Jenach Stärke der Nachbarschaftsbeziehungen werden weitere Knoten visualisiert.Wegen der begrenzen Darstellungsfläche werden nur sehr nahe Knoten dargestellt. Verbindungen zu nicht dargestellten Knoten werden nur angedeutet.

Diese Form der Visualisierung wäre ohne eine interaktive Darstellung nicht möglich.

3.5 GUI

Was liegt näher als eine graphische Benutzeroberfläche im WYSIWYG (What you see is what you get) zu bearbeiten. Deshalb bieten die meisten Programmiersprachen derartige Editoren.

Einige Sprachen gehen den Schritt noch etwas weiter. In Delphi werden auch nicht sichtbare Objekte wie z.B. Timer in einem Fenster plaziert. Für den Entwickler ist das deshalb besonders praktisch, weil schnell ersichtlich wird, welche Elemente sich auf das Fenster auswirken. Mit wenigen Klicks können die Parameter der Objekte verändert werden oder Quellcode für Messages erzeugt werden.

Es gibt aber auch Sprachen, in denen selbst die Logik hinter den Objekten in dieser Darstellung editierbar ist. Timer und Buttons haben dann z.B. ein Ziel, das aktiviert wird. Damit lassen sich sehr schnell komplexe GUI Systeme zusammenbauen. Das ist für die Softwareentwicklung vorteilhaft, weil der Kunde sehr schnell einen Eindruck vom entstehenden System bekommt und frühzeitig seine Änderungswünsche einbringen kann. Die interne Logik kann weiterhin in klassischem Stil programmiert werden.

3.6 Integrierte Entwicklungsumgebungen

Man erwartet heute von einer modernen Programmiersprache einige Zugaben. Dazu gehört eine integrierte Entwicklungsumgebung. Die bekannteste ist derzeit das Visual Studio von Microsoft. Bereits diese IDE besitzt einige visuelle Features, die den Programmiervorgang vom reinen Texteditor-Tippen abheben. So ist z.B. das Syntax Highlighting ist eine Form der Kodierung die dem Leser das schnelle Erfassen der syntaktische Verwendung eines Textteiles ermöglicht.

Bereits im Texteditor ist eine Blockstruktur möglich, die dem Leser schnell die hierarchische Gliederung des Quelltextes zeigt. Auch Kommentarblöcke können vom Benutzer eingesetzt werden, um z.B. Funktionsköpfe hervorzuheben.

Man könnte eine klassische IDE visuell aufwerten, indem man Objekte in den Quelltext einbetten kann. Damit wäre es möglich Dokumentation mit im Quelltext zu halten, Codeteile könnten wie in einem Falteneditor weggeklappt werden, andere Codeteile durch eine visuell bessere Darstellung ersetzt werden. Ein Klick auf ein Hyperlink Objekt befördert den Benutzer in eine andere Quellcodedatei und Zeile. Das Objekt müßte nur einige elementare Operationen beherrschen, wie Speichern, Laden, Quelltext abliefern oder Eingaben von Maus und Tastatur abarbeiten.

Mit einem ähnlichen Ziel, aber mit tiefgreifenderen Änderungen beschäftigen sich derzeit einige Microsoft Mitarbeiter. Das System wird ,,Intentional Programming" genannt. Da man sich vom konkreten Quelltext trennt, ist es möglich verschiedene Darstellungformen des Codes zu bearbeiten und zu kombinieren. Denkbar wäre sogar, daß zwei Programmierer den gleichen Code in verschiedenen Sprachen bearbeiten können.

4 Eigene Meinung

Ich habe versucht einen Teil des Themas ,,Visuelle Sprachen" herauszugreifen, um die Konzepte näherzubringen. Damit konnte ich leider viele Aspekte des Themas nur anschneiden oder sie blieben unerwähnt. Den interessieren Zuhören oder Leser empfehle ich Literatur, die das Thema auch kritisch betrachtet (siehe Kapitel Literatur).

Wir kommen meiner Meinung nach in ein spannendes Zeitalter, in dem nicht mehr die Technik die Möglichkeiten diktiert, sondern wir. Der experimentelle Umgang mit visuellen Sprachen ist meiner Meinung nach ein Indiz, daß wir uns der Methoden, um die Möglichkeiten zu erschließ en, noch nicht bewußt sind. Ich persönlich bin skeptisch gegenüber den meisten visuellen Systemen. Oft skaliert ein solches System am Anfang sehr gut, doch mit steigender Komplexität nehmen Schwierigkeiten übermäßig zu. Ich halte aber auch die klassischen textbasierenden Sprachen für sehr antiquiert. Ich denke in einer gesunden Kombination beider Systeme liegt etwas besseres, wir müssen es nur finden.

5 Literatur und WWW-Links

- Verlag: Heise Verlag, Fachzeitschrift ix, Ausgabe 11/2000, S.142 ,,Intentional Programming: Programmieren ohne Sprache"
- Verlag: Addison Wesley, Autor: Stefan Schiffer ,,Visuelle Programmierung"
- Verlag: Hanser, Autor: Jörg Poswig, ,,Visuelle Programmierung"
- http://www.inxight.com, Hyperbolic Tree
- http://www.cinegrfx.com, ShadeTree