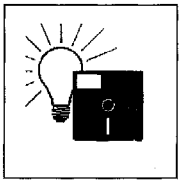
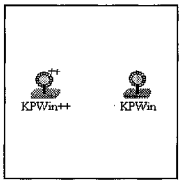




Zwei in einem: KnowledgePro

Michael Haas

Die Programmierumgebung KnowledgePro für Windows war bereits Thema einer umfangreichen Besprechung in jur-pc *(Peter Ebenhoch, Visuelle Entwicklungsumgebungen und KnowledgePro Windows; in: jur-pc 1994, 2443 und 2512; vgl. zur Diskettenbeilage mit der KnowledgePro-Demoversion jur-pc 1994, S. 2473 f.)*. Das Konzept der Programmiersprache, das durch einige Besonderheiten gerade für Problemlösungen in der Rechtsinformatik geeignet scheint, rechtfertigt es aber, der neuen Programmversion erneut Aufmerksamkeit zu schenken.

Grundlegendes

KnowledgePro liegt in zwei voneinander unabhängigen Entwicklungssystemen vor: KPWin und KPWin++. Ersteres basiert auf einem Interpreter und stellt Werkzeuge zum interaktiven Programmtest zur Verfügung. KPWin++ ist ein Precompiler, der KnowledgePro-Programme in C++ Programme umwandelt. Der erzeugte C++ Code kann dann von einem gängigen C-Compiler zu einer direkt ausführbaren Programmdatei übersetzt und gebunden werden. Diese Programme bringen im Vergleich zur Ausführung in KPWin einen erheblichen Geschwindigkeitsvorteil und sind obendrein unabhängig von einer Runtime verteilbar *(siehe hierzu Kasten 1)*. Laut Handbuch soll in einer späteren Version die Zweiteilung der Umgebung entfallen. Dann wird es möglich sein, in einer Entwicklungsumgebung Programme im Interpreterbetrieb zu testen und anschließend über den Zwischenschritt der C++ Codegenerierung in eine EXE-Datei umzuwandeln.
Da in der derzeitigen Version der Programmtest, das sog. Debugging, nur in KPWin möglich ist, stellt dies die primäre Entwicklungsumgebung dar. Erst wenn das Programm komplett entworfen und getestet ist, erstellt man mit KPWin++ eine EXE-Datei. Aus diesem Grund wird in der Besprechung auch von KPWin ausgegangen. In einem gesonderten Abschnitt werden die Besonderheiten von KPWin++ dargestellt.
Dieser Artikel beruht auf den Erfahrungen mit dem Entwurf einer kleinen Beispielsanwendung. Die Anwendung realisierte ein kleines Expertensystem, das einige besondere Möglichkeiten von KnowledgePro ausnutzt.

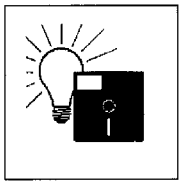
Die KP-Programmiersprache

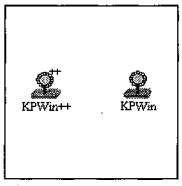
Die Programmiersprache KnowledgePro unterscheidet sich grundlegend von den einander prinzipiell ähnlichen Sprachen Basic, Pascal und C. Diese sind dadurch gekennzeichnet, daß sie streng zwischen Programm und Daten trennen. Programme werden durch Funktionen und Prozeduren repräsentiert, Daten durch Variable *(siehe Kasten 2)*. KnowledgePro ersetzt all dies durch

Compiler, Interpreter etc.

Wenn von Programmieren gesprochen wird, ist meist das Abfassen von Befehlszeilen in der Form einer bestimmten Programmiersprache gemeint. Mit diesen Befehlen kann der Computer in aller Regel so nichts anfangen, es sei denn man bedient sich der sog. Maschinensprache, das ist eine besondere Sprache, die vom verwendeten System abhängt. Damit nun der Rechner mit den Programmzeilen einer hohen Programmiersprache (eine Sprache, die nicht seine Maschinensprache ist) etwas anfangen kann, müssen sie in Maschinensprache übersetzt werden. Dazu verwendet man besondere Übersetzungsprogramme, sog. Compiler und Interpreter.
Beide tun prinzipiell dasselbe, sie übersetzen eine Programmiersprache in Maschinensprache. Der Unterschied liegt darin, daß ein Interpreter die Übersetzung vornimmt, während das Programm abgearbeitet wird. Das heißt, jedesmal wenn man das Programm startet, übersetzt der Interpreter den sog. Quellcode von neuem. Ein Compiler hingegen übersetzt das gesamte Programm auf einmal und speichert es dann in ausführbarer Form ab. Es ist offensichtlich, daß Programme, die ein Compiler erzeugt, schneller sind als solche, die interpretiert werden, da der Übersetzungsvorgang wegfällt. Ein weiterer Nachteil interpretierter Programme ist, daß sie eine sog. Runtime benötigen (bekannt sind die VisualBasic Runtimes VBRUNx00.DLL). Eine Runtime enthält alle spezifischen Funktionen, die die jeweilige Programmiersprache zur Verfügung stellt, diese werden bei compilierten Programmen in die ausführbare Datei eingebunden. Wenn man ein interpretiertes Programm verteilen will, muß man immer auch die Runtime mitverteilen. Dies kann technisch und rechtlich problematisch sein. Um den Weg zurück zu KnowledgePro zu finden, ist noch der Begriff des Precompilers einzuführen. Es handelt sich dabei um einen Compiler, der Code aus einer hohen Programmiersprache in eine andere übersetzt. Durch diesen Zwischenschritt wird es ermöglicht, Programmcode einer Sprache zu compilieren, für die kein eigener Compiler existiert.

Michael Haas ist Mitarbeiter am Institut für Rechtsinformatik der Universität des Saarlandes.





Prozeduren, Funktionen und Variable

Dies sind Begriffe, die in KnowledgePro nicht vorkommen, deren Bedeutung man aber dennoch kennen sollte. An ihnen läßt sich das verbreitete Programmierkonzept der sog. *imperativen Programmierung* festmachen. KnowledgePro vereint dieses Konzept mit Elementen der sog. *funktionalen Programmierung*.
Grundlegend für imperative Programmierung ist die Trennung von Programm und Daten. Das Programm besteht aus einer Folge von Befehlen (deshalb imperative Programmierung), die die Daten verändern. Einzelne Befehlsfolgen können zu Prozeduren und Funktionen zusammengefaßt werden. Variable sind Elemente, die zum Speichern von Daten dienen. Eine Variable kann z.B. eine Zahl oder einen Text enthalten. Weitere wichtige Merkmale dieser Sprachstruktur sind Verzweigungen (if ... then) und Schleifen (while, for, ...). Die funktionale Programmierung kommt ohne Befehle und Variablen im Sinne der imperativen Programmierung aus. Ein Programm besteht hier aus der verschachtelten Anwendung von Funktionen. Die Daten, mit denen das Programm arbeitet, liegen in den Parametern dieser Funktionen vor. Eine Zuweisung eines Wertes an eine Variable (z.B. i=1) ist nicht möglich. Um dennoch auch mit umfangreichem Datenmaterial umgehen zu können, bieten funktionale Sprachen die Möglichkeit, Listen zu verarbeiten. Verzweigungen sind bei funktionaler Programmierung dadurch zu erreichen, daß der Wert, den eine Funktion zurückgibt, von einer Bedingung abhängig gemacht wird. Die einzige Möglichkeit Schleifen zu bilden, ist die Rekursion. Die Besonderheiten des funktionalen Konzeptes machen eine eigene "Programmierphilosophie" erforderlich. Sie haben aber auch dazu geführt, daß funktionale Programmiersprachen (besonders Lisp) erste Wahl sind, wenn es um die Entwicklung von Programmen im Bereich der künstlichen Intelligenz geht.
Wie gesagt, vereint KnowledgePro Elemente von beiden Konzepten. Ein Topic kann in imperativer Programmierung als Variable, Prozedur oder Funktion verwendet werden. Es ist aber ebenso möglich, ihn funktional als rekursive Funktion oder Liste einzusetzen. Der Programmierer hat also die Wahl, mit welcher Sprachstruktur er arbeiten will, und er kann beide Konzepte in einem Programm gleichzeitig verwenden. Damit stellt KnowledgePro ein flexibles, vielseitiges und leistungsfähiges Sprachkonzept zur Verfügung. Diese Flexibilität fordert vom Programmierer aber auch ein erhöhtes Maß an Konzeptuierung. Das Programmkonzept sollte schon detailliert durchdacht sein, bevor man mit dem Codieren anfängt, da man sich sonst in der Flexibilität verliert.

Topics. Ein Topic kann eine Prozedur, eine Funktion oder auch eine Variable sein. Als Funktion oder Variable kann ein Topic einen oder mehr als 65.000 Werte haben. Das heißt, daß auch Listen verarbeitet werden können. Dazu stehen umfangreiche Funktionen wie Suchen und Sortieren zur Verfügung.

?t

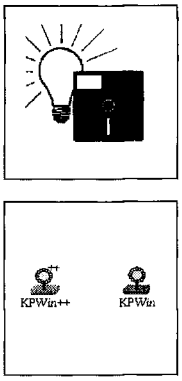
Um auf den Wert eines Topics t zuzugreifen, muß man die Funktion value_of(t) aufrufen, was man als ?t abkürzen kann. Dies führt beim Umstieg von z.B. Basic auf KnowledgePro zu erheblichen Schwierigkeiten. Die Erhöhung eines Schleifenzählers geschieht in Basic durch i=i+1, in KnowledgePro muß es i=?i+1 heißen. Man kann sich leicht denken, wie oft das Fragezeichen vergessen wird. Das Fehlen eines Fragezeichens stellt in der Programmiersprache keinen Fehler dar, so daß es vom Übersetzer nicht moniert wird. Erst wenn es zu unerwarteten Ergebnissen kommt, weiß man, daß man sich auf die Suche nach fehlenden Fragezeichen machen sollte. Zu ähnlichen Problemen kommt es in C bei der sog. Dereferenzierung von Pointern, was nicht unerheblich zu C's Ruf als ein schwieriges Expertenwerkzeug beigetragen hat. In beiden Fällen ist das Ergebnis ein nicht zu vernachlässigender Unsicherheitsfaktor bei den fertigen Programmen. Es ist daher anzuraten, beim Programmtest besonders gründlich vorzugehen, da ein fehlendes Fragezeichen an einer selten aufgerufenen Stelle unangenehme Folgen haben kann. Sieht man von diesem Problem ab, stellt das Topic-Konzept ein extrem flexibles und leistungsfähiges Konzept dar. So kann z.B. der Programmcode zur Laufzeit verändert werden, was dazu genutzt werden kann, dem Endbenutzer einer KPWin-Anwendung die eigenständige Erweiterung dieser Anwendung zu ermöglichen.

Topics hierarchisch

Besonders interessant ist die Möglichkeit, Topics hierarchisch zu gliedern. Es ist z.B. möglich einen Topic 'Rechtsgeschäft' anzulegen, der einen Untertopic 'Vertrag' enthält, der wiederum die Untertopics 'Antrag' und 'Annahme' enthält. Die Werte der Topics könnten relevante Gesetzestexte und sonstige Erläuterungen enthalten. Auf diese Weise läßt sich problemlos ein Begriffssystem aufbauen. In anderen Programmiersprachen müßte man dazu zunächst komplizierte Datenstrukturen und Algorithmen implementieren, die erhebliche Programmierkenntnisse voraussetzen.

Topics regelbasiert

Als besondere Eigenschaft von KPWin weist das Handbuch aus, daß die Topics "regelbasiert" (rule based) verwendet werden kön-



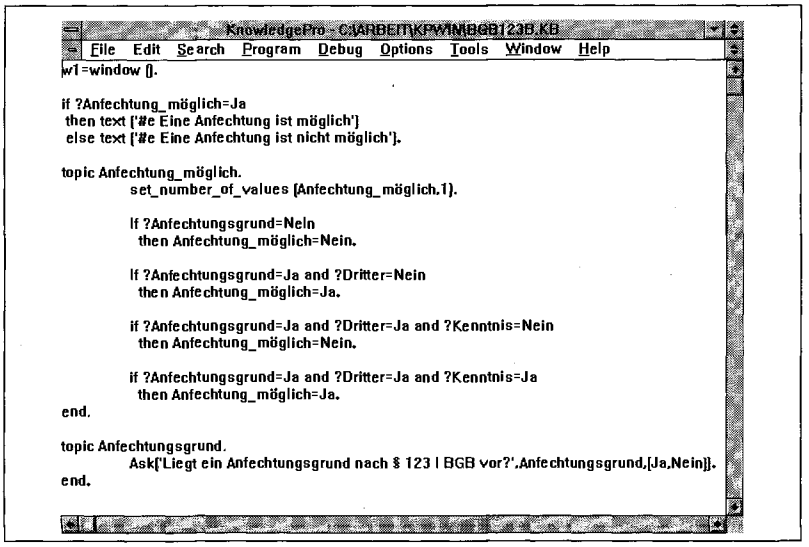


Abb. 1: Ein Knowledge-Pro-Programm

nen und es deshalb ohne viel Aufwand möglich sei, Expertensysteme zu entwickeln.

"Expertensystem"

Unter einem Expertensystem versteht man ein »Programmsystem, das "Wissen" über ein spezielles Gebiet speichert und ansammelt, aus dem Wissen Schlußfolgerungen zieht und zu konkreten Problemen des Gebiets Lösungen anbietet«[1]. Das Wissen wird dabei in Fakten und Regeln abgebildet. Regeln sind Allaussagen der Art "alle Mörder werden mit lebenslänglicher Haft bestraft". Fakten sind konkrete Aussagen wie "A ist ein Mörder". Aus dieser Regel und diesem Faktum läßt sich schließen, daß A mit lebenslanger Haft bestraft wird (werden soll). Regeln im Sinne von KPWin »sind im Grunde if ... then Anweisungen, die eine oder mehrere Anweisungen ausführen, wenn eine bestimmte Folge von Bedingungen erfüllt ist.«[2]. Dies stimmt mit der Terminologie der Expertensysteme überein, da sich die o.g. Regel wie folgt in eine if-Anweisung umwandeln läßt:
if Täter="Mörder"
then Strafe="lebenslange Haft"

Das Faktum, daß A ein Mörder ist, läßt sich abbilden als A="Mörder". Die Wissensbasis eines Expertensystems kann also in der Tat ohne Probleme in KPWin modelliert werden. Möglichkeiten, Schlußfolgerungen aus dem Wissen zu ziehen, findet man nicht, diese muß man programmieren. (Im Beispiel würde die if-Anweisung nach der Zuweisung Täter=A die entsprechende Strafe als Ergebnis liefern.) Aber KPWin will ja auch kein Expertensystem sein, sondern dessen Entwicklung erleichtern. Dazu trägt bei, daß die dritte Komponente eines Expertensystems, die Suche von konkreten Problemlösungen, gut unterstützt wird. Wenn für ein abstraktes Problem ein Lösungsschema vorhanden ist, läßt sich dies mühelos in ein Programm umwandeln, das dann im Dialog mit dem Benutzer die Lösung für einen konkreten Fall entwickelt.

Anfechtung

Das Beispiel der Anfechtung wegen Täuschung nach § 123 BGB soll dies verdeutlichen. Diese Anfechtung setzt gemäß Abs. 1 zunächst voraus, daß der Erklärende durch eine arglistige Täuschung zu der Erklärung bestimmt wurde. Absatz 2 S. 1 bestimmt, daß die Täuschung durch einen Dritten nur dann erheblich ist, wenn der Anfechtungsgegner sie kannte oder kennen mußte. Dies läßt sich in folgendem Schema darstellen:

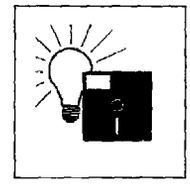
- ■ Liegt eine Täuschung vor?
  - □ nein:
    Anfechtung nicht möglich
  - □ ja:
    War der Täuschende Dritter?
    - nein:
      Anfechtung möglich
    - ja:
      Kannte der Anfechtungsgegner die Täuschung oder mußte sie kennen?
      - ja:
        Anfechtung möglich
      - nein:
        Anfechtung nicht möglich

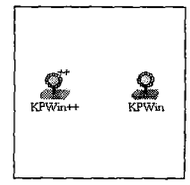
Ein solches Schema kann in einen Satz von Regeln umgewandelt werden, so daß die notwendigen Fragen gestellt werden und das Ergebnis ausgegeben wird. Damit ist ein kleines, sehr einfaches Expertensystem erstellt. Abbildung 1 zeigt einen Ausschnitt aus einem entsprechenden Knowledge-Pro-Programm. Die Regeln sind die vier mit IF beginnenden Zeilen.
Hervorzuheben ist schließlich die Möglichkeit, Hypertext und Grafiken mit Hyperregionen darzustellen. Dadurch lassen sich Programme hervorragend um eine erläuternde Komponente erweitern.[3] Zum Beispiel könnte das gezeigte Mini-Expertensystem um den Gesetzestext und Beispiele für arglistige Täuschung erweitert werden.
Diese Möglichkeiten machte sich das Beispielprogramm zunutze. Es erlaubt, Schemata wie das obi-

1 Duden »Informatik« (2. Aufl., Mannheim, 1993), unter "Expertensystem".

2 Knowledge Pro Windows, User Manual, S. 135 (Übersetzung des Autors).

3 dazu ausführlich Ebenhoch, a. a. O. S. 2450.





ge einzugeben, wobei jedem Punkt eine Erläuterung zugeordnet werden kann. Anschließend kann man das Schema regelbasiert abarbeiten. Dadurch wird der Aufwand vermieden, der nötig ist, ein Schema in einen Regelsatz umzuwandeln. Außerdem ist es so möglich, die Regeln ohne Programmierkenntnisse zu erstellen.

Die Entwicklungsumgebung

Als Stand der Windows-Programmiertechnik gilt heute der visuelle Entwurf der Benutzerschnittstelle. Das bedeutet, daß die Elemente, über die der Benutzer mit dem Programm kommuniziert (Buttons, Eingabefelder etc.), auf einem Zeichenbrett quasi gemalt werden. Man muß also nicht die Befehle kennen, die ein solches Element auf den Bildschirm bringen. Es gibt zwei verschiedene Wege, die zu diesem Ziel führen. Vor allem VisualBasic von Microsoft ist ein Beispiel für eine komplett visuelle Umgebung, bei der der gesamte Programmentwurf am Zeichenbrett geschieht. Hier wird der Programmcode am Zeichenbrett in einem speziellen Codefenster eingegeben. Der zweite Weg geht von einer traditionellen Programmentwicklung aus, bei der in einem Texteditor Programmcode eingegeben wird. Zusatzprogramme erlauben es, die Benutzeroberfläche des Programms am Zeichenbrett zu entwerfen und erstellen aus der Zeichnung Programmcode, der in den Programmtext eingefügt wird.

Letzteres ist die Vorgehensweise von KPWin und KPWin++. Dem Paket liegen einige Tools zur visuellen Entwicklung bei, die alle mit KPWin++ erzeugt wurden. Sogar die Quelltexte werden mitgeliefert. Das wichtigste Werkzeug ist der Screen Designer, mit dem die Objekte auf der Benutzeroberfläche plaziert und ihre Eigenschaften bestimmt werden können. Durch Doppelklicken auf ein Objekt wird ein Fenster geöffnet, in dem die Eigenschaften, engl. Properties, eingestellt werden könnten *(siehe Abb. 2)*. Dieser Vorgang gestaltet sich mit dem Designer recht umständlich, da man sich durch mehrere Dialogfenster hangeln muß, um alle nötigen Einstellungen vornehmen zu können. Besonders störend ist, daß es nicht möglich ist, den Namen eines Objekts zu ändern, unter dem man es im Programm erreichen kann, ohne ihm einen Ereignistopic zuzuordnen. Dies ist eine Funktion, die ausgeführt werden soll, wenn das Objekt aktiviert wird. Insbesondere bei Texteingabefeldern wird ein Ereignistopic nur selten benötigt.

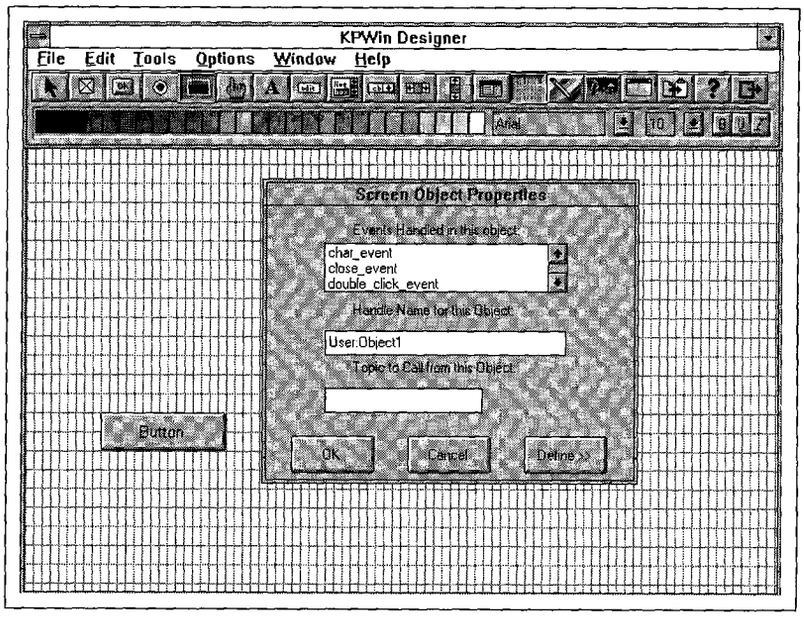


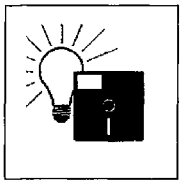
Abb. 2: Arbeiten mit dem 'Screen Designer'

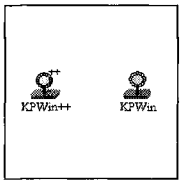
Man kann sich nur dadurch helfen, daß man als Ereignistopic einen "Dummytopic" angibt, der keine Funktion ausführt.

Ein entscheidender Gesichtspunkt bei Systemen mit Trennung von Programmierumgebung und Screen Designer ist es, ob die Integration von beiden gelungen ist. Der Weg aus dem Designer in die Programmierumgebung und zurück muß möglichst einfach sein, und alle Eigenschaften der Objekte müssen in der visuellen Umgebung erreichbar sein. Ist dies nicht der Fall werden die Vorteile der visuellen Entwicklung zunichte gemacht. Abgesehen von wenigen Einschränkungen bietet der Screen Designer alle Möglichkeiten der Programmierumgebung. Leider kann man in dem Dialogfenster, in dem eingestellt wird, auf welche Ereignisse ein Objekt reagieren soll, auch Ereignisse einstellen, die das aktuelle Objekt gar nicht verarbeiten kann. Man muß also im Handbuch nachsehen, welches Objekt auf welche Ereignisse reagieren kann. Genau dieses Nachschlagen soll die visuelle Programmentwicklung eigentlich unnötig machen. Die Übernahme der Benutzerschnittstelle aus dem Designer über die Zwischenablage nach KPWin ist gelungen. Dieser Weg ist jedoch eine strikte Einbahnstraße. Es gibt keine Möglichkeit, den Entwurf einer Schnittstelle in den Designer zu laden. Das heißt, daß man die Schnittstelle entweder in einem Durchgang entwickeln oder spätere Änderungen im Code in KPWin ausführen muß – eine unnötige und unsinnige Einschränkung.

Debugging

Selbst bei genauester Planung eines Programms werden sich ab einer gewissen Komplexität Fehler nicht vermeiden lassen. Dabei ist zwischen Fehlern in der Syntax und in der Semantik zu unterscheiden. Syntaxfehler sind





Abweichungen des Programmcodes von der Form, die die jeweilige Programmiersprache vorschreibt; sie werden vom Übersetzer der Programmiersprache gefunden und angezeigt. Semantikfehler sind Abweichungen der tatsächlichen Funktion des Programmcodes von der vom Programmierer gewollten. Solche Fehler kann das System natürlich nicht entdecken, sie müssen vom Programmierer selbst ausfindig gemacht werden. Diesen Vorgang nennt man Debugging, was im Englischen "Entwanzen" heißt und seit der Zeit, als Wanzen die Relais von Computern verklemmten, als Synonym für Fehlerverfolgung und Fehlerbeseitigung benutzt wird.

Dem Debugging kommt bei KPWin eine besondere Bedeutung zu, da der Zugriff auf Topicwerte eine große Fehlerquelle darstellt (die Sache mit dem Fragezeichen). Da es sich dabei nicht um Syntaxfehler handelt, die vom Übersetzer gefunden würden, wünscht man sich die Möglichkeit, den Ablauf des Programms möglichst genau verfolgen zu können. Mit seinem Trace-Modus ist KPWin leider auf der Stufe einfachster Basic-Systeme stehen geblieben. In einem separaten Fenster wird die aktuelle Stelle im Programmablauf ausgegeben, aber nicht etwa die Zeilennummer im Quelltext, sondern eine vorübersetzte Form des Quelltextes. Anhand dieser Ausgabe ist es kaum möglich, die entsprechende Zeile im Quelltext zu finden. Standard ist heute, daß die gerade ausgeführte Quelltextzeile optisch hervorgehoben wird.

Auch die wenigen weiteren Debug-Möglichkeiten bieten kein besseres Bild. Man ist deshalb darauf angewiesen, durch Einfügen von Ausgabebefehlen usw. der Sache von Hand nachzuhelfen.

Zuverlässigkeit

Ein entscheidender Gesichtspunkt bei der Bewertung der Effizienz eines Programmiersystems ist seine Zuverlässigkeit. Beim Programmieren kommt es immer wieder zu Fehlern, die den Zustand der Programmierumgebung und des Betriebssystems stören. Wenn dann Programmiersprache oder System jedesmal abstürzen, ist ein effizientes Arbeiten nicht möglich, da nach jedem Absturz Minuten vergehen, bis der ursprüngliche Zustand wiederhergestellt ist. So weit es eben geht, muß die Programmierumgebung deshalb solche Fehler abfangen, damit es nicht zum Absturz kommt. Ein häufiger Fehler bei Programmieren mit Rekursionen ist es, daß man keine sichere Abbruchbedingung definiert hat. Das Programm befindet sich dann in einer Endlosschleife. Da bei jedem Schleifendurchlauf Speicher belegt wird, ist dieser irgendwann voll. Sichere Programmiersysteme brechen dann das Programm ab und zeigen die Stelle, an der abgebrochen wurde. KPWin meldet, daß der Speicher voll ist und beendet das Programm und sich selbst gleich mit. Sollte man sein Programm vorher nicht gespeichert haben, ist es verloren.

Insgesamt macht KPWin leider nicht den Eindruck, daß es sich um ein in allen denkbaren Umgebungen intensiv getestetes System handelt. Es kommt häufiger zu Situationen, in denen nur ein Neustart des Programms oder gar des Systems hilft. Einen nicht unbeträchtlichen Anteil daran dürfte auch Windows haben, das solche Probleme noch steigert. Aber daß man mit diesen Problemen besser fertig werden kann, beweisen andere Programmierumgebungen. Hier bleibt noch viel Raum für Verbesserungen.

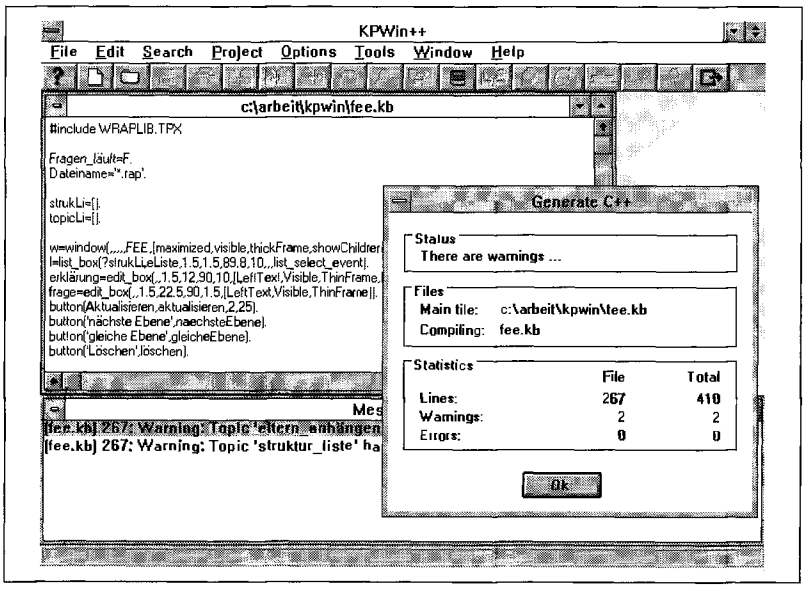
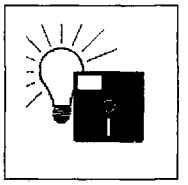
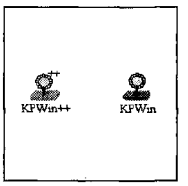


Abb. 3: Ein Blick in die "Kompilierwerkstatt"

KPWin++

Die Funktion von KPWin++ im Zusammenspiel von KnowledgePro und C++ wurde bereits erläutert. Im Regelfall wird man nach Abschluß der Programmentwicklung KPWin++ einmal auf den Quelltext anwenden, um die EXE-Datei zu erzeugen *(vgl. Abb. 3)*. Damit man dann keine unliebsamen Überraschungen erlebt, ist es ratsam, sich klarzumachen, daß KPWin++ systembedingte Einschränkungen gegenüber KPWin enthält. Dies liegt daran, daß das compilierte Programm zur Laufzeit nicht die Möglichkeit hat, auf den Übersetzer zurückzugreifen, dem interpretierten Programm ist dies möglich. Unter KPWin++ stehen deshalb alle KPWin-Funktionen, die in direkter oder indirekter Verbindung mit dem Interpreter stehen, nicht zur Verfügung. Dies sind insbesondere die erwähnten Funktionen, die es ermöglichen, das Programm zur Laufzeit zu verändern. Bevor man zweifelhafte Funktionen verwendet, sollte man ins KPWin++ Handbuch schauen!





Beim Übersetzen des Beispielprogramms meldete der C++ Compiler (nicht KPWin++) einen Fehler. Ein Blick an die Fehlerstelle des von KPWin++ erzeugten C++ Codes zeigt eine überaus sinnvolle Eigenschaft dieses Programms: Es schreibt den original KPWin-Code als Kommentar an die entsprechenden Stellen in C++. Dadurch ist es ohne weiteres möglich, die Stelle im KnowledgePro-Programm zu finden, die dem C-Compiler Probleme bereitet. So war es durch eine kleine Änderung im KnowledgePro-Code möglich, das Compilieren des Beispiels zu ermöglichen.
[Für KnowledgePro-Kenner: Der Fehler trat bei (?vater):(?kind)=". Es sollte also ein neuer Topic angelegt werden, der durch zwei Variable spezifiziert wurde. Für KPWin++ darf man einen Topic nur durch eine Variable ansprechen, deshalb muß man schreiben:
neu=(?vater):(?kind). (?neu)=".]
Ansonsten kann man von der Arbeit mit KPWin++ wenig berichten: Es war ohne Einschränkungen möglich, auch die Quelltexte des ScreenDesigners zu übersetzen. Wenn man die genannten Funktionseinbußen beachtet, wird man mit KPWin++ keinerlei Probleme haben. Die Geschwindigkeit bei der Programmausführung steigt im Vergleich mit KPWin um den Faktor 2 bis 20, abhängig von der Funktion des Programms.[4]

Fazit

KnowledgePro ist eine sehr leistungsfähige Programmiersprache, deren Einsatz in der Rechtsinformatik viele Möglichkeiten eröffnet, die mit anderen Sprachen nur mit erheblich mehr Aufwand zu erreichen wären. Für den Einsteiger ist sie leicht zu erlernen, und auch der Umsteiger von Basic oder Pascal wird sich zurecht finden.
Die konkrete Umsetzung der Sprache in Form von KPWin läßt noch viele Wünsche offen, was effizientes Arbeiten angeht.
Die Möglichkeit, mittels KPWin++ in Verbindung mit einem C-System ausführbare Dateien zu erzeugen, ist vor allem für den professionellen Entwickler interessant, der dadurch seine Programme ohne Zugabe einer Runtime vertreiben kann.
Es bleibt zu hoffen, daß in der nächsten Version KPWin++ zu einer vollwertigen Entwicklungsumgebung ausgebaut wird und dabei gleich einige der genannten Schwachstellen beseitigt werden.

[4] Ebenhoch, a. a. O. S. 2516.