EIN HYPERTEXT-EDITOR ZUR SOFTWARE-WARTUNG

Johannes Sametinger, Alois Stritzinger

Johannes-Kepler-Universität Linz
Institut für Wirtschaftsinformatik
A-4040 Linz

ZUSAMMENFASSUNG

Dieser Artikel beschreibt einen Hypertext-Editor, der es erleichtert, Softwaresysteme zu warten. Mit Hilfe dieses Editors ist es möglich, die Beziehungen zwischen Klassen und Methoden, bzw. Moduln und Prozeduren, sowie die Beziehungen zwischen Dateien und sogar den Bezeichnern eines Softwaresystems auf einfache Art und Weise zu identifizieren und zum Navigieren durch das System zu verwenden.

Die Anwendung dieses Hypertext-Werkzeuges beschränkt sich nicht auf die Wartung von Softwaresystemen, sondern kann selbstverständlich auch bei der Entwicklung gewinnbringend eingesetzt werden. Bei der Konzeption des Editors wurde aber besonderes Augenmerk auf die Tätigkeiten der Wartung gelegt.

EINLEITUNG

Die Wartung nimmt den größten Teil des Software-Lebenszyklus ein. Nicht selten wird mehr als die Hälfte der Zeit für diese Tätigkeit aufgewendet [6], [10]. Es ist daher naheliegend, daß einer Verminderung des Wartungsaufwandes große Bedeutung zukommt.

Das Schwierigste beim Ändern eines Softwaresystems ist das Nachvollziehen der Gedanken der ursprünglichen Entwickler (siehe [8], [9]). Diese Analysetätigkeit nimmt seinerseits wiederum mehr als die Hälfte der Wartungstätikgeit ein. Gute Dokumentation kann hier eine entscheidende Verbesserung bringen. Meistens ist die Dokumentation jedoch weder vollständig noch am aktuellen Stand. In diesem Fall sind Werkzeuge von großem Vorteil, die das Verstehen von Programmen auf Quellcode-Ebene unterstützen (z.B. [2], [3], [11]) und den Dokumentationsprozeß automatisieren oder zumindest die Konsistenzhaltung von Dokumenten unterstützen ([5], [7]).

Wir beschreiben einen Hypertext-Editor, der das Verstehen von Programmen erleichtert, indem er dem Benutzer hilft, die Übersicht in einem großen System zu bewahren, Beziehungen zwischen Systemkomponenten zu identifizieren und bequem an für das Verstehen des Systems benötigte Informationen heranzukommen.

DIE NETZWERK-STRUKTUR VON SOFTWARE-SYSTEMEN

Hypertext ist eine Art von Informations-Management, mit dem in einem Netzwerk von Knoten (Informationsblöcke) navigiert werden kann ([1], [4]). Die Knoten sind aufgrund bestimmter Beziehungen miteinander verbunden. Um Hypertext für das Navigieren in einem Softwaresystem zu verwenden, müssen solche Informationsblöcke, sowie deren Verbindungen untereinander definiert werden.

Im folgenden beschreiben wir kurz einige typische Charakteristika für den Aufbau von Softwaresystemen. Wir unterscheiden dabei objekt-orientierte Systeme, die in C++, und modul-orientierte Systeme, die in Modula-2 implementiert sind. Die Implementierungssprache beeinflußt einerseits den Aufbau (die Struktur) eines Softwaresystems und andererseits (durch ihre syntaktische und semantische Eigenschaften) die Art und Weise, wie Beziehungen zwischen den einzelnen Systemkomponenten beschrieben werden.

Objekt-orientierte Softwaresysteme

Ein objekt-orientiertes in C++ geschriebenes Programm besteht aus einer Reihe von Dateien, die Klassen-Definitionen, Methoden-Beschreibungen und globale Deklarationen enthalten ([14]). Globale Deklarationen können in mehreren Klassen und Definitionen verwendet werden. Was den Inhalt einer Datei betrifft, so gibt es keinerlei Einschränkungen. Eine Datei kann mehrere Klassen-Definitionen, Methoden-Beschreibungen und auch globale Deklarationen enthalten (z.B. verschiedene Dialogkomponenten für eine graphische Benutzerschnittstelle). Die Beschreibung einer Klasse kann auch auf mehrere Dateien verteilt sein (beispielsweise um die Methoden für die Ein-/Ausgabe mehrerer Klassen zusammenzufassen). Um in einer Datei den Inhalt einer anderen Datei verwenden zu können wird diese *inkludiert* (dies geschieht mit einer speziellen Preprozessor-Anweisung).

Es bestehen somit eine Reihe von Beziehungen zwischen den Dateien, Klassen und Methoden in einem in C++ geschriebenen Softwaresystem:

- Eine Klasse wird in einer Datei beschrieben.
- Eine Klasse erbt von einer anderen Klasse.
- Eine Methode wird ebenfalls in einer bestimmten Datei beschrieben.
- Eine Methode gehört zu einer bestimmten Klasse.
- Eine Methode wird in einer Unterklasse überschrieben.
- Eine Datei wird von anderen Dateien *inkludiert*.

Es existieren noch weitere Beziehungen auf der Ebene der Bezeichner:

- Ein Bezeichner wird in einer Klasse oder Methode definiert, oder er steht in einer Datei und ist global verfügbar.
- Die Verwendung eines Bezeichners steht in Verbindung mit der Definition dieses Bezeichners und mit allen anderen Verwendungen desselben Bezeichners.
- Ein Kommentar kann Erläuterungen zu einem Bezeichner enthalten, z.B. die Beschreibung einer Klasse, einer Methode, oder einer Instanzvariablen (das sind Variablen einer Klasse).

Modul-orientierte Softwaresysteme

Ähnlich ist der Aufbau von modulorientierten Systemen. Für Modula-2 können beispielsweise folgende Beziehungen zwischen Moduln, Prozeduren (bzw. Funktionen) und Dateien definiert werden ([17]):

- Ein Modul besteht aus einem Definitions- und einem Implementierungsteil.
- Ein Modul importiert andere Modul(definitione)n.
- Eine Prozedur gehört zu einem bestimmten Modul.
- Sowohl Definition als auch Implementierung eines Moduls stehen in einer eigenen Datei.
- Eine Modulimplementierung gehört zu einer bestimmten Moduldefinition.

Für Bezeichner gilt gleiches wie für objekt-orientierte Systeme, d.h. auch sie stehen untereinander in Verbindung und können in Kommentaren erläutert werden.

Wir verwenden diese Beziehungen, um mit dem Hypertext-Editor durch ein Software-System zu navigieren und nützliche Informationen für das Verstehen und Warten eines Systems bereitzustellen.

DER HYPERTEXT-EDITOR

Der Hypertext-Editor zerlegt den Quellcode automatisch beim Einlesen eines Softwaresystems (das sind einfach eine Reihe von C++- oder Modula-2-Dateien) in oben beschriebene Komponenten (Klassen und Methoden, bzw. Moduln und Prozeduren). Außerdem werden — durch syntaktische Analyse — die Beziehungen zwischen diesen Komponenten und den Bezeichnern in diesem System hergestellt. Mit Hilfe dieser Informationen wird dem Benutzer sowohl ein sehr einfaches Navigieren durch das System ermöglicht, als auch eine Reihe von sehr nützlichen Informationen zum Verstehen des Systems bereitgestellt (siehe auch [12]).

Benutzerschnittstelle

Die Benutzerschnittstelle basiert auf den Konzepten moderner graphischer Dialogschnittstellen (siehe [13], [15], [16]).

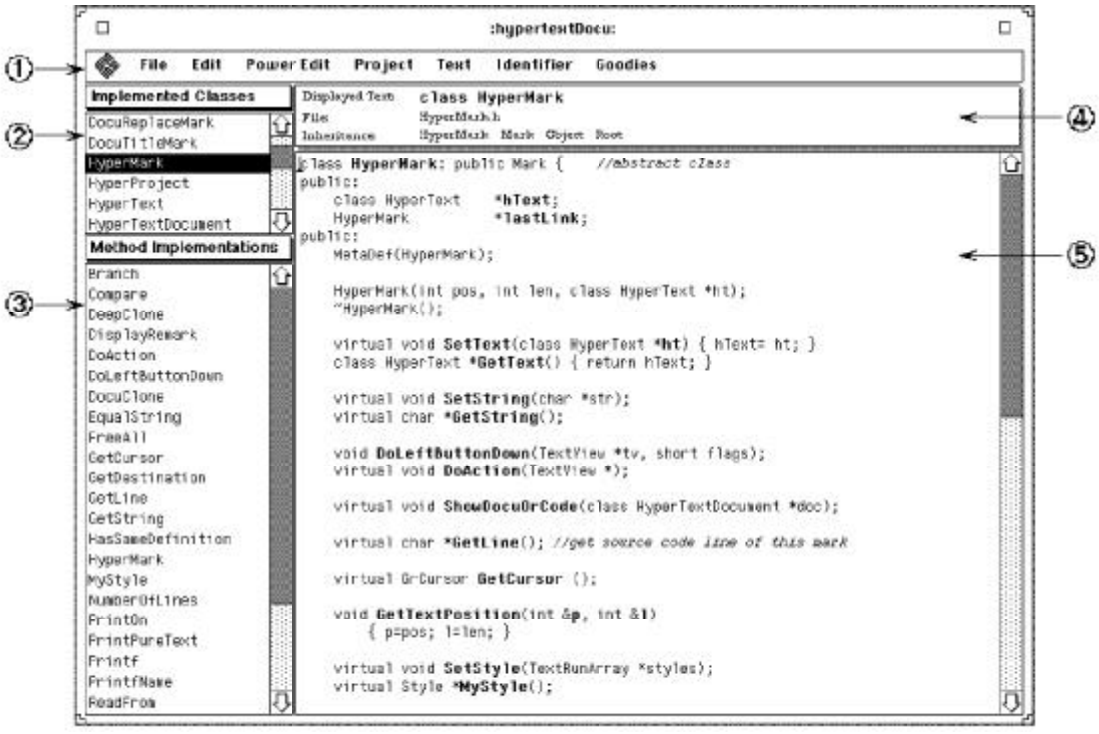


Abb. 1: Benutzerschnittstelle

Sie besteht aus einer Menüzeile (siehe ¿ in Abb. 1), zwei Listen (¡ und ¬), einer Informationsbox (v), sowie einem Texteditor (*ƒ*). Mit Hilfe der Menüs können verschiedenste Kommandos ausgeführt werden, die obere Liste zeigt Dateien, Klassen oder Moduln des Systems an, und die zweite Liste zeigt jeweils die Methoden, Prozeduren, Unterklassen oder *inkludierte*, bzw. importierte Dateien des in der oberen Liste ausgewählten Elementes. In der Informationsbox wird angezeigt, welcher Text soeben im Texteditor bearbeitet werden kann, aus welcher Datei dieser Text stammt, und (bei Klassen und Methoden) welche Oberklassen (Vererbungshierarchie) es zu einer Klasse gibt.

Browsing-Möglichkeiten

Es gibt verschieden Möglichkeiten, durch ein System zu *browsen*. Die einfachste Art und Weise ist, einfach ein Element aus den beiden linken Listen auszuwählen. In diesem Fall erscheint dann im Editor die entsprechende Klasse, Methode, Prozedur oder Datei.

Von großem Vorteil ist die Möglichkeit, einer der vorhin beschriebenen Beziehungen zu folgen und somit sehr einfach wie folgt zu verzweigen:

- von einer Klasse zu deren Oberklasse oder einer der Unterklassen,
- von einer Methode zu derselben Methode in einer der Oberklassen
- von einer Klasse oder Methode zu jener Datei, in der die Klasse oder Methode enthalten ist,
- von einer Datei zu einer der *inkludierten* Dateien,
- von einer Prozedur zu jenem Modul, in dem diese Prozedur enthalten ist,
- …

Es ist überdies möglich einen Bezeichner im Text auszuwählen und beispielsweise

- zur Definition dieses Bezeichners (unabhängig davon wo man sich gerade befindet)
- zur vorhergehenden oder nächsten Verwendung desselben Bezeichners

zu verzweigen.

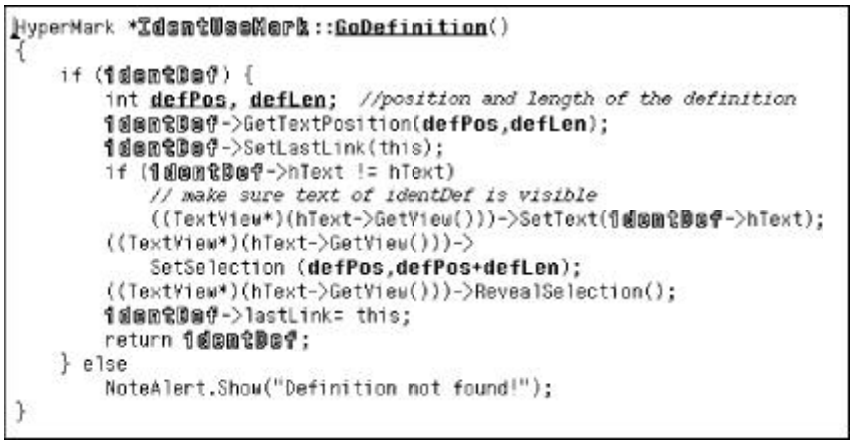
Informationen über Bezeichner
Über Bezeichner können jederzeit wichtige Informationen angezeigt werden. Dazu gehören:

- die Stelle, an der dieser Bezeichner deklariert wurde (welche(r) Klasse, Methode, Modul, Prozedur, Datei),
- die Zeile, in der die Deklaration enthalten ist,
- eine kurze Beschreibung des Bezeichners (falls vorhanden).

Beispiel eines Wartungs-Szenarios
Anhand dieses Beispiels wollen wir zeigen, wie mit Hilfe des vorgestellten Hypertext-Editors das Verstehen von fremden Code erleichtert wird. Nehmen wir an, wir wollen wissen, wie eine bestimmte (in C++ geschriebene) Methode genau funktioniert.

Zu diesem Zweck wählen wir zunächst die Klasse der betreffenden Methode in der oberen Liste aus (siehe ¡ in Abb. 1). In der zweiten, unteren Liste werden dann sämtliche Methoden dieser Klasse angezeigt, und wir können unsere Auswahl treffen. Im Editor erscheint der Quellcode und in der Informationsbox sehen wir die Bezeichnung der Methode, die Datei, die sie enthält, sowie die Vererbungshierarchie ihrer Klasse. Wenn man die verwendete Klassenbibliothek kennt, dann kann diese Information alleine schon einiges über die Eigenschaften einer Klasse aussagen.

Wir können nun beispielsweise alle lokalen Bezeichner (mit einer besonderen Schrift) hervorheben, das sind jene Bezeichner, in dieser Methode definiert werden (Parameter eingeschlossen). Weiters interessiert uns vielleicht, welche Instanzvariablen der Klasse verwendet werden. Zu diesem Zweck verzweigen wir kurz zur Klasse (durch einfaches Anklicken in der oberen Liste), bewirken dort ein Hervorheben aller lokalen Größen (zur besseren Unterscheidung diesmal mit einer anderen Schrift) und kehren wieder zu unserer Methode zurück und können nun auf einen Blick sowohl lokale Größen (in Abb. 2 *outlined* dargestellt) als auch Instanzvariablen der Klasse (in Abb. 2 fett dargestellt, Bezeichnerdefinitionen zusätzlich unterstrichen) erkennen.



```
HyperMark *IdentUseMark::GoDefinition()
{
    if (identDef) {
        int defPos, defLen;  //position and length of the definition
        identDef->GetTextPosition(defPos,defLen);
        identDef->SetLastLink(this);
        if (identDef->hText != hText)
            // make sure text of identDef is visible
            ((TextView*)(hText->GetView()))->SetText(identDef->hText);
        ((TextView*)(hText->GetView()))->
            SetSelection (defPos,defPos+defLen);
        ((TextView*)(hText->GetView()))->RevealSelection();
        identDef->lastLink= this;
        return identDef;
    } else
        NoteAlert.Show("Definition not found!");
}
```

Abb. 2: Informationen über eine Methode

Alle Namen, die in der Methode noch in Normalschrift angezeigt werden sind entweder Schlüsselwörter (auch die könnten wir in einer beliebigen Schrift anzeigen lassen), oder sind in einer der anderen Oberklassen oder sonstwo global in einer Datei definiert. Es kann nun wichtig sein, einfach und schnell Informationen über diese Bezeichner zu kriegen. Wir können dazu entweder direkt zu den Definitionen der Bezeichner verzweigen, oder wichtige Informationen einblenden. Dies ist beim Verstehen von Code von besonderer Wichtigkeit. Wir erhalten so sehr einfach Name, Deklaration, Datei und Vererbung eines beliebigen Bezeichners. Außerdem wird — sofern vorhanden — eine kurze Beschreibung dieses Bezeichners eingeblendet.

ZUSAMMENFASSUNG

Es wurde ein Hypertext-Editor zur Wartung von Software-Systemen vorgestellt. Die Implementierung erfolgte auf SUN Workstations in C++ [14] unter Verwendung des *Application Frameworks* ET++ ([15], [16]). Der Editor wird seit Mitte dieses Jahres sowohl an unserem Institiut bei Forschungsprojekten und für die Lehre als auch bei Partnern aus der Industrie eingesetzt.

LITERATUR

[1] Bigelow J.: Hypertext and CASE, IEEE Software, pp. 23-27, March 1988.

[2] Cleveland L.: An Environment for Understanding Programs, Proc. of the 21st Annual Hawaii Int. Conf. on System Sciences, Vol. 2, 1988.

[3] Cleveland L.: A User Interface for an Environment to Support Program Understanding, Proceedings of the Conference on Software Maintenance, pp. 86-91, 1988.

[4] Conklin J.: Hypertext: An Introduction and Survey, Computer Vol. 20, No. 9, pp 17-41, Sept.87.

[5] Fletton N. T., Munro M.: Redocumenting Software Systems Using Hypertext Technology, Proceedings of the Conference on Software Maintenance, pp. 54-59, 1988.

[6] Gibson V. R., Senn J. A.: System Structure and Software Maintenance Performance, CACM, Vol. 32, No. 3, pp. 347-358, 1989.

[7] Landis L. D., et al.: Documentation in a Software Maintenance Environment, Proceedings of the Conference on Software Maintenance, pp. 66-73, 1988.

[8] Letovsky S., Soloway E.: Delocalized Plans and Program Comprehension, IEEE Software, pp. 41-49, May 1986.

[9] Parikh G., Zvegintzov N.: Tutorial on Software Maintenance, IEEE Computer Society, pp. 61-62, 1983.

[10] Parikh G.: Techniques of Program and System Maintenance, Second Edition, QED Information Sciences, Inc. 1988.

[11] Rajlich V., et al.: VIFOR: A Tool for Software Maintenance, Software—Practice and Experience, Vol. 20, No. 1, pp. 67-77, January 1990.

[12] Sametinger J.: A Tool for the Maintenance of C++ Programs, Proceedings of the Conference on Software Maintenance, 1990.

[13] Shneiderman B., et al.: Display Strategies for Program Browsing: Concepts and Experiment, IEEE Software, pp. 7-15, May 1986.

[14] Stroustrup B.: The C++ Programming Language, Addison-Wesley, 1886.

[15] Weinand A., Gamma E., Marty R.: ET++ — An Object Oriented Application Framework in C++, OOPSLA ‘88, SIGPLAN Notices, Vol. 23, No. 11, pp. 46-57, 1988.

[16] Weinand A., Gamma E., Marty R.: Design and Implementation of ET++, a Seamless Object-Oriented Application Framework, Structured Programming, Vol. 10, No.2, Springer International 1989.

[17] Wirth N.: Programming in Modula-2, 3rd corrected edition, Springer-Verlag, New York, NY, 1985.