Eine graphische Oberfläche für numerische Programme

U. Nowak, U. Pöhle, R. Roitzsch

Konrad–Zuse–Zentrum für Informationstechnik Berlin (ZIB), Heilbronner Straße 10, 10711 Berlin

1 Einleitung

Bei der Weiterentwicklung von numerischen Verfahren im Scientific Computing werden häufig bereits existierende numerische Basis–Module modifiziert, sei es durch Verallgemeinerung oder Spezialisierung. Ein derartiges Vorgehen setzt gute Kenntnisse über vorhandene (und zugängliche) Basis–Module und deren Arbeitsweise voraus.

Für diese Aufgabe stellen graphische Benutzeroberflächen das ideale Hilfsmittel dar, da hier keine störenden technischen Hürden den Einstieg in ein neues Stück numerischer Software erschweren oder gar scheitern lassen.

Das Konrad–Zuse–Zentrum entwickelt und unterhält eine Sammlung numerischer Programm–Pakete (CodeLib) zur Lösung von nichtlinearen Gleichungssystemen, Systemen von gewöhnlichen Differentialgleichungen (steif und nicht–steif), differentiell–algebraischen Systemen und partiellen Differentialgleichungen (vom parabolischen oder elliptischen Typ).

Die Programme der CodeLib sind im Verzeichnis `/pub/elib/codelib` *des anonymen ftp–servers* `elib.zib-berlin.de` *des ZIB zu finden oder über die WWW-Seite* `http://elib.zib-berlin.de:88/`.

Um eine Anpassung an spezielle Problemstrukturen zu erlauben oder besonderen Anforderungen an die numerische Lösung nachzukommen, bieten die Verfahren z.T. eine Vielzahl von Optionen an. Eine derartige Optionsvielfalt führt bei klassischen Benutzerschnittstellen (z.B. Unterprogrammaufruf, Parameterdatei) häufig zu einer unhandlichen und fehleranfälligen Bedienung.

Durch die Entwicklung einer graphischen Benutzeroberfläche (GUI) sollen daher die folgenden Ziele erreicht werden:

- *einfaches Ausprobieren anhand vordefinierter Testprobleme,*
- *Kennenlernen numerischer Steuergrößen und Verfahrensvarianten,*
- *einfache Eingabe neuer Probleme,*
- *einfache Nutzung graphischer Ausgabemöglichkeiten und*

- *einheitliche Darstellung gleicher oder ähnlicher Optionen.*

Auch wenn mit den hier beschriebenen GUIs durchaus interessante Problemklassen des Scientific Computing einer unmittelbaren Lösung zugänglich sind, sind diese GUIs nicht als allgemeine Problemlösungsumgebung zu verstehen.

2 Beschreibung der graphischen Oberfläche

Zur Zeit werden mit der graphischen Oberfläche die folgenden numerischen Anwendungen bedient:

Kaskade 3.0: *ein Werkzeugkasten zur Lösung partieller Differentialgleichungen mit Finiten Element Methoden (in 1, 2 oder 3 Raumdimensionen) ([2], [5]).*

NLEQ1: *ein Verfahren zur Lösung von Systemen nichtlinearer Gleichungen mittels eines gedämpften affin–invarianten Newton–Verfahrens ([10], [3]).*

EULSIM: *ein Verfahren zur Lösung von Systemen steifer gewöhnlicher Differentialgleichungen mit Ordnungs– und Schrittweitensteuerung ([4]).*

PDEX1M: *ein Verfahren zur Lösung von nichtlinearen parabolischen Systemen in einer Raumdimension ([9]).*

MEXX: *ein Verfahren zur Zeitintegration von gekoppelten mechanischen Systemen mit Zustandsbeschränkungen ([8], [7]).*

Das GUI ist für die verschiedenen Anwendungsprogramme, soweit es sinnvoll ist, einheitlich aufgebaut. Die hier verwendeten Beispiele beziehen sich auf Kaskade und PDEX1M.

Zu jeder Anwendung gehört ein Hauptfenster, das über das laufende Anwendungsprogramm informiert und im wesentlichen zwei Steuerungselemente, nämlich eine Menüleiste und eine Reihe von Steuerknöpfen, bietet.

Die Menüleiste enthält bei allen Anwendungen mindestens die Menüs `Example`, `Running`, `Settings` *und* `Information`. *In dem Menü* `Example` *werden alle vordefinierten Beispiele zur Auswahl angeboten. Bezeichnungen, die auf „User“ enden, deuten auf Beispiele hin, die vom Anwender definiert werden. Dafür sind dann je nach Anwendung geeignete Programmstücke in C, FORTRAN oder Reduce einzugeben (siehe Abschnitt 2.3).*

2.1 Ablaufsteuerung des Anwenderprogramms

In der Regel wird der Anwender zunächst eines der unter `Example` *vordefinierten Beispiele auswählen, dann unter* `Settings` *problemabhängige oder den Algorithmus beeinflussende Optionen einstellen (siehe Abschnitt 2.2) und anschließend mit dem* `Start`*–Knopf das Anwendungsprogramm starten. Dieser benennt sich dann in* `Restart` *um. Erneutes Drücken dieses Knopfes startet also das Programm von vorn (siehe Abbildung 1).*

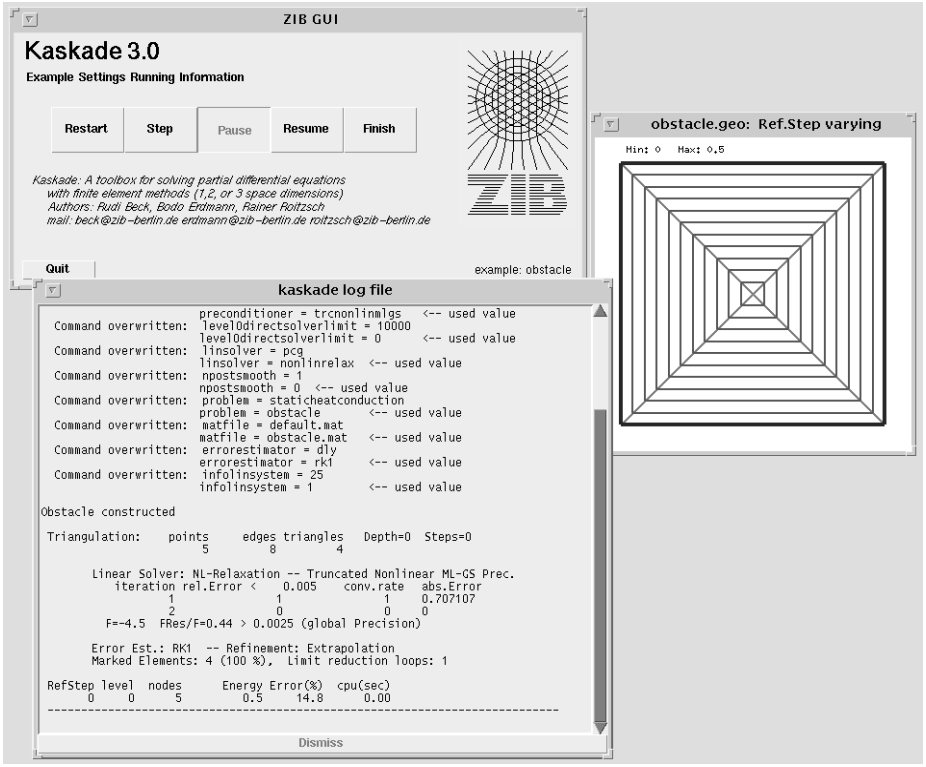

Abbildung 1 Bildschirm nach Start des Kaskade-Beispiels „obstacle“

Programmablauf. *Zur weiteren Steuerung des Programmablaufs dienen die Knöpfe* `Step`, `Pause`, `Resume` *und* `Finish`.

Es ist möglich, einzelne Iterationsschritte (`Step`*) oder bis zum Erreichen einer Lösung (*`Resume`*) zu rechnen und dann auf eine Benutzereingabe zu warten.*

Die fortlaufende Rechnung kann durch `Pause` *wieder in den Einzelschrittmodus umgeschaltet werden.*

Die Anwendung läuft während eines Rechenschrittes entkoppelt vom GUI und reagiert erst wieder am Ende eines Schrittes auf inzwischen gedrückte Knöpfe.

Synchronisation. *Die durch die Oberfläche einstellbaren Optionen werden beim Start der Anwendung übertragen. Eventuelle Änderungen während eines Programmlaufs wirken sich also erst beim nächsten* `Start` *oder* `Restart` *aus. Der Knopf* `Finish` *erlaubt, das Anwendungsprogramm nach dem nächsten Rechenschritt abzubrechen.*

Eine formale Spezifikationsmethode mit Graph–Grammatiken [12] wird in [13] für die Beschreibung dieser Knöpfe angewendet.

Programmausgaben. *Textausgaben des Anwendungsprogramms werden in einem gesonderten Fenster (Überschrift „log“) angezeigt. Darüber hinaus verfügen alle Anwendungen über eigene graphische Ausgabemöglichkeiten.*

Eine Nachbearbeitung, die spezielle Informationen aus den Programmausgaben herausfiltert und zur Beurteilung des Rechenfortschritts innerhalb des GUI aufbereitet, ist in Arbeit.

2.2 Einstellen von Optionen

Unter dem Menü `Settings` *können verschiedene Gruppen von Optionen ausgewählt werden. Damit sind logisch zusammengehörige Optionen übersichtlich zusammengefaßt, beispielsweise Optionen zum numerischen Algorithmus oder Optionen zur Problembeschreibung, die vom jeweiligen Beispiel abhängen (siehe Abbildung 2).*

Optionsfenster. *Für jede Gruppe von Optionen wird ein Fenster eröffnet, in dem die augenblicklich gültigen Einstellungen angezeigt und zur Änderung angeboten werden. Die möglichen Optionen hängen natürlich von der Anwendung ab (vergleiche Abbildungen 2 und 3 mit entsprechenden Optionsgruppen von Kaskade und PDEX1M).*

Dabei haben alle Optionsfenster den gleichen, leicht wiederzuerkennenden Aufbau: Titelzeile, Blöcke von logisch zusammengehörenden Optionen und am unteren Rand eine Knopfleiste mit den folgenden Funktionen:

- `Apply` *übernimmt die geänderten Werte.*
- `Default` *setzt alle angezeigten Optionen auf intern festgelegte Voreinstellungen.*

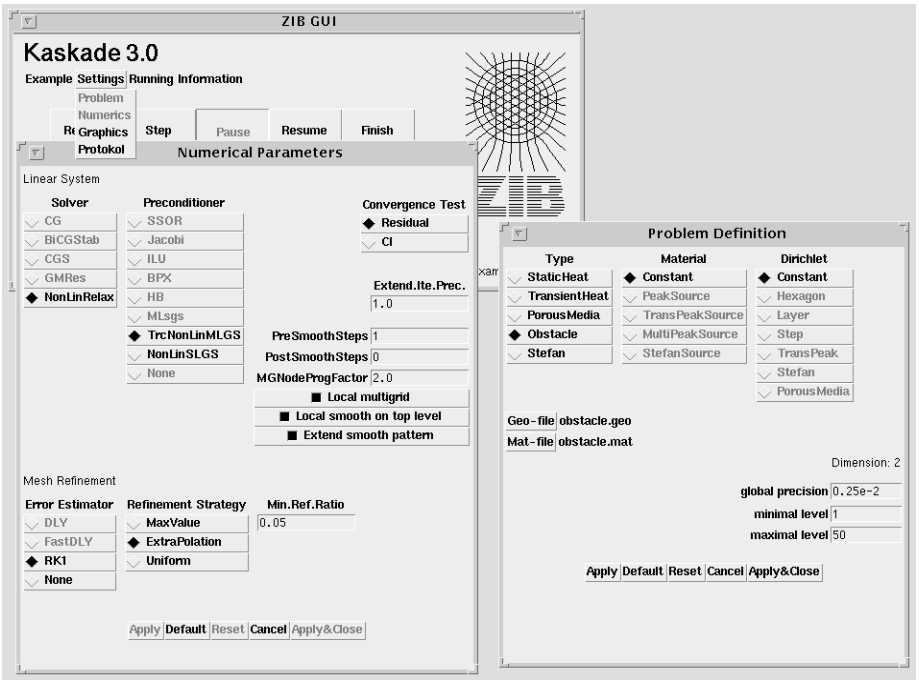

Abbildung 2 Setzen von Optionen für Kaskade

- `Reset` *macht alle Änderungen rückgängig, für die noch nicht* `Apply` *gegeben wurde.*
- `Cancel` *verwirft alle Änderungen, für die noch nicht* `Apply` *gegeben wurde, und schließt das Optionsfenster.*
- `Apply & Close` *übernimmt die geänderten Werte und schließt das Optionsfenster.*

Trotz inhaltlicher Verschiedenheit ist die Darstellung der Optionen meist sehr ähnlich, da es nur wenige Optionstypen gibt. Im wesentlichen sind das

- *Alternativen, aus denen jeweils genau eine ausgewählt wird („radio button“),*
- *Schalter, die ein– oder ausgeschaltet sein können („check button“) und*
- *Werte, z.B. in Form von Zahlen, Dateinamen oder Schiebereglern.*

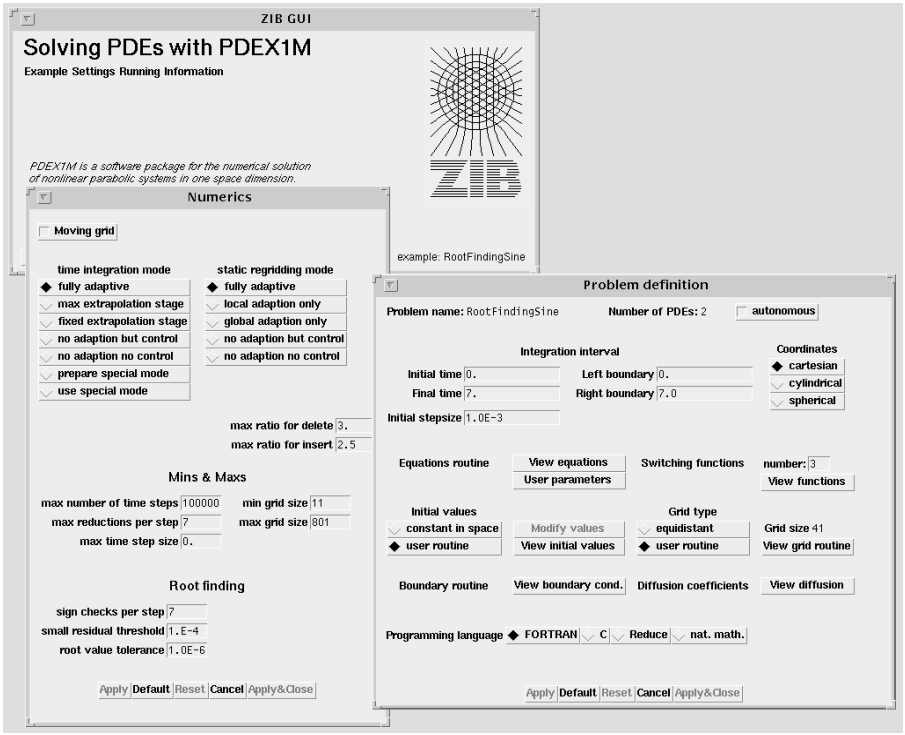

Abbildung 3 Setzen von Optionen für PDEX1M

Benutzerführung. *Neben der übersichtlichen Gruppierung der Optionen bietet eine graphische Oberfläche weitere Vorteile für den Anwender. Optionen, die in dem jeweiligen Zusammenhang keine Bedeutung haben oder sinnvollerweise nicht gesetzt werden können, sind entweder im Optionsfenster blaß dargestellt oder werden gar nicht erst gezeigt. Damit können diese Optionen auch nicht irrtümlich gesetzt werden.*

Zum Beispiel sind in Abbildung 2 bei den Optionen „Linear System“ bzw. „Preconditioner“ die Alternativen, die für das gewählte Beispiel nicht sinnvoll sind, gesperrt. Beim Vergleich der numerischen Optionen für PDEX1M in der Abbildung 4 sieht man, daß z.B. ein Wert für die „grid viscosity“ nur angezeigt wird, wenn der Schalter „Moving grid“ gesetzt ist, oder daß es von der Wahl des „time integration mode“ abhängt, ob die Angabe der „time step size“ erforderlich ist.

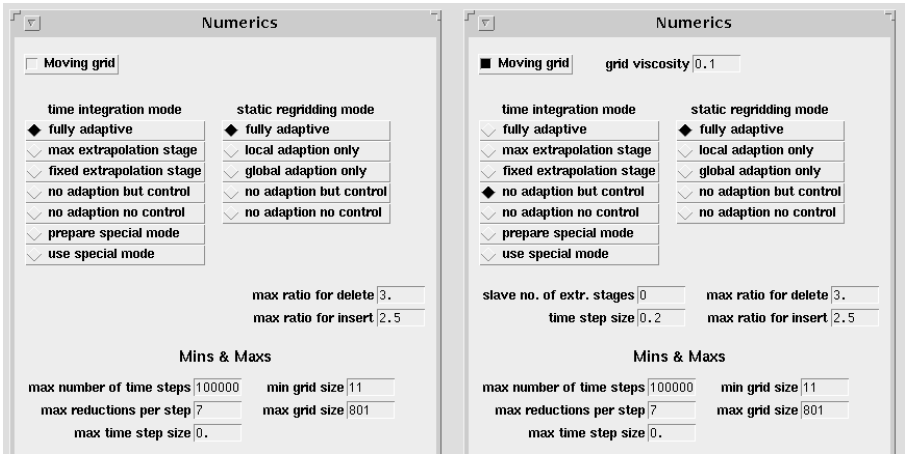

Abbildung 4 Zwei Varianten von numerischen Optionen für PDEX1M

2.3 Benutzerdefinierte Beispiele

Die Problembeschreibung unserer numerischen Anwendungen besteht im allgemeinen aus einigen Zahlenwerten und einem oder mehreren Unterprogrammen oder Programmstücken, die der Anwender liefern muß. PDEX1M benötigt z.B. Werte für Intervallgrenzen und Anfangswerte sowie Unterprogramme zur Beschreibung der rechten Seite der Differentialgleichungen oder zur Darstellung der Randbedingungen.

Unter der Optionengruppe `Problem definition` *können die Zahlenwerte direkt eingegeben werden. Für die erforderlichen Unterprogramme werden Fenster mit editierbarem Text ohne Prozedurköpfe, zugehörige Variablenvereinbarungen oder ähnliches gezeigt (siehe Abbildung 5).*

Die so erzeugten Programmstücke werden mit den vordefinierten Prozedurrahmen zusammengesetzt, gegebenenfalls übersetzt und gebunden. Je nach Anwendung können die Sprachen FORTRAN, C oder Reduce verwendet werden.

3 Hinweise für Erstellung von GUIs

In diesem Abschnitt tragen wir Hinweise zum Aufbau graphischer Benutzeroberflächen für numerisch–orientierte Programme zusammen. Die dabei gemachten Beobachtungen führen uns zu einigen speziellen Anforderungen an den Baukasten, mit dem man das GUI zusammenstellt. Die daraus resultierenden Imple-

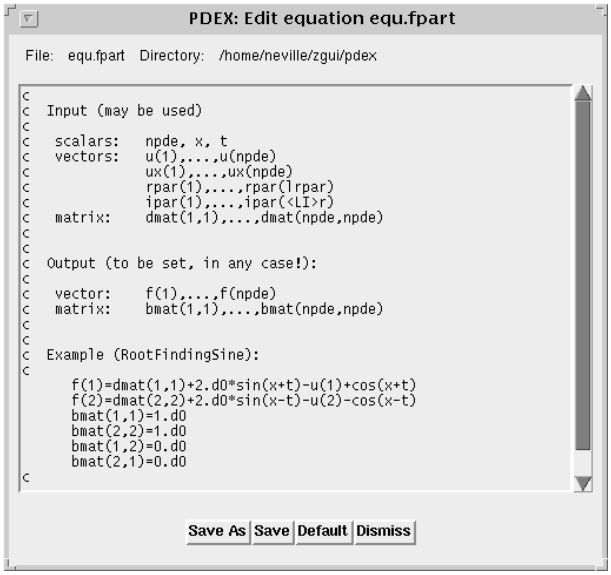

Abbildung 5 FORTRAN-Zeilen zur Darstellung der rechten Seite in PDEX1M

mentierungstechniken werden im nächsten Abschnitt beschrieben.

Wir setzen voraus, daß der Anwender/Entwickler numerischer Software Zugang zu einem Rechner hat, der die Konstuktion graphischer Benutzeroberflächen unterstützt. Solche Systeme sind etwa Unix mit X11(Motif), Apple's Macintosh oder PCs mit einer Windows–Variante. Für die jeweiligen Systeme versuchen die Hersteller, zum Teil mit den unterschiedlichsten Methoden, eine konsistente Benutzeroberfläche sicherzustellen. Die bei der Bedienung verwendeten Techniken hat der Benutzer, häufig ohne es zu wissen, verinnerlicht. Neue Komponenten, wie die hier beschriebenen GUIs, müssen sich an diese lokalen Gegebenheiten anpassen. Die low–level Details, wie das Aussehen der Fenster, wird man in der Regel durch den Window–Manager und die Verwendung eines GUI–Baukastens für das jeweilige System automatisch einhalten. Für die weiteren, lokalen Konventionen ist man auf einen Style Guide (etwa [1]) und/oder einen Baukasten mit höherer Funktionalität angewiesen.

Im folgendem wollen wir Hinweise für die Darstellung von Optionen durch GUIs sammeln.

Namenskonventionen. *Die benannten Elemente der graphischen Benutzeroberfläche sollten möglichst selbsterklärend und kurz sein. Diese Binsenweisheit*

ist im Kontext der numerischen Programme ebenfalls sinnvoll und führt auf die bekannten Probleme, wenn Benutzer und Implementierer nicht aus dem gleichen „Kulturkreis“ (der Numerik) stammen. Abzuwägen ist ein häufig sich widersprechender Gebrauch von Worten gegen deren konsistente Verwendung. Als ein Beispiel verweisen wir auf die Bezeichnungen `eps`, `tol` *oder* `globError` *als Namen der Option für die verlangte Genauigkeit der numerischen Lösung. Selbst die etwas längeren Namen sagen noch nicht aus, ob eine relative oder absolute Zahl gemeint ist und um welche Fehlernorm es sich handelt. Für die Erklärung der Begriffe sollte ein Hilfe–Mechanismus verwendet werden.*

Gruppierung. *Logisch zusammenhängende Bedienfunktionen sollten auch optisch zusammen liegen. Nicht relevante Funktionen können ausgeblendet werden, um die darzustellende Informationsmenge zu begrenzen. Es muß abgewogen werden zwischen dem manchmal irritierenden Neuaufbau von Fenstern und der Tatsache, daß Möglichkeiten, die man sieht, ohne daß man sie verwenden kann, trotzdem eine interessante Information darstellen. Beispiel für eine angemessene Informationsreduktion ist etwa, daß bei Kaskade erst mit der Selektion des linearen Lösers* `gmres` *ein Eingabefeld zum Einstellen der Zahl der Hilfsvektoren sichtbar wird.*

Wichtige und spezielle Optionen, die nur der Spezialist verwendet, sollten nicht auf dem gleichen optischen Niveau angeboten werden. Hier kann eine geometrische, hierarchische Gruppierung helfen.

Abhängigkeiten. *Kombinationen von Optionen oder Werten sind manchmal sinnlos. Eine gute Benutzeroberfläche erlaubt sichern solcher Wertekombinationen erst gar nicht und macht das auch optisch sichtbar. Das Wissen um solche Abhängigkeiten und deren Implementierung ist alles andere als trivial. Es muß hier dieselbe Arbeit, die in dem numerischen Programm (hoffentlich!) geleistet wurde, nochmals realisiert werden. Für den Anfang kann man sich natürlich auch auf das Programm und dessen Fehlerverhalten verlassen. Beispiele für solche Abhängigkeiten sind etwa die Auswahl vordefinierter Modellprobleme und eine eingeschränkte Auswahl der zulässigen Lösungsverfahren.*

Hilfe–Mechanismus. *Durchgängig muß dem Benutzer die Möglichkeit angeboten werden, zusätzlich Informationen über die Bedeutung der einzelnen Bedienelemente zu erhalten. Dazu sind Mechanismen besonders gut geeignet, die auf Anforderung zu Bereichen, auf die mit der Maus gezeigt wird, schnell die entsprechende Information liefern.*

Arbeitsminimierung. *Bei der Benutzung eines GUIs ist etwas schiefgelaufen, wenn man bei jeder Benutzung eine Reihe von Handgriffen wiederholen muß, wie*

etwa das Verschieben von Fenstern auf geeignete Positionen. Das GUI sollte die Möglichkeiten anbieten, sich Einstellungen und Anpassungen zu merken.

4 Implementierung

Das hier beschriebene GUI ist mit Hilfe des Tcl/Tk–Baukastens implementiert. Es besteht aus einem Hauptprogramm und einem Satz Tcl–Prozeduren, die es erlauben, für die verschiedenen numerischen Anwendungen eine konkrete Benutzerschnittstelle zusammenzustellen. Damit die Anwendungen vernünftig mit dem GUI zusammenwirken, haben wir eine Programmierschnittstelle für die Anwendungsprogramme (API) definiert, die die kritischen Probleme regeln soll.

4.1 Tcl/Tk

John K. Ousterhout's Tcl/Tk [11] ist eine Kommandosprache (Tcl) mit einem reichen Satz von Kommandos (Tk) für den Aufbau graphischer Benutzerschnittstellen. Die „Main Event Loop" ist dabei im Basissystem integriert. Die Unix-Version implementiert den Motif „look and feel", so daß sich der Anwender nicht mehr um die X11–Programmierung kümmern muß. Hier einige der Vorteile von Tcl/Tk:

Verfügbar. *Die Tcl/Tk Quellen werden frei verteilt, es sind keine speziellen Lizenzbedingungen zu erfüllen. An dem System wird weiter entwickelt. So sind neue Versionen für Apple's Macintosh und MicroSoft Windows zu erwarten. Die Installation ist problemlos.*

Einfach zu lernen und zu benutzen. *Tcl ist zwar einfach zu erlernen, zur Erstellung umfangreicher Software allerdings nicht immer gut geeignet. Bei größeren Projekten machen sich die Defizite der Sprache, wie die fehlenden Datentypen oder der globale Namensraum, negativ bemerkbar. Auf der anderen Seite erleichtert das Interpreter–Konzept das Ausprobieren der Tk–Widgets und damit auch das Testen.*

Es gibt eine Reihe guter Bücher über Tcl/Tk (Ousterhout [11], Welch [14], and Libes [6]), die zum Erlernen und als Referenz verwendbar sind. Für die aktuellen Probleme findet man Unterstützung in der Internet Newsgruppe `comp.lang.tcl`.

Erweiterbar. *Der Funktionsumfang von Tcl/Tk kann durch Tcl–Prozeduren oder durch die Programmierung neuer Funktionen in einer kompilierbaren Programmiersprache (etwa C) erweitert werden. Durch die Tcl–Programmierung*

wird eine Art „rapid prototyping“ unterstüzt. Aus Effizienz– oder Bequemlichkeitsgründen kann man diese Prozeduren später in C neu schreiben. Schließlich besteht auch noch die Möglichkeit, das GUI mit der Anwendung direkt zu verschmelzen.

Bewährt. *Tcl/Tk funktioniert einfach.*

4.2 Modell der Schnittstelle zwischen GUI und Anwendung

Wir verwenden ein einfaches Modell der Schnittstelle zwischen GUI und Anwendung. Der Benutzer stellt mit Hilfe des GUI eine Datei mit Anweisungen/Optionen für die Anwendung zusammen. Auf Befehl des Benutzers startet das GUI die Anwendung und reagiert auf vorher definierte Eingabeanforderungen (Prompts) der Anwendung. Vorgesehen sind Prompts für den Anfang, den Zwischenschritt und das Ende der Anwendung (siehe Abbildung 6). Zur einfachen Anpassung der Anwendung stehen eine Reihe von Routinen (C, Fortran) zur Verfügung, mit denen man z.B. die Prompts realisieren oder die Parameterdatei abfragen kann.

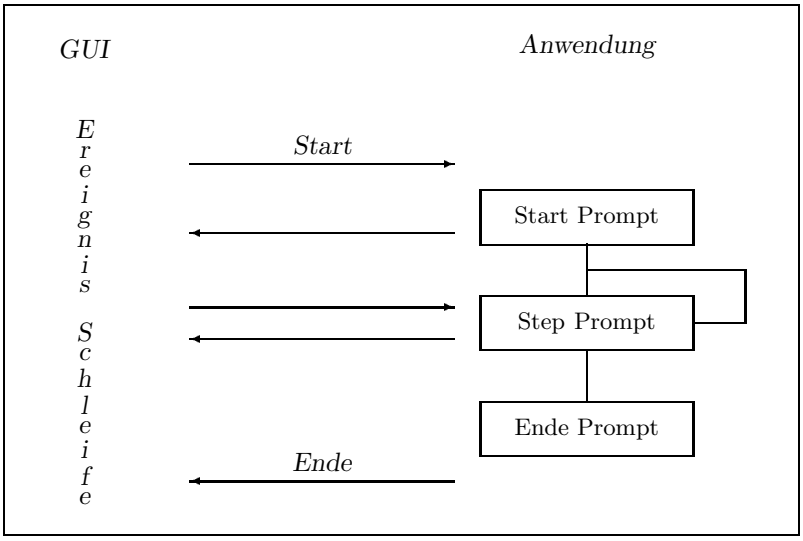

Abbildung 6 Prozess Interaktionen

Die Unix–Version des GUI verwendet zur Realisierung die Standard Ein/Ausgabe und das Pipeline–Konzept.

4.3 Bibliothek der zusammengesetzten Widgets

Eine konkrete Benutzerschnittstelle für eine Anwendung wird durch den Aufruf geeigneter Tcl–Prozeduren, die zusammengesetze Widgets aufbauen und in den richtigen Kontext setzen, realisiert. Damit wird dafür gesorgt, daß die anwendungsspezifischen Daten einheitlich präsentiert werden. Wir möchten das an einigen Beispielen zeigen:

Hauptfenster. *Jedes GUI muß eine Routine zur Definition des Hauptfensters aufrufen. Die Parameter dienen dabei zum Füllen des Layouts in Abbildung 7.*

Abbildung 7 Layout des Hauptfensters

Menüs. *Zur Verwaltung der Menüpunkte* `Examples` *und* `Settings` *haben wir eine Reihe von Tcl–Prozeduren entwickelt. Im ersten Fall werden Unterpunkte aus der Existenz von Beispieldateien, deren Namen einer bestimmten Regel entsprechen, automatisch generiert. Zusätzlich wird dem Benutzer ein Mechanismus (*`Save`, `SaveAs` *und* `Default`*) zum Ändern bzw. Erstellen weiterer Beispiele an die Hand gegeben. Mit dem* `Settings` *Menüpunkt können Fenster zum Zeigen und Ändern von anwendungsspezifischen Optionen geöffnet werden.*

Anwendungskontrolle. *Die Anwendung wird über eine Anwendungskontrolleiste gesteuert. Neben den Knöpfen zum Starten, Fortsetzen, Anhalten und Beenden der Anwendung wird in einer Statuszeile Information über Namen, Rechenzeit und Speicherverbrauch gezeigt.*

Die Anwendung läuft im Einschrittmodus, d.h. nach jedem Rechenschritt gibt sie einen vereinbarten Text („prompt“) aus und wartet auf eine Eingabe durch das GUI, bevor der nächste Rechenschritt ausgeführt wird. Diesen Befehl zur

Fortsetzung schickt das GUI entweder einmalig, wenn der Anwender den `Step`*–Knopf drückt, oder automatisch immer wieder, wenn der* `Resume`*–Knopf gedrückt wurde, und solange, bis die Anwendung fertig ist oder der Anwender* `Pause` *oder* `Finish` *gedrückt hat.*

Optionsverwaltung. *Alle Fenster, mit denen man sich einen Ausschnitt der Optionen der Anwendung ansehen kann, haben die gleiche Struktur, siehe Abbildungen 2, 3 und 8. Im unterem Bereich befinden sich die Knöpfe, zum Akzeptieren oder Verwerfen der gemachten Änderungen. Alle anderen Elemente darüber, etwa für*

- *die Auswahl von Werten: „radio buttons“, „check buttons“,*
- *die Eingabe von Zahlen,*
- *die Definition von Feld– oder Matrixwerten oder*
- *die Einstellung (Auswahl) von Dateien*

sind mit diesen Knöpfen geeignet verbunden. So ist z.B. sofort sichtbar, ob irgendeine Option geändert wurde oder ob Voreinstellungen existieren. Diese Verbindung haben wir durch Tcl–Prozeduren realisiert, die die Zusammenstellung eines GUI zu einer Anwendung erheblich vereinfachen.

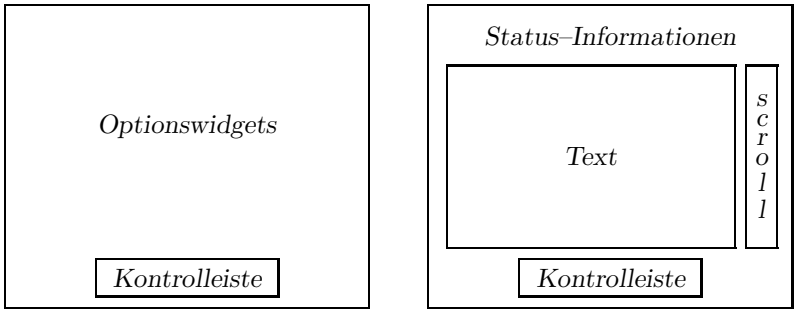

Abbildung 8 Options– und Textfenster

Textfenster. *Längere Texte werden in speziellen Textwidgets dargestellt, die sich lediglich im Rahmen unterscheiden, siehe Abbildungen 5 und 8. So können Dateien zum Ansehen oder Editieren angeboten, Programmausgaben in Logbüchern gesammelt oder auch feste Texte gezeigt werden.*

Hilfe. *Hilfstexte können an Widgets gebunden werden, so daß sie mit einem Mausklick+Zusatztaste jederzeit abrufbar sind. Mit Hilfe des Menüpunktes* `Information` *stehen ausführliche Beschreibungen zur Verfügung.*

4.4 Verwaltung der Optionssätze

Der Zugriff auf Optionen ist über eine Programmierschnittstelle für C und Fortran (für die numerische Anwendung) und Tcl (für das GUI) realisiert. Zu den Leistungen dieses API (application programming interface) gehören

- *Kreieren und Löschen eines Optionsatzes,*
- *Lesen/Schreiben einer Optionsdatei,*
- *Lesen/Schreiben und Löschen einzelner Optionen,*
- *Konvertierungen von Optionswerten und*
- *Lesen/Schreiben von Feldern und Matrizen.*

Literaturverzeichnis

[1] Apple Computer, Inc.: *Macintosh Human Interface Guidelines.* Addison–Wesley, Reading (1992)

[2] R. Beck; B. Erdmann; R. Roitzsch: *An Object-Oriented Adaptive Finite Element Code.* Konrad–Zuse–Zentrum, Technical Report TR 95–04 (1995)

[3] Peter Deuflhard: *A Modified Newton Method for the Solution of Ill-Conditioned Systems of Nonlinear Equations with Applications to Multiple Shooting.* Numer. Math. **22**, p. 289–315 (1974)

[4] Peter Deuflhard: *Recent Progress in Extrapolation Methods for ODE's.* SIAM Review **27**, p. 505–535 (1985)

[5] P. Deuflhard; P. Leinen; H. Yserentant: *Concepts of an adaptive hierarchical finite element code.* IMPACT **1**, (1989)

[6] Don Libes: *Exploring Expect.* O'Reilly, Sebastopol (1995)

[7] Ch. Lubich; Ch. Engstler; U. Nowak; U. Pöhle: *Numerical Integration of Constrained Mechanical Systems Using MEXX.* Mech. Struct. & Mach., **23**(4), p. 473–495 (1995)

[8] Ch. Lubich: *Extrapolation integrators for constrained multibody systems.* IMPACT Comp. Sci. Eng. **3**, p. 213–234 (1991)

[9] Ulrich Nowak: *Adaptive Linienmethoden für nichtlineare parabolische Systeme in einer Raumdimension.* Konrad–Zuse–Zentrum, Technical Report TR 93–14 (1993)

[10] U. Nowak; L. Weimann: *A Family of Newton Codes for Systems of Highly Nonlinear Equations.* Konrad–Zuse–Zentrum, Technical Report TR 91–10 (1991)

[11] John K. Ousterhout: *Tcl and the Tk Toolkit.* Addison–Wesley, Reading (1994)

[12] Bettina E. Sucrow: *Formale Spezifikation graphischer Benutzungsschnittstellen mit Hilfe von Graph–Grammmatiken.* In W. Mackens und S. Rump, editors, *Proceedings des Workshops* Software Engineering im Scientific Computing. (1996)

[13] U. Nowak; U. Pöhle; R. Roitzsch; B. E. Sucrow: *Formale Spezifikation ZIB–GUI mit Hilfe von Graph–Grammatiken.* In W. Mackens und S. Rump, editors, *Proceedings des Workshops* Software Engineering im Scientific Computing. (1996)

[14] Brent B. Welch: *Practical Programming in Tcl and Tk.* Prentice Hall, Upper Saddle River (1995)