Das Frontend von LooPo

Ausarbeitung zum Programmierpraktikum

Nils Ellmenreich
Nils@Uni-Passau.De

12. März 1996

1 Einführung

Im Rahmen des LooPo (kurz für: *Loop Parallelization in the Polytope Model*) Projektes wurde mit mehreren Diplomarbeiten und Programmierpraktika ein Prototyp eines automatischen Parallelisierers für Schleifenprogramme entwickelt. Der Aufbau des Projektes ist modular, d.h. jedes von einer Person entwickelte Modul ist in einem eigenen ausführbaren Programm enthalten — diese Module bilden zusammen LooPo. Während der Laufzeit ist nur ein Modul ständig aktiv, das sogenannte Frontend. Es regelt den Ablauf durch Aufruf der einzelnen Module mit den entsprechenden Parametern und dient als Schnittstelle zwischen dem Benutzer und den einzelnen Modulen.

Motivation für die Modularisierung war zum einen, die einzelnen Programmteile der verschiedenen Programmierer möglichst unabhängig von einander zu halten, damit im Falle von Implementationsschwierigkeiten in einem Modul nicht das ganze Projekt zum Stillstand kommt. Außerdem ist LooPo durch diese Aufteilung vergleichsweise speicherfreundlich, da jeweils nur wenige Module gleichzeitig aktiv sind und diese nach Beendigung ihren Speicher freigeben.

Als wichtigster Grund besteht aber die Absicht, den Programmtext der Schleifenparallelisierung freizuhalten von allem, was direkt mit der Graphikoberfläche in Verbindung steht. Als Graphiktoolkit wurde Tcl/Tk in Verbindung mit C++ gewählt; da es aber möglich bleiben soll, die Graphikoberfläche von LooPo auf ein anderes Toolkit umzustellen, wurde diese im Frontend-Modul gekapselt. Somit müßte bei einer Umstellung nur das Frontend neugeschrieben werden.

1.1 Voraussetzungen für den Betrieb

Voraussetzung für das Übersetzen von LooPo sind

- ◊ gcc-2.5.8 oder höher (incl. libg++)
- ◊ GNU make

◊ X11R5 oder höher

◊ Tcl 7.4 und Tk 4.0 oder höher

◊ Linux, SunOS 4.1.3 oder Solaris 2.x

◊ PIP (siehe dazu [Fea88]; PIP ist als PD erhältlich)

Da die genannte Software auch auf anderen Betriebssystemen zu finden ist, sollten auch andere UNIX Plattformen möglich sein.

1.2 Die Module von LooPo

Neben dem Frontend existieren folgende Module, deren Gesamtheit LooPo bildet:

Scanner/Parser liest den Sourcecode des zu parallelisierenden Quellprogramms ein und konstruiert daraus der Parsebaum [Gu95].

Dependence Analysis untersucht das Quellprogramm im Parsebaum auf Abhängigkeiten zwischen den Statements [Kei95].

Scheduler/Allocator berechnen das „Space-Time-Mapping“, also Schedule und Allokation nach dem Polyedermodell [Wi95].

Target Generation konstruiert den Parsebaum des parallelisierten Zielprogramms unter Zuhilfenahme von Abhängigkeiten, Schedule und Allokation [We95].

Target Output wandelt den Zielparsebaum in eine textuelle Repräsentation des parallelisierten Programms incl. Synchronisationsstatements [Fa95].

Filter ermöglicht das Herausfiltern einer Teilmenge der von Depencence Test gefundenen Abhängigkeiten für die Eingabe von Dispo2

Dispo2 Anzeige des Indexraumes (max. 3 dim) und der vorhandenen Abhängigkeiten [Wü95].

Die Module Scanner/Parser bis Target Output müssen genau in der angegebenen Reihenfolge durchlaufen werden. Filter und Dispo 2 sind optionale Module zur Visualisierung, die nach Bestimmung der Abhängigkeiten durch den Dependence Test aufgerufen werden können.

2 Aufbau des Frontends

Das Umfang dieses Programmierpraktikums besteht aus zwei getrennten Teilen. Zum einen gibt es das eigentliche Frontend, das aus dem ausführbaren Programm `$LOOPODIR/frontend/loopo` sowie dem Tcl-Skript `$LOOPODIR/lib/MainWin.tcl` besteht. Der Benutzer selbst ruft nur das loopo-Binary auf; dieses öffnet das Hauptfenster und gibt Gelegenheit, per Mausklick die weiteren Module zu starten. Der zweite Teil des Frontends ist für den Benutzer nicht direkt sichtbar. Er besteht aus der Dialog-Library, die zu den einzelnen Modulen hinzugelinkt wird und diesen C++-Klassen für Dialoge wie z.B. Fehlerboxen zur Verfügung stellt. Auf diese Weise können die Module auch direkt mit dem Benutzer kommunizieren, ohne selbst Code der Graphikoberfläche zu enthalten.

Außer in diesen beiden Teilen des Frontends kommt nur noch im `Dispo2`-Modul [Wü95] expliziter Code für die Graphikoberfläche vor. Dieses Modul dient zur graphischen Darstellung der Indexräume von Schleifennestern und den in ihnen enthaltenen Datenabhängigkeiten. Bei diesem Modul war eine Trennung von Graphik und C++-Code nicht sinnvoll, da hier die Verbindung zwischen beiden sehr eng ist.

2.1 Der source-Baum

Wenn `$LOOPODIR` die Wurzel des Source-Baumes von LooPo darstellt, dann befinden sich alle C++-Quelltexte des Frontends in `$LOOPODIR/frontend/`. Zur Zeit (12. März 1996) befinden sich dort folgende Dateien:

```
Dialogs.cc      MainWin.h       definitions.h   fifo.cc
Dialogs.h       TKWindow.cc     error.cc        fifo.h
MainWin.cc      TKWindow.h      error.h         main.cc
```

Im einzelnen sind das:

TKWindow enthält die Basisklasse zur Verwaltung eines Tcl/Tk Fensters von C++. Enthalten sind u.a. Funktionen zum Laden des entsprechenden Skriptfiles, zum Anzeigen und Löschen des X11-Fensters. Die Klasse `TKWindow` basiert auf `CFWindow` von Carsten Friedrich [1].

Dialogs besteht aus mehreren Klassen, die jeweils eine Dialogbox zur Benutzerinteraktion enthalten. Diese Klassen sind von `TKWindow` abgeleitet. Weiterhin ist die Klasse `sessionlog` enthalten, über die die Module mit Hilfe einer FIFO (siehe Abschnitt 3.3) mit dem Frontend kommunizieren können. Das Kompilat von Dialogs befindet sich in der `libdlg` im `general`-Verzeichnis.

MainWin beinhaltet praktisch die ganze „Intelligenz" des Frontends. Hier befindet sich die Schnittstelle zwischen dem Tcl/Tk Fenster und C++. Über Tcl-Library Funktionen können die Optionen in den Menüs des Hauptfensters gesetzt und ausgelesen werden. Von hier werden auch die anderen LooPo-Module mit ihren Optionen gestartet, nachdem der Benutzer auf den entsprechenden Button des Hauptprogramms geklickt hat.

fifo enthält eine Klasse, mit der die FIFO (oder `named pipe`) Kommunikation zwischen dem Frontend und den Modulen realisiert wird. Sie wird in Dialogs zum Schreiben in die FIFO, im Hauptprogramm (main.cc) zum Auslesen benutzt.

[1] email: `friedric@fmi.uni-passau.de`

main ist schließlich das kurze Hauptprogramm, das die FIFO anlegt, das Hauptfenster startet und zum Schluß das temporäre Verzeichnis löscht.

error besteht nur aus einer Funktion zur Ausgabe einer standardisierten Fehlermeldung.

Die Dateien `TKWindow.cc`, `main.cc` und `fifo.cc` sind vergleichsweise klein und werden bei der Beschreibung des Kernstücks vom Frontend, dem MainWin, mit erwähnt. Weiterhin ist Dialogs kein eigentlicher Bestandteil des Frontends, sondern quasi die Bereitstellung von Frontend-Funktionalität an die Module. Der Aufbau ist nicht so umfangreich wie der von MainWin; die dort vorhandenen Klassen sind aber ebenfalls von `TKWindow` abgeleitet, so daß hinter jedem Dialogfenster das gleiche Prinzip wie bei MainWin steht. Es kommen keine wesentlichen Eigenschaften hinzu, die MainWin nicht auch hätte. Eine Beschreibung von Dialogs findet sich in Abschnitt 4.

Im weiteren wird der Schwerpunkt auf der Beschreibung des eigentlichen Frontends liegen, dessen Kern die Klasse MainWin bildet. Neben der Darstellung des Ist-Zustandes werden später Erweiterungsmöglichkeiten aufgezeigt.

2.2 Das Hauptfenster

Bei Start des LooPo-Programmes wird in der Klasse MainWin in `MainWin.cc` das Tcl-Script `MainWin.tcl` geladen und nach Initialisierung des Interpreters erscheint das Hauptfenster von LooPo (Abb. 1 - hier in einem späteren Stadium).

Im oberen Teil unterhalb der Menüleiste werden die Dateinamen des Quellprogramms sowie der Parserausgabe angezeigt. Darunter befindet sich die Buttonleiste, mit der die einzelnen Module von LooPo aufgerufen werden. Später sollen die Buttons nur anklickbar sein, falls der Aufruf des entsprechenden Moduls auch sinnvoll ist (siehe Abschnitt 5).

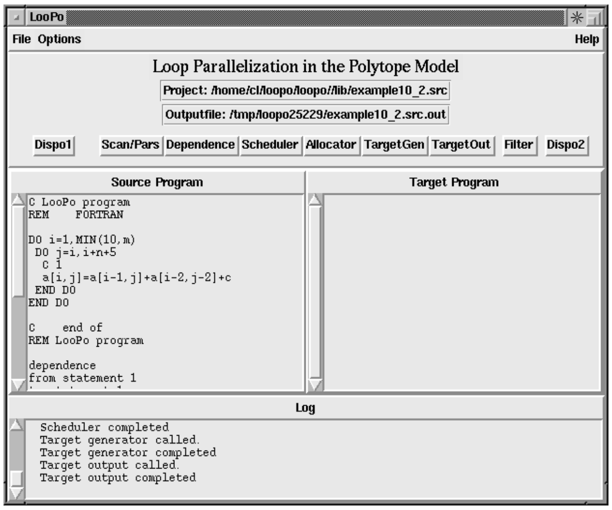

Abbildung 1: Das LooPo-Hauptfenster

Die beiden Teilfenster unter der Buttonleiste ermöglichen eine Gegenüberstellung von Quell- und Zielprogramm. Das Quellprogramm wird nach erfolgreichem Beenden des Scanner/Parsers in das linke Fenster geladen, wobei nicht der originale Prgorammtext verwendet wird, sondern eine vom Target Ouput auf Basis des erzeugten Parsebaums „verschönerte“ Version. Das Zielprogramm erscheint nach Fertigstellung durch das Target Output-Modul in das rechte Fenster.

Der Rest des Hauptfensters wird vom Sitzungsprotokoll (*session log* – siehe auch Abschnitt 3.3) ausgefüllt. Hier werden Status- und Fehlermeldungen der einzelnen Module dargestellt. Diese Meldungen werden über die FIFO von den Modulen an das Frontend zurückgegeben, da die Module als eigenständige UNIX-Prozesse keine Funktionen des Frontends zur Nachrichtenübertragung ausführen können. Der Exit-Status des jeweiligen Moduls wird dabei mit speziellen Schlüsselwörtern übergeben, die nicht im Log-Fenster angezeigt werden.

3 Die Klasse MainWin

Wie schon in der Einleitung erwähnt, kommt der Klasse MainWin zentrale Bedeutung innerhalb des Frontends zu, weshalb sie hier ausführlich beschrieben werden soll. Zuerst soll aber auf die Initialisierung in der Hauptfunktion eingegangen werden.

Die wesentlichen Initialisierungen der Hauptfunktion sind:

◊ Anlegen eines temporären Verzeichnisses, das alle während des Laufs von LooPo angelegten Dateien enthalten wird,

◊ Instanziierung der `fifo`- und `MainWin`-Klassen,

◊ Anlegen eines von Tk gesteuerten Timer-Events, der einmal pro Sekunde eine Funktion zum Auslesen der FIFO aufruft,

◊ Starten der Hauptfunktion des Fensters — diese läuft solange, bis der Benutzer über das File-Menü LooPo verläßt.

Zum Schluß finden noch einige Aufräumaktionen statt: die FIFO wird geschlossen, der Event gelöscht und das temporäre Verzeichnis entfernt.

3.1 Interaktion zwischen Tcl, Tk und C++

Tcl [Ou94] ist eine interpretierte Skriptsprache, deren Interpreter in C geschrieben ist. Tk ist eine Erweiterung von Tcl zur Implementierung von OSF-Motif(tm)-ähnlichen Widgets als Graphiktoolkit. Es bietet sich zum einen die Möglichkeit, den Interpreter direkt zu starten und ihn Tcl/Tk Skripts ausführen zu lassen. In diesem Fall wird das gesamte Programm in Tcl geschrieben und interpretiert. Zum anderen kann man von einer high-level-Sprache wie z.B. C oder C++ per Funktionsaufruf einen Tcl-Interpreter starten und ihn ein Skript ausführen lassen.

Bei LooPo wurde der zweite Weg gewählt; das Tcl/Tk Skript wird bis auf wenige Ausnahmen allein zur Implementierung des Graphikinterfaces benutzt. Nur wenige Tcl-Codesegmente dienen dazu, C++ Code aufzurufen, Optionenfenster anzuzeigen oder über einen Fileselektor das Quellprogramm zu laden. Diese Codesegmente werden direkt duch Anwählen von Buttons bzw. Menüpunkten ausgeführt und sind in Anhang A aufgeführt.

Die Erstellung der Tcl-Skripten wurde zu Beginn des Projektes nicht direkt vorgenommen, sondern über den Widget-Builder XF [De94] erledigt. Da XF einige Limitationen in der Manipulation des Codes aufweist, mußte das Skript damals schon manuell verbessert werden. Mit Erscheinen des nicht aufwärtskompatiblen Tk 4.0 wurde die Umstellung und Weiterentwicklung per Hand notwendig, da XF für diese Version nicht mehr zur Verfügung steht.

Die Verbindung zwischen einem laufenden Tcl-Skript und einem C/C++-Programm kommt auf mehrere Arten zustande. Dabei werden Funktionen aus der Tcl-Library verwendet. Diese stellt C-Funktionen zur Verfügung, um Zugriff auf den Tcl-Interpreter zu erhalten. Diese Funktionen beginnen typischerweise mit `Tcl_` und bekommen als ersten Parameter einen Pointer auf den Interpreter.

- ◊ Während der Initialisierung des Hauptfensters wird mit `Tk_CreateTimerHandler` ein Timerevent erzeugt. Das hat zur Folge, daß der Tcl-Interpreter nach Ablauf einer bestimmten Zeitspanne eine C++-Funktion aufruft. Da dieser Event nur einmal definiert ist, muß in der aufgerufenen Funktion der jeweils nächste Event erzeugt werden.
- ◊ mit Hilfe der Tcl-Library-Funktionen `Tcl_GetVar` und `Tcl_SetVar` können globale Tcl-Variablen innerhalb des laufenden Interpreters gesetzt und gelesen werden.
- ◊ mit `Tcl_CreateCommand` wird ein neuer Tcl-Befehl innerhalb des laufenden Interpreters erzeugt. Bei Verwendung dieses Befehls wird dann eine benutzerdefinierte C/C++-Funktion aufgerufen, der auch Parameter übergeben werden können.
- ◊ per `Tcl_Eval` können Tcl-Befehle von C++ aus in dem Interpreter abgearbeitet werden.

Diese Möglichkeiten werden bei LooPo folgendermaßen genutzt: Durch den Timer-Event wird regelmäßig eine C++-Funktion aufgerufen, die den FIFO ausliest und die übermittelten Nachrichten im *session log* zur Anzeige bringt. Die Inhalte von selektierten Dateinamen, Zuständen von Radiobuttons, usw. stehen in Tcl-Variablen, welche von C++ mit Tcl-Funktionen gelesen bzw. gesetzt werden können.

Um von Tcl aus z.B. als Reaktion auf das Drücken eines Buttons C++-Code auszuführen, wurde bei Initialisierung der Klasse MainWin das „neue" Tcl-Kommando `callback` bei dem Tcl-Interpreter angemeldet. Die Verwendung von `callback` bewirkt den Aufruf einer C++-Funktion, der Argumente übergeben werden können. Um den Overhaed gering zu halten, wurden keine weiteren Befehle definiert; verschiedene Aktionen werden nach den `callback` übergebenen Parametern unterschieden. Das Drücken auf einen Modulbutton bewirkt z.B. die Anwendung von `callback` — die C++-Funktion startet dann das gewünschte Modul. Schließlich werden direkte Tcl-Befehle von C++ aus verwendet, um beispielsweise den Zustand von Buttons zu ändern oder einen Text in ein Textwidget einzufügen.

3.2 Der Konstruktor von MainWin

Im Konstruktor wird das `MainWin` naturgemäß initialisiert. Das beinhaltet das Starten eines Tcl-Interpreters, Anmelden des `callback`-Befehls innerhalb dieses Interpreters, Laden des Tcl-Skriptes `MainWin.tcl`, welches praktisch die Implementation der Graphikoberfläche enthält, setzen von Tcl-Pfadvariablen sowie dem Initialisieren aller Zustände von Buttons mit ihren Defaultwerten.

3.3 Kommunikation zwischen Tcl und C++ zur Laufzeit

Nachdem in der `main`-Funktion das Hauptfenster gestartet wurde, ist die Kontrolle an das Tcl übergeben worden. C++-Code wird jetzt nur ausgeführt, wenn der Timer-Event eintrifft oder nach Drücken eines Buttons der `callback`-Befehl ausgeführt wird.

Aus diesem Grund ist auch nicht verwunderlich, daß die beiden Member-Funktionen `mainwin::callback` und `mainwin::appendlog` den größten Anteil am Code der Klasse MainWin ausmachen. Die Implementation von `callback` ist ziemlich offensichtlich: der Tcl-Befehl callback wird immer mit genau einem Parameter aufgerufen, der das aufzurufende Modul spezifiziert. Die dazu benutzten Schlüsselwörter sind in der Datei `definitions.h` deklariert. Daher ist callback sinngemäß ein großes CASE-Statement, das zu dem gewüschten Modul die UNIX-Kommandozeile mit allen geforderten Parametern zusammensetzt. Einige dieser Parameter hängen von Optionen ab, die der Benutzer in entsprechenden Fenstern des Frontends wählen kann. Die Werte der Optionenvariablen werden daher per `Tcl_GetVar` ausgelesen. Im Anschluß an die Fallunterscheidung wird die gerade zusammengesetzte Kommandozeile per `system`-Call in einer Shell ausgeführt; das Modul wird also als UNIX-Kindprozeß im Hintergrund gestartet.

Die `appendlog`-Funktion ist aufwendiger, da hier mögliche Ergebnisse der Moduln eintreffen und ausgewertet werden. Nachdem ein Modul per `system` gestartet wurde, besteht keine enge Beziehung mehr zum Frontend, wie es z.B. beim geforkten Prozeß wäre. Von den verschiedenen Möglichkeiten, die UNIX zur Interprozeß-Kommunikation bietet, wurde die FIFO ausgewählt, da sie vergleichsweise einfach und portabel zu programmieren ist und völlig ausreichende Funktionalität bietet. Die FIFO selbst wird in der `main`-Funktion von `Tk_TimerProc` ausgelesen; der empfangene String wird an `mainwin::appendlog` übergeben. Die Implementation der Klasse `fifo` in `fifo.cc` ist an ein Beispiel in [Ro85] angelehnt.

Nach dem Start eines Moduls gibt es daher nur noch die Richtung des Informationsflusses vom Modul zum Frontend über die FIFO. Sie erfüllt zwei verschieden Zwecke: zum einen werden Nachrichten, die in `definitions.h` deklarierte Schlüsselwörter enthalten, von `appendlog` abgefangen und daraufhin Aktionen ausgeführt. Zum anderen werden alle übrigen Nachrichten wörtlich in das *session log* Teilfenster eingefügt. Zu den besonderen Nachrichten gehören z.B. die Meldung eines Moduls, daß es erfolgreich oder auch fehlerhaft beenden wird. Als Beispiel sei hier der Scanner/Parser angeführt, nach dessen erfolgreicher Beendigung der Target Output mit dem gerade erzeugten Parsebaum eine formatierte Version des Quellprogramms erzeugt, die in das linke Teilfenster von Frontend eingefügt wird. Sollte der Parser einen Fehler finden, so wird im Log eine entsprechende Fehlermeldung ausgegeben und das mit Zeilennummern versehene, fehlerhafte Quellprogramm geladen, um dem Benutzer einen Hinweis zur Fehlersuche zu geben.

3.4 Die Optionenfenster

In den vorangegangen Abschnitten wurde die Möglichkeit erwähnt, über Optionen Eigenschaften der Module zu verändern. Dazu haben die Module jeweils ein eigenes Optionenfenster, daß sich unter dem „Options" Menüpunkt im Hauptfenster verbirgt (siehe Abb. 1).

Als Beispiel sei in Abbildung 2 das Optionenfenster des Target Generation Modules abgebildet. Die häufigste Form der Auswahl besteht in Radiobuttons, die Eigenschaften triggern können. Im Beispiel können Zielschleifen synchron oder asynchron sein. Entsprechend ist die Auswahl im Optionenfenster möglich. Durch Anklicken der Buttons werden die Werte von Tcl-Variablen verändert. In `definitions.h` wird die Tcl-Variable einem define „zugewiesen“, hier also:

```
#define TARGET_LOOP "target_loop"
```

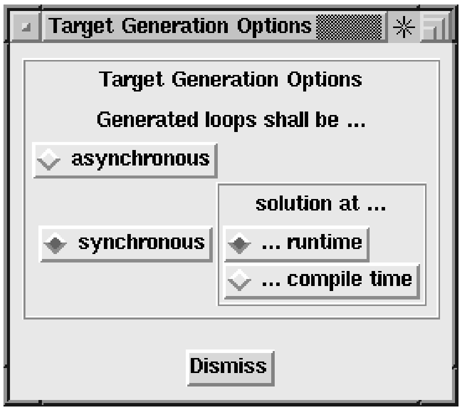

Abbildung 2: Das Target Generation Optionen-Fenster

Jetzt muß noch der Defaultwert gesetzt werden; dazu werden für jede Variablen alle Werte definiert, die diese Variable annehmen kann. Der Default wird auf einen dieser Werte gesetzt:

```
#define SYNC "sync"
#define ASYNC "async"
#define TARGET_LOOP_DEFAULT SYNC
```

Wie in Abschnitt 3.2 dargestellt, wird im Konstruktor von MainWin der Default an den Tcl-Interpreter übergeben.

```
Tcl_SetVar(interp, TARGET_LOOP,
           TARGET_LOOP_DEFAULT, 0);
```

Beim Aufruf des Target Generators wird der aktuelle Wert der Tcl-Variable, der direkt vom Zustand der Buttons abhängt, ausgelesen und die entsprechende Option an die auszuführende Kommandozeile (hier: `filename`) angehängt:

```
if(!strcmp(Tcl_GetVar(interp,
           TARGET_LOOP ,0), SYNC)){
    filename += " -sync ";
    [...]
  }else
    filename += " -async ";
```

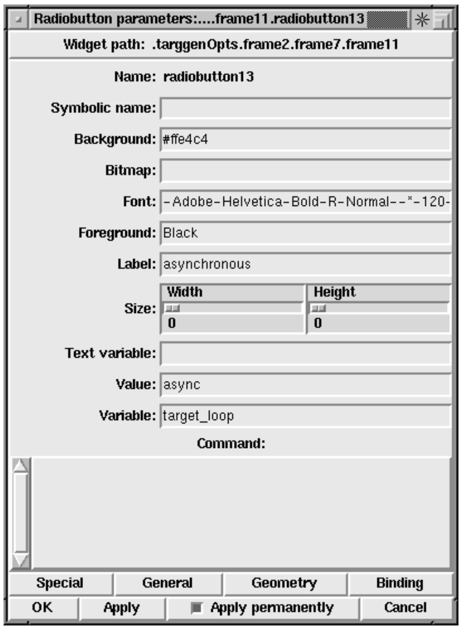

Abbildung 3: Das Optionen-Fenster im XF

An den zuvor beschriebenen Stellen im C++ Code ist ersichtlich, wo Erweiterungen nötig sind, um eine zusätzliche Funktion einzubauen. Daneben ist natürlich auch eine Modifkation des Tcl-Skriptes notwendig. Eine genaue Beschreibung der Benutzung von Tcl/Tk und XF kann hier nicht gegeben werden, detaillierte Beschriebungen findet man in [Ou94] und

[De94]. Innerhalb der Struktur des Optionen-Fensters muß ein Rahmen angelegt werden, in dem dann die zur Auswahl der Option erforderlichen Objekte unterzubringen sind.

Als Beispiel kann hier ein Radiobutton benutzt werden. Das dazugehörige Optionen-Fenster im Widget-Builder XF ist in Abb. 3 zu sehen. Die drei wesentlichen Eintragungen sind `Value`, `Variable` und `Label`. Letzteres bezeichnet einfach nur den Text, der auf dem Button steht. `Variable` bezeichnet die Tcl-Variable, in der der in `Value` bezeichnete Wert gespeichert wird, falls jemand auf den Button drückt. Dabei ist zu beachten, daß `Variable` und `Value` in `definitions.h` entsprechend definiert werden, damit der Zugriff von C++ erfolgen kann.

Die Verwendung von defines in C++-Code verfolgt die Absicht, C++ und Tcl möglichst unabhängig voneinander zu halten und insbesondere keinen expliziten Tcl-Code (sei es eine Variable oder ein Widget-Name) im C++-Code außerhalb von `definitions.h` zu benutzen. Diese wurden durch die entsprechenden defines ersetzt. Kleinere Änderungen im Tcl haben daher nur Auswirkung auf die defines in `definitions.h`.

3.5 Die Ausgabe der Module

LooPo ist als X-Windows basiertes Tool geplant worden, weshalb zum Abschluß der ersten Entwicklungsphase sämtliche Ausgaben in die Shell verschwunden sein werden. Einige Module sollen aber dem Benutzer ihr Ergebnis direkt zur Verfügung stellen (das betrifft Dependence Test, Scheduler/Allocator und Target Generation), damit dieser über die Optionen auf das Ergebnis einwirken kann. Bei LooPo kommt es nicht nur auf das Ergebnis an, sondern auch darauf, wie es zustande kam, d.h. mit welchen Zwischenergebnissen. Dazu wird die `stdout`-Ausgabe der Module in eine Datei geschrieben, deren Inhalt nach Beendigung in ein Textfenster mit Scrollbalken geladen wird. Solche Ausgabefenster existieren bisher für den Dependence Test sowie Scheduler/Allocator. Diese Fenster können bei Bedarf entfernt werden.

4 Die Dialogs-Library

Neben dem eigentlichen Frontend, das durch die Klasse `MainWin` repräsentiert wird, wird den Modulen die Dialogs-Library zur Verfügung gestellt. Sie umfaßt die Klassen `errorbox` und `messagebox`, die jeweils eine Dialogbox mit einer Information oder Fehlermeldung für den Benutzer erscheinen lassen, sowie das schon früher erwähnte *session log*. Diese drei Klassen wurden in der Datei `Dialogs.cc` zusammengefaßt.

Abbildung 4: Eine Errorbox

Da die drei Klassen aber von `fifo` bzw. `TKWindow` abgeleitet sind, wurden alle Objektdateien, die nötig sind, um die Dialogboxes eigenständig lauffähig zu machen, in der Bibliothek `libdlg.a` vereinigt. Im einzelnen sind das:

```
Dialogs.o++ TKWindow.o++ error.o++ fifo.o++
```

Diese Bibliothek befindet sich im `general`-Verzeichnis des LooPo-Projektes und wird von den Modulen hinzugelinkt. Die notwendigen Klassendefinitionen befinden sich in `Dialogs.h`.

Eine Dialogbox umfaßt drei Methoden: den Konstruktor, den Destruktor und die `start`-Methode. Dem Konstruktor wird ein String übergeben, der die in der Box anzuzeigende Nachricht enthält. Danach kann die Box jederzeit am Bildschirm durch Aufruf der `start`-Methode angezeigt werden. Sie bleibt so lange stehen, bis der Benutzer das Fenster durch Drücken des OK-Buttons entfernt. Der Destruktor wird automatisch am Ende des Sichtbarkeitsbereiches der Klasse aufgerufen und löscht den Tcl-Interpreter.

Folgende Befehlssequenz zeigt eine Errorbox an, wie sie in Abb 4 zu sehen ist:

```
errorbox *Box = new errorbox("No schedule exists!");
Box->start();
```

Die Benutzung des *session logs* ist noch einfacher; nach der Instanziierung der Klasse wird mit der Methode `append` ein String über die FIFO an das Log geschickt. Ein Beispiel wäre:

```
sessionlog *Log = new sessionlog();
Log->append("Eine Meldung an das Log ... \n");
```

Eine Beschreibung der Funktionsweise wurde in Abschnitt 3.3 gegeben.

5 Mögliche Erweiterungen

Auch nach Abschluß des Programmierpraktikums ging die Entwicklung von LooPo weiter. Neben der Erweiterung um neue Module (z.B. Verarbeitung von WHILE-Schleifen in der Eingabesprache) ist noch folgendes geplant:

◊ Die einzelnen Module hängen zum Teil von der Ausgabe anderer Module ab; d.h. sie können nicht ablaufen, bevor diese Ausgabe zur Verfügung steht. Deshalb ist die Ausführungsreihenfolge der Module nicht beliebig, es kann nicht zu jedem Zeitpunkt jedes Modul aufgerufen werden. Daher ist geplant, die Buttons der Module zu „disablen“, die noch nicht aufgerufen werden können und diesen Status ständig zu aktualisieren.

◊ Die Module haben zum Teil viele Optionen, die vom Benutzer jeweils vor dem Modulaufruf gesetzt werden können. So kann ein Modul mehrfach hintereinander aufgerufen werden, um die Ausgabe mit jeweils anderen Optionen zu testen. Eine solche LooPo-Sitzung kann daher länger dauern. Daher soll in Zukunft die Möglichkeit geschaffen werden, den aktuellen Status von LooPo abzuspeichern, um später nach erneutem Programmstart an der gleichen Stelle fortfahren zu können.

◊ Die neuen Möglichkeiten von Tk Version 4.0 sollten benutzt werden, um die Benutzerfreundlichkeit zu erhöhen.

6 Zusammenfassung

Das Frontend von LooPo bildet ein abgeschlossenes C++ Programm, das mit Hilfe von Tcl-Skripten die graphische Oberfläche implementiert und den Aufruf der LooPo-Module steuert. Mit Ausnahme des graphik-orientierten Dispo2 ist allein im Frontend Code für die Oberfläche vorhanden.

Es sind im Frontend Fenster zur Einstellung von Optionen und Darstellung der Ausgabe der LooPo-Module vorhanden. Die auch zum Frontend gehörende Dialogs-Library stellt den Modulen Dialogboxen in Form von C++-Klassen zur Verfügung.

Das Frontend bildet also die Klammer um alle LooPo Module und regelt deren Aufruf. Da sich die Module in Zukunft weiter verändern werden, wird auch das Frontend erweitert werden müssen.

A Die TCL-Skripten hinter Buttons und Menüpunkten

Die im folgenden aufgeführten Codesegmente gehören zusammen mit dem C++ Code zur Programmlogik und unterscheiden sich daher von dem Rest des Tcl-Codes, der nur die grafische Oberfläche implementiert.

A.1 Das FILE-Menü

Set Projectname

```
set inputname [FSBox "Select Projectfile:"
               [format "%s%s" $LOOPODIR "/lib"] ]
if {$inputname != ""} {
 set projectname $inputname
 set projectnameText "Project: $projectname"
 set filename [string range $projectname
               [expr [string last / $projectname] +1 ]
               [string length $projectname] ]
 set outfilename [format "%s/%s.out" $TEMPDIR $filename ]
 set outfileText "Outputfile: $outfilename"
}
```

Set Outputfile

```
set outfilename [FSBox "Select Outputfile:" $LOOPODIR]
set outfileText "Outputfile: $outfilename"
```

Quit

```
destroy .
```

A.2 Options-Menu

Scheduler/Allocator ...

```
ShowWindow.moduleOpts
```

Target Generation ...

```
ShowWindow.targgenOpts
```

Target Output

```
ShowWindow.targetOpt
```

A.3 Help Menu

About

```
ShowWindow.top3
```

A.4 Buttons

Der Dispo1-Button

```
callback "dispo1_start"
```

Der Scan/Pars2-Button

```
set ScanParsDir $LOOPODIR
set ScanParsInputFilename $projectname
set ScanParsOutputFilename $outfilename
callback "scanpars2_start"
```

Der Dependence-Button

```
callback "dependences_start"
```

Der Scheduler-Button

```
callback "scheduler_start"
```

Der Allocator-Button

```
callback "allocator_start"
```

Der TargetGen-Button

```
callback "targgen_start"
```

Der TargetOut-Button

```
callback "targout_start"
```

Der Filter-Button

```
callback "filter_start"
```

Der Dispo2-Button

```
callback "dispo2_start"
```

Literatur

[Fea88] Paul Feautrier, Parametric Integer Programming, Operations Research Vol. 22, Nr. 3, 1988, S. 234–268.

[Wü95] Alexander Wüst, Dispo2, Programmierpaktikum im LooPo-Projekt, Universität Passau, 1995.

[Wi95] Christian Wieninger, Automatische Methoden zur Parallelisierung im Polyedermodell, Diplomarbeit, Passau 1995.

[We95] Sabine Wetzel, Automatic Code Generation in the Polyhedron Model, Diplomarbeit, Passau 1995.

[Kei95] Harald Keimer, Vergleich von Verfahren zur Datenabhängigkeitsanalyse , Diplomarbeit, Passau 1995.

[Ou94] John K. Ousterhout, Tcl and the TK Toolkit, Addison-Wesley, 1994.

[De94] Sven Delmas, XF – an interactive Interface Builder, Berlin, 1993.

[Ro85] Marc Rochkind, Advanced UNIX Programming, Prentice Hall, 1985.

[Gu95] Robert Günz, Scanner/Parser, Programmierpraktikum im LooPo-Projekt, 1995.

[Fa95] Peter Faber, Der Target Output von LooPo, Programmierpraktikum im LooPo-Projekt, 1995.