Woop v1.0

Martin Homik

Zusammenfassung

Aktuelle Workflow Management Systeme sind in der Lage, die Durchlaufzeit einzelner Prozesse schon in der Planungsphase zu optimieren. Sie vernachlässigen jedoch die Optimierung von Organisation und Infrastruktur. Dies gilt insbesondere für flexible Prozesse mit kontinuierlicher Versorgung.

Eine Optimierung von Organisation und Infrastruktur wird aufgrund von zunehmender Komplexität notwendig. Dies betrifft vor allem die Wirtschaft, in der Zeit und Kapital eine wesentliche Rolle spielt.

Das hier vorgestellte Produkt heißt Woop . Es löst komplexe Workflow Probleme durch einen constraintbasierten Ansatz. Woop basiert auf dem System *Mozart* und der Sprache *Oz*.

Inhaltsverzeichnis

1 Einleitung **5**

1.1 Motivation 5

1.2 Kriterien 5

1.2.1 Kriterien der Problemstellung 5

1.2.2 Kriterien der Problemlösung 6

1.3 Installation 6

2 Starten **7**

3 Frontend **9**

3.1 Ablaufgraph 9

3.2 Globale Information 9

3.3 Lokale Information 9

3.4 Interaktiver Auswahlbereich 11

4 Suche **13**

4.1 Strategien und Suchmaschinen 13

4.2 Heuristik 13

4.2.1 Einführung 14

4.2.2 Klassische Heuristiken 15

4.2.3 Woop spezifische Heuristiken 16

4.3 Optimierung 16

4.4 Heuristics Controler 17

4.5 Dialog: Suche 18

5 Menü **21**

5.1 File . . . 21

5.1.1 Verifikation . . . 21

5.2 Browse . . . 22

5.3 Optionen . . . 22

5.3.1 Propagierer . . . 22

6 Eingabe **25**

6.1 Mozart Grunddatenstrukturen . . . 25

6.2 Woop Datenstrukturen . . . 26

6.2.1 Initialisierung . . . 26

6.2.2 Aktivitäten . . . 26

6.2.3 Teilnehmertypen . . . 27

6.2.4 Constraints . . . 28

6.2.5 Verifikation . . . 28

6.3 Weitere Eingabedateien . . . 30

7 Benutzerdefinierte Nebenbedingungen **31**

7.1 Kosten . . . 31

7.2 Teilnehmertypen . . . 31

7.3 Aktivitäten . . . 32

7.4 Sonstige . . . 32

8 Woop Editor **33**

8.1 Zeichnen . . . 33

8.2 Verschieben . . . 34

8.3 Sessionverwaltung . . . 34

8.4 Graphverwaltung . . . 34

8.4.1 Separater und vermischter Ablauf . . . 35

8.4.2 Verifikation . . . 35

9 Tips **37**

Kapitel 1

Einleitung

1.1 Motivation

Aktuelle Workflow Management Systeme sind in der Lage, die Durchlaufzeit einzelner Prozesse schon in der Planungsphase zu optimieren. Sie vernachlässigen jedoch die Optimierung von Organisation und Infrastruktur. Dies gilt insbesondere für flexible Prozesse mit kontinuierlicher Versorgung.

Eine Optimierung von Organisation und Infrastruktur wird aufgrund von zunehmender Komplexität notwendig. Dies betrifft vor allem die Wirtschaft, in der Zeit und Kapital eine wesentliche Rolle spielt.

Das hier vorgestellte Produkt heißt Woop . Es löst komplexe Workflow Probleme durch einen constraintbasierten Ansatz. Woop basiert auf dem System *Mozart* und der Sprache *Oz*.

1.2 Kriterien

Es gilt zwischen zwei Gruppen von Kriterien zu unterscheiden: den Kriterien für die Problemstellung und den Kriterien für die Problemlösung.

1.2.1 Kriterien der Problemstellung

Die erste Gruppe umfaßt die Anforderungen an einen Prozess, um damit seine Wirtschaftlichkeit auszudrücken. Darunter befinden sich folgende Punkte:

Invest. Dem Invest liegt die Infrastruktur, die aus Ressourcentypen und deren Anzahl zusammengesetzt ist, zugrunde. Die Verwendung billiger Ressourcen senkt die Kosten. Eine Umverteilung von Arbeitsschritten ermöglicht unter Umständen einen Verzicht auf teure Ressourcen.

Taktzeit. Die Taktzeit spielt in einem kontinuierlich versorgten Prozeß eine wesentliche Rolle. Eine geschickte Organisation, d.h. eine Zuordnung von Arbeitsschritten zu Ressourcen, minimiert die Taktzeit und fördert die Ausbalancierung einzelner Ressourcen, um Wartezeiten zu vermeiden.

Redundanz. Der Ausfall einer Ressource bedingt eine Verlangsamung oder gar den Stopp des Prozesses. Um dennoch die geforderte Taktzeit zu erfüllen, bedarf es an Ersatzressourcen. Deren Anzahl ist proportional zur Stabilität einer Ressource und zum allgemein angenommenen Verlust.

Flexible Prozesse. Eine bestehende Organisation und Infrastruktur hat eine vorgegebene Menge von Prozessen zu erfüllen. Jeder Geschäftprozeß ist in Bezug auf die Faktoren Invest, Taktzeit und Redundanz optimiert. Somit ist jeder Geschäftsprozeß dieser Menge ohne weiteres innerhalb der Infrastruktur einsatzbereit.

Attributierte Ressourcen. Ressourcen haben Eigenschaften (Attribute). Sie unterscheiden sich durch ihre Fähigkeiten, ihre Stabilität, ihre Effizienz und ihre Kosten.

Für die reale Geschäftswelt bedeutet die Optimierung der oben genannten Punkte eine Verbesserung der Produktivität, sowie eine größere Zufriedenheit der Kunden.

1.2.2 Kriterien der Problemlösung

Zusätzlich zu den oben genannten Kriterien werden weitere Anforderungen an das Lösungsverfahren bzw. an die Lösung selbst gestellt.

Modellierung. Das Problem muß verständlich und so einfach wie möglich in ein mathematisches Modell übetragen werden.

Generische Lösung. Eine Lösung muß aus partiellen Informationen, die entweder aus dem Modell hervorgehen, oder durch den Anwender angegeben werden, generiert werden.

Korrektheit und Vollständigkeit. Sämtliche gefundenen Lösungen sind korrekt. Es darf keine Lösung übersehen werden.

Optimalität. Die optimale Lösung hat die geringsten Kosten. Existieren mehrere Lösungen mit geringsten Kosten, so ist nur die Lösung optimal, die gleichzeitig günstig ist und den kürzesten Takt hat. Die optimale Lösung muß schnell approximiert bzw. bestimmt werden.

Effizienz und Skalierbarkeit. Eine (optimale) Lösung wird schnell und unter geringem Ressourcenverbrauch gefunden. Des weiteren funktioniert das Verfahren für verschiedene Problemgrößen vorhersagbar.

Benutzerdefinierte Modellierung. Der Benutzer kann über eine Schnittstelle weitere partielle Informationen hinzufügen.

1.3 Installation

Dieses Paket ist auf dem aktuellen Stand vom 30.04.2002. Zum Betrieb von Woop v1.0 sind des weiteren folgende Pakete notwendig:

1. Emacs 20.4.1
2. Mozart 1.2.3

Bei Fragen oder Problemen kontaktieren Sie:

Martin Homik homik@ps.uni-sb.de

Kapitel 2

Starten

Woop kann auf zweierlei Arten gestartet werden. Entweder durch Anklicken des Woop Icons auf dem Desktop oder über ein Shell Kommando.

Beim letzteren können Parameter angegeben werden:

`conf`
: Pfad zur Konfigurationsdatei.

`mode`
: Arbeitsmodus (1 = WorkflowI (Standard), 2 = WorkflowII)

`lang`
: Sprachauswahl (eng=Englisch (Standard), ger=Deutsch)

`locale`
: Pfad zu einer anderen Localedatei. Dieser Punkt hat eine höhere Priorität als `lang`.

`help`
: Gibt eine Auskunft.

Beispielsweise startet

```
./woop --mode 2 --lang eng --conf configs/base2.init
```

das Programm im WorkflowII Modus in englischer Sprache und lädt zu Beginn das Problem `base2` ein.

Kapitel 3

Frontend

Nach dem Start von Woop erscheint das Hauptfenster, so wie in Abbildung 3.1 zu sehen ist. Es ist grob in zwei Bereiche aufgeteilt. Den größten Platz nimmt der Ablaufgraph ein. Rechts daneben werden technische Daten dargestellt. Dies sind von oben nach unten gesehen: globaler Informationsbereich, interaktiver Auswahlbereich und lokaler Informationsbereich.

3.1 Ablaufgraph

Der Ablaufgraph wird mit Hilfe des Werkzeugs *Woop Graph Editor* erstellt. Nach dem Laden einer Konfigurationsdatei, wird der Graph in seiner Rohfassung angezeigt. Sonderknoten haben die Hintergrundfarbe weiß; alle anderen die Farbe grau. Eine Kante zwischen zwei Knoten drückt eine Vorrangsrelation aus.

Wurde eine Lösung gefunden und aktiviert, so werden die Knoten entsprechend ihrer Zugehörigkeit zu einem Block gefärbt. Knoten mit derselben Farbe befinden sich im selben Block. Lediglich die Sonderknoten bleiben weiß.

3.2 Globale Information

Globale Informationen sind Angaben zur Konfiguration und zur gewählten Lösung. Zum ersteren gehört der Verlustfaktor und die maximale Taktvorgabe. Zum letzteren die berechneten Kosten, die Anzahl benötigter Workflow Teilnehmer und die effektive Taktzeit der Lösung.

3.3 Lokale Information

Lokale Informationen werden angezeigt, falls eine Lösung aktiviert ist und die Maus über einen Knoten geführt wurde. In den lokalen Informationen sind folgende Angaben enthalten: Abbildung einer Aktivität auf einen Block, Teilnehmertyp des Blocks, Anzahl benötigter Teilnehmer und die Taktzeit des Blocks. Die letzten beiden Punkte können je nach Wahl des Ablaufs verschieden sein.

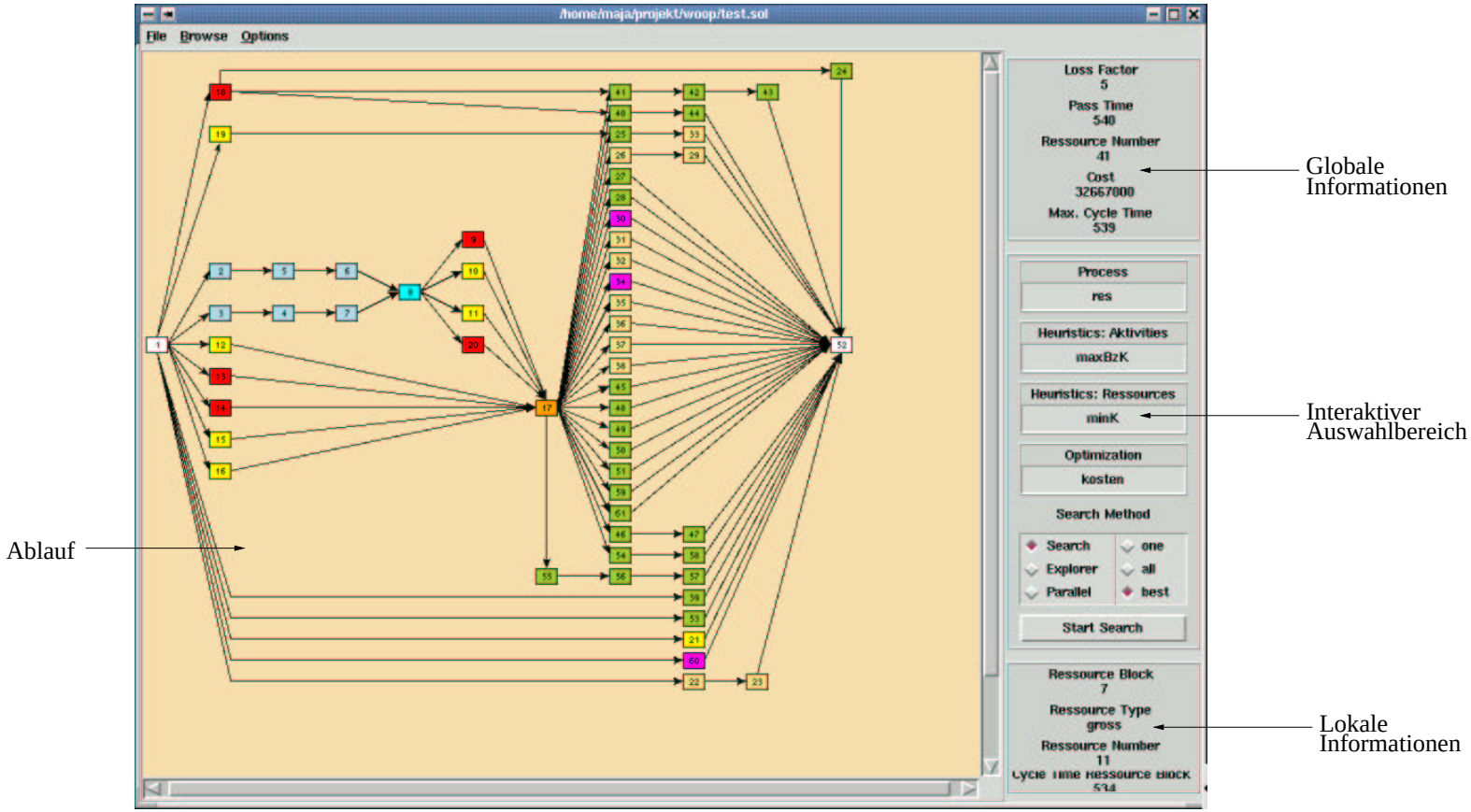

Abbildung 3.1: Woop Frontend

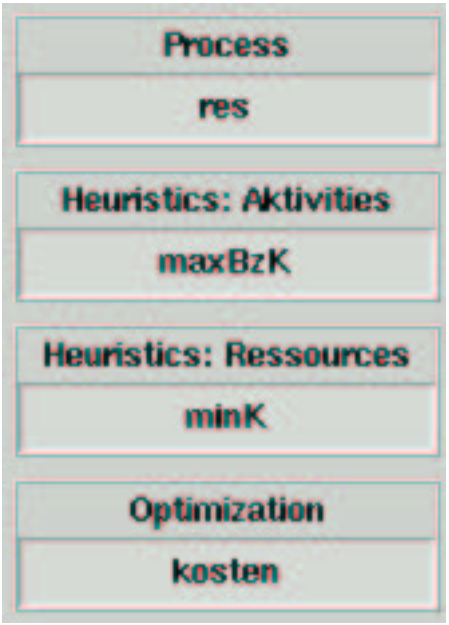

Abbildung 3.2: Heuristikauswahl

3.4 Interaktiver Auswahlbereich

Im interaktiven Auswahlbereich kann man mehrere Einstellungen vornehmen. Unter PROCESS wird ein spezieller Ablauf gewählt. Auf dem Bildschirm wird dazu der entsprechende Graph angezeigt. Liegt eine Lösung vor, so ist der Graph entsprechend gefärbt, andernfalls sind alle Knoten grau. Eine Einstellung `sep` bedeutet, daß die Vereinigung aller Abläufe angezeigt wird. Die Einstellung `mix` aktiviert den vermischten Datenfluß. Dieser, ob rein-vermischt oder kombiniert-vermischt, hängt von der Wahl unter *Options* $\rightarrow$ *Propagators* ab. Der kombiniert-vermischte Datenfluß unterscheidet sich vom rein-vermischten durch die Berechnung der Anzahl von Workflow Teilnehmern.

In den darauffolgenden Punkten *Heuristics: Activities* und *Heuristics: Ressources* werden die Heuristiken festgelegt. Hierbei heißt `none`, daß keine Suche stattfindet. Dies ist nur bei Benutzung des OzExplorer sinnvoll. Dadurch erfährt man, wieviele Informationen noch vor der Suche dem System bekannt sind, und wieviel Zeit für die Suche eingerechnet werden soll.

Sucht man nach der besten Lösung, wählt man die gewünschte Strategie unter OPTIMIZATION. Die Suchmaschine wird unter SEARCH METHOD eingestellt.

Sind die Einstellungen zufrieden stellend, so wird über START SEARCH der Suchvorgang begonnen.

Weitere Informationen zur Suche befinden sich im Kapitel 4.

Kapitel 4

Suche

4.1 Strategien und Suchmaschinen

Neben der Beschreibung des Workflow Problems mittels interner Strukturen und benutzerdefinierter Randbedingungen, nimmt die Suche einen sehr großen Stellenwert ein. Die Beschreibung des Problems alleine reicht bei komplexen Szenarien nicht aus, um schnelle Ergebnisse zu erhalten. Es bleiben noch viele Unbekannte übrig. Somit ist Suche notwendig. Während die Wahl einer Suchstrategie oder Suchmaschine einfach ausfällt, ist die Wahl einer guten Heuristik wesentlich schwieriger und hängt vom gestellten Problem ab.

Mozart bietet mehrere Suchmaschinen, von denen drei Woop verwendet:

Einfache Suche. Die Suchmaschine arbeitet im Hintergrund und meldet sich nur, wenn eine neue Lösung gefunden wurde, indem sie diese an Woop weiterreicht.

Explorer. Die Suchmaschine bietet eine komplette Visualisierung des Suchbaums. Durch diese zusätzliche Berechnung gestaltet sich die Suche etwas langsamer als bei der einfachen Suche.

Parallele Suche. Steht ein Netzwerk von Rechnern zur Verfügung, so bringt die parallele Suche die schnellsten Ergebnisse. Siehe hierzu auch [1].

Es existieren drei grundlegende Suchstrategien:

Strategie	Beschreibung
Eine	Es wird nur eine Lösung gesucht.
Alle	Es werden alle Lösungen gesucht.
Beste	Es wird die beste Lösung gesucht.

4.2 Heuristik

Es hat sich herausgestellt, daß je nach Anforderung mehrere zutreffende Heuristiken anwendbar sind. Aus diesem Grund bietet Woop neben der Wahl einer vorgegebenen Heuristik die Möglichkeit zur Erstellung einer eigenen Heuristik.

Abbildung 4.1: Suchmaschinen und -strategien

4.2.1 Einführung

Grundlage für die Zerlegung ist eine Menge von Elementen, in unserem Fall die Menge von Aktivitäten, die wir einem Block zuweisen wollen. Da die Zerlegung von Teilnehmertypen ähnlich ist, wird darauf nur gelegentlich eingegangen. Reicht also die Enschränkung der Belegungsmöglichkeiten mittels aller Problembeschreibungen nicht aus, so müssen die verbliebenen Zuweisungsmöglichkeiten probiert werden.

Die Zerlegung eines Knotens geschieht nach folgenden Kriterien:

Filter	Welche Aktivitäten kommen in Betracht?
Order	Welche Ordnung auf gefilterten Aktivitäten soll gelten?
Select	Welche Aktivität ist auszuwählen?
Value	Welchem Block wird die selektierte Aktivität zugewiesen?

Das Filtern schließt determinierte Aktivitäten aus (*undet*), d.h. über schon zugewiesene Aktivitäten kann keine weitere Fallunterscheidung mehr erfolgen.

Anschließend werden die Aktivitäten bezüglich eines Kriteriums sortiert. Es sind folgende Kriterien für eine neue Ordnung verfügbar:

Order	Beschreibung
naive	Keine neue Ordnung.
size	Nach aufsteigender Anzahl der Aktivität-Zuweisungsmöglichkeiten, d.h. zuerst Aktivitäten mit geringsten Zuweisungsmöglichkeiten.
min	Nach kleinstmöglichem Blockindex, d.h. erst die Aktivität, die an den frühesten Block zugewiesen werden kann.
max	Nach größtmöglichem Blockindex, d.h. erst die Aktivität, die an den spätesten Block zugewiesen werden kann.
nbSusps	Nach den meisten Verwicklungen in blockierte Berechnungen.
random	Zufällige Aktivität-Anordnung.
maxBz	Absteigende Ordnung nach Bearbeitungszeit.

Die Selektion wählt meist die erste durch Order zurückgegeben Aktivität (*id*). Lediglich bei Nutzung von *maxBz* muß auch Selektion mit *maxBz* belegt werden.

Die Blockzuweisung einer Aktivität bietet wiederum eine größere Auswahl. Hierzu gibt es folgende Auswahlmöglichkeiten:

Value	Beschreibung
min	Wähle den erstmöglichen Block.
max	Wähle den letztmöglichen Block.
mid	Wähle den mittleren Block.
splitMin	Halbiere die Blockauswahl und betrachte nur die untere Hälfte.
splitMax	Halbiere die Blockauswahl und betrachte nur die obere Hälfte.
billig	Wähle den Block mit geringsten Kosten, d.h. den Block, für den der billigste Teilnehmertyp eingesetzt wird.
gSlack	Wähle den Block, der das größte relative Zeitfenster inne hat, d.h. teile eine Aktivität dem Block zu, dessen Anzahl von Teilnehmern mit dieser neuen Aktivität nicht erhöht wird.
random	Wähle einen zufälligen Block.

4.2.2 Klassische Heuristiken

Im folgenden werden die bekanntesten klassischen Heuristiken vorgestellt. Sie sind in ihrer Effizienz unterschiedlich zu beurteilen. Je nach Aufgabe erfüllen sie zwar ihren Zweck, sind aber schlechter im Optimierungsverhalten.

Jede angegebene Heuristik ist durch die Wahl der oben genannten Kriterien (*Filter, Order, Select, Value*) charakterisierbar.

naive Dies ist die einfachste Form der Fallunterscheidung. Wir nehmen die erstbeste Aktivität und weisen ihr den frühestmöglichen Block zu. Da es hierbei an Wissenseinwirkung fehlt, ist diese Heuristik nicht für schnelle Suche und Optimierung geeignet. Die Kriterien sind: (undet, naive, id, min).

ff Diese Heuristik steht abkürzend für First Fail und eignet sich insbesondere zur Vermeidung von Flaschenhals Problemen. Hierbei wird erst die Aktivität mit kleinstmöglicher Blockauswahl (kleiner Freiheitsgrad) zur Fallunterscheidung herangezogen. Diese Heuristik ist im Allgemeinen sehr gut und für Woop geeignet. Die Kriterien sind: (undet, size, id, min).

split Ähnlich wie bei First Fail wird in dieser Heuristik die Aktivität mit kleinstmöglicher Blockauswahl zur Fallunterscheidung hinzugezogen. Allerdings wird nicht der frühestmögliche Block gewählt. Es wird stattdessen ein Wert m ermittelt, wobei m der mittlere Wert des Blockbereichs ist. In der Fallunterscheidung wird anschließend der Constraint $x \leq m$ zum Suchbaum hinzugefügt. Hier wird also die Aktivität auf einen kleineren Bereich reduziert. Das Ergebnis des Verfahrens ist unbefriedigend und sollte nicht angewendet werden. Die Kriterien sind: (undet, size, id, splitMin).

nbSusps In dieser Heuristik fällt die Wahl auf eine Aktivität, die in die meisten wartenden Berechnungen verwickelt ist. Ihr wird der frühestmögliche Block zugewiesen. Dieses Verfahren ist für das gegebene Problem nicht geeignet, kann jedoch in Einzelfällen interessante Resultate bringen. Die Kriterien sind: (undet, nbSusps, id, min).

4.2.3 Woop spezifische Heuristiken

Neben den allgemeinen Verfahren wurden weitere speziell auf das Workflow Problem zugeschnittene Heuristiken entwickelt.

maxBzK Diese Heuristik bietet eine sehr gute Optimierung. Es wird eine Aktivität mit längster Bearbeitungszeit gewählt und dem Block mit dem billigstmöglichen Teilnehmertyp zugewiesen. Der Grund für die Wahl einer solchen Aktivität liegt darin, daß die Anzahl der Teilnehmer sich u.a. aus der Gesamtbearbeitungszeit berechnet. Aktivitäten mit langer Bearbeitungszeit nehmen einen großen Teil der Gesamtbearbeitungszeit ein. Sie approximieren die notwendige Anzahl der Teilnehmer für diesen Block am besten. Gleichzeitig approximieren sie die Teilnehmerkosten. Die Kriterien sind: (undet, maxBz, bz, minKAkt).

maxBzS Ein anderer Ansatz besteht darin, eine Aktivität einem Block zuzuweisen, dessen Anzahl an Teilnehmern sich durch diese zusätzliche Aktivität nicht ändert. Entsprechend ändert sich auch nicht der Teilnehmerinvest. Hier kommt es darauf an, wie groß die einzelnen relativen Zeitfenster der Blöcke sind. In der Regel sind diese Fenster zu klein. Somit fallen die Ergebnisse kostenintensiv aus. Dennoch können Szenarien mit großen Zeitfenstern erstellt werden. In diesem Fall kann die Strategie nützlich sein. Die Kriterien sind: (undet, maxBz, bz, maxS).

billig2 Im Gegensatz zu *maxBzK* werden die Bearbeitungszeiten vernachläßigt. Stattdessen wird versucht, die Flaschenhalsproblematik mit günstigen Teilnehmerkosten zu vereinen, d.h. es wird die Aktivität mit kleinstem Freiheitsgrad gewählt und dem billigstmöglichen Block zugewiesen. Obwohl die Optimierung zufriedenstellend ist, ist die *maxBzK* Heuristik wesentlich zielgerichteter. Die Kriterien sind: (undet, min, id, minKAkt).

random Diese Heuristik wählt eine zufällige Aktivität und weist diese Aktivität einem zufälligen Block zu. Die resultierenden Ergebnisse sind unterschiedlich zu bewerten. Die Kriterien sind: (undet, random, id, random).

maxAKTs Diese Heuristik ist speziell für Teilnehmertypen ausgelegt. Da die Liste der Teilnehmertypen nach Kosten sortiert vorliegt, wird durch den Zugriff auf das minimale Element, der billigstmögliche Teilnehmertyp für diesen Block gewählt. Die hier aufgeführte Heuristik richtet sich jedoch nach der Anzahl der vom Teilnehmertypen bereitgestellten Aktivitäten. Durch geringe Einschränkungen ist der Freiheitsgrad der Aktivität Zuweisung hoch, da sie in fast allen Blöcken einen Platz finden. Der Nachteil ist, daß derartige allgemeine Teilnehmertypen teuer sind. Der große Vorteil dagegen ist, daß schnell eine Lösung gefunden wird. Die Kriterien sind: (undet, size, id, maxAKTs).

4.3 Optimierung

Woop bietet fünf verschieden Optimierungskriterien an:

k Es werden nur die Kosten optimiert.

kAT Es werden die Kosten und die effektive Taktzeit optimiert.

- Kosten der neuen Lösung sind kleiner
- Kosten sind gleich, aber die neue Lösung hat eine bessere effektive Taktzeit.

Formalisiert sieht es wie folgt aus:

$$kosten(Alt) + takt(Alt) > kosten(Neu) + takt(Neu)$$

Diese Optimierung ist korrekt solange gilt: Ist $x = Stelligkeit(maxT)$, mit maxT gleich maximale Taktzeit, so sind

$$Gesamtkosten \mod 10^x = 0$$

Beispiel: Angenommen Lösung 1 hat Kosten 100000 und einen effektiven Takt von 900 und Lösung zwei hat Kosten 100001 und einen effektiven Takt von 500. Obwohl letztere Lösung teurer ist, ist sie bezüglich des Optimierungskriteriums günstiger. Die nächstgültigen Kosten sind daher entweder 101000 oder 99000, falls der effektive Takt immer kleiner 1000 ist.
Der Vorteil dieser Optimierung ist eine schnelle Approximation.

kDT Im Gegensatz zu *kAT* funktioniert diese Optimierung ohne Einschränkung. Insgesamt ist sie jedoch langsamer, da der Suchbaum im geringeren Maße beschnitten wird.

takt Es wird nur der effektive Takt minimiert. Mit der Minimierung des Taktes geht eine Ausbalancierung der Auslastung der Blöcke einher. Da hier die Kosten nicht betrachtet werden, ändern sie sich fortwährend.

takt2 Auch hier wird der Takt minimiert, jedoch zu fixen Kosten. Diese müssen entweder genau vorgegeben werden, oder es werden die Kosten der ersten Lösung übernommen.

Eine Empfehlung besteht darin, erst hinsichtlich der Kosten (*k*) zu optimieren. Wird nach einer ausreichenden Zeit keine bessere Lösung gefunden, so soll eine weitere Suche mit dem Optimierungskriterium *takt2* durchgeführt werden, wobei die fixen Kosten den minimalen Kosten der ersten Suche entsprechen.

4.4 Heuristics Controler

Zur Herstellung einer eigenen Heuristik eignet sich der *Heuristik Controler*. Des weiteren gibt er Auskunft über die Kriterien der einzelnen Heuristiken. Betrachte hierzu Bild 4.2.

Der Heuristik Controler wird über den Menüpunkt *Options* → *Heuristics* für jeweils Aktivitäten oder Teilnehmertypen aufgerufen. Im oberen Bereich kann über ein Popup Menu eine Heuristik gewählt werden, deren Kriterien im unteren Bereich dargestellt sind.

Eine neue Heuristik wird über den Menüpunkt *File* → *New Generic* erstellt. Im neuen Fenster muß ein eindeutiger Name für die Heuristik eingegeben werden. Anschließend lassen sich die Kriterien bequem über Popup Menus einstellen. Durch `Ok` kehrt man wieder zurück.

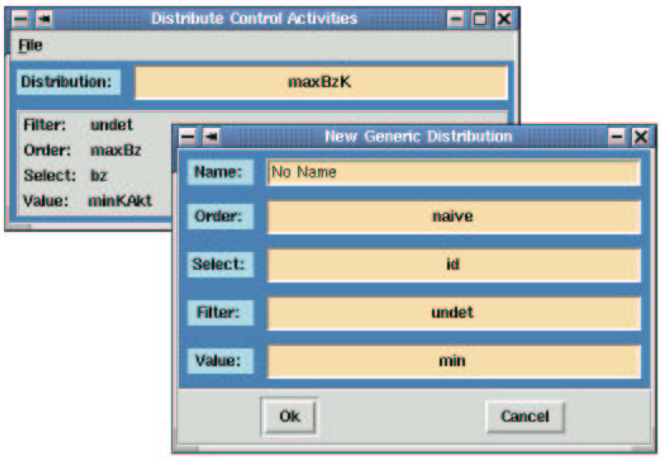

Abbildung 4.2: Heuristik Controler

Will man die neu erstellte Heuristik nutzen so kommt man über den Menüpunkt *File* → *Use* wieder zurück zum Woop Hauptschirm. Anschließend findet man die neue Heuristik in der entsprechenden Auswahl.

Nach jedem sorgfältigen Verlassen von Woop über *File* → *Exit* speichert das Programm alle Heuristiken in der Datei `.wooprc`.

4.5 Dialog: Suche

Sind alle Parameter eingegeben und wurde der Button START SEARCH gedrückt, so erscheint auf dem Bildschirm ein Such-Dialog (siehe Abbildung 4.3).

Im oberen Bereich werden die eingegangenen Lösungen angezeigt. Darunter geben zwei Felder Auskunft über den Status und die Dauer der aktuellen Suche. Der Status `ready` zeigt die Betriebsbereitschaft an. Wird eine Suche gestartet, so wechselt der Status nach `running`. Wird sie unterbrochen oder beendet, so gilt der Status `stop`.

Im unteren Bereich befinden sich vier Buttons. Eine Suche kann mit START SEARCH begonnen werden. Sie läßt sich mit STOP SEARCH unterbrechen und mit RESTART SEARCH wieder aufnehmen. Der Button CLOSE beendet den Dialog.

Lediglich die Suche mit einer Search Suchmaschine kann unterbrochen werden. Der OzExplorer kann hier genausowenig kontrolliert werden wie die Parallele Suche. Im Gegensatz zum OzExplorer verschickt die Parallele Suchmaschine Ergebnisse an diesen Dialog. Um auch Lösungen des OzExplorer in Woop aufzunehmen, muß man Lösungen (grüne Rauten) anklicken. Alternativ kann eine Lösung textuell angezeigt werden. Dazu muß man im Menü *Information* → *Action* → *Inspect* aktivieren.

Die Lösungen im Dialog geben erste Informationen an. Hierzu gehören die Gesamtkosten, die effektive Taktzeit, die Anzahl benötigter Blöcke, die Anzahl erforderlicher Teilnehmer und die Dauer der Suche.

Im Dialog eingegangene Lösungen lassen sich grafisch in Woop anzeigen, indem sie mit der Maus angeklickt werden. Die Knoten der Abläufe werden gefärbt. Globale und Lo-

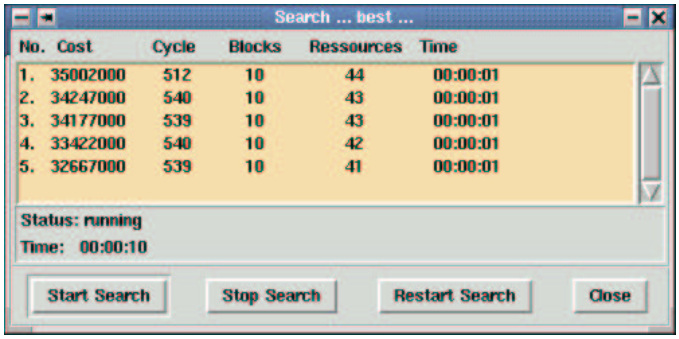

Abbildung 4.3: Such-Dialog

kale Informationen werden berechnet und angezeigt. Die nun gewählten Lösungen lassen sich weiterbearbeiten, beispielsweise speichern. Alle Lösungen gehen verloren, wenn der Dialog verlassen wird. Gleichzeitig erscheint der Graph wieder in Graudarstellung.

Kapitel 5

Menü

5.1 File

Der Menüpunkt *File* → *New* bewirkt, daß eine aktuell angezeigte Lösung wieder gelöscht wird. Dies ist insbesondere nach der Verifikation zu empfehlen.

Eine Lösung wird über *File* → *Save Solution* gesichert, während sie mit *Load Solution* zusammen mit der zugehörigen Konfiguration nach Woop geladen wird.

Eine neue Konfiguration wird über den Menüpunkt *File* → *Load Configuration* gewählt.

Für die parallele Suche ist standardmäßig die Datei `hosts/base.hosts` eingestellt. Alternativ kann über *File* → *Load Hosts* eine andere Datei bestimmt werden.

Mit Hilfe des OzPanel, *File* → *Open Panel*, erhält man Informationen über das Laufzeitverhalten.

Lösungen können nach Ascii, *File* → *Export* → *Ascii*, exportiert werden.

5.1.1 Verifikation

Woop ist in der Lage, per Hand errechnete Lösungen zu verifizieren und anzuzeigen. Voraussetzung hierfür ist das Vorhandensein der dafür benötigten Konfiguration. Die Verifikation testet auf folgende Fehlerquellen:

Gültige Teilnehmertypnamen. Alle Typnamen der Lösung müssen von der Konfigurationsdatei bereitgestellt werden. Ansonsten ist dieser Teilnehmertyp unbekannt.

Gültige Partitionen. Alle Aktivitäten müssen so auf die Blöcke verteilt sein, daß der zugehörige Teilnehmertyp alle bereitstellt.

Gültige Abläufe. Alle Aktivitäten müssen so auf die Blöcke verteilt sein, daß keine Vorrangsrelation verletzt wird.

Die Lösungen sind in Oz/Mozart Datenstruktur anzugeben (siehe Kapitel 6).

```
root(gKosten: <Integer>
     nBlocks: <Integer>
     maxT:    <Integer>
     akts:    <Tupel>
     wfts:    <Tupel>)
```

Das Feld unter dem Feature `akts` ist ein Tupel von Integern. Die Feature des Tupels sind die Aktivität-Namen, während die Felder auf die Blöcke verweisen, zu der die Aktivität gehört.

Die Teilnehmertypen werden in dem Tupel `wfts` mit ihrem Namen als Atome benannt.

5.2 Browse

Die Ansicht bietet die Möglichkeit, einzelne Angaben zu betrachten. Es dient zur Einsicht in die internen Vorgänge, beispielsweise welche zusätzlichen Informationen generiert werden. Des weiteren sind Fehler besser nachvollziehbar.

Sehr nützlich erweist sich der Punkt *User-Defined Constraints*. Darin sind zusätzlich zu den eigenen Constraints weitere, vorberechnete Constraints enthalten. Diese sind an der Vorsilbe `pre_` zu erkennen. Vorberechnete Constraints gehen nicht in die Berechnungen für vermischte Fertigung ein.

5.3 Optionen

Woop bietet mehrere Einstellungsmöglichkeiten.

Unter *Options* → *Heuristics* → *Activities* und *Options* → *Heuristics* → *Ressources* lassen sich neue Heuristiken erstellen. Siehe hierzu Kapitel 4.4. Neu erstellte Heuristiken werden automatisch in der Datei `.wooprc` gespeichert.

Ist eine Konfiguration gewählt, so kann man über *Options* → *Parameter* einige Angaben zur Suche variieren. Hierzu gehören der maximale Takt, der Verlustfaktor, die min./max. Anzahl von Teilnehmern, die min./max. Anzahl von Blöcken, die Anzahl der Schwesterwerkzeuge und eine obere Grenze für Gesamtkosten. Neu eingestellte Werte lassen sich mit dem Ok Button dauerhaft ins Programm übernehmen. Soll der Dialog während der Suche erhalten bleiben, so reicht die Wahl USE. Mit RESET erhält man wieder die vorherigen Werte.

Über den Menüpunkt *Options* → *Logging* kann man bestimmen, ob eine E-Mail Benachrichtigung bzw. eine Datei Protokollierung erwünscht ist. Beim Ersteren muß die E-Mail Adresse und beim Letzteren der Pfad angegeben werden. Diese Angaben werden beim Verlassen des Programms automatisch in der Datei `.wooprc` gespeichert.

5.3.1 Propagierer

Woop generiert aus seinen Parametern die für die Suche notwendigen Constraints und übergibt sie an die Suchmaschine. Alle Constraints sind in Gruppen organisiert, die über Schalter ein-, aus- bzw. umgeschaltet werden. Die Constraintgruppen können so eingestellt

werden, daß sie die Kriterien von WorkflowI oder WorkflowII erfüllen. Darüber hinaus ist ein benutzerdefinierter Modus vorhanden, bei dem die Gruppen beliebig miteinander kombinierbar sind. Im Folgenden sind die Gruppen nach der Thematik aufgeführt:

Allgemein Wie schon oben erwähnt, sind hier drei Modi einstellbar: WorkflowI, WorkflowII und Benutzerdefiniert. Zusätzlich kann zwischen separatem und vermischtem Datenfluß gewählt werden. Beim letzteren muß man sich zwischen rein-vermischtem und kombiniert-vermischten Datenfluß entscheiden. Der vermischte Datenfluß ist erst aktiv, wenn unter *Abläufe* `mixed` gewählt ist.

Globale/Lokale Zeit. Wähle zwischen lokaler und globaler Zeit. Einer dieser Punkte ist immer eingeschaltet.

Benutze Werkzeuge. Grundsätzlich stellt sich für den benutzerdefinierten Modus die Frage, ob Werkzeuge ebenfalls in die Berechnung eingehen.

Benutzerdefinierte Constraints In diesem Punkt wird die Einbindung von benutzerdefinierten Nebenbedingungen ausgeblendet.

Slack Algorithmus Die Suche kann durch einen sogenannten Slack Algorithmus unterstützt werden. Da die flexible Prozesse komplex sind, ist der aufgebaute Suchbaum sehr tief. Der Slack Algorithmus verringert die Tiefe, indem er schaut, ab welcher Tiefe es sich nicht mehr lohnt weitere Aktivitäten zuzuteilen, weil die Kosten sich nicht mehr ändern. Die bisherigen Probleme haben leider keine deutliche Reduktion erbracht, so daß dieser Punkt standardmäßig ausgeschaltet bleibt. Dieser Algorithmus funktioniert nur für globale Zeit. Außerdem sollte beachtet werden, daß Lösungen mit geringerer Taktzeit übersehen werden können.

Obige Bedenken schließen jedoch nicht aus, daß sinnvolle Szenarien existieren, die dennoch den Slack Algorithmus effizient nutzen könnten. Hierzu ist die Voraussetzung notwendig, daß Teilnehmertypen größere absolute Zeitfenster aufweisen. Absolute Zeitfenster sind Maß dafür, wieviel Bearbeitungszeit der Teilnehmer noch aufnehmen kann, bevor ein weiterer benötigt wird.

Kapitel 6

Eingabe

6.1 Mozart Grunddatenstrukturen

In diesem Abschnitt stellen wir diejenigen Grunddatenstrukturen von Mozart vor, die auch in den Eingabedateien vom Woop vorkommen.

In allen Eingabedateien erlauben wir nur *Integer* als Zahlen. Des weiteren erlauben wir die boolschen Werte `true` und `false`.

Eine wichtige und häufig gebrauchte Basisdatenstruktur sind *Atome*. Ein Atom ist eine symbolische Struktur, das aus einer Sequenz von alphanumerischen Zeichen besteht. Sie beginnt entweder mit einem klein geschriebenen Buchstaben oder mit beliebigen in Anführungszeichen eingebetteten Zeichen. Beispiele sind:

```
a   foo   '='   ':='   'OZ 3.0'   'Hello World'
```

Records sind zusammengesetzte Datenstrukturen. Sie bestehen aus einem *Label* und einer festen Anzahl von Komponenten oder Argumenten. Folgender Record

```
tree(key: I value: Y left: LT right: RT)
```

besteht aus vier Argumenten und dem Label `tree`. Jedes Argument besteht aus einem Paar *Feature:Field*. Die Features in unserem Beispiel sind `key, value, left` und `right`. Korrespondierend dazu sind die Felder `I, Y, LT` und `RT`.

Durch Auslassen der Features eines Records wird die Struktur zu einem *Tupel* reduziert. Folgendes Tupel hat dasselbe Label und dieselben Features wie unser oben genanntes Record:

```
tree(I Y LT RT)
```

Es handelt sich hierbei schlicht um eine Vereinfachung der Notation für ein Record:

```
tree(1:I 2:Y 3:LT 4:RT)
```

Eine weitere wichtige Klasse von Datenstrukturen in Mozart sind *Listen*. Listen sind entweder das Atom `nil` oder ein Tupel, bestehend aus dem Infixoperator `|` und zwei Argumenten, die den Kopf und den Rest der Liste repräsentieren. Hierzu ein Beispiel:

```
1|2|3|nil
```

Wir vereinfachen die Notation durch eine sogenannte *geschlossene Liste*, die wie folgt aussieht:

```
[1 2 3]
```

Woop benutzen wir lediglich geschlossene Listen.

6.2 Woop Datenstrukturen

In diesem Kapitel stellen wir die Eingabedateien vor, in denen die Informationen zur Berechnung enthalten sind.

6.2.1 Initialisierung

Da die Eingabe aus vielen Informationen besteht, wurden diese in mehrere Dateien aufgeteilt. Dabei bilden die Dateien mit der Endung `.init` die übergeordneten Dateien. Hierdrin befinden sich Angaben, nämlich Pfade, zu den weiteren Dateien.

Die Intialisierungsdatei ist ein Record mit dem Label `init` und den Features `akts, wfts, constraints` und `graph`. Die Felder sind in Apostrophen eingebettete Atome. Sie enthalten den Pfad vom Hauptprogramm zu den Konfigurationsdateien. Hier ein Beispiel:

```
init(
   akts:        'configs/base.akt'
   wfts:        'configs/base.wft'
   graph:       'configs/base.graph'
   constraints: 'configs/base.constraints'
)
```

Mit dem Woop Editor werden Abläufe erstellt und als Graph Dateien exportiert. Die restlichen drei Dateitypen werden in den nachfolgenden Kapiteln vorgestellt.

6.2.2 Aktivitäten

Dieser Dateityp hat die Endung `.akt`. Darin enthalten ist ein Record mit dem Label `conf` und dem Feature `akts`. Das Feld unter dem Feature ist ein Tupel bestehend aus einem Label `akts` und Records mit Angaben zu jeder Aktivität.

Dieser Aktivität-Record besteht wiederum aus einem Label `akt` und den Features `name, time` und codetools. Sowohl im Feld von `name` als auch von `time` werden nur Integer

akzeptiert, während `tools` ein Tupel (Label `t`) von Integern enthält und damit die notwendigen Werkzeuge beschreibt. Die Anzahl der Werkzeuge muß nicht explizit angegeben werden, sie wird automatisch berechnet. Bei der hier angegebenen Zeit handelt es sich um die *globale* Zeit.

```
conf(akts:
       akts(akt(name: <Integer>
                time: <Integer>
                tool: <Tupel>)
            )
    )
```

6.2.3 Teilnehmertypen

Dieser Dateityp hat die Endung `.wft`. Darin enthalten ist ein Record mit dem Label `conf` und dem Feature `wfts`. Das Feld unter dem Feature ist eine Liste von Records mit Angaben zu jedem Teilnehmertyp.

Dieser Teilnehmertyp Record besteht wiederum aus einem Label `wft` und den Features `name, akts, rz, st, kost, mag, lokal` und `typ`. Es gilt folgende Syntax:

```
conf(wfts:
       [wft(akts:     <Tupel>
            name:     <Atom>
            lokal:    <Tupel>
            rz:       <Integer>
            st:       <Integer>
            kost:     <Integer>
            mag:      <Integer>
            typ:      <Atom>
            )
       ]
    )
```

Die unter dem Feature `lokal` angegebenen Aktivitäten stellt der Teilnehmertyp prinzipiell zur Verfügung, wobei die angegebene Zeit nun eine *lokale* Zeit ist.

```
b(<Integer1>: <Integer2> ...)
```

Hier steht `Integer1` für den Namen der Aktivität und `Integer2` für die lokale Zeit.

Des weiteren enthält das Feature `akts` die vom Teilnehmertyp prinzipiell zur Verfügung gestellten Aktivitäten. Diese werden in einem Tupel als Einzelwerte oder als Bereich (`von#bis`) angegeben. Beispiel: `akts(15 17 20#24)`.

Die Features `rz, st` und `kost` werden mit Integern belegt. `rz` steht für die Spannzeit, `st` für Stabilität und `kost` für die Teilnehmerkosten. Hierbei sollte man beachten, daß `st` eine Prozentangabe ist und bei falschen Werten falsche Berechnungen anstellt.

Des weiteren werden Teilnehmertypen durch Werkzeugangaben ergänzt. Es existiert ein Werkzeugmagazin, dessen Plätze beschränkt sind (`mag`). Während der Suche werden u.a.

Aktivitäten einzelnen Blöcken zugeteilt. Jede Aktivität gibt die für sie notwendigen Werkzeuge vor. Überschreitet die Anzahl aller Werkzeuge die Anzahl der Magazinplätze, so können nicht alle Aktivitäten aufgenommen werden. Demzufolge kann ein Teilnehmer nur dann viele Aktivitäten aufnehmen, wenn diese Aktivitäten sich in ihren Werkzeugen gleichen.

6.2.4 Constraints

Dieser Dateityp hat die Endung `.constraints`. Darin enthalten ist ein Record mit dem Label `conf` und dem Feature `constr`. Das Feld unter dem Feature ist ein Record bestehend aus einem Label `constr` und und den Features `uz, pStart, pEnd, maxT, v` und `user`.

```
config(constr:
         constr(uz:     <Integer>
                maxT:   <Integer>
                v:      <Integer>
                pStart: <Integer>
                pEnd:   <Integer>
                nMin:   <Integer>
                nMax:   <Integer>
                numT:   <Integer>
                user:   <Liste>
               )
       )
```

In `uz` steht die globale Übergangszeit. Sie ist für alle Aktivität Paare gleich.

Für die Berechnung sind die Konstanten `maxT` die maximale Taktzeit, und `v` der Verlust notwendig.

Mit `pStart` und `pEnd` wird festgelegt, über wieviel Blöcke sich die Suche erstrecken soll und `nMin` bzw. `nMax` bestimmt die minimale bzw. maximale Anzahl geforderter Teilnehmer.

Der Punkt `numT` bestimmt die Anzahl der Schwesterwergzeuge.

Der wichtigste Punkt ist `user`. Hier stehen die benutzerdefinierten Constraints, gesammelt in einer Liste. Sie schränken das Problem weiter ein.

6.2.5 Verifikation

Bevor die Arbeit beginnt, müssen die Eingabedaten auf ihre Richtigkeit überprüft werden. Die logische Zusammensetzung des Graphen verifiziert der Woop Editor. Die weiteren textuellen Eingaben betrachtet Woop selbst. Die Fehlermeldungen haben folgende Syntax:

```
Feature [Position] File
```

Dabei besagt `Feature`, an welcher Stelle in der Datei `File` der Fehler aufgetreten ist. Handelt es sich um einen Aufzählungstypen (Record, Tupel oder Liste) so ist die Position zusätzlich angegeben.

Unbekannte Fehlermeldung. Ein unbekannter Fehler ist eingetreten.

Folgende Punkte sind nicht vorhanden: `(rz st) 3 (*.wft)` Im Input der Teilnehmerkonfigurationsdatei fehlen die Angaben zur Rüstzeit und zur Stabilität bezüglich des dritten Teilnehmertyps.

Datenstruktur ist keine Record: `akts 3 (*.wft)` In der Teilnehmerkonfigurationsdatei wurde bei der Beschreibung der Aktivität Zeiten des dritten Teilnehmertyps eine vom Record abweichende Datenstruktur gewählt.

Datenstruktur ist kein Integer: `maxT (*.constraints)` Im Input der Constraintkonfigurationsdatei wurde bei der Beschreibung der maximalen Taktzeit eine vom Integer abweichende Datenstruktur benutzt.

Datenstruktur ist keine Liste: `user (*.constraints)` In der Constraintskonfigurationsdatei wurde bei der Beschreibung der benutzerdefinierten Nebenbedingungen eine von einer Liste abweichende Datenstruktur benutzt.

Datenstruktur ist kein Tupel: `akts 3 (*.wft)` In der Teilnehmerkonfigurationsdatei wurde bei der Beschreibung der Aktivitäten des dritten Teilnehmertyps eine vom Tupel abweichende Datenstruktur gewählt.

Datenstruktur ist kein Atom: `name 3 (*.wft)` In der Teilnehmerkonfigurationsdatei wurde bei der Beschreibung des Namens des dritten Teilnehmertyps eine vom Atom abweichende Datenstruktur gewählt.

Gewählter Wert ist zu klein: `maxT (*.constraints)` In der Constraintskonfigurationsdatei wurde bei der Beschreibung des maximalen Takts ein zu kleiner Wert gewählt.

Gewählter Wert ist zu groß: `v (*.constraints)` Im Input der Constraintskonfigurationsdatei wurde bei der Beschreibung des Verlustfaktors ein zu großer Wert eingesetzt.

Falsche Aktivität Bereichsangabe: `akts 3 (*.wft)` Im Input der Teilnehmerkonfigurationsdatei wurde bei der Beschreibung der Aktivitäten des dritten Teilnehmertyps eine falsche Angabe gemacht. Wahrscheinlich ist der Bereich falsch angegeben.

Zu viele Sonderteilnehmertypen: `(*.wft)` Im Input der Teilnehmerkonfigurationsdatei wurden mehr als zwei Sondertypen angegeben. Erlaubt sind lediglich zwei: eine für den Start und eine für das Ende.

Aktivitäten Teilnehmertypangaben stimmen nicht überein: `lokal 3 (*.wft)` Eingaben von Daten in der Teilnehmerkonfigurationsdatei sind nicht korrekt. Die Beschreibung der Aktivitäten des dritten Teilnehmertyps stimmt nicht mit den verwendeten Aktivitäten der lokalen Aktivität-Zeiten überein.

pEnd ist kleiner als pStart: `(*.constraints)` In der Constraintskonfigurationsdatei wurde ein falscher Suchbereich angegeben, d.h, pStart ist kleiner als pEnd. Keine Iteration über Anzahl Blöcke findet statt.

Widerspruch! nMax ist kleiner als nMin: `(*.constraints)` Im Input der Datei für Constraintkonfigurationen wurden die Schranken für die minimale und maximale Anzahl der Teilnehmer falsch gesetzt. d.h. die untere Schranke ist größer als die obere.

6.3 Weitere Eingabedateien

Im Unterverzeichnis `hosts` sollte eine Datei `base.host` vorhanden sein. Ihr Inhalt ist für die parallele Suche wichtig. Darin enthalten sind die Hostnamen, die an der Suche beteiligt werden sollen. Darüber hinaus wird die Anzahl der Prozesse angegeben und über welche Shell kommuniziert werden soll. Mehrere Prozesse sind auf Multiprozessor System vorteilhaft. Eine Beispieldatei sieht wie folgt aus:

```
[
 host(name: fry      tasks: 1 fork: ssh)
 host(name: amy      tasks: 1 fork: ssh)
 host(name: bender   tasks: 1 fork: ssh)
 host(name: zoidberg tasks: 1 fork: ssh)
 host(name: a        tasks: 1 fork: ssh)
 host(name: pazo     tasks: 1 fork: ssh)
 host(name: grieg    tasks: 1 fork: ssh)
]
```

Des weiteren legt Woop eine `.wooprc` Datei an, in der globale Angaben nach dem Beenden des Programms gespeichert werden. Sie kann nicht textuell verändert werden.

Kapitel 7

Benutzerdefinierte Nebenbedingungen

Benutzerdefinierte Randbedingungen haben sich als sehr nützlich erwiesen und geben dem Anwender die Möglichkeit, ein gegebenes Problem zu modellieren und näher zu spezifizieren. Je mehr über ein Problem bekannt ist, umso eingeschränkter ist der Suchraum.

Wir gliedern die benutzerdefinierten Randbedingungen in vier Gruppen: Kosten, Teilnehmertypen, Aktivitäten und Sonstige.

7.1 Kosten

gKosten(<**Integer**> **)** Die in `Integer` angegebenen Kosten darf eine Lösung nicht überschreiten.

mingKosten(<**Integer**> **)** Die in `Integer` angegebenen Kosten muß eine Lösung überschreiten.

bKosten(<**Integer**> **)** Die in `Integer` angegebenen Kosten beschreiben den notwendigen Invest pro Block.

tKosten(<**Integer**> **)** Die in `Integer` angegebenen Kosten beschreiben den notwendigen Invest pro Werkzeug.

7.2 Teilnehmertypen

isSonder(block: <**Integer**> **)** Der in `block` angegebene Block ist ein Sonderblock. Der Teilnehmertyp ist unbekannt.

notSonder(block: <**Integer**> **)** Der in `block` angegebene Block ist kein Sonderblock. Der Teilnehmertyp ist unbekannt.

nWftMax(<**Integer**> **)** Die in `Integer` angegebene Gesamtanzahl von Teilnehmern darf eine Lösung nicht überschreiten. Sonderteilnehmer werden nicht mitgerechnet.

nWftMin(<**Integer**> **)** Die in `Integer` angegebene Gesamtanzahl von Teilnehmern darf eine Lösung nicht unterschreiten. Sonderteilnehmer werden nicht mitgerechnet.

nMinAllBlock(<Integer>) Dieser Constraint besagt, daß jeder Block aus mindestens `Integer` Teilnehmern besteht. Ausgeschlossen sind Blöcke mit Sonderteilnehmern.

nMaxAllBlock(<Integer>) Dieser Constraint besagt, daß jeder Block aus höchstens `Integer` Teilnehmern besteht. Ausgeschlossen sind Blöcke mit Sonderteilnehmern.

nMinWftType(name: <Atom> min: <Integer>) Es sollen mindestens `min` Teilnehmer vom Typ `name` vorhanden sein.

nMaxWftType(name: <Atom> min: <Integer>) Es sollen höchstens `max` Teilnehmer vom Typ `name` vorhanden sein.

nMinBlock(block: <Integer> min: <Integer>) In Block `block` sollen mindestens `min` Teilnehmer vorhanden sein.

nMaxBlock(block: <Integer> min: <Integer>) In Block `block` sollen höchstens `max` Teilnehmer vorhanden sein.

isTypeOf(block: <Integer> type: <Atom>) Dem Block `block` wird im vorhinein der Teilnehmertyp `type` zugewiesen.

isNotTypeOf(block: <Integer> type: <Atom>) Der Teilnehmertyp `type` wird im Block `block` ausgeschlossen.

7.3 Aktivitäten

aktsInBlock(block: <Integer> akts: <Tupel>) Die in `akts` angegebenen Aktivitäten sind im Block `block` enthalten.

aktsNotInBlock(block: <Integer> akts: <Tupel>) Die in `akts` angegebenen Aktivitäten sind nicht im Block `block` enthalten.

aktsSameBlock(<Tupel>) Alle hier angegebenen Aktivitäten liegen in jeder Lösung im selben Block.

maxAKTsAll(<Integer>) Jeder Block besitzt maximal `Integer` Aktivitäten.

aktsOn(name: <Atom> akts: <Tupel>) Die in `akts` angegebenen Aktivitäten werden alle vom Teilnehmertyp `name` ausgeführt.

aktsNotOnWft(name: <Atom> akts: <Tupel>) Die in `akts` angegebenen Aktivitäten werden keinesfalls vom Teilnehmertyp `name` ausgeführt.

7.4 Sonstige

maxT(<Integer>) Vorgegebene maximale Taktzeit.

maxToolsChange(<Integer>) Die vorgegebene Zahl repräsentiert die maximal erlaubte Anzahl von Werkzeugwechseln, die bei einem Ablauftausch anfallen kann. Hierzu werden alle Ablaufpaare blockweise betrachtet. Diese Nebenbedingung wird nicht beim vermischten Datenfluß angewendet.

Kapitel 8

Woop Editor

Der Editor ist ein interaktives Werkzeug zur Erstellung und Verifikation von Abläufen.

8.1 Zeichnen

Die Zeichenfläche nimmt den größten Teil ein. Rechts daneben befinden sich die Zeichenwerkzeuge.

Eine Aktivität bzw. ein Knoten wird durch Drücken des Button NEW ACT. instanziert, wobei der Knoten erst durch Drücken der linken Maustaste über der Zeichenfläche entsprechend plaziert wird. Diese Aktivität hat standardmäßig den Namen *-1*. Ein Doppelklick mit der linken Maustaste öffnet einen Dialog, indem der Name geändert werden kann.

Durch Anklicken von DEL ACT. und einer Aktivität wird diese von der Zeichenfläche gelöscht.

Welche Aktion gerade aktiv ist, zeigt die farbliche Kennzeichnung des Buttons.

Zusätzlich zum Knoten kann ein Punkt mittels NEW POINT erstellt werden. Er dient als Hilfsmittel, um gerade Pfeile (Vorrangsrelation) zu ermöglichen. Im Punkt treffen sich zwei Linien. Über DEL POINT wird ein Punkt gelöscht. Der Punkt und seine Kennzeichnung bleiben in Woop unsichtbar.

Um zwei Aktivitäten (oder Punkte) miteinander zu verbinden, muß NEW REL aktiviert werden. Anschließend wird man in der Statusanzeige aufgefordert, die Start Aktivität auszuwählen. Danach erscheint die Aufforderung, die Ziel Aktivität zu bestimmen, woraufhin ein Pfeil (bzw. eine Linie) gezeichnet wird. Der Pfeil tritt standardmäßig rechts aus der Start Aktivität aus und endet auf der linken Seite der End Aktivität. Dies kann über einen Doppelklick auf den Pfeil verändert werden. Im *Relation Dialog* kann der Eintritt und Austritt der Kante neu bestimmt werden.

Das Löschen einer Kante übernimmt DEL REL.

Über PROCESS PROPERTIES wird dem Graph ein neuer Ablaufname zugewiesen. Des weiteren ist die Festlegung eines Quota Wertes für den Graphen möglich. Dieser Wert gibt an, zu welchem Teil dieser Graph in den vermischten Datenfluß eingeht. Die Summe aller Quotas, außer `sep` und `mix` beträgt 100.

Jeder Ablauf benötigt eine globale Start- und End-Aktivität. Dies sind in der Regel Aktivität 1 und Aktivität 100. Davon abweichend kann man über *Options* → *Activities* eigene Vorgaben definieren.

8.2 Verschieben

Aktivitäten und Points können durch einmaliges Drücken der linken Maustaste, diese dabei gedrückt haltend und die Maus bewegend verschoben werden. Die Pfeile werden automatisch nachgezeichnet.

Ein gesamter Graph wird verschoben, indem anstatt der linken die mittlere Maustaste gedrückt wird.

Wird ein Element über die Grenze der Zeichenfläche hinaus verschoben oder neu erstellt, so wird die Zeichenfläche automatisch vergrößert.

Das Verschieben oder Setzen eines Elements orientiert sich an einem Raster. Dieses Raster ist normalerweise unsichtbar und existiert für jeden Graph einzeln. Die Darstellung des Rasters geschieht über *Options* → *Use Grid*.

8.3 Sessionverwaltung

Eine Session beinhaltet in der Regel mehrere Graphen. Um die Daten nicht zu verlieren sollte man sie regelmäßig abspeichern. Benutze hierzu *File* → *Save Session*. Das Laden einer Session geschieht über *File* → *Load Session*. Eine aktuelle Session kann verworfen werden, indem man *File* → *New Session* wählt.

Entspricht eine Session den geforderten Abläufen, so kann sie ins entsprechende Woop Format über *File* → *Export Session* exportiert werden. Für den Export muß ein separater und ein vermischter Ablauf errechnet werden.

Vor dem Export wird die gesamte Session verifiziert. Dies kann auch manuell über *File* → *Verify Session* gemacht werden. Hierbei wird jeder Graph einzeln verifiziert.

8.4 Graphverwaltung

Einen neuen Graphen erstellt man über *Graph* → *New Graph*. Diesen kann man über *Graph* → *Delete Graph* wieder löschen, oder falls nur der Inhalt neu gestaltet werden soll, reicht *Graph* → *Clear Graph.*

Den Inhalt eines Graph kann man mit *Graph* → *Copy Graph* in die Zwischenablage kopieren, um ihn dann mit *Graph* → *Paste Graph* wieder neu zu zeichnen. Dies ist insbesondere dann nützlich, wenn mehrere Abläufe ähnlich gestaltet sind.

Das Abspeichern eines Graphen geschieht über *Graph* → *Graph Speichern*, wohingegen das Laden eines einzelnen Graphen über *Graph* → *Graph Laden* zu vollziehen ist.

8.4.1 Separater und vermischter Ablauf

Es reicht aus, nur die Abläufe zu zeichnen. Der Editor ist selbst in der Lage, einen separaten und einen vermischten Ablauf (für chaotische Fertigung) zu erstellen. Hierzu dienen die Menüpunkte *Graph* → *Separate Graph* und *Graph* → *Admixed Graph*.

Beim separaten Ablauf werden die Positionen der Aktivitäten genauso übernommen. Kommen Aktivitäten doppelt vor, so wird nur die erstere beachtet. Somit muß man schon während des Zeichnens auf die Positionen der Aktivitäten achten. Es kann vorkommen, daß eine Aktivität eine andere im Res Graph überlagert.

Die Gestaltung des vermischten Ablaufs ist einfacher. Bei allen Abläufen bleibt die Start- und End-Aktivität erhalten. Die Aktivitäten der Abläufe werden umbenannt und deren Position neu bestimmt. Die Umbenennung besteht hauptsächlich aus der Addition eines Offset Wertes auf den aktuellen Aktivität Namen. Die Festlegung des Offset Wertes findet in *Options* → *Aktivities* statt.

8.4.2 Verifikation

Während der Verifikation kann der Editor einen Fehler feststellen. Folgende Fehlerquellen sind bekannt:

Path Error. Ein Hilfspunkt hinterläßt eine Lücke im Graphen, d.h. die über ihn verlaufende Kante verbindet keine zwei Aktivitäten. Beende Kante durch Verbindung des Punktes.

No Start Activity. Es existiert kein Knoten mit der Bezeichnung der Startaktivität. Entweder wurde eine Benennung eines Knotens vergessen, oder eine neue Bezeichnung muß unter *Options* → *Aktivities* neu definiert werden.

No End Activity. Es existiert kein Knoten mit der Bezeichnung der Schlußaktivität. Entweder wurde eine Benennung eines Knotens vergessen, oder eine neue Bezeichnung muß unter *Options* → *Aktivities* neu definiert werden.

No Successor. Eine Aktivität hat keinen Nachfolger. Eine neue Kante mit der betroffenen Aktivität als Ausgangsaktivität muß erstellt werden.

No Predecessor. Eine Aktivität hat keinen Vorgänger. Eine neue Kante mit der betroffenen Aktivität als Eingangsaktivität muß erstellt werden.

Cycle Error. Die Quelle und Senke einer Kante ist gleich. Lösche Kante.

Duplicate Names. Es existieren mehrerer Knoten mit derselben Bezeichnung. Benenne eine Aktivität um.

Kapitel 9

Tips

1. Vorsicht bei Nutzung von Minimalangaben. Eine Einschränkung auf minimal x Teilnehmer, wird eine Lösung mit weniger Teilnehmer nicht finden.

2. Benutze wo immer es geht eine parallele Suchmaschine. Durch sie ist es möglich, eine Suche über mehrere Blöcke gleichzeitig zu führen. Dies führt zu einer reduzierten Rechenzeit.

3. Schwesterwerkzeuge gehen nur geringfügig in die Berechnungen ein. Es wird einmalig festgelegt, wieviele Schwesterwerkzeuge jeweils eingesetzt werden. Werden beispielsweise x Schwesterwerkzeuge erlaubt und hat ein Teilnehmer ein Magazin mit y Plätzen, so gehen das Modell nunmehr von $y \ mod \ x$ Plätzen aus.

   Auch in die Kosten gehen Schwesterwerkzeuge nicht ein.

4. Nicht immer sind Einschränkungen befriedigend. Die benutzerdefinierte Nebenbedingung `maxToolsChange` führt zwar zu einer kleineren Lösungsmenge, allerdings steigt die Rechenzeit und der Speicherverbrauch. Besser ist es, diese Bedingung auszulassen und die berechneten Lösungen auf diese Bedingung zu überprüfen, indem man sie exportiert.

Literaturverzeichnis

[1] Christian Schulte. Parallel search made simple. Technical Report TRA9/00, School of Computing, National University of Singapore, 55 Science Drive 2, Singapore 117599, September 2000. To appear.