Designumgebung
für
Design – Graphgrammatiken

Studienarbeit

Sven Brandt

Fachbereich Informatik
Universität Paderborn

Paderborn, November 2001

Erklärung

Hiermit versichere ich, daß ich diese Arbeit selbständig angefertig habe und keine anderen, als die angegebenen und bei Zitaten kenntlich gemachten Quellen und Hilfsmittel benutzt habe.

Ort, Datum	Unterschrift

Inhaltsverzeichnis

Kapitel I
Einleitung

Das Management eines technischen Systems schließt neben der eigentlichen Planung des Designs auch die anschließende Verwaltung der Änderungen der Systemstruktur ein. Diese Veränderungen treten hauptsächlich bei sich ändernden Anforderungen oder späterer Optimierung des technischen Systems auf. Im Detail muß das Management folgende Aufgaben umfassen:

- Die Erschaffung eines technischen Systems (Synthese)
- Die Berechnung einer adäquaten Topologie
- Die Berechnung und Überprüfung des Verhaltens der Topologie
- Die Überprüfung des Systems (Analyse)
- Die Überprüfung der korrekten Verschaltung des Systems
- Die Überprüfung der korrekten Funktion des Systems
- Die Anpassung und Optimierung des Systems

Ein computergestütztes Management des technischen Systems muß daher Möglichkeiten schaffen, diese Aufgaben zu erleichtern.

Die Synthese erfordert das Anordnen einzelner Subsysteme oder Bauteile, wohingegen die Anpassung und Optimierung eine eindeutig definierte Manipulation des technischen Systems beschreibt. Eine natürliche Darstellung eines technischen Systems ist die Graphendarstellung. Hierbei stellen die Graphenknoten die einzelnen Bauteile des Systems dar, während die Kanten des Graphen den Wertstoffluß oder den Energie– bzw. Informationsaustausch zwischen den Bauteilen darstellen. Die Manipulation dieser Graphendarstellung verändert einzelne Bauteile oder ganze Subsysteme.

Der Schwerpunkt dieser Arbeit beschäftigt sich mit der Manipulation von Graphen, die als Design – Graphgrammatiken definiert werden kann **[SSK2001]**. Diese Arbeit beschreibt neben den Grundlagen der Design Graphgrammatiken einen Eingabeeditor, der eine einfache grafikbasierte Definition dieser Grammatiken ermöglicht. Im letzten Kapitel wird dieser Editor mit den Tools der Programmiersprache PROGRES verglichen, die ein mächtiges Werkzeug zur Beschreibung und Durchführung von Graphtransformationen darstellt.

Kapitel II
Grundlagen der Design Graphgrammatiken

In diesem Kapitel werden die grundlegenden Anforderungen und Eigenschaften vorgestellt, die an einen „technischen Graphen“ gestellt werden, der für das Management eines technischen Systems angewendet werden soll.

Die Anforderungen an einen technischen Graphen lassen sich in zwei große Bereiche einteilen.

Zum einen muß der Graph in der Lage sein ein real existierendes technisches System möglichst einfach für den Anwender abzubilden. Der Anwender sollte in der Lage sein, ohne großes Fachwissen über Graphen, den technischen Graphen lesen und verstehen zu können. Außerdem muß sichergestellt sein, daß alle zur Arbeitsweise des technischen Systems benötigten Informationen innerhalb des Graphen abgebildet werden können.

Der zweite große Bereich beschäftigt sich mit den Manipulationsmöglichkeiten des Graphen. Hier müssen die grundlegenden Arbeitsschritte bei dem Management des technischen Systems innerhalb des Graphen möglich sein.

1 Darstellung eines realen technischen Systems

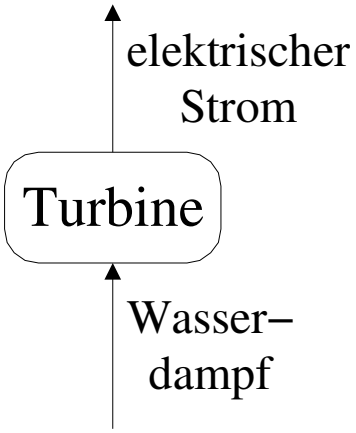
Ein real existierendes technisches System ist gekennzeichnet durch einen Wertstoffffluß, der von den Rohstoffen über die Verarbeitung bis hin zu den fertigen Produkten bzw. Abfallstoffen reicht. Von besonderem Interesse ist hierbei die Verarbeitung, da die Rohstoffe bei dem Design des Systems als gegeben vorausgesetzt werden können und die Produkte bzw. Abfallstoffe sich direkt aus dem Design der Verarbeitung ergeben. Daher soll die Verarbeitung im allgemeinen und der Produktionsablauf innerhalb der Verarbeitung im besonderen Gegenstand der Darstellung des technischen Graphen werden.

Die wichtigsten Objekte der Verarbeitung sind die Produktionseinheiten, in denen eine Wertstoffumwandlung stattfindet. Diese Produktionseinheiten besitzen jeweils mindestens einen Wertstoffeingang und einen Wertstoffausgang. Es bietet sich also an, die Produktionseinheiten als Knoten des Graphen darzustellen. Um die Arbeitsweise der Produktionseinheiten zu spezifizieren, wird dem Knoten eine Beschreibung ***(Label)*** beigefügt.

So wird beispielsweise eine Turbine, die aus Wasserdampf durch die Drehung ihrer Rotoren elektrische Energie erzeugt, als ein Knoten mit einem Eingang, einem Ausgang und dem Label *„Turbine“* dargestellt.

Neben den Produktionseinheiten existiert innerhalb der Verarbeitung ein Transport von Werkstoffen, der die einzelnen Produktionseinheiten verbindet. Der Transport ist daher als Kante innerhalb des Graphen wiederzufinden. Auch bei den Kanten wird eine Bezeichnung (***Label***) eingefügt, die eine Beschreibung des transportierten Werkstoffs bzw. der transportierten Informationen enthalten kann. Da die Transport– und Informationswege eines technischen Systems in der Regel gerichtet sind, sollten auch alle Kanten des entstandenen technischen Graphen gerichtet sein. Eine ungerichtete Verbindung kann mit Hilfe zweier gerichteter Kanten nachgebildet werden.

In unserem Beispiel wird also ein Knoten mit dem Label *„Turbine“*, eine auf den Knoten gerichtete Kante (***Eingang***) mit dem Label *„Wasserdampf“* und eine von dem Knoten gerichtete Kante (***Ausgang***) mit dem Label *„elektrischer Strom“* in den Graphen integriert.



Definition der labeled Graphen

Ein labeled Graph ist ein Tupel $G=\langle V_G, E_G, \sigma_G \rangle$ mit:

V_G *ist eine Menge von Knoten*

$E_G \subseteq V_G \times V_G$ *ist die Menge der gerichteten Kanten*

σ *ist die Labelfunktion*

σ_G: $V_G \cup E_G \rightarrow \Sigma$ *mit:*

Σ *ist die Menge von Symbolen, die das Label−Alphabet bilden*

Definition der Kanten

$(v_1, v_2, label)$ repräsentiert eine gerichtete Kante von dem Knoten v_1 zu dem Knoten v_2 mit dem Label ***label***.

$(v_1, v_2, label) \in E_G$

$v_1, v_2 \in V_G$

$label \in \dot{\Sigma}$

Ungerichtete Kanten werden durch zwei gerichtete (v_1, v_2, l) *und* (v_2, v_1, l) repräsentiert. Sie werden als $\{v_1, v_2, l\}$ beschrieben.

Kanten ohne Label können als (v_1, v_2) *bzw.* $\{v_1, v_2\}$ beschrieben werden.

2 Transformation des Graphen

Die Synthese, Anpassung und Optimierung eines technischen Systems erfordern eine Reihe von Transformationen des Graphen. So ist es notwendig einzelne Einheiten einzufügen oder zu löschen. Ebenfalls muß es möglich sein die Spezifikation der Einheiten zu ändern. Diese Änderungen betreffen nicht nur einzelne Knoten und Kanten, sondern müssen auch auf Teilgraphen anwendbar sein.

Die Manipulation eines Graphen wird durch Regeln der Form $target \rightarrow replacement$ beschrieben. Um eine exakte Definition dieser Regel angeben zu können, werden Design Graphgrammatiken zu Hilfe genommen. Eine zentrale Rolle kommt dabei der Definition der linken Regelseite zu, die in dem Ausgangsgraphen gefunden und anschließend ersetzt werden muß. Zur Bestimmung der linken Seite wird das Isomorphismus bzw. Matching Konzept benutzt (siehe auch **[Jungnickel1999]**).

Definition des Isomophismus

Seien $G = \langle V_G, E_G \rangle$ und $H = \langle V_H, E_H \rangle$ zwei Graphen.

Isomophismus ist eine bijektive Abbildung $\varphi: V_G \rightarrow V_H$ für die gilt:
$\forall a, b \in V_G: \{a,b\} \in E_G \Leftrightarrow \{\varphi(a), \varphi(b)\} \in E_H$

Die Graphen G und H werden isomorph genannt, falls es eine solche Abbildung gibt.

G und H werden isomorph mit Labels genannt, falls G und H isomorphe labeled Graphen mit den Label – Funktionen σ_G *und* σ_H sind und folgende Bedingung gilt:
$\forall e \in E_G$ *mit* $\varphi(e)=\{\varphi(a),\varphi(b)\}: \quad \sigma_G(e) = \sigma_H(\varphi(e)),$ *falls* $e = \{a,b\}$

Definition des Matching und des Context

Seinen $G = \langle V, E, \sigma \rangle$ und C labeled Graphen. Jeder Subgraph $\langle V_C, E_C, \sigma_C \rangle$ von G, der isomorph zu C ist, heißt Matching von C in G.

Falls C nur aus einem Knoten besteht, heißt das Matching knotenbasiert, andernfalls graphbasiert.

Sei T ein Subgraph von C und sei $\langle V_T, E_T, \sigma_T \rangle$ ein Matching von T in G und innerhalb des Matching von C. Ein Matching von C in G steht in mindestens einer Beziehungen zu $\langle V_T, E_T, \sigma_T \rangle$:

1. $V_T \subset V_C, \quad V_T \neq \emptyset$ Dann heißt der Graph $\langle V_C, E_C, \sigma_C \rangle$ Context von T in G.
2. $\langle V_C, E_C, \sigma_C \rangle = \langle V_T, E_{T,} \sigma_T \rangle$ Dann heißt T contextfrei

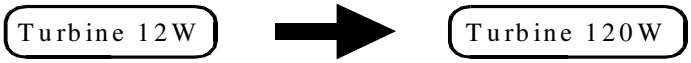
Im weiteren wird es keine Unterscheidung zwischen dem Graph und seiner ismorphen Kopie geben.

3 Beispiele für die Manipulation von Graphen

Typ I. **Ersetzen eines Knoten durch einen anderen.**

Diese Manipulation stellt eine contextfreie Transformation dar, die auf den Knotenlabeln basiert. Die Änderung wird ausreichend durch eine Labeländerung des betroffenen Knotens beschrieben.

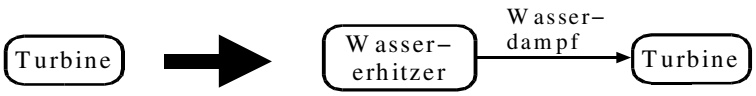
So wird beispielsweise eine Turbine durch eine leistungsfähigere ersetzt.



Typ II. **Ersetzen eines Knoten durch einen Graphen.**

Bei dieser Manipulation handelt es sich um eine contextfreie Transformation, die auf den Knotenlabeln beruht.

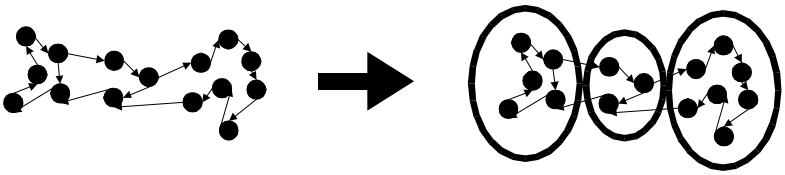
Hierbei wird beispielsweise der Knoten „Turbine“ durch den Graphen mit den Knonen „Wassererhitzer“ und „Turbine“ und einer Kante „Wasserdampf“ zwischen ihnen ersetzt.



Typ III. **Ersetzen eines Knoten mit Context durch einen anderen Knoten.**

Hierbei handelt es sich um eine knotenbasierte Transformation, die auf den Knoten– und Kantenlabeln beruht.

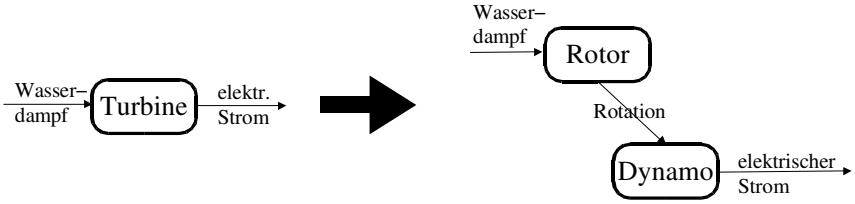
Diese Manipulation ermöglicht eine Umgebung berücksichtigende Aufteilung und Zusammenfassung eines Graphen. Beispielsweise können die Wege des elektrischen Stroms in die Graphen „Stromerzeugung“, „Stromtransport“ und „Stromverbrauch“ eingeteilt werden.



Typ IV. **Ersetzen eines Knoten mit Context durch einen Graphen**

Diese Manipulation beschreibt eine knotenbasierte Transformation, die auf den Knoten– und Kantenlabeln beruht.

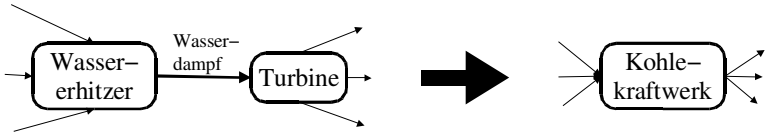
Mit Hilfe dieser Manipulation kann der Knoten „Turbine“ mit dem Eingang „Wasserdampf“ und dem Ausgang „elektrischer Strom“ in die Knoten „Rotor“ und „Dynamo“ mit der Kante „Rotation“ umgewandelt werden.



Typ V. **Ersetzen eines Graphen durch einen anderen Graphen**

Es handelt sich hierbei um eine contextfreie Transformation, die auf Graphen ohne Kantenlabeln basiert.

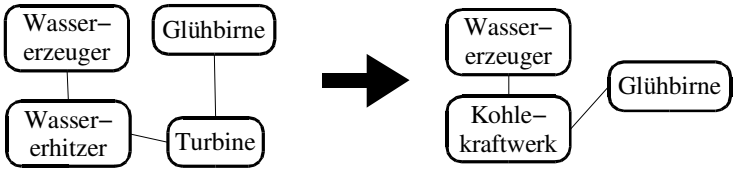
Diese Manipulation ermöglicht beispielsweise einen Graphen mit dem Knoten „Wassererhitzer“ und „Turbine“ in einen Graphen mit dem Knoten „Kohlekraftwerk“ umzuwandeln



Typ VI. **Ersetzen eines Graphen mit Context durch einen anderen Graphen**

Diese Manipulation beschreibt eine graphenbasierte contextsensitive Transformation, die auf den Kantenlabeln basiert.

Die Umwandlung eines Graphen mit den Knoten „Wassererzeuger“, „Wassererhitzer“, „Turbine“ und „Glühbirne“ in einen Graphen mit den Knoten „Wassererzeuger“, „Kohlekraftwerk“ und „Glühbirne“ wird mit dieser Transformation beschrieben.



Typ VII. **Löschen eines Subgraphen**

Diese Manipulation ist eine Sonderform des Typ VI.

Typ VIII. **Löschen eines Knotens**

Eine Transformation dieses Typen stellt eine Sonderform des Typ IV dar.

Alle Arten der Transformationen werden mit Hilfe von Design Graphgrammatiken definiert, die in contextfreie und contextsensitive Grammatiken unterteilt werden. Beide Arten werden im folgenden beschrieben.

4 Design Graphgrammatik

Die Design Graphgrammatik repräsentiert eine Menge von Transformationsregeln. Jede dieser Regeln wandelt einen Graphen G in einen neuen Graphen G' um, wobei ein Knoten bzw. ein Subgraph in einen neuen Knoten bzw. Subgraphen überführt wird. Es werden also Teile eines Graphen **G** in einen Graphen **R** umgewandelt: **R wird in G eingebettet.**

Im folgenden wird die Design Graphgrammatik nur Graphgrammatik genannt.

Definition des Hostgraphen (G)
Der Hostgraph beschreibt die Struktur, auf der die Transformationen durchgeführt werden. Der Hostgraph repräsentiert das komplette technische System.

Definition des Contextgraphen (C)
Eine zum Graph C' isomorphe Struktur C, die ein Matching innerhalb des Hostgraphen repräsentiert, wird Contextgraph genannt. Der Contextgraph findet sich auf der linken Seite einer Transformationsregel wieder.

Definition des Targetgraphen (T)
Der Targetgraph T repräsentiert eine zu einem Graphen T' isomorphe Struktur, die während einer Transformation in eine andere Struktur umgewandelt wird. Der Targetgraph ist also ein Subgraph des Contextgraphen. Auch der Targetgraph wird auf der linken Seite der Transformationsregeln definiert.
Besteht der Targetgraph nur aus einem Knoten, wird er Targetknoten genannt und mit **t** gekennzeichnet.

Definition des Replacementgraphen (R)
Der Replacementgraph R repräsentiert eine zu einem Graphen R' isomorphe Struktur, die bei der Anwendung einer Transformationsregel den Targetgraph ersetzt. Der Replacementgraph wird auf der rechten Seite der Transformationsregel definiert.

Definition des Cut – Node
Ein Cut – Node ist ein Knoten des Hostgraph, der eine direkte Kante zu dem Matching des Targetgraph hat.

4.1 Contextfreie Graphgrammatik

Bei der contextfreien Graphgrammatik sind der Targetgraph und der Contextgraph identisch. Bei Anwendung einer Transformationsregel der contextfreien Graphgrammatik wird also der Targetgraph durch eine isomorphe Kopie des Replacementgraph ersetzt.

Definition der contextfreien Graphgrammatik

Eine contextfreie Graphgrammatik ist eine Tupel $\zeta = \langle \Sigma, P, s \rangle$:

Σ *ist das Label−Alphabet für Knoten und Kanten*

P *ist eine endliche Menge Transformationsregeln*

s *ist das Initial−Symbol*

Die Transformationsregeln P werden in der Form $T \to \langle R, I \rangle$ definiert. Dabei gilt:

$T = \langle V_T, E_T, \sigma_T \rangle$ *ist der zu ersetzende Targetgraph*

$R = \langle V_R, E_R, \sigma_R \rangle$ *ist der möglicherweise leere ersetzende Replacementgraph*

I *ist die Menge der Einbettungsvorschriften für den Replacementgraph*

Die Menge der Einbettungsvorschriften I bestehen aus den Tupeln $((h,t,e),(h,r,f))$ mit

$h \in \Sigma$ *ist ein Label des Knoten* $v \in G \setminus T$

$t \in \Sigma$ *ist ein Label des Knoten* $w \in V_T$

$e \in \Sigma$ *ist ein Label der Kante* $\{v, w\}$

$f \in \Sigma$ *ist ein anderes Kantenlabel, das nicht unterschiedlich zu e sein muß*

$r \in V_R$ *ist ein Knoten des Replacementgraph*

Bei der Anwendung der Grammatik wird im ersten Schritt ein Matching des Targetgraphen T in dem Hostgraphen G gesucht. Alle Kanten zwischen den Cut – Nodes und dem Targetgraphen werden im nächsten Schritt gelöscht. Anschließend wird eine isomorphe Kopie des Replacementgraphen R unter Beachtung der Einbettungsvorschriften mit den Cut – Nodes des Hostgraphen verbunden.

Die Einbettungsvorschrift $((h,t,e),(h,r,f))$ wird wie folgt interpretiert:

Falls eine Kante mit dem Label **e** von einem Knoten mit dem Label **h** zu einem Knoten mit dem Label **t** existierte, so wird in dem neuen Graphen eine Kante mit dem Label **f** von dem Knoten mit dem Label **h** zu dem Knoten mit dem Label **r** eingefügt.

4.1.1 Beispiel einer contextfreien Graphgrammatik

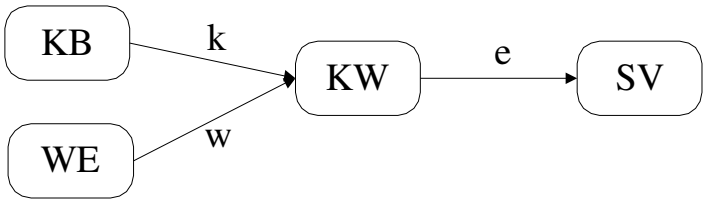
Ein Stromerzeuger entschließt sich ein altes Kohlekraftwerk durch die Kombination von Solartechnik und Brennstoffzellen zu ersetzten.

Der alte Graph besteht aus vier Knoten mit den Labeln:

- ➔ Wassererzeuger (**WE**)
- ➔ Kohlebergbau (**KB**)
- ➔ Kohlekraftwerk (**KW**)
- ➔ Stromverbraucher (**SV**).

Zusätzlich gibt es drei Kanten mit den Labeln:

- ➔ Wasser (**w**)
- ➔ Kohle (**k**)
- ➔ elektrische Energie (**e**).

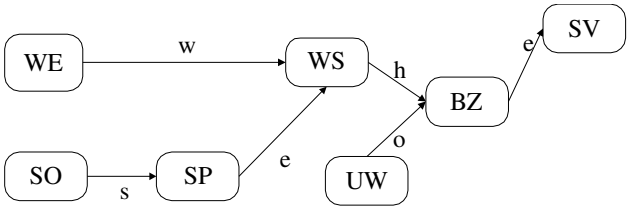


Dieser Graph soll in einen Graphen umgewandelt werden mit den Knoten:

- ➔ Wassererzeuger (**WE**)
- ➔ Sonne (**SO**)
- ➔ Solarpanel (**SP**)
- ➔ Wasserstofferzeugung (**WS**)
- ➔ Brennstoffzelle (**BZ**)
- ➔ Umwelt (**UW**)
- ➔ Stromverbraucher (**SV**)

und den Kanten:

- ➔ Sonnenenergie (**s**)
- ➔ Wasser (**w**)
- ➔ Wasserstoff (**h**)
- ➔ Sauerstoff (**o**)
- ➔ elektrische Energie (**e**)



Die contextfreie Graphgrammatik ζ kann für diese Aufgabe wie folgt definiert werden:

$$\zeta = \langle \Sigma, P, s \rangle$$

$$\Sigma = \{WE, KB, KW, SV, SO, SP, WS, BZ, UW, \quad w, k, e, s, h, o\}$$
$$P = \{r\}$$
$$s = \emptyset$$

Für die Transformationsregel $r: \; T \rightarrow \langle R, I \rangle$ gilt:

$$T = \langle \{1,2\}, \; \{(1,2)\}, \; \{(1, KB), (2, KW), ((1,2), k)\} \rangle$$

$$R = \langle Nodes, Edges, Label \rangle$$
$$Nodes = \{1, 2, 3, 4, 5\}$$
$$Edges = \{(1,2),(2,3),(3,4),(5,4)\},$$
$$Label = \{(1, SO),(2, SP),(3, WS),(4, BZ),(5, UW)\ ((1,2), s),((2,3), e),((3,4), h),((5,4), o)\}$$

$$I = \langle \; (\; (WE, KW, w),(WE, SP, w)), \; ((SV, KW, e),(SV, BZ, e) \;) \; \rangle$$

4.2 Contextsensitive Graphgrammatik

Die contextsensitive Graphgrammatik gleicht einen Nachteil der contextfreien Graphgrammatik aus. Diese ersetzen jedes Vorkommen des Targetgraphen, ohne die Umgebung zu beachten, während die contextsensitive Graphgrammatik neben dem Targetgraph auch seine nähere Umgebung einbezieht. Es wird also nur der Hostgraph verändert, bei dem sowohl die Umgebung des Targetgraphen, als auch der Targetgraph selbst als isomorphe Kopie vorhanden ist. Um dieses Verhalten zu erreichen, wird ein neuer Graphtyp, der Contextgraph **(II 4)** in die Definition der Graphgrammatik eingeführt.

Definition der contextsensitiven Graphgrammatik

Eine contextsensitive Graphgrammatik ist eine Tupel $\zeta = \langle \Sigma, P, s \rangle$:

Σ *ist das Label−Alphabet für Knoten und Kanten*

P *ist eine endliche Menge Transformationsregeln*

s *ist das Initial−Symbol*

Die Transformationsregeln P werden in der Form $\langle T, C \rangle \rightarrow \langle R, I \rangle$ definiert. Dabei gilt:

$T = \langle V_T, E_T, \sigma_T \rangle$ *ist der zu ersetzende Targetgraph*

C *ist der Contextgraph, für den* $T \subseteq C$ *gilt*

$R = \langle V_R, E_R, \sigma_R \rangle$ *ist der möglicherweise leere ersetzende Replacementgraph*

I *ist die Menge der Einbettungsvorschriften für den Replacementgraph*

Die Menge der Einbettungsvorschriften I besteht aus den Tupeln $((h,t,e),(h,r,f))$ mit

$h \in \Sigma$ *ist ein Label des Knoten* $v \in G \setminus T$

$t \in \Sigma$ *ist ein Label des Knoten* $w \in V_T$

$e \in \Sigma$ *ist ein Label der Kante* $\{v, w\}$

$f \in \Sigma$ *ist ein anderes Kantenlabel, das nicht unterschiedlich zu e sein muß*

$r \in V_R$ *ist ein Knoten des Replacementgraph*

Bei der Anwendung der contextsensitiven Graphgrammatik wird im ersten Schritt ein Matching des Contextgraphen im Hostgraph gesucht. Anschließend werden alle Verbindungen zu dem Targetgraph gelöscht. Schließlich wird im dritten Schritt eine isomorphe Kopie des Replacementgraphen nach den Anweisungen der Einbettungsvorschriften in den Hostgraph eingebettet, wobei die Einbettungsvorschriften wie bei der contextfreien Graphgrammatik angewendet werden.

Das Beispiel zur Anwendung der contextfreien Graphgrammatik kann auch als contextsensitive Graphgrammatik formuliert werden. Für die Definition als contextsensitive Graphgrammatik muß lediglich die Definition des Contextgraphen eingefügt bzw. die Definition der Transformationsregeln angepaßt werden.

Für den Contextgraph ergibt sich:

$$C = \langle Nodes, Edges, Label \rangle$$

$$Nodes = \{3, 4, 5, 6\}$$

$$Edges = \{(3,5),(4,5),(5,6)\}$$

$$Label = \{(3,KB),(4,WE),(5,KW),(6,SV),((3,5),k),((4,5),w),((5,6),e)\}$$

Existierende Graph – Grammatiken bieten gegenüber dem hier vorgestellten Konzept den Nachteil, daß sie hauptsächlich für Software – Engineering Probleme und nicht für technische Systeme entwickelt wurden **[Rozenberg1997, EEKR1999, EKMR1999, ESP1998, Kaul1986, 1987, Korff1991, Lichtblau1991, RS1995, SWZ1995, Schürr1997]**.

4.3 Strict degree matching

Das strict degree matching schreibt innerhalb der Graphgrammatik die Anzahl der Kanten eines Knoten vor. Bei der Anwendung einer Graphgrammatik mit einem Knoten, der als strict degree matching markiert wurde, wird nur ein Matching des Target– bzw. Contextgraphen gesucht, bei dem genau die Kanten vorhanden sind, die in der Graphgrammatik definiert wurden. Es wird also insbesondere ausgeschlossen, daß ein Knoten zusätzliche Verbindungen zu dem Hostgraphen hat.

4.4 Variable Label

Bei der Definition der Graphgrammatik besteht die Möglichkeit variable Label zu benutzen. Die Einbettungsvorschrift aus dem Beispiel **II 4.1.1** läßt sich mit den neuen Variablen X und Y folgendermaßen formulieren.

$$I = \{ ((X, Y, w), (X, SP, w)) , ((X, Y, e), (X, BZ, e)) \}$$

Die benutzten Variablen repräsentieren eine konkrete Label – Instanz. Dies bedeutet, daß sich zugewiesene Werte der Variablen während der Ersetzung nicht ändern.

Durch die Benutzung von variablen Labeln kann es vorkommen, daß mehrere Einbettungsvorschriften dieselbe Kante ersetzen. Beispielsweise ersetzen $((X, Y, l), (X, n, k))$ und $((X, m, l), (X, n, k))$ die Kante (X, m, l) . Eine Anwendung kann zur Folge haben, daß mehrere Kanten eingefügt werden. Um diesen Konflikt aufzulösen werden nur die Einbettungs–vorschriften angewendet, die den höchsten Spezifikationsgrad haben (siehe auch **[RN1995]**).

4.5 Speicherung der Graphgrammatik in einer Datei

In diesen Abschnitt wird das konkrete Dateiformat definiert, das für die Ersetzung in Graphen eingesetzt wird.

Das komplette Dateiformat besteht aus zwei Dateien. Die erste Datei beschreibt den aktuellen Hostgraph, auf den die Transformationsregeln angewendet werden sollen. Diese Datei hat die Endung ***gdf*** (**G**raph – **D**efinition – **F**ile). Die zweite Datei beschreibt die Transformationsregeln. Sie hat die Endung ***rdf*** (**R**ule – **D**efinition – **F**ile).

In beiden Dateien wird die Groß– und Kleinschreibung nicht beachtet. Dies gilt jedoch nicht für Strings, die zur Kennzeichnung in Hochkommata eingeschlossen werden.

Kommentare werden in beiden Dateien mit dem Prozentzeichen % gekennzeichnet. Sie können an jeder Position in der Datei stehen und kommentieren den Rest der Zeile aus.

Zu Beginn der Datei sollte eine kurze Beschreibung des Inhalts der Datei stehen.

Auf die Beschreibung des Inhalts kann eine Versionsnummer folgen. Die Version wird im Format **V = 1.0** abgespeichert.

Anschließend folgt jeweils der eigentliche Inhalt der Dateien.

Bei der Beschreibung der Dateien bezeichnen Elemente, die in ’[’ und ’]’ bzw. ’<’ und ’>’ eingeschlossen sind optionale bzw. individuelle Elemente.

4.5.1 Beschreibung des Graph – Definition – File

Die Definition eines Graphen muß die Knoten mit ihren Labeln und die Kanten mit ihren Labeln umfassen.

Der erste Abschnitt beschreibt die Knoten des Graphen. Er wird durch die Anzahl der Knoten eingeleitet.

nodes <n> **n** ist hierbei die Anzahl der Knoten des Graphen.

Im nächsten Teil wird den Knoten ein Label zugeordnet. Jeder Knoten besitzt eine eindeutige Kennzahl. Daraus ergibt sich für die Beschreibung eines Knoten folgende Form

<x> <Label> **x** ist hierbei die eindeutige Kennzahl des Knoten, während **Label** den Label
des Knoten beschreibt. Dieser wird als String angegeben und daher in Hochkommata eingeschlossen.

Im letzten Teil werden die Kanten des Graphen spezifiziert. Jede Kante besitzt ebenfalls eine eindeutige Kennzahl, einen Label, den Anfangsknoten und den Endknoten der Kante. Zusätzlich muß festgelegt werden, ob es sich um eine gerichtete oder ungerichtete Kante handelt. Eingeleitet wird die Beschreibung der Kanten durch die Anzahl der Kanten.

edges <k> **k** ist die Anzahl der Kanten.

Die Beschreibung einer Kante besteht aus folgenden fünf Komponenten:

e<n> <i> <j> <d> <Label>

n steht für die eindeutige Kennzahl der Kante.
i beschreibt die eindeutige Kennzahl des Anfangsknoten.
j beschreibt die eindeutige Kennzahl des Endknoten.
d definiert, ob die Kante ungerichtet (***u***) oder gerichtet (***g***) ist.
Label bezeichnet den Namen der Kante. Es handelt sich hierbei um einen String, der also in Hochkommata eingeschlossen werden muß.

4.5.2 Beispiel eines Graph – Definition – File

V 1.0

nodes 4
*1 "WE" **
*2 "SV" **
3 "KB"
4 "KW"

edges 1
e1 3 4 g "k"

4.5.3 Beschreibung des Rule – Definition – File

Das Rule – Definition – File besteht aus einer endlichen Menge von Transformationsregeln. Diese haben einen eindeutigen Namen und werden in der Form

rule <Name der Regel>

eingeleitet. Bei diesem Namen handelt es sich um einen frei wählbaren String der in Hochkommata eingeschlossen wird.

Jede Regel besteht aus vier Teilen. Der erste Teil beschreibt den Contextgraph, der zweite den Targegraph und der dritte den Replacementgraph. Alle drei Teile werden identisch zu der Beschreibung des Graph – Definition – File definiert. Die Definition des Contextgraphen wird durch die Zeile **c** eingeleitet. Der Targetgraph beginnt mit einem **t**, während der Replacementgraph durch ein **r** eingeleitet wird.

Der letzte Teil des File beschreibt die Einbettungsvorschriften der Regel. Diese werden durch ein **i** eingeleitet.

Die Anzahl der Einbettungsvorschriften wird durch die Zeile

entries <k>

definiert. Dabei ist **k** die Anzahl der Einbettungsvorschriften.

Auch die Einbettungsvorschriften haben eine eindeutige Kennzahl und bestehen aus dem Label eines Knoten des Hostgraph, dem Label eines Knoten des Targetgraph, einem Kantenlabel, dem Label des neuen Knoten aus dem Hostgraph, dem Label des neuen Knoten des Replacementgraph und dem neuen Kantenlabel. Konkret sieht die Definition daher folgendermaßen aus:

i <H–L> <T–L> <K–L> <H–L> <R–I> <K2–L>

i ist hierbei die eindeutige Kennzahl der Einbettungsvorschrift
H–L ist ein Label von einem Knoten des Hostgraphen
T–L ist ein Label von einem Knoten des Targetgraphen
K–L ist ein Kantenlabel
R–I ist eine Knoteninstanz aus dem Replacement – Graph
K2–L ist ein neues Kantenlabel

Ist es für die Einbettungsvorschrift unerheblich, zu welchem Knoten des Hostgraphen eine Verbindung ersetzt bzw. erzeugt werden soll, wird anstelle des eindeutigen Knotenlabel **<H–L>** die Variable **L** eingefügt **(II 4.4)**.

4.5.3.1 Besonderheiten des Rule – Definition – File

Für die Definition wäre die obige Beschreibung bereits ausreichend. Im folgenden werden einige Besonderheiten beschrieben, die ihren Ursprung in der leichteren Verarbeitung der Datei haben.

Um variable Label in der Einbettungsvorschrift erkennen zu können, werden diese ohne Anführungsstriche geschrieben.

Da ein String Leerzeichen enthalten könnte, wird in der Einbettungsvorschrift ein Semikolon als Trennzeichen zwischen den einzelnen Objekten eingeführt. Diese Änderung erleichtert lediglich das Parsen der Datei.

Das Konzept des strict degree matching **(II 4.3)** erfordert es, daß ein Stern an den betreffenden Knoten gehängt wird.

Damit ein Knoten in der Einbettungsvorschrift exakt identifiziert werden kann wird zusätzlich zu dem Label die eindeutige Kennzahl des Knoten durch einen Doppelpunkt getrennt an den Label gehängt.

Mit diesen Besonderheiten ergibt sich folgende geänderte Schreibweise für Knoten in allen drei Grapharten:

<n> <Label> [*]

Für die Schreibweise einer Einbettungsvorschrift ergibt sich:

$$i;\ \langle H-L\rangle : n_1;\ \langle T-L\rangle : n_2;\ \langle K-L\rangle;\quad \langle H-L\rangle : n_1;\ \langle R-I\rangle : n_3;\ \langle K2-L\rangle$$

4.5.4 Beispiel eines Rule – Definition – File

Beschreibung des Beispiels siehe **II 4.1.1**

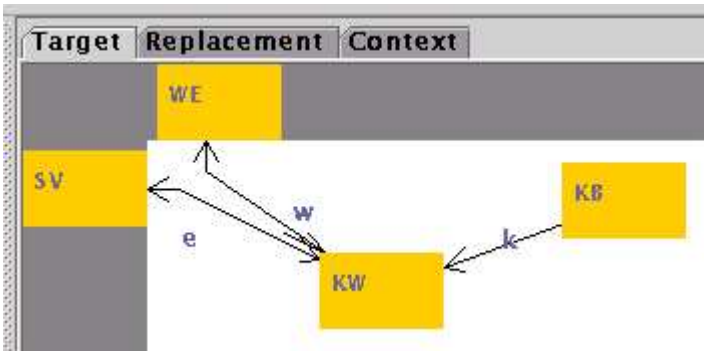
V 1.0

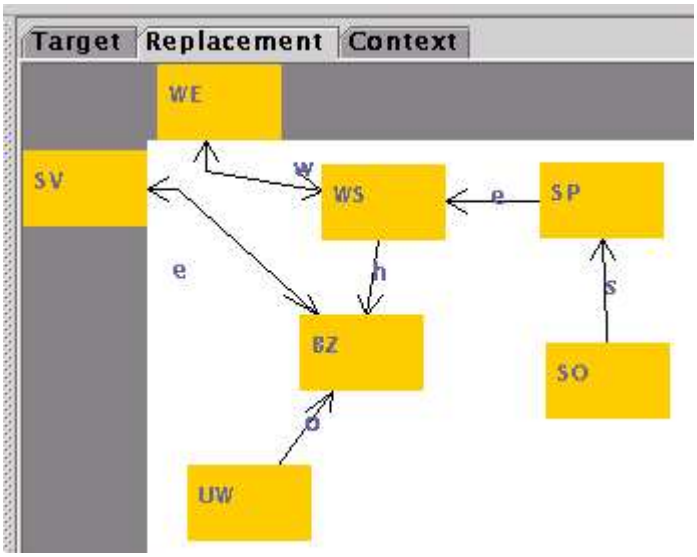
rule "rule0"
c
nodes 4
*1 "WE" **
*2 "SV" **
3 "KB"
4 "KW"
edges 1
e1 3 4 g "k"

t
nodes 2
3 "KB"
4 "KW"
edges 1
e1 3 4 g "k"

r
nodes 5
5 "SO"
6 "WS"
7 "UW"
8 "BZ"
9 "SP"
edges 4
e2 5 9 g "s"
e3 9 6 g "e"
e4 6 8 g "h"
e5 7 8 g "o"

i
entries 2
1; "WE":1; "KW":4; "w"; "WE":1; "WS":6; "w";
2; "SV":2; "KW":4; "e"; "SV":2; "BZ":8; "e";





Kapitel III

Eingabeeditor für Graphgrammatiken

In diesem Kapitel wird der Eingabeeditor RUBED (**Ru**le – **B**ased – **Ed**itor) für Graphgrammatiken vorgestellt, der es ermöglicht Rule – Definition – Files zu erzeugen. Der Editor arbeitet komplett graphikbasiert, wodurch es Designern von technischen Systemen sehr einfach möglich ist, Transformationsregeln für die Graphmanipulationen zu definieren.

Im ersten Teil dieses Kapitels werden die Anforderungen und grundsätzlichen Ideen des Editors vorgestellt, während im zweiten Teil die Umsetzung der Ideen und eine Funktionsbeschreibung des Editors folgt.

Nähere Informationen zur Java – Programmierung des RuBEd finden sich in **[Hawlitzek2000, Krüger2000]**.

1 Anforderungen an einen Eingabeeditor für Graphgrammatiken

Die Anforderungen an einen solchen Eingabeeditor können auf den Aufbau eines Graph – Definition – bzw. Rule – Definition – File beschränkt werden, da diese Dateien wie in **II 4.5** beschrieben alle Informationen der Graphgrammatik enthalten. Der RUBED beschränkt sich hierbei auf die Definition des Rule – Definition – File, da dieses File die eigentlichen Veränderungen eines Graphen beschreibt. Diese Datei schließt die allgemeine Beschreibung der einzelnen Komponenten genauso ein, wie die in **II 4.5.3.1** beschrieben Besonderheiten.

Aus der Struktur des Rule – Definition – File ergeben sich bereits einige Vorgaben an einen Editor. So muß der Anwender in die Lage versetzt werden, die drei Graphenarten in dem Editor zu erkennen. Für den Target– und den Replacementgraph ergeben sich dadurch keine Probleme, da es sich hierbei um komplett definierte Graphen handelt, die ohne ihre Umgebung aussagekräftig dargestellt werden können. Die beiden Graphen können so dargestellt werden wie es in vielen Graphen – Tools üblich ist.

Die Darstellung des Contextgraph ist nicht so einfach möglich. Hierbei ist zu beachten, daß es sich bei dem Targetgraph um einen Subgraphen des Contextgraphen handelt. Dies widerspricht der allgemeinen Bedeutung des deutschen Wortes Kontext, das den Graphen auf die Differenz $^{\text{Context}}/_{\text{Target}}$ beschränken würde. Da das Tool möglichst intuitiv zu benutzen sein soll, ist die Darstellung des Contextgraphen auf diese Differenz zu beschränken.

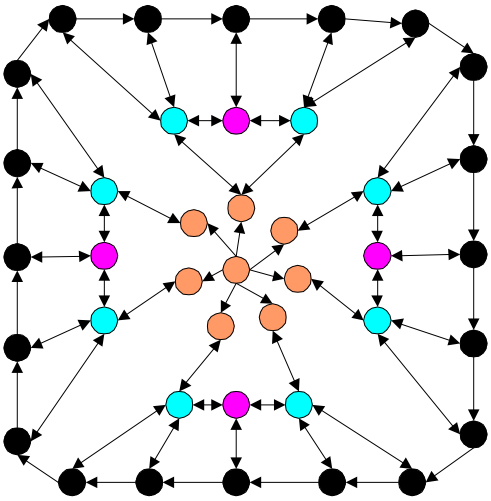
Neben den drei Graphenarten muß der Eingabeeditor eine Möglichkeit schaffen die Einbettungsvorschriften zu definieren. Bei der Anwendung einer Einbettungsvorschrift wird eine Kante von dem Host– bzw. Contextgraph zu dem Targetgraph durch eine Kante zum Replacementgraph ersetzt. Die Ausgangskante befindet sich bei der contextfreien Graphgrammatik zwischen dem Target– und dem Hostgraph, wohingegen sie sich bei der contextsensitiven Graphgrammatik zwischen dem Target– und dem Contextgraph befindet. Die neu einzufügende Kante befindet sich zwischen dem Replacementgraph und dem Host– bzw. Contextgraph.

Daraus ergeben sich für den graphischen Eingabeeditor zwei mögliche Fälle.

Der erste Fall beschreibt die contextsensitive Graphgrammatik. Bei dieser Grammatik sind alle betroffenen Kanten bereits in dem Contextgraph vorhanden. Es ist also in dem Eingabeeditor nur notwendig eine Möglichkeit zu schaffen, Kanten zwischen dem Target– bzw. Replacement– und dem Contextgraph einzufügen.

Im zweiten Fall muß die contextfreie Graphgrammatik betrachtet werden. Bei dieser Grammatik ergibt sich das Problem, daß die Cut – Nodes nicht direkt in dem Target– bzw. Replacementgraph vorhanden sind. Es muß also innerhalb des Eingabeeditors eine Möglichkeit geben, Knoten einzufügen, die nicht in den Graphdefinitionen vorkommen. Zusätzlich können diese Cut – Nodes labellos definiert werden.

Um beide Fälle in gleicher Art abzudecken, wird eine neue Knotenart eingefügt. Diese Knoten werden im folgenden Context – Nodes genannt, da sie in der contextsensitiven Graphgrammatik innerhalb des Contextgraphen liegen würden.



Contextsensitive Graph Grammatik

Knoten des Hostgraphen

Knoten des Contextgraphen

Knoten des Target–bzw. Replacementgraphen

Contextfreie Graph Grammatik

Knoten des Hostgraphen

Knoten des Target– bzw. Replacementgraphen

Die Grafik zeigt eine mögliche Anordnung der Knoten. Die oben beschriebenen Context – Nodes sind in diesem Beispiel helltürkis dargestellt. Die hellmagenta Knoten stellen Knoten dar, die zwar innerhalb des Contextgraphen liegen, aber weder Bestandteil des Target– bzw. Replacementgraphen sind, noch einen Context – Node darstellen. Diese Knoten werden im folgenden Difference – Nodes genannt.

Die contextfreie Graphgrammatik definiert die Knoten des Target– bzw. Replacementgraphen und die Context – Nodes. In der contextsensitiven Graphgrammatik werden zusätzlich die Difference – Nodes definiert.

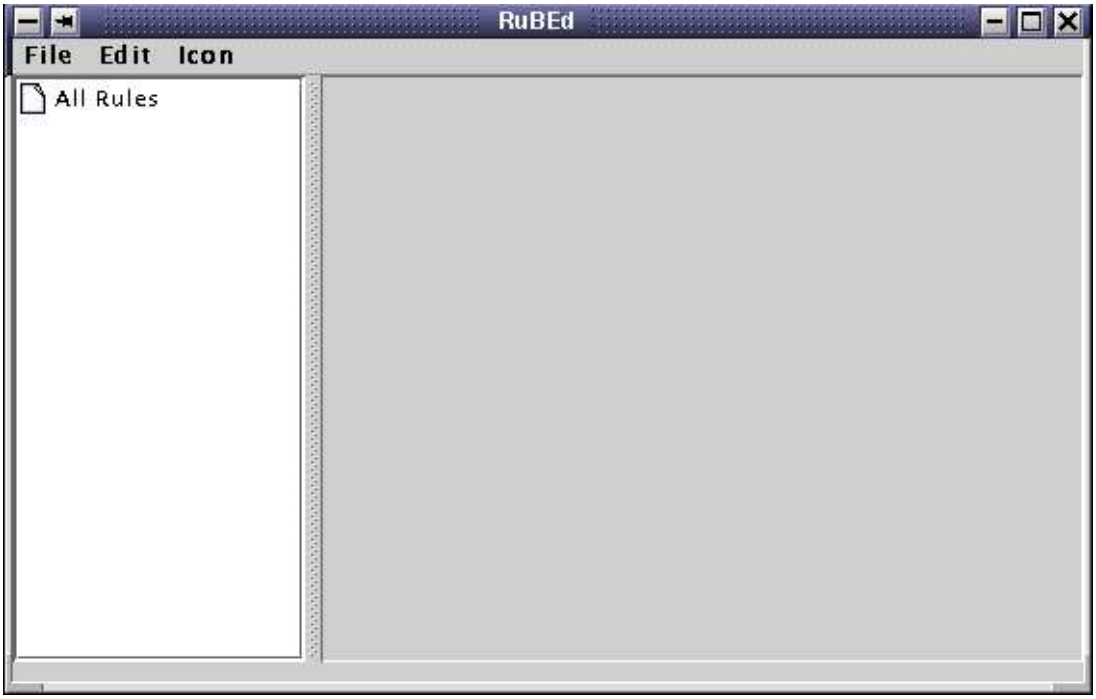
Die Graphen werden also innerhalb des Eingabeeditors in drei Eingabebereiche definiert. Im ersten Bereich kann der Anwender den Targetgraph, im zweiten den Replacementgraph vollständig definieren. Im dritten Bereich wird dem Anwender die Möglichkeit gegeben, sowohl die Context – Nodes, als auch die Difference – Nodes zu definieren. Der Anwender macht also keine Unterschiede zwischen der Definition von contextfreier und contextsensitiver Graphgrammatik. Lediglich die Anwesenheit von Difference – Nodes entscheidet über die Art der Graphgrammatik.

Um die Einbettungsverbindungen zwischen den Graphen einfügen zu können, werden die entsprechenden Context – Nodes zusätzlich in den beiden ersten Eingabebereichen dargestellt.

2 Beschreibung des RUBED

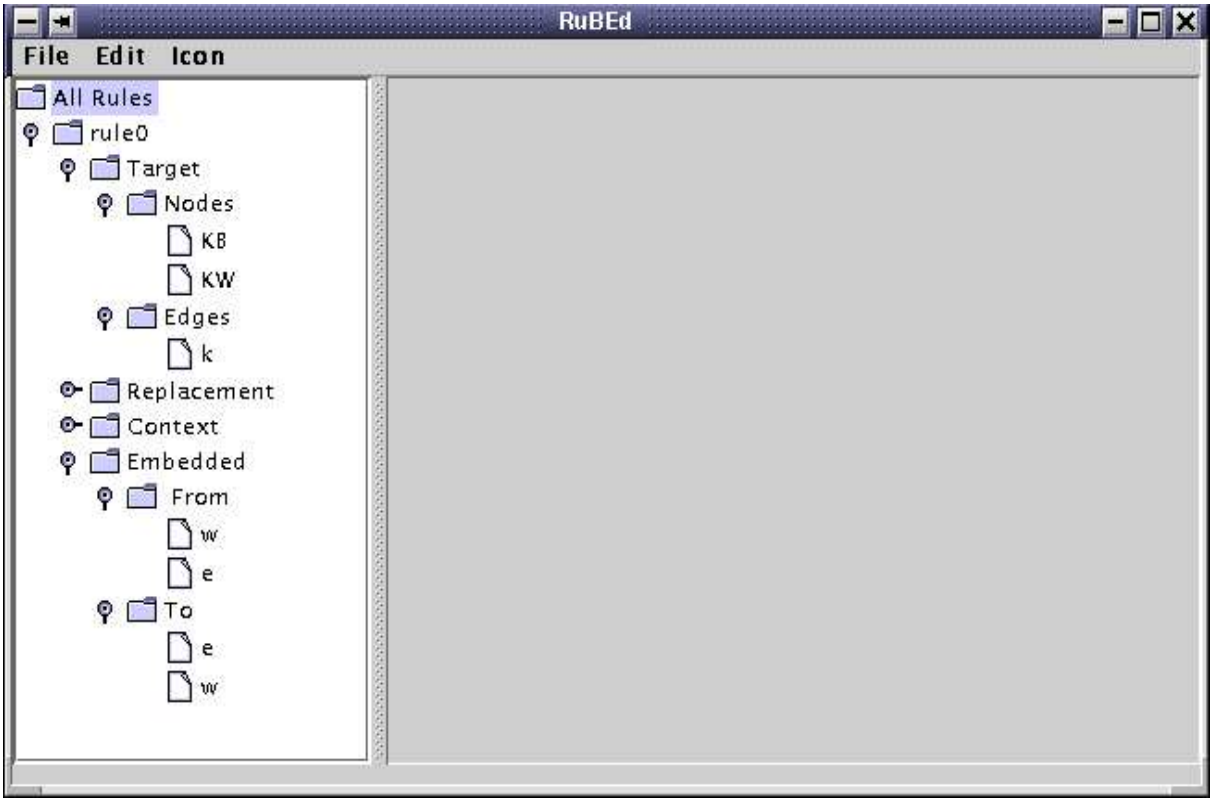
Der Eingabeeditor verwaltet eine endliche Zahl Transformationsregeln, die in einem Projekt zusammengefaßt werden. Ein solches Projekt beschreibt vollständig jeweils ein Rule – Definition – File.

Die Ansicht des Eingabeeditors ist in zwei Bereiche eingeteilt. Im ersten Bereich, der Projekt – Ansicht, werden alle Knoten und Kanten des Projekts in einem übersichtlichen Baum dargestellt. Die eigentliche graphische Eingabe findet im zweiten Bereich, der Graph – Ansicht statt. Dieser ist in drei Tabpages geteilt, die über einen Reiter angewählt werden.



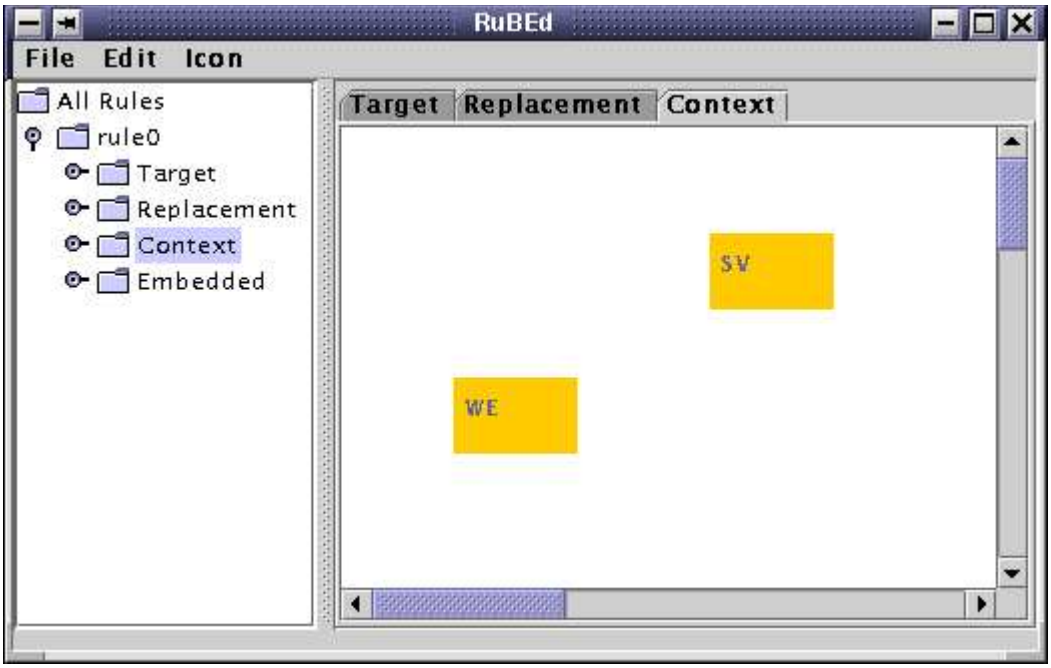
2.1 Die Projekt – Ansicht

Die Projekt – Ansicht besteht aus einer Baumdarstellung aller Knoten und Kanten. Die erste Ebene dieses Baumes bildet die einzelnen Regeln. Die Knoten und Kanten jeder Regel werden in die Bereiche Target, Replacement und Context eingeteilt. Die Einteilung entspricht der eigentlichen Definition des Target–, Replacement– bzw. Contextgraphen. Der Embedded – Zweig beschreibt die Einbettungsvorschriften. Dabei werden die Ausgangskanten in dem From – Zweig und die ersetzenden Kanten in dem To – Zweig dargestellt.



2.2 Graph – Ansicht

Die eigentliche Definition der Graphgrammatik geschieht in dem rechten Teil der Ansicht. Hier befinden sich drei Tabpages, in die der Anwender jeweils Knoten und Kanten einfügen kann. Die Tabpages sind mit den in **III 1** beschriebenen Eingabebereichen identisch.



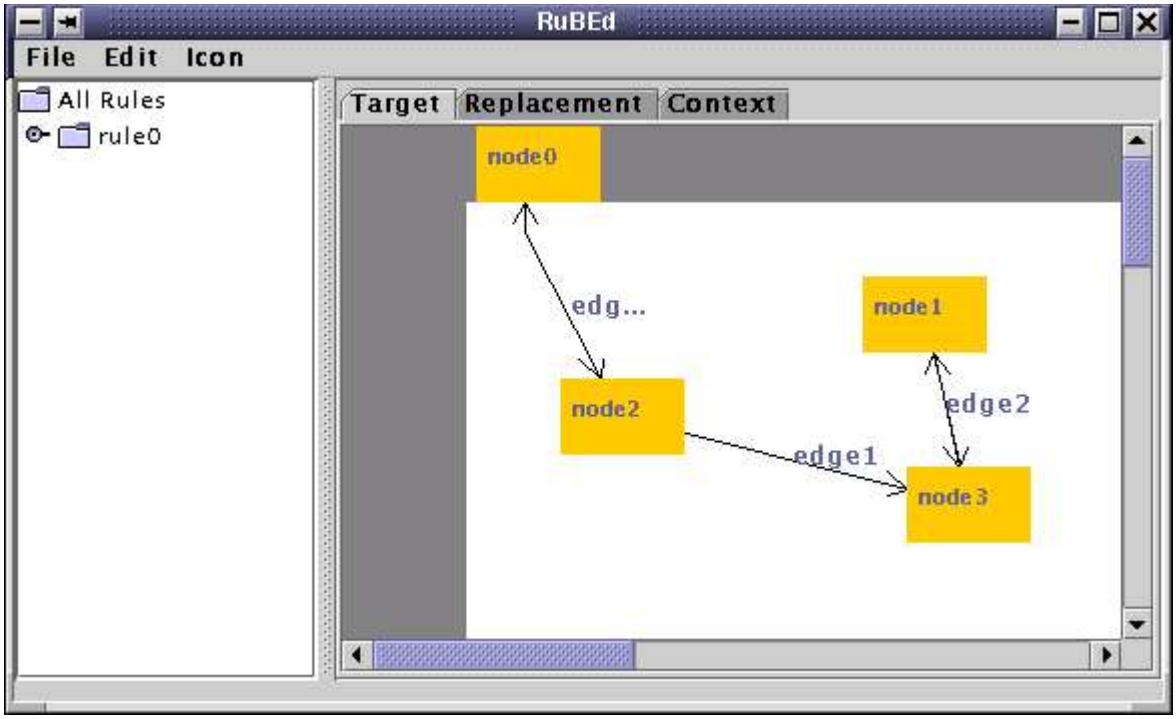
Um einen Knoten einzufügen, reicht ein Mausklick an die gewünschte Position. Zum Einfügen einer Kante markiert der Anwender den Ausgangsknoten, indem er auf diesen mit der linken Maustaste klickt. Anschließend klickt er auf den Endknoten, um die Kante einzufügen. Benutzt der Anwender dabei die linke Maustaste, wird eine ungerichtete, bei der rechten Maustaste eine gerichtete Kante eingefügt. Auf diese Weise definiert der Anwender sowohl den Target– und den Replacementgraph, als auch die Context – bzw. Difference – Nodes.

Der Anwender hat jederzeit die Möglichkeit, die Position der Knoten seinen Wünschen anzupassen. Dazu zieht er einen Knoten bei gedrückter Maustaste an die gewünschte Position.

Der RUBED stellt die Knoten als Rechtecke dar, die innerhalb das Label des Knoten tragen. Die Kanten des Graphen werden als verbindende Pfeile dargestellt. Dabei werden ungerichtete Kanten als Verbindungen mit zwei Pfeilspitzen dargestellt. Das Kantenlabel wird an die jeweiligen Kanten geschrieben. Diese Darstellung entspricht der allgemein üblichen Darstellung eines Graphen.

2.2.1 Einfügen der Context – bzw. Difference – Nodes

Die Context – bzw. Difference – Nodes werden auf der Context – Tabpage eingefügt. Nach dem Einfügen eines Knoten auf dieser Tabpage ist dieser standardmäßig ein Difference – Node. Die Knoten haben also keine Verbindungen zum Target– bzw. Replacementgraph. Dazu werden die Knoten mit Hilfe des Property – Dialogs in einen Context – Node umgewandelt **(III 2.3.1)**. Anschließend erscheint der Context – Node sowohl auf der Target –, als auch auf der Replacement – Tabpage. Der Anwender ist nun in der Lage, die externen Verbindungen des Target – bzw. Replacement – Graphen wie gewohnt auf der jeweiligen Tabpage einzufügen. Dabei ist zu beachten, daß die Einbettungsvorschriften keine Richtung der Kanten definieren. Die Richtung dieser Kanten wird also ignoriert. Außerdem werden die Kanten beim Einladen eines Rule – Definition – File als ungerichtet dargestellt.



2.3 Die Property – Dialoge

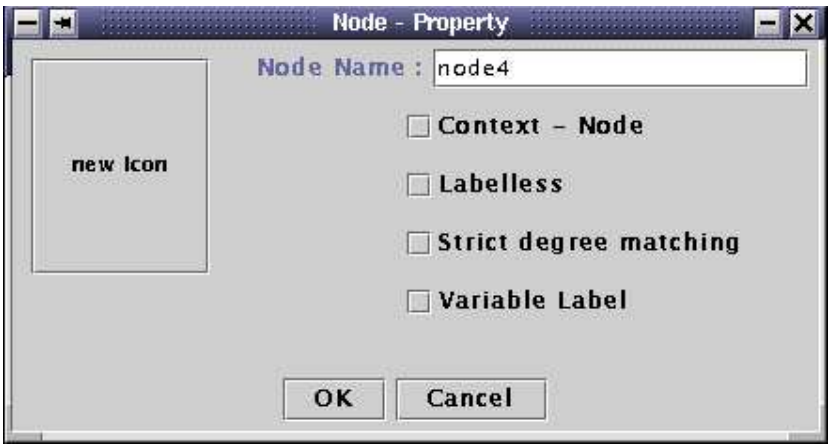
Um die Eigenschaften einer Regel, eines Knoten oder einer Kante zu ändern, stehen dem Anwender die jeweiligen Property – Dialoge zur Verfügung. Das entsprechende Objekt wird in der Projekt – Ansicht markiert. Anschließend wird über den Menüpunkt **Edit –> Property** der entsprechende Property – Dialog angezeigt.

Für die Knoten und Kanten besteht eine weitere Möglichkeit darin den Dialog direkt in der Graph – Ansicht zu öffnen. Dazu genügt ein Doppelklick auf das Kantenlabel. Der Dialog eines Knoten wird mit einem rechten Maustasten – Klick auf dem entsprechenden Knoten geöffnet. Der Dialog eines Context – Nodes kann dabei nur auf der Context – Tabpage geöffnet werden.

Mit allen Property – Dialogen ist eine Änderung des jeweiligen Namens möglich. Dies gilt insbesondere für den Regel – Property – Dialog, da die Namensänderung die einzig mögliche Änderung ist.

2.3.1 Der Knoten – Property – Dialog

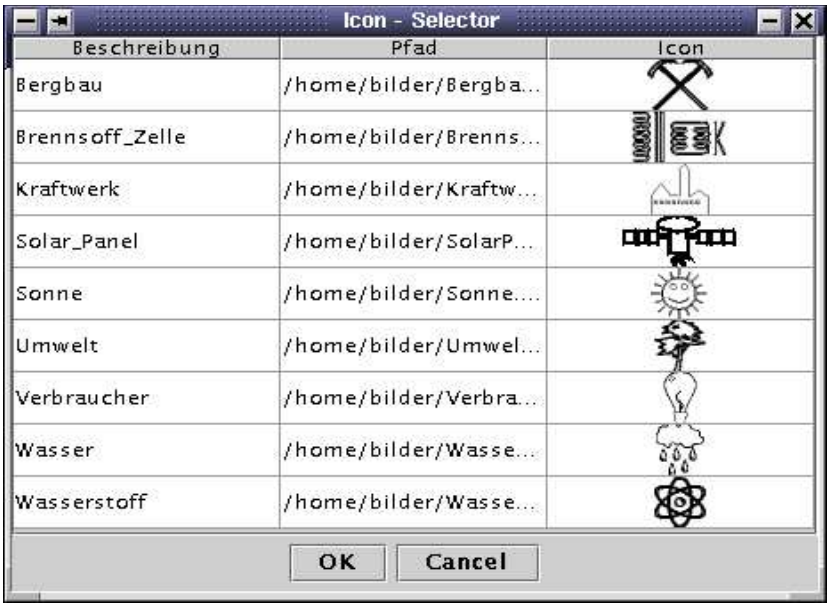
Neben der Namensänderung bietet dieser Dialog die Möglichkeit, den Knoten für das strict degree matching zu markieren. Außerdem kann der Anwender mit Hilfe dieses Dialogs beeinflussen, ob es sich um ein eindeutiges Label oder um eine Variable handelt.



Die beiden ersten Radiobutton betreffen die Einstellungen der Knoten des Contextgraphen. Wird der erste Radiobutton (**Context – Node**) aktiviert, wird aus dem Difference – Node ein Context – Node. Dieser kann also eine Verbindung zu dem Target– bzw. Replacementgraph eingehen. In Folge dieser Einstellung wird der Knoten auf den beiden ersten Tabpages eingefügt und kann vom Anwender benutzt werden.

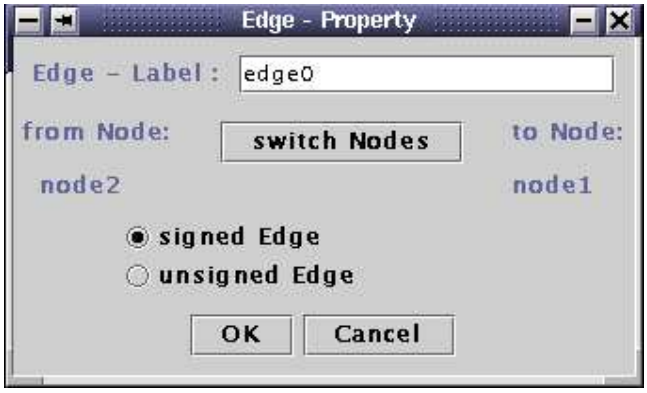
Der zweite Radiobutton (**Labelless**) schaltet einen Context – Node labellos. Dies bewirkt, daß in der Einbettungsvorschrift des Rule – Definition – File an Stelle des Knotenlabel die Variable **L** erscheint.

Neben den Einstellungen, die direkt das Rule – Definition – File betreffen, kann der Anwender mit Hilfe dieses Dialogs dem Knoten eine Grafikdatei zuweisen. Dazu muß eine Bilderdatenbank eingeladen worden sein **(III 2.4.3)**. Wird der **new Icon** – Button innerhalb des Dialogs betätigt, erscheint ein Dialog, der alle möglichen Graphik – Dateien zur Auswahl darstellt. Wird anschließend der Knoten – Property – Dialog beendet, aktualisieren sich die entsprechenden Knoten.



2.3.2 Der Kanten – Property – Dialog

Dieser Dialog bietet dem Anwender die Möglichkeit, aus einer gerichteten Kante eine ungerichtete oder umgekehrt zu machen. Zusätzlich ist es in diesem Dialog möglich, die Richtung der Kante zu ändern.

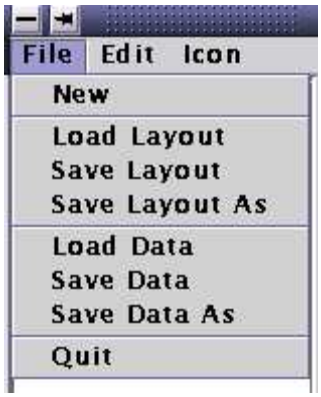


2.4 Die RUBED – Menü – Leiste

Die Menü – Leiste des RUBED besteht aus drei Untermenüs. In dem ersten Untermenü (**File**) befindet sich die allgemeine Projektbearbeitung. Die Möglichkeiten zur Änderung der Einstellungen der einzelnen Knoten, Kanten und Regeln sind in dem zweiten Untermenü (**Edit**) zusammengefaßt. Das dritte Untermenü (**Icon**) gibt dem Anwender die Möglichkeit, Bilder einzuladen, die den Knoten zugeordnet werden können.

2.4.1 Projektbearbeitung des RUBED

Der RUBED kann jeweils nur ein Projekt gleichzeitig bearbeiten. Die Projektverwaltung befindet sich unter dem Menüpunkt **File**.



Hier ist es möglich, ein neues Projekt anzulegen (**New**), eine Ansicht zu laden (**Load Layout**) oder zu speichern (**Save Layout // SaveLayout As**), ein Rule – Definition – File zu erzeugen (**Save Data // Save Data As**) oder einzuladen (**Load Data**). Zusätzlich kann in diesem Untermenü das Eingabetool beendet werden (**Quit**).

2.4.2 Objektbearbeitung des RUBED

Das Untermenü **Edit** dient dem Anwender dazu, die Objekte des Projekts zu verwalten und zu ändern. Zu diesen Objekten gehören die Knoten und Kanten aber auch die einzelnen Regeln eines Projekts.

Mit **new Rule** kann eine neue Regel in das Projekt eingefügt werden. Diese Aktion ist zu Beginn der Arbeit notwendig, um Knoten und Kanten einfügen zu können.

Alle Objekte können jederzeit aus dem Projekt entfernt werden. Dazu müssen sie in der Projekt – Ansicht markiert und mit Hilfe des **delete** Menüpunkts gelöscht werden. Wird eine Regel aus dem Projekt gelöscht, werden automatisch alle Knoten und Kanten dieser Regel mitgelöscht. Bei dem Löschen eines Knoten werden alle Kanten, die in Verbindung mit diesem Knoten stehen, ebenfalls gelöscht.

Um die Arbeit des Anwenders zu vereinfachen kann mit dem Menüpunkt **copy Rule** eine komplette Regel kopiert werden. Die Anordnung der einzelnen Knoten und Kanten bleibt dabei erhalten. Lediglich ein neuer Name wird der Regel zugeteilt.

Mit dem **Layout** Untermenü können die Knoten der einzelnen Tabpages automatisch angeordnet werden. Dabei werden die Knoten nach Art eines Schachbretts angeordnet. Da diese Funktion lediglich dazu dient Überschreibungen von Knoten zu entfernen, wird auf eine intelligente Anordnung der Knoten verzichtet.

Die Arbeitsweise des **Property** Menüpunkts wurde bereits in **III 2.3** beschrieben.

2.4.3 Knoten – Icons

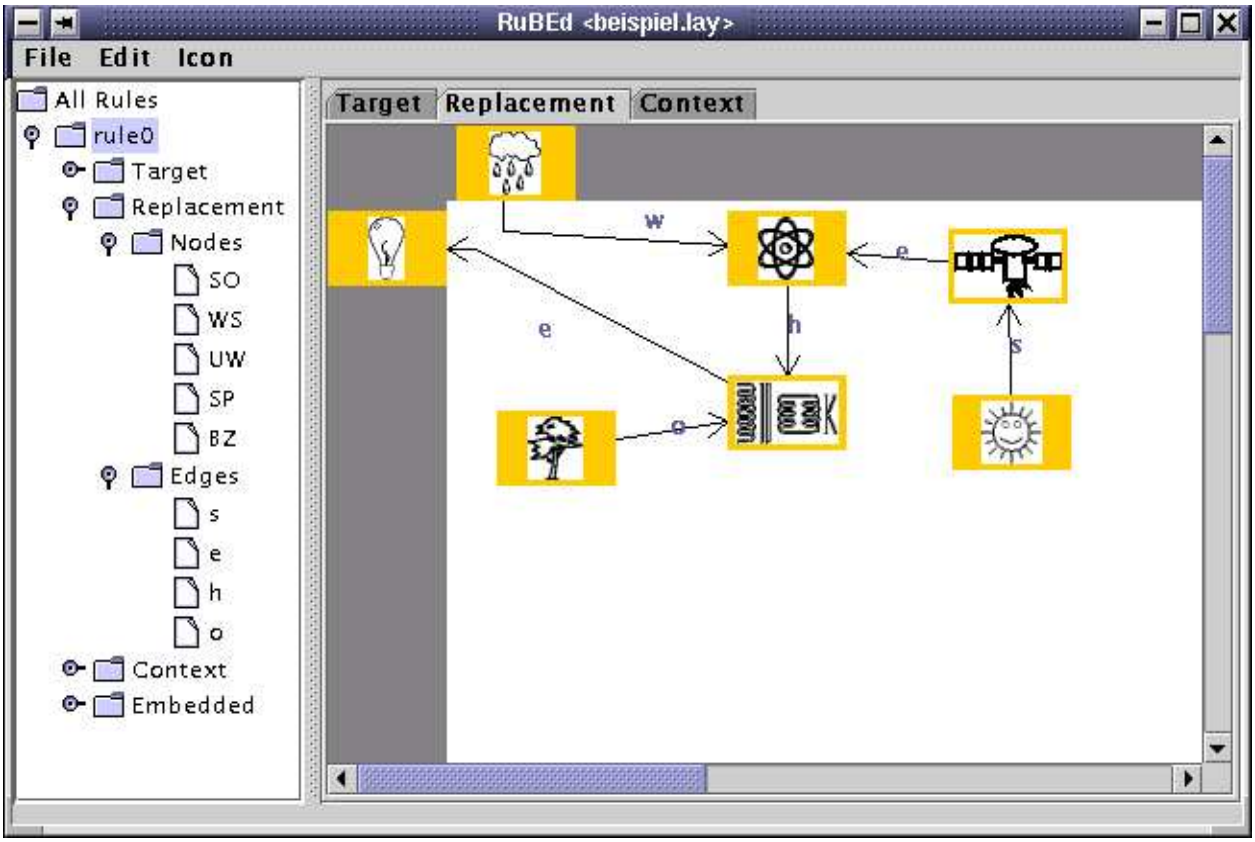
Neben der Projektverwaltung und der Objektbearbeitung bietet das Eingabetool die Möglichkeit, dem Knoten nicht nur Label, sondern auch Bilder zuzuordnen. Als Bilder können alle gif – Bilddateien verwendet werden. Um diese Bilder dem Eingabetool zugänglich zu machen, ist es notwendig in einer Datei alle gewünschten Bilddateien mit ihrem absoluten Pfad und einer Beschreibung anzugeben. Diese Datei erhält die Endung ***idb*** (**I**con – **D**ata**b**ase). Die Datei besteht also aus Zeilen der Form:

Beschreibung; absoluter Pfad

Eine Datei könnte also beispielsweise diesen Inhalt haben:

Bergbau; /home/bilder/Bergbau.gif
Brennstoff_Zelle; /home/bilder/BrennstoffZelle.gif
Kraftwerk; /home/bilder/Kraftwerk.gif
Solar_Panel; /home/bilder/SolarPanel.gif
Sonne; /home/bilder/Sonne.gif
Umwelt; /home/bilder/Umwelt.gif
Verbraucher; /home/bilder/Verbraucher.gif
Wasser; /home/bilder/Wasser.gif

Die Icon – Database wird über den Menüpunkt **Icon –> Load Icon – Database** eingeladen und interpretiert. Über den Knoten – Property – Dialog kann jedem Knoten ein Bild zugeordnet werden **(III 2.3.1)**.



Kapitel IV
Gegenüberstellung RUBED <–> PROGRES

Dieses Kapitel vergleicht den RUBED mit PROGRES. Dabei wird besonderer Wert auf die Design – Anforderungen gelegt, die an den RUBED gestellt wurden. Diese Anforderungen werden im ersten Teil dieses Kapitels dargestellt. Im zweiten Teil wird PROGRES kurz vorgestellt, während der eigentliche Vergleich im dritten Teil dieses Kapitels zu finden ist.

1 Design – Anforderungen an den RUBED

Der RUBED ist speziell für die Eingabe von Graphgrammatiken konzipiert worden. Dabei wurde besonderer Wert auf eine möglichst einfache und intuitive Bedienung gelegt. So werden die Transformationsregeln grafisch aufgebaut, was den Vorteil hat, daß der Anwender gedanklich auf der Graphebene arbeiten kann und nicht zwischen graphischer Denkweise und textueller Definition wechseln muß. Dieser Effekt wird durch die Benutzung der Maus in den überwiegenden Anwendungen noch verstärkt. Außerdem kann der Anwender dem Knoten beliebige Icon zuordnen, um eine für ihn möglichst realitätsnahe Darstellung des Graphen zu erreichen.

Neben der einfachen Bedienung ist der RUBED speziell für die Erzeugung des Rule – Definition – File konzipiert worden. Die Label der Knoten und Kanten sind also die einzigen Informationsträger der Knoten und Kanten innerhalb des Editors, da der spätere Einsatz des Rule – Definition – File ein auf Labeln basierte Graphersetzung ist. Der RUBED beschränkt sich daher auf die Darstellung der Label, der Knoten und Kanten.

Die dritte Anforderung an den RUBED betrifft den ausschließlichen Einsatz des Editors in technischen Systemen. Daher müssen keine Sonderfälle beachtet werden, die beispielsweise eine besondere Anordnung der Knoten in dem Graphen fordern würden. Außerdem handelt es sich bei den Knoten um real existierende Gegenstände, die außerhalb des Graphen bereits definiert wurden. Auf diese Definition kann uneingeschränkt zugegriffen werden, so daß keine Möglichkeiten für Kommentare oder nähere Beschreibungen der Knoten und Kanten geschaffen werden müssen.

Zusammenfassend kann man sagen, daß bei der Entwicklung des RUBED folgenden drei Punkten besondere Beachtung geschenkt wurde:

1. **Der Eingabeeditor muß graphikbasiert arbeiten und einfach zu bedienen sein.**
2. **Der Eingabeeditor soll möglichst klein sein.**
3. **Der Eingabeeditor soll speziell für technische Systeme konzipiert sein.**

2 Beschreibung von PROGRES

In diesem Abschnitt wird die Graphen – Definitions – Programmiersprache PROGRES **[Schürr2000a]** vorgestellt. Hierbei ist zu bemerken, daß innerhalb dieser Arbeit nicht alle Funktionen und Möglichkeiten von PROGRES genannt werden können. Die Beschreibung beschränkt sich daher auf die für den anschließenden Vergleich notwendigen Teile von PROGRES. Auch werden diese Teile nicht in allen Einzelheiten besprochen, sondern nur die Ideen und grundsätzlichen Möglichkeiten dargelegt. Weitere Informationen zu PROGRES finden sich in **[Schürr2000a, Schürr2000b]**.

PROGRES hat sich im Laufe der Jahre zu einem Tool mit umfangreichen Möglichkeiten für die Arbeit mit Graphen entwickelt. So ist es nicht nur möglich, Graphen zu definieren, sondern auch Anfragen an diese Graphen zu stellen bzw. einzelne Teile durch neue zu ersetzen.

Die Arbeit mit PROGRES geschieht sowohl text–, als auch grafikbasiert. PROGRES ist als eine visuelle Programmiersprache entwickelt worden, bei dessen Entwicklung aber darauf geachtet wurde, gleichzeitig die textuelle Programmierung auch weiterhin zu ermöglichen. Es wurde versucht, die graphikbasierte Beschreibung auf angemessene Bereiche zu beschränken.

2.1 Definition eines Graphen in PROGRES

PROGRES betrachtet Graphen als die Definition von Knoten und Kanten, die Label enthalten können.

Für die Knoten ist es zusätzlich möglich, Attribute zu definieren. Als Datentypen dieser Attribute können neben den einfachen Integer – und String – Datentypen auch benutzer–definierte Datentypen vorkommen. Diese werden vom Anwender in einer eingebetteten Programmiersprache vorgenommen. Als Programmiersprachen kommen dabei C und Modula in Frage.

Eine vereinfachte Darstellung einer Turbine kann folgendermaßen definiert werden:

```
node class TURBINE                    (*Ein Knoten mit dem Label TURBINE wird erzeugt*)
                                      (*Der Knoten at folgende Attribute*)
    Power : integer = 20000;
    Powerunit . string = "KW"
    Rotornumber : integer = 100;
    ...
end;                                  (*Die Definition wird beendet*)
```

Die Verbindungen im Graphen sind stets gerichtete Kanten, bei denen eine Definition von Attributen nicht möglich ist. Für eine spätere Analyse der Kanten können den anliegenden Knoten aber Kardinalitäten zugeordnet werden.

Eine Kante von einer Turbine zu den Verbrauchern wird folgendermaßen definiert:

edge type *strom : TURBINE [1:1] → VERBRAUCHER [0:n];*

Die Kardinalitäten sagen aus, daß genau eine Turbine beliebig vielen Verbrauchern Strom liefern darf.
Zusätzlich zu diesem Konzept wurde bei PROGRES ein Ansatz ähnlich der objektorientierten – Programmierung verfolgt. So ist es möglich, Oberklassen zu definieren, die bereits einige Attribute für ihre Kind –Knoten definieren.

Neben den Knoten und Kanten können in PROGRES Funktionen definiert werden. Diese Möglichkeit wird hier aber nicht weiter beschrieben, da eine Funktion bei der Definition von Graphgrammatiken nicht unbedingt erforderlich ist.

2.2 Anfragen an einen Graphen in PROGRES

In PROGRES existiert eine umfangreiche Möglichkeit, Anfragen an einen existierenden Graphen zu stellen. PROGRES bietet sowohl die Möglichkeit nach Subgraphen zu suchen, als auch weitere Anforderungen an einen zu suchenden Subgraphen zu stellen. Solche Anforderungen können Bedingungen im Graphen oder auch konkrete Werte der Attribute sein. Zusätzlich kann eine Bedingung an den Graphen durch die Abwesenheit eines Knoten oder Kante definiert werden.

Die Ausgabe einer solchen Suche kann sowohl eine komplette Liste aller gefundenen Subgraphen, als auch nur einen Subgraph enthalten. Zusätzlich können beliebige Attribute jedes gefundenen Subgraphen ausgegeben werden. Anschließend können die Ergebnisse in beliebiger Art und Weise weiterverarbeitet werden.

Die Definition einer Anfrage in PROGRES wird sowohl mit textuellen, als auch mit graphischen Mitteln vorgenommen. Die zu suchenden Subgraphen werden dabei graphisch beschrieben, während eine Verknüpfung der Ergebnisse auf textueller Basis definiert wird.

Neben der Suche nach Subgraphen können auch iterativ Wege in einem Graphen gesucht werden. Zu diesem Zweck stehen in PROGRES Kontrollstrukturen zur Verfügung. So können Wege mit Hilfe von Schleifenaufrufen gesucht werden. Diese Möglichkeit ist jedoch für die Definition einer Graphgrammatik nicht sinnvoll, da Graphgrammatiken auf der Suche nach Subgraphen basiert. Diese Methode wird daher hier nicht weiter erläutert.

2.3 Ersetzen eines Subgraphen in PROGRES

In PROGRES besteht die Möglichkeit, einzelne Subgraphen durch neue zu ersetzen. Dazu werden die Subgraphen wie in **IV 2.2** beschrieben gesucht und anschließend ersetzt. Zusätzlich können auch bei dem Ersetzen Kontrollstrukturen eingesetzt werden. Diese Möglichkeit wird hier aber ebenfalls nicht weiter beschrieben.

Die Definition einer Ersetzungsvorschrift wird in zwei Bereiche geteilt. Die linke Seite beschreibt den zu ersetzende Subgraphen mit allen Bedingungen, während die rechte Seite den neuen Subgraphen darstellt. Als Bedingungen für den zu ersetzenden Graphen kommen die in **IV 2,2** beschriebenen Bedingungen in Frage.

Das Ersetzen eines Subgraphen geschieht in PROGRES nach den vier folgenden Regeln:

1. Alle positiven Knoten, die sowohl auf der linken, als auch auf der rechten Seite vorkommen werden mit allen Attributen und zwischen ihnen liegenden Knoten gesichert.
2. Alle positiven Knoten und alle Kanten auf der linken Seite, die kein Gegenstück auf der rechten Seite haben, werden gelöscht. Diese Regel wird nur bei nicht geschützten Knoten und Kanten angewendet.
3. Alle Knoten und Kanten der rechten Seite, die kein Gegenstück auf der linken Seite haben, werden eingefügt.
4. Neue Attribute werden in dem neuen Subgraphen eingefügt. Dabei können sowohl neue Übergabeparameter, als auch alte Attributwerte benutzt werden.

Neben dem einfachen Ersetzen eines Subgraphen können Einbettungsvorschriften angegeben werden, um den neuen Subgraphen in den Hostgraphen einzubinden. Dabei können die in dem ursprünglichen Graphen vorhandenen Knoten auf einen neuen Knoten umgebogen werden. Diese Methode ist mit den Einbettungsvorschriften der Design – Graphgrammatik vergleichbar.

3 Vergleich des RUBED mit PROGRES

Dieser Abschnitt vergleicht die Möglichkeiten und die Umsetzung des RUBED mit denen von PROGRES. Dabei wird besonderes Augenmerk auf die in **IV 1** beschriebenen Anforderungen gelegt.

Wie bereits in dem vorherigen Abschnitt beschrieben, bietet PROGRES sehr umfangreiche Möglichkeiten Graphen zu manipulieren. Dabei beschränken sich die Möglichkeiten von PROGRES nicht nur auf die Manipulation von Graphen allein. Es können außerdem die Graphen vollständig aufgebaut werden bzw. mit der Definition von Produktionsregeln induktiv definiert werden.

Die Möglichkeiten gehen also sehr weit über die eigentlich geforderten hinaus. Die einzige Einschränkung muß bei der Definition des Contextgraph gemacht werden. Während die Graphgrammatik die Definition des Contextgraph explizit vorsieht, muß dieser bei PROGRES implizit auf der linken und rechten Seite der Transfomationsregeln definiert werden.

Der große Funktionsumfang von PROGRES hat zur Folge, daß die Bedienung sehr komplex und aufwendig erlernbar ist. So muß der Anwender nicht nur die grafische Definition verstehen, sondern zusätzlich eine, wenn auch kleine Programmiersprache, lernen. Diese lange Einarbeitungsphase soll durch die Graphgrammatiken vermieden werden. Mit dem Konzept soll erreicht werden, daß der Anwender nur grafische Eingaben vornehmen muß.

Da der RUBED nur für die Definition der Transformationsregeln konzipiert wurde, kann auf eine umfangreiche Programmiersprache verzichtet werden. Der RUBED bietet ein sehr einfaches und leicht verständliches Frontend, das der Anwender ohne lang Einarbeitungsphase benutzen kann. Dieser Vorteil ist der wesentliche Grund, der zur Entwicklung des RUBED geführt hat.

Da die Anwendung der Graphgrammatiken nur auf den Labeln der Knoten und Kanten beruht, kann auf die Definition von Attributen, wie bei PROGRES möglich, komplett verzichtet werden. Dies ist ein weiterer Grund keine Programmiersprache zu verwenden. Der Verzicht auf Attribute stellt aber keine wesentliche Einschränkung des Funktionsumfangs dar, da es theoretisch möglich ist alle Informationen in den Labeln zu vereinen. Dies Vorgehen ist jedoch größtenteils unnötig, da die Graphgrammatiken im Bereich der technischen Systeme eingesetzt werden sollen. Bei der Anpassung oder Optimierung solcher Systeme ist jedoch die genaue Beschreibung der einzelnen Einheiten entweder unnötig oder sie ist bereits in dem Label enthalten. Mit Hilfe der Vereinfachung der Einheiten des technischen Systems auf Label, ist der RUBED in der Lage diese Einheiten sehr einfach und verständlich grafisch darzustellen.

PROGRES ist in der Lage nahezu beliebige Bedingungen an die Ersetzung eines Subsystems zu knüpfen. Diese Eigenschaft wird aber bei der Definition von Graphgrammatiken nicht benötigt, da alle Informationen der Subsysteme bereits in den Labeln der Knoten enthalten sind und diese bei der Ersetzungsprozedur implizit ausgewertet werden. Auch die Definition von Knoten – Oberklassen ist aus diesem Grund nicht für Graphgrammatiken geeignet, da sich Ober– und Kindknoten in dem Label unterscheiden und somit in jedem Fall anders behandelt werden. Außerdem fehlt bei der Graphgrammatik der hauptsächliche Grund für die Einführung einer Objekt– Orientierten – Komponente, da es keine vererbbaren Attribute gibt.

Zusammenfassend kann man sagen, daß der RUBED die Teilkomponente von PROGRES, die für die Definition der Transfomationsregeln einer Graphgrammatik nötig wäre, durch ein intuitiv zu benutzendes Tools ersetzt, daß sich die einschränkenden Eigenschaften der Graphgrammatiken und der technischen Systeme zu nutze macht, um eine einfache Darstellung zu realisieren.

Anhang A Literaturliste

[EEKR1999] **H. Ehrig, G. Engels, H.–J. Kreowski und G. Rozenberg:**
Handbook of Graph Grammars and Computing by Graph Transformation,
volume 2 Applications, Languages and Tools.
World Scientific 1999

[EKMR1999] **H. Ehrig, H.–J. Kreowski, U. Montanari und G. Rozenberg:**
Handbook of Graph Grammars and Computing by Graph Transformation,
volume 3 Concurrency, Parallelism and Distribution.
World Scientific 1999

[ESP1998] **M. van Eekelen, S. Smetsers und R. Plasmeijer:**
Graph Rewriting Systems for Functional Programming Languages.
Technical report, Computing Science Institute,
University of Nijmegen 1998

[Hawlitzek2000] **F. Hawlitzek:**
Java 2.
Addison – Wesley 2000

[Jungnickel1999] **D. Jungnickel:**
Graphs, Networks and Algorithms,
volume 5 of Algorithms and Computation in Mathematics.
Springer 1999

[Kaul1986] **M. Kaul:**
Syntaxanalyse von Graphen bei Präzedenz – Graph – Grammatiken.
PhD thesis, Fakultät für Mathematik und Informatik,'
Universität Passau 1986

[Kaul1987] **M. Kaul:**
Practical Applications of Precedence Graph Grammars.
In H. Ehrig, M. Nagel, G. Rozenberg und A. Rosenfeld:
Graph Grammars and Their Application to Computer Science.
Nummer 291 in Lecture Notes in Computer Science, Seite 326–342.
Berlin 1987 Springer–Verlag

[Korff1991] **M. Korff:**
Application of Graph Grammars to Rule–based Systems.
In H. Ehrig:
Graph Grammars and Their Application to Computer Science.
Nummer 532 in Lecture Notes in Computer Science, Seite 505–519.
Berlin 1991 Springer–Verlag

[Krüger2000] **G. Krüger:**
Go To Java 2, Handbuch der Java–Programmierung, 2. Auflage.
Addison–Wesley, 2000

[Lichtblau1991] **U. Lichtblau:**
Recognizing Rooted Context−Free Flowgraph Languages in Polynomial Time.
In H. Ehrig: Graph Grammars and Their Application to Computer Science.
Nummer 532 in Lecture Notes in Computer Science, Seite 538−548.
Berlin 1991 Springer−Verlag

[RN1995] **S. Russel und P. Norvig:**
Artificial Intelligence: A Modern Approach.
Prentice − Hall, Englewood Cliffs, N.J. 1995

[RS1995] **J. Rekers und A. Schürr:**
A Graph Grammar Approach to Graphical Parsing.
Technical Report 95−15, Deparment of Computer Science,
Leiden University 1995

[Rozenberg1997] **G. Rozenberg:**
Handbook of Graph Grammars and Computing by Graph Transformation,
volume 1 Foundations.
World Scientific 1997

[Schürr1997] **A. Schürr:**
Developing Graphical (Software Engineering) Tools with PROGRES.
In Proc. ICSE, Seite 618−619.
IEEE Computer Society Press 1997

[Schürr2000a] **A. Schürr:**
PROGRES for Beginners.
RWTH Aachen 2000

[Schürr2000b] **A. Schürr:**
A Guided Tour Through the PROGRES Environment.
University of the Federal Armed Forces Munich 2000

[SSK2001] **A. Schulz, B. Stein und A. Kurzok:**
On Automated Design of Technical Systems.
Fachbereich Informatik, Universität Paderborn 2001

[SWZ1995] **A. Schürr, A. Winter und A. Zündorf:**
Visual Programming with Graph Rewriting Systems.
In Proc. 11th Intl. IEEE Symposium on Visual Languages.
IEEE Computer Society Press 1995