AGG / GenGED

Iris Wangerin
235 835

Betreut von Dipl.-Inf. Ulrike Ranger

Zusammenfassung

GenGED ist ein Werkzeug, welches für eine visuelle Sprache einen visuellen Editor generiert. In diesem können Diagramme editiert, geparst und Abläufe auf dem Diagramm simuliert werden. Die gewünschte visuelle Sprache wird mit Hilfe eines Alphabet- und eines Grammatikeditors eingegeben. Unterstützt wird GenGED von dem integrierten Graphtransformationsprogramm AGG. AGG wendet auf einen eingegebenen attributierten Graphen Regeln an, nach denen der Graph automatisch transformiert wird. Diese Transformationen sind dynamische Zustandsänderungen des Graphen.

Inhaltsverzeichnis

1 Einleitung

Der Mensch nimmt Informationen schneller auf, wenn sie in Bildern dargestellt sind statt in reiner Textform. Komplexe Softwaresysteme lassen sich durch Diagramme abstrahieren, sie werden übersichtlich dargestellt und schneller verstanden. Zum Beispiel können Zusammenhänge zwischen Komponenten in umfangreichen Softwareprojekten durch Diagramme übersichtlich und verständlich dargestellt werden. Mit der Komplexität der heutigen Software sind visuelle Sprachen wie zum Beispiel Klassendiagramme, Zustandsdiagramme oder Petrinetze unverzichtbare Hilfsmittel geworden. Mit Klassendiagrammen können zum Beispiel Programmarchitekturen modelliert werden. Zustandsdiagramme und Petrinetze können dynamische Veränderungen im Programmablauf und den Informationsfluß während der Programmausführung modellieren.
Visuelle Sprachen definieren sich über visuelle Grammatiken, denen visuelle Alphabete und Regeln zugrunde liegen. Die Entwicklung verschiedener visueller Sprachen kann mit einem Generator unterstützt werden. GenGED (*Generation of Graphical Environments for Design*) ist ein solcher Generator. In der zu generierenden Umgebung können syntaxgesteuert oder freihand Diagramme der gewünschten Sprache erstellt werden.
Die Idee hinter GenGED ist, daß modellgetriebene Softwareentwicklung durch beliebige visuelle Sprachen unterstützt wird. Durch aussagekräftige Symbole und eingeschränkte Operationen lassen sich die durch GenGED generierbaren Sprachen für sehr eingegrenzte Domänen konkretisieren. Zum Beispiel läßt sich eine Sprache generieren, mit der nur Ausleihprozeduren einer Videothek simuliert werden, dafür aber können die Symbole aussehen wie Videokassetten, Häuser und Menschen. Die generierte Sprache wird dem Anwender auch unsinnige Entscheidungen verbieten. Zum Beispiel könnte eine Sprache für die Videotheksausleihe dem Anwender keine Möglichkeit bieten, Kunden an andere Kunden zu verleihen, Videos als Arbeitskräfte einzustellen oder anderes nicht Vorgesehenes zu modellieren.
GenGED basiert auf der Transformation von Graphen, welche von dem integrierten Programm AGG (Kapitel 2) unterstützt wird. Die Transformation wird mit Hilfe von Graphregeln durchgeführt, welche aus zwei Graphen bestehen. Eine Transformation verändert die Graphstruktur oder führt Berechnungen auf Attributen durch. In GenGED werden Diagramme visuell mit Hilfe von Editoren definiert (Kapitel 3). Das visuelle Alphabet wird im Alphabeteditor zusammengesetzt. Die Graphtransformationen basieren auf Graphregeln, welche im Grammatikeditor definiert werden. In einem Spezifikationseditor wird die Entwicklungsumgebung konfiguriert. GenGED generiert dann aus dem Alphabet und den Graphregeln automatisch eine Entwicklungsumgebung für die spezifizierte Sprache (Kapitel 4).

Dazu gehört auch ein Simulator, der Simulationsschritte auf einem Laufzeitgraphen durchführt. Die Entwicklungsumgebung ist eine eigenständige Komponente und kann unabhängig von GenGED genutzt werden.

2 AGG

AGG (*Attributed Graph Grammar*) [6; 7; 5] ist die wichtigste Komponente von GenGED. AGG wird seit 1997 unter der Leitung von Gabriele Taentzer an der TU Berlin entwickelt. In GenGED wird AGG als Interpreter für Graphtransformationen genutzt. AGG unterstützt die Erstellung und Transformation attributierter Graphen. Mit einem Typgraphen werden die Objekttypen und die Graphstruktur festgelegt und darauf Attribute deklariert (Abschnitt 2.1). Der Laufzeitgraph instanziiert den Typgraphen. Ein Startgraph wird vom Benutzer geladen oder gezeichnet und von AGG als Laufzeitgraph mit Hilfe der Regeln transformiert.
Auf der Regelebene werden Graphregeln definiert (Abschnitt 2.2). Eine Regel besteht aus zwei Graphen. Eine Transformation eines Laufzeitgraphen wird definiert durch den Unterschied zwischen der linken und der rechten Regelseite. Die linke Seite der Regel stellt Anwendungsbedingungen dar. Zusätzlich können beliebig viele negative Anwendungsbedingungen als Graphen definiert werden. Durch die rechte Regelseite wird die Transformation des Laufzeitgraphen definiert. Wenn eine Regel angewendet wird, werden Morphismen von der linken Seite auf den Laufzeitgraphen gesucht. Falls eine Graphstruktur der linken Seite genügt, kann die Regel angewendet werden (Abschnitt 2.3) und der Graph wird transformiert.

2.1 Graphen

Im folgenden wird zwischen mehreren Ebenen von Graphen unterschieden. Zunächst wird ein Graphmodell vorgestellt, wie es allgemein in AGG verwendet wird. Dann wird der Unterschied und Zusammenhang eines Laufzeitgraphen zu einer Typisierung hervor gehoben.

Das Graphmodell von AGG: Im Allgemeinen besteht ein Graph aus beliebig vielen gerichteten Kanten und Knoten, den Graphobjekten. Jedes Objekt ist von genau einem bestimmten Typ. In AGG muß eine Kante zwei Knoten miteinander verbinden, einen Quellknoten und einen Zielknoten. Dabei sind Schleifen erlaubt, d.h. der Quell- und Zielknoten kann ein und das selbe Objekt sein. Es können mehrere Kanten verschiedenen oder gleichen Typs zwischen denselben Knoten bestehen. Die Knoten und Kanten können Attribute haben. Zu einem Attribut gehört ein Attributname, ein Datenttyp und ein Wert. Der Datentyp kann ein beliebiger Java-Datentyp sein, wie zum Beispiel String und Integer. Es können jedoch keine neuen Datentypen direkt in AGG definiert werden.

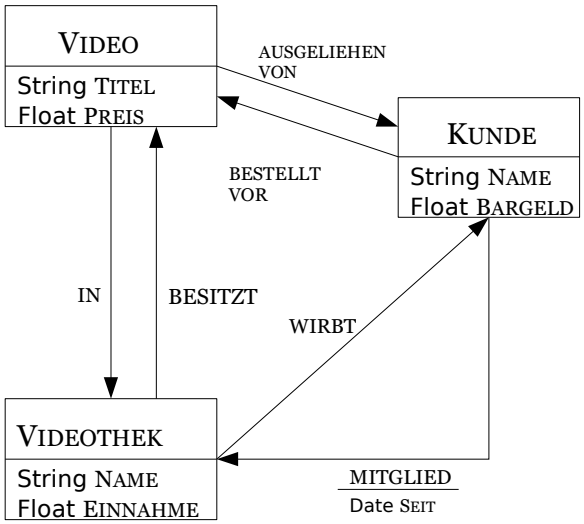

Abbildung 1: Ein Typgraph für eine Videothek

Die Attributnamen und Datentypen der Attribute werden für jeden Objekttypen deklariert. Alle Objekte desselben Typs haben also dieselben Attribute, nur die Werte unterscheiden sich.

Typgraphen: Ein Typgraph legt die Struktur und die Komponenten von Graphen fest. Wird ein neuer Typ definiert, kann ihm in AGG eine bestimmte geometrische Form (Rechteck, Ellipse, Kreis...) und Farbe zugewiesen werden. Für jeden Typ können beliebig viele Attribute deklariert werden. Der Typgraph schränkt die Graphinstanzen insofern ein, daß deren Kanten und Knoten nur dort auftauchen dürfen, wo sie im Typgraphen vorgesehen sind.

Laufzeitgraphen: Ein Laufzeitgraph ist eine Instanz des Typgraphen. Seine Objekte können nur von den Typen sein, die auch im Typgraphen existieren. Die Relationen der Objekte in dem Laufzeitgraphen müssen den Relationen ihrer Typen im Typgraphen entsprechen. Während im Typgraphen jeder Typ einzigartig ist, können in den Graphinstanzen auch mehrere oder keine Objekte eines Typen existieren. Es ist auch möglich, im Typgraphen Kardinalitäten für Kanten zu setzen, die sicher stellen, daß im Laufzeitgraphen jede Kante nur höchstens eine bestimmte Anzahl von Knoten bestimmter Typen verbindet (Maximumgrenze)

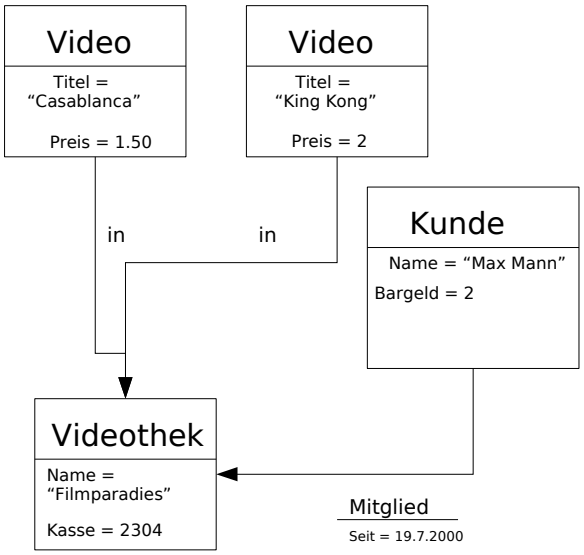

Abbildung 2: Eine Instanz des Typgraphen aus Abbildung 2

oder daß mindestens eine bestimmte Anzahl existieren muß (Minimumgrenze). Jedes Objekt hat genau die Attribute seines Typs, die in den Instanzgraphen Werte haben. Die Abbildungen 1 und 2 zeigen den Zusammenhang eines einfachen Typgraphen und einer Instanz. Modelliert wird eine Videothek, in der es Kunden gibt und Videos. Die Videos können sich in einer Videothek befinden oder entliehen sein. Der Kunde kann Mitglied der Videothek sein. Im Instanzgraphen dürfen Videos, Kunden und Videotheken existieren, sowie Kanten vom Typ in, ausgeliehen von, Mitglied, besitzt und bestellt vor. Für die Kanten ist genau festgelegt, zwischen welchen Knotentypen sie sich befinden dürfen. Für einige der Typen sind Attribute deklariert. So hat das VIDEO einen TITEL vom Datentyp String und einen PREIS vom Datentyp Float.
Im Laufzeitgraphen aus Abbildung 2 werden zwei VIDEOS dargestellt mit Namen und Verleihpreis. Sie befinden sich in einer Videothek, die einen Namen und einen Kassenbestand hat. Außerdem ist ein Kunde mit Namen und Bargeld seit dem 19.7.2000 Mitglied in dieser Videothek.
Die Benutzung von Typgraphen in AGG ist optional. Sie hat jedoch den Vorteil, daß Typgraphen bereits Einschränkungen an die Laufgraphengraphen stellen.

Wird ein Typgraph benutzt, können nur Laufzeitgraphen erstellt werden, die diesen instanziieren.

2.2 Regeln

Eine Regel definiert eine dynamische Veränderung eines Graphen. Die Regel besteht aus zwei Graphen, die Instanzen eines vorher definierten Typgraphen sind. Die Objekte sind als Variabeln aufzufassen, die im Laufzeitgraphen an konkrete Objekte gebunden werden. Die linke Regelseite L steht für den 'Vorher'-Zustand des Graphen und die rechte Regelseite R für den 'Nachher'-Zustand. Die linke Seite der Regel stellt Bedingungen an den Laufzeitgraphen G dar. Nur wenn diese Bedingungen erfüllt sind, kann die Regel ausgeführt werden. Es wird überprüft, ob ein Graphmorphismus von der linken Seite auf den Laufzeitgraphen existiert. Das bedeutet, daß im Laufzeitgraphen eine Struktur existieren muß, deren Objekte von denselben Typen sind wie die aus dem linken Regelgraphen und die auf die gleiche Weise miteinander verknüpft sind. Es existiert ein Morphismus zwischen den beiden Seiten der Regel. Er wird durch gleiche Nummerierung der sich aufeinander beziehenden Graphobjekte ausgedrückt. Die rechte Regelseite stellt keine Bedingung an den Laufzeitgraphen dar, sondern wird allein für die Transformation gebraucht. Der Unterschied zwischen der linken und der rechten Regelseite definiert die Transformation des Laufzeitgraphen.
Zum Beispiel existiert ein Morphismus von der linken Regelseite aus Abbildung 3 auf den Graphen aus Abbildung 2. Da es dort zwei mögliche Übereinstimmungen gibt, kann der Spezifikator eine passende Übereinstimmung selber wählen, oder er wählt einzelne Objekte aus und der AGG vervollständigt die Übereinstimmung. Eine ausgewählte Übereinstimmung wird durch gleiche Nummererierung der Objekte mit der linken Regelseite dargestellt. Wird im Laufzeitgraphen trotz Mehrdeutigkeiten keine Übereinstimmung vom Benutzer ausgewählt, wählt AGG zufällig eine aus.

Negative Anwendungsbedingungen Die bislang beschriebenen Bedingungen auf der linken Regelseite sind alle *positiv* in dem Sinne, daß sie vorgeben, was im Laufzeitgraphen existieren muß. Es gibt auch die Möglichkeit, negative Bedingungen auszudrücken, also Sachverhalte, die im Instanzgraphen *nicht* existieren dürfen, wenn die Regel angewendet werden soll. In AGG werden für diese negative Bedingungen neue Graphen N eingeführt. Sie drücken negativen Bedingungen auf die gleiche Weise aus, wie die linke Regelseite die positiven Bedingungen.
Beispielsweise soll ein Videothekskunde kein Video ausleihen können, wenn er kein Bargeld bei sich hat (Abbildung 3). In der linken Regelseite kann nicht ausgedrückt werden: "If $Kunde.Bargeld > 0$ then do ...". Es ist ein Nachteil von

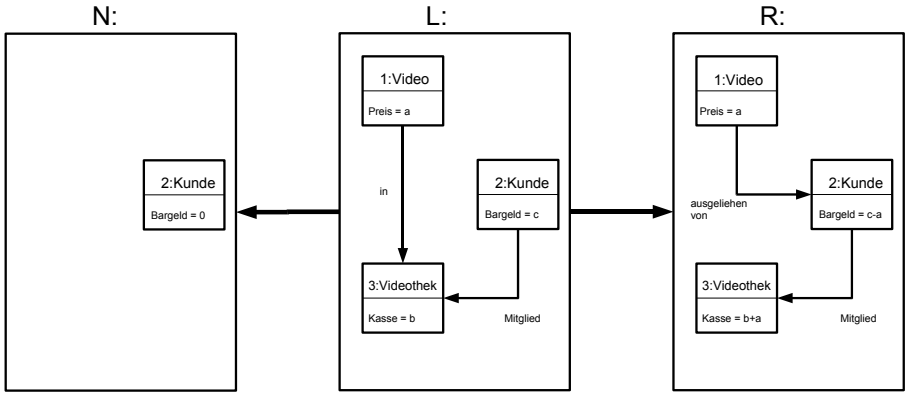

Abbildung 3: Regel mit negativer Bedingung: VideoAusleihe

AGG, daß Ungleichungen an dieser Stelle nicht benutzt werden können, welcher in GenGED verbessert wird (Kapitel 3). In AGG wird wird die Bedingung durch den neuen Graphen N angegeben. Bei der Auswertung wird geprüft, ob die Bedingungen aus der linken Seite L auf den Laufzeitgraphen G zutreffen und ob die Bedingung aus N nicht zutrifft. Das Kundenobjekt 2 aus N soll genau dem Kunden aus L entsprechen. Es heißt: ”Genau dieser Kunde hat kein Bargeld”, und nicht: ”Es gibt irgendeinen Kunden, der kein Bargeld mehr hat.”. Auch dieser Morphismus wird durch Nummerierung der Objekte festgelegt.
Es können beliebig viele dieser negativen Graphen für eine Regel spezifiziert werden. Beispielsweise soll die negative Anwendungsbedingung aus Abbildung 3 erweitert werden um die Bedingung 'Der Kunde darf nicht Kunde einer anderen Videothek sein.' Würde dies im selben Graphen N modelliert, würde damit zuviel gesagt: Ein Kunde darf nicht sowohl Kunde einer anderen Videothek als auch kein Bargeld haben. Ein Kunde, der zwar Mitglied der richtigen Videothek ist, aber kein Bargeld mit sich führt, wird von der negativen Anwendungsbedingung nicht erfaßt. Um also verschiedene, von einander unabhängige negative Anwendungsbedingungen auszudrücken, werden für jede dieser Bedingungen neue Graphen N_i spezifiziert.

2.3 Regelanwendungen

Wurde ein Morphismus von der Regel auf den Laufzeitgraphen gefunden, kann die Regel ausgeführt werden. Ganz naiv erklärt: Der auf die Bedingung passende Teilgraph aus der linken Regelseite wird aus dem Laufzeitgraphen ausgeschnitten

und durch einen der rechten Seite entsprechendem Teilgraph ersetzt. Der Laufzeitgraph wird durch die Anwendung der Regel nach G' transformiert.
Abbildung 4 zeigt, wie die Regel aus Abbildung 3 auf den Laufzeitgraphen aus Abbildung 2 angewendet wird. Es gibt mehrere Übereinstimmungen zwischen Regel und Laufzeitgraph, also mehrere Videos, die ausgeliehen werden können. Der Benutzer wählt 'King Kong' manuell aus, AGG vervollständigt die Übereinstimmung mit der linken Regelseite. Wird kein bestimmtes Video ausgewählt, nimmt AGG zufällig ein zulässiges Video. Nun wird die Regel angewendet: Die Kante IN wird gelöscht und eine neue Kante AUSGELIEHEN VON erstellt, welche als Quell-

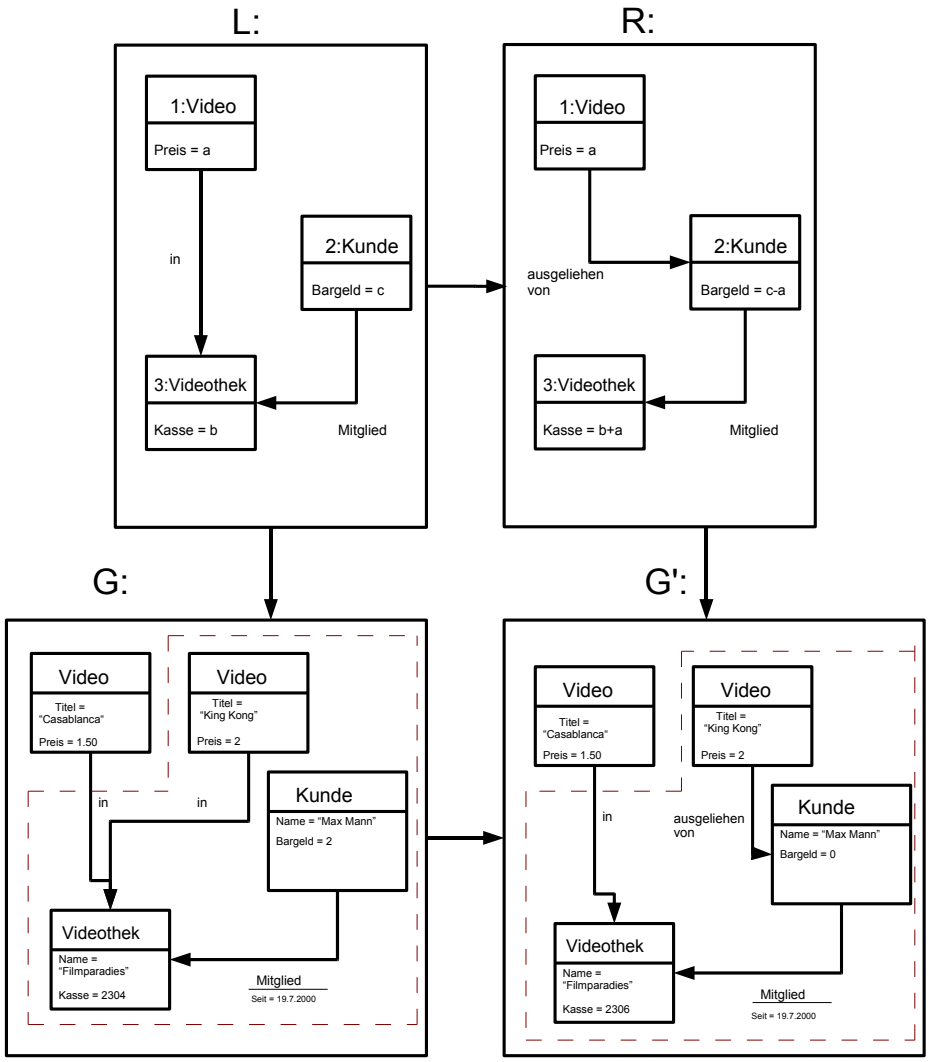

Abbildung 4: Videotheksmodell nach angewandter Regel

knoten das VIDEO 1 und als Zielknoten den Kunden hat. Die Regel wird nur auf das Video angewandt, welches ausgeliehen werden soll und nicht auf alle Videos, die die Bedingung erfüllen.
Allgemein wird bei der Regelanwendung folgendes Schema befolgt: Jedes Objekt auf der linken Regelseite L hat eine Entsprechung im Laufzeitgraphen G. Wenn es auch eine Entsprechung auf der rechten Regelseite R gibt, wird das Objekt aus G erhalten, ansonsten wird es entfernt. Ein Objekt, welches nur in R existiert, aber nicht in L, wird in G' neu erstellt. Objekte, die der Morphismus nicht erfaßt, werden auch nicht verändert (im Beispiel das Video 'Casablanca').
Eine Ausnahme bilden Kanten, deren Quell- oder Zielknoten gelöscht wurden. Sie heißen *hängende Kanten*. Diese Kanten werden automatisch gelöscht. Es dürfen also keine Kanten übrig bleiben, die nicht zwei Knoten miteinander verbinden. Nach der Transformation besteht ein Morphismus zwischen G' und der rechten Regelseite R.

Verschmelzen von Objekten: Es ist auch möglich, daß eine Regel zwei oder mehr Objekte verschmilzt. Eine Regel kann mehrere Knoten in einen verschmelzen und die Kanten, die vorher die beiden zu verschmelzenden Knoten miteinander verbanden, werden gelöscht. Für Kanten, die zwischen einem der zu verschmelzenden Knoten und anderen Knoten verlaufen, muß manuell entschiede werden, ob sie bestehen bleiben. Der Benutzer muß auch entscheiden, welche Attributwerte für das neue Objekt übernommen werden oder ob dessen Attribute neue Werte bekommen.
Abbildung 5 ist ein einfaches Beispiel. Die Regel sagt: Zwei Videos werden zu einer Spezialedition-Box zusammengefaßt und nur noch als ein Videoobjekt betrachtet. Der Benutzer muß nach der Transformation entscheiden, wieviel das neue Video kosten soll. **Nicht-injektive Regeln:** Zwei Objekte gleichen Typs einer Regel können sich auf das selbe Objekt im Laufzeitgraphen beziehen. Beispielsweise

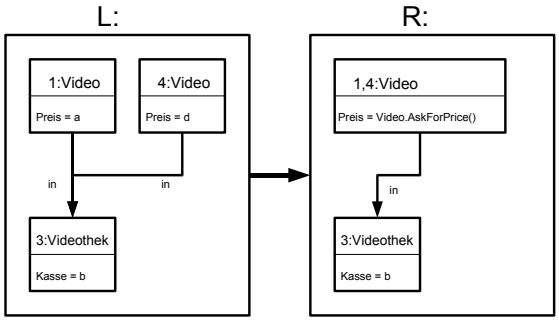

Abbildung 5: Regel zum Verschmelzen

soll eine Kante zwischen zwei Knoten geölscht werden. Im Laufzeitgraphen existiert jedoch nur eine Kante, deren Quell- und Zielknoten ein und der selbe sind. Die beiden Knoten des Graphen finden in dem Knoten des Laufzeitgraphen eine Übereinstimmung und die Kante wird gelöscht. Ob nicht-injektive Regeln zugelassenw erden, kann durch eine lobale Einstellung festgelegt werden.

Berechnungen auf Attributen: Graphregeln können nicht nur Graphstrukturen verändern, sondern auch Berechnungen auf Objektattributen ausführen. AGG beschränkt sich nicht nur auf einfache arithmetische Operationen, sondern kann beliebige Javamethoden aufrufen. Die Rückgabe der Methode muß allerdings mit dem Datentyp des Attributes kompatibel sein. Auf der linken Seite der Regel werden Variablen und Konstanten definiert. Auf der rechten Seite hingegen darf jeder beliebig komplizierte Ausdruck stehen. Dieser darf sich aber nur auf die Variablen der linken Seite in derselben Regel beziehen. Das ist eine Einschränkung von AGG: Es können für die Berechnungen keine Variablen verwendet werden, die nicht schon auf der linke Regelseite definiert wurden.
In Abbildung 4 werden einige Werte verändert. Auf der linken Seite L hat der Kunde b Bargeld bei sich und in der Kasse der Videothek sind c Geldeinheiten. Es kostet a Geldeinheiten, sich ein Video auszuleihen. Nach der Graphtransformation hat die Videothek diese a Geldeinheiten mehr in der Kasse und der Kunde ist um diese Geldeinheiten ärmer.

Ausführung der Regeln und zusätzliche Funktionen: Es ist möglich, auf einem Laufzeitgraphen alle definierten Regeln direkt hintereinander auszuführen. Die Regeln und negativen Anwendungsbedingungen sind in einer Listenstruktur angeordnet. Dabei sind die negativen Anwendungsbedingungen Unterstrukturen der zugehörigen Regeln. Der Laufzeitgraph wird mit einem Startgraphen initiiert, der manuell eingeben wird.
Die Regeln werden in der Reihenfolge ausgewählt, in der sie definiert wurde. Kann eine Regel nicht angewendet werden, wird die nächste ausgesucht. Der Lauf endet, wenn keine Regel mehr angewendet werden kann.
Eine Möglichkeit, die Reihenfolge der Regelanwendung nachträglich zu beeinflussen, ist die Verwendung von Ebenen (*Layers*). Die Regeln werden in verschiedene Ebenen eingeteilt. Bei der Ausführung werden die Ebenen nacheinander durch gegangen und nur die Regeln der aktuellen Ebene ausgeführt. Dabei wird jede Ebene so lange wiederholt durchlaufen, bis keine Regel der aktuellen Ebene mehr anwendbar ist.
Es ist in AGG auch möglich, eine Kritische-Paar-Analyse durchzuführen, die dem Benutzer bei der Zuteilung der Ebenen unterstützen soll. Dabei wird überprüft, ob zwei Regelanwendungen in beiden möglichen Reihenfolgen ausgeführt, immer das selbe Ergebnis liefern. Im schlechtesten wird durch die Ausführung der einen Regel die Ausführung der zweiten Regel verhindert. Auf die Ebenen und

die Kritische-Paar-Analyse wird in Abschnitt 4.2 noch einmal eingegangen.
Es ist nicht möglich, die Abfolge der Regelanwendungen durch Kontrollstrukturen wie Schleifen oder bedingte Anweisungen zu bestimmen. Einen solchen Ansatz bietet der Simulator von GenGED (Abschnitt 4.3).

Abbildung 6 zeigt einen Screenshot von der AGG Umgebung. ① zeigt die definierten Typen, ② und ③ die beiden Regelseiten. In ④ wird ein Startgraph definiert und später der aktuelle Laufzeitgraph angezeigt. In ⑤ werden alle bereits definierten Regeln in einer Listenstruktur aufgelistet.

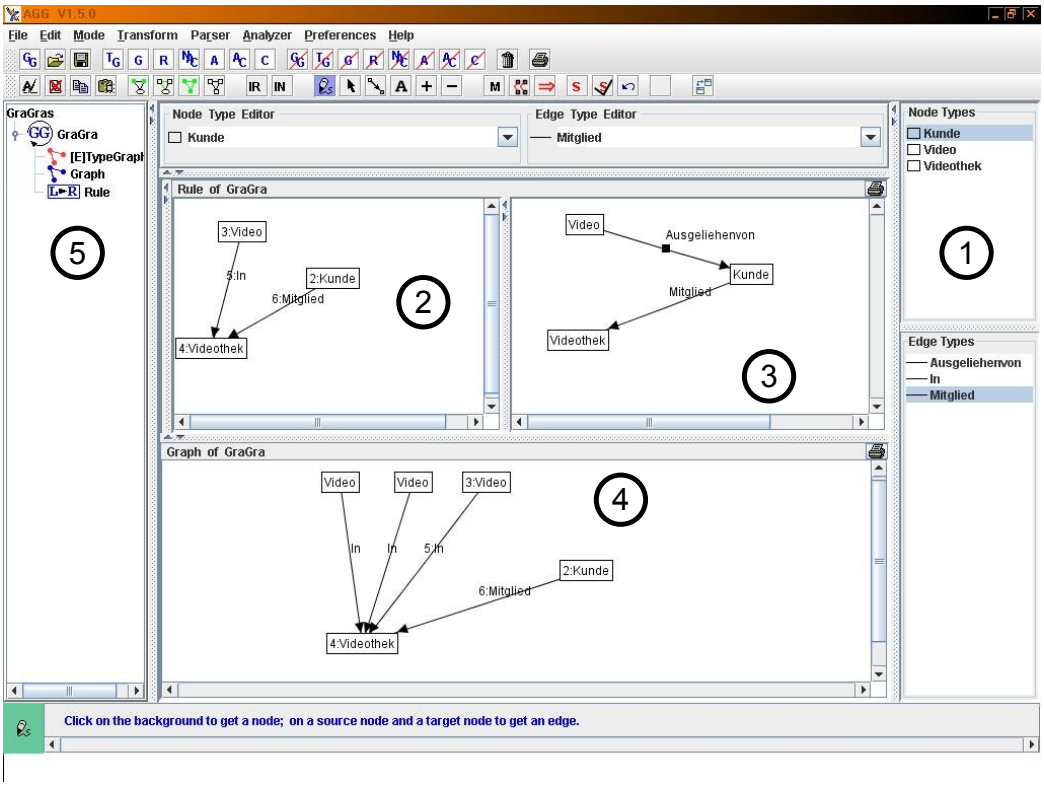

Abbildung 6: Die AGG Umgebung

3 GenGED

Mit GenGED wird eine visuelle Umgebung für eine visuelle Sprache generiert [8]. In der Umgebung können Diagramme in der Sprache erstellt, editiert, geparst und simuliert werden. Die Spezifizierung dieser visuellen Umgebung geschieht in mehreren Schritten. Abbildung 7 zeigt den Aufbau von GenGED im Überblick.
Ein visuelles Alphabet wird mit dem *Alphabeteditor* (Abschnitt 3.1) definiert. *Constraints (Einschränkungen)* legen das Layout und die Struktur der Diagramme fest. Der integrierte Constraintsolver *ParCon* (Abschnitt 3.4) löst die durch die Constraints aufgestellten Gleichungssysteme und paßt die Struktur und das Layout des Diagramms gegebenenfalls an. Durch die Struktur und das Layout wird ein Teil der Semantik des Graphen visualisiert. In dem *Grammatikeditor* (Abschnitt 3.2) werden Graphregeln definiert. Mit Hilfe dieser Regeln können später Diagramme erstellt, auf Gültigeit überprüft oder Veränderungen der Diagrammstruktur simuliert werden. Diese Veränderungen beruhen auf der Transformation der Graphstruktur. Diese Aufgabe übernimmt das integrierte Programm *AGG*. Im *Virtual Language (VL) Spezifikationseditor* (Kapitel 4) wird die visuelle Umgebung spezifiziert und konfiguriert. Es wird gewählt, ob es eine Parsegrammatik und einen Simulator gibt und die entsprechenden Regeln für diese ausgewählt. Nach der Konfiguration generiert GenGED den Code für die VL Umgebung (Abschnitt 3.3). Die VL Umgebung ist ein eigenständiges Programm, das auch unabhängig von GenGED benutzt werden kann.

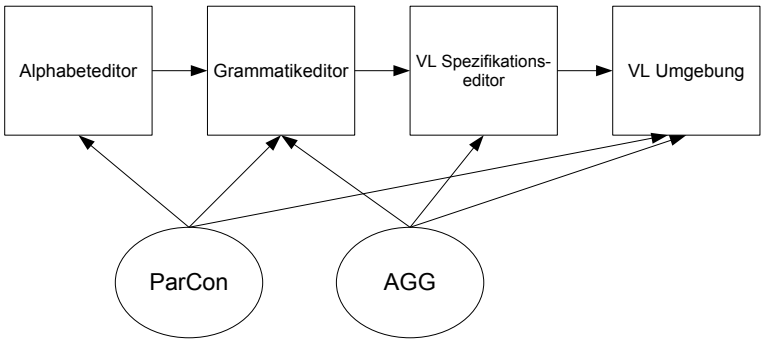

Abbildung 7: Der Systemaufbau von GenGED

3.1 Alphabeteditor

Mit dem Alphabeteditor wird ein visuelles Alphabet erstellt, das aus visuellen Symbolen besteht. Durch Verbindungen wird eine Graphstruktur vorbestimmt. Diese ist ein Typgraph, so wie er für AGG in Abschnitt 2.1 definiert wurde und spielt im folgenden die gleiche Rolle für GenGED. Der Unterschied zu der Graphstuktur von AGG ist, daß beliebige Typen beliebig miteinander kombiniert werden können und die Verbindungen werden nur als Relation angegeben. Der Alphabeteditor setzt sich aus drei Teilen zusammen. Dem Symboleditor, dem Verbindungseditor und einem Constraintmenü.

Symboleditor: Mit dem Symboleditor werden beliebige Symbole ausgewählt und gezeichnet. Es können geometrische Formen wie Kreise, Ellipsen und Rechtecke und andere einfachen Figuren gewählt werden. Eigene Symbole können freihand gezeichnet oder aus den vorgefertigten Symbolen zusammen gesetzt werden. Mit Hilfe einer Farbpalette können die Symbole farbig unterschieden werden. Die verschiedenen Symbole stehen für jeweils einen Objekttypen und erhalten einen eindeutigen Namen. Abbildung 8 zeigt den Symboleditor. Ganz links werden die Typen aufgelistet. Abbildung 9 zeigt eine Vergrößerung der Listendarstellung der Symbole. Für den Kundentypen wird automatisch, sobald das Symbol gezeich-

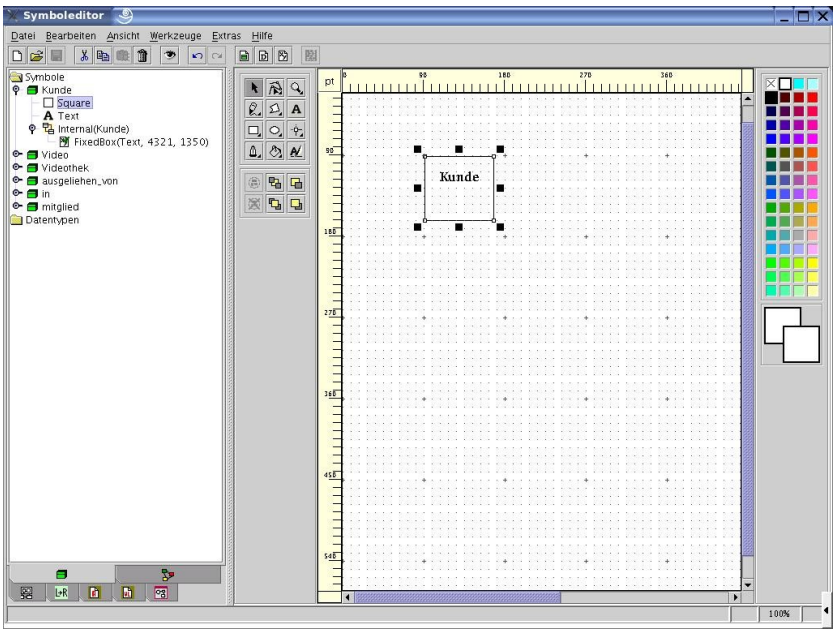

Abbildung 8: Der Symboleditor

Abbildung 9: Symbole in der Listendarstellung

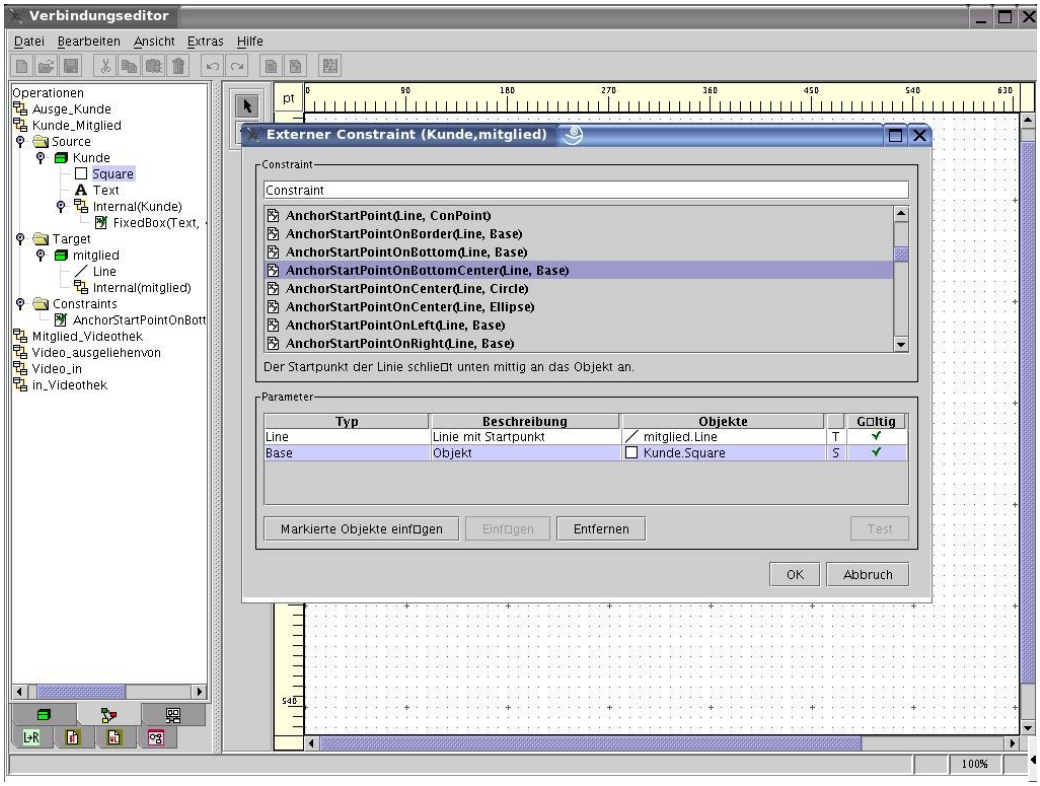

Abbildung 10: Der Verbindungseditor mit Constraintmenü

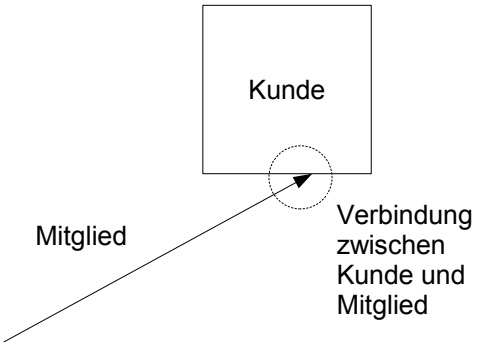

Abbildung 11: Verbindung zwischen zwei Symbolen

net wird, ein interner Constraint *fixedbox* festgelegt, welches Text und Maße des Quadrates beinhaltet.

Verbindungseditor: Mit dem Verbindungseditor werden Verbindungen zwischen den Symbolen hergestellt. In einem Operationsmenü werden aus den vorher definierten Symbolen ein Quell- und ein Zielsymbol für eine Verbindung ausgewählt. Abbildung 10 zeigt den Verbindungseditor, in dem eine Verbindung zwischen den beiden Symbolen KUNDE und MITGLIED definiert ist. Im Hintergrund ist die Verbindung in der Listendarstellung zu sehen. Im Vordergrund wird für diese Verbindung ein neuer Constraint ANCHORSTARTPOINTONBOTTOMCENTER im Constraintmenu definiert, der das Startende des Symbols MITGLIED mit der unteren Kante des Symbols KUNDE verbindet (Abbildung 11). Die Verbindungen in GenGED werden später als Kanten an AGG geschickt. Sie bedeuten in GenGED aber nicht nur eine Abhängigkeit zwischen Typen, sondern durch die definierbaren Constraints auch eine graphische Relation.

Attribute: Für die Typen können auch Attribute deklariert werden. Die Deklaration geschieht im Symboleditor und im Verbindungseditor werden wird das Attribut einem Typen zugewiesen. Wird ein Symbol später verwendet, muß dem Attribut ein Wert zugewiesen werden. Abbildung 12 zeigt links, daß im Verbindungseditor einem Typen ein Attribut (dargestellt durch **D**) zugewiesen wurde. Auf der rechten Seite wird der Wert im Grammatikeditor abgefragt, da ein neues Symbol dieses Typen verwendet werden soll.

Constraintmenü: Ein Constraint definiert die Eigenschaft eines Objektes oder einer Relation zwischen mehreren Objekten. Das Alphabet wird durch sie erweitert, indem den Symbolen noch weitere Informationen hinzugefügt werden. Jeder Constraint besteht aus einem Gleichungssystem (meist) linearer Un-/Gleichungen, das von dem Constraintsolver PARCON gelöst wird. Die Cons-

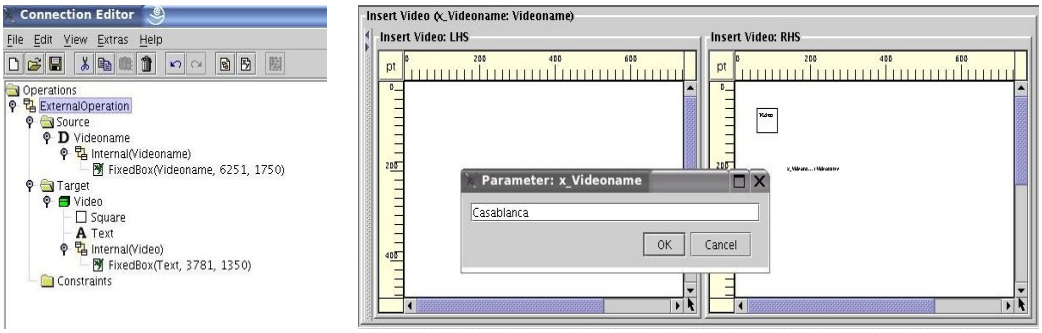

Abbildung 12: Deklarierter Datentyp

traints erhalten verschiedene Parameter, die durch das Aussehen der Symbole gegeben sind, zum Beispiel Höhe, Breite, Mittelpunkt oder manuell gesetzte *Aufhängungspunkte*.

Im Symboleditor werden einige interne Constraints automatisch gesetzt, wenn ein neues Symbol gewählt wird, zum Beispiel die Proportion zwischen den Kanten eines Rechteckes. Wenn eine Verbindung zwischen zwei Symbolen definiert wird, können beliebig viele Constraints definiert werden. In Abbildung 10 ist das Constraintmenü zu sehen, in dem ein Constraint zwischen KUNDE und MITGLIED ausgewählt wird. Der Constraint ANCHORSTARTPOINTONBOTTOMCENTER verbindet eine (gerichtete) Linie mit der unteren Kante eines geometrischen Objektes. Es ist auch möglich, die Bibliothek des Constraintmenüs um eigene Constraints zu erweitern.

Gegeben seien beispielsweise zwei Quadrate. Für das Layout soll gewährleistet sein, daß sich die beiden Symbole nicht überschneiden. Ein Quadrat sei definiert durch seine Kantenlänge k und seine x- und y-Koordinaten der linken unteren Ecke. Beispielsweise die zwei quadratischen Symbole VIDEO und KUNDE kön-

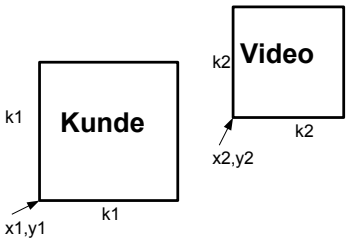

Abbildung 13: Zwei quadratische Symbole sollen sich nicht überschneiden

nen durch Benutzerinteraktion bewegt werden oder in ihrer Größe verändert werden (Abbildung 13). Die Constraints seien dann:

$$Video.x + Video.k <= Kunde.x$$
oder
$$Video.y + Video.k <= Kunde.y$$
oder
$$Kunde.x + Kunde.k <= Video.x$$
oder
$$Kunde.y + Kunde.k <= Video.y$$

Alle vier Ungleichungen müssen dafür aufgestellt werden. Sie sind durch *oder* verbunden, es muß also nur eine der Gleichungen gültig sein, damit die Bedingung erfüllt ist. Das Quadrat KUNDE ist dann über, unter, links oder rechts des Quadrats VIDEO, schneidet es jedoch nicht. Sind die Gleichungen nicht erfüllt, wird ParCon eine neue, ähnliche Variablenbelegung finden, und die Symbole verschieben (siehe Abschnitt 3.4)

Die in GenGED benutzten Constraints kommen zumeist mit den arithmetischen Operationen $+$ und $-$ aus. Jedoch schießt die Anzahl der auftretenden Gleichungen und Ungleichungen mit der Anzahl der Symbole in die Höhe. Je mehr Symbole zueinander in Beziehung gesetzt werden, desto mehr Constraints müssen gelöst werden. Wird ein drittes Symbol zu den beiden Quadraten hinzu genommen, müßten diese Constraints für jede Kombination der Symbole gelöst werden. Mit der Größe der Diagramme dauert das Lösen sehr lange.

Testeditor: Noch bevor Regeln definiert werden, kann die Struktur eines Diagrammes im *Diagram Test Editor* getestet werden. Dazu wird ein neues Diagramm freihand aus den im Alphabeteditor definierten Symbolen gezeichnet. Durch die Verbindungen und Constraints soll es validiert werden. Für jede neu eingefügte Verbindung werden sofort die Constraints gelöst. Um die Constraints zu lösen, ruft GenGED das integrierte Programm ParCon auf (Abschnitt 3.4). Gegebenfalls wird eine Diagrammkomponente gelöscht oder verrückt, wenn die Constraints nicht erfüllt werden können, so daß sie erfüllt sind.

3.2 Grammatikeditor

Im folgenden wird zwischen drei Typen von Grammatiken für visuelle Sprachen unterschieden: Die Syntaxgrammatik, die Parsegrammatik und die Simulationsgrammatik. Die *Syntaxgrammatik* gibt Editieroperationen vor, um Diagramme

in der visuellen Sprache syntaxgesteuert zu erstellen (Abschnitt 4.1). Die *Parsegrammatik* unterstützt freihand Editieren durch Parseoperationen, um ein beliebiges Diagramm für gültig oder ungültig zu erklären (Abschnitt 4.2). Die *Simulationsgrammatik* gibt Simulationsoperationen vor, um das dynamische Verhalten eines Modells zu simulieren (Abschnitt 4.3).

Für jede neue Grammatik wird ein vorher im Alphabeteditor definiertes visuelles Alphabet benutzt. Für das Alphabet generiert GenGED automatisch Syntaxregeln für die Basisoperationen: Für jedes Symbol eine Einfüge-Regel und eine Lösch-Regel. Mit dem Grammatikeditor werden nun Graphregeln definiert. Der Editor ist in mehrere Felder unterteilt, in denen Regeln gezeichnet werden können. Abbildung 14 zeigt einen Screenshot von GenGED. In den oberen Editoren wird die aktuell benutzte Graphregel angezeigt. Die unteren beiden Felder sind sogenannte *Working Displays* in der entweder eine neue Regel erstellt werden kann oder ein Startdiagramm spezifiziert wird. Für das Startdiagramm wird das rechte Working Display nicht gebraucht. In der Leiste auf der linken Seite sind alle Regeln der aktuellen Grammatik aufgelistet. Sowohl die Spezifizierung des Start-

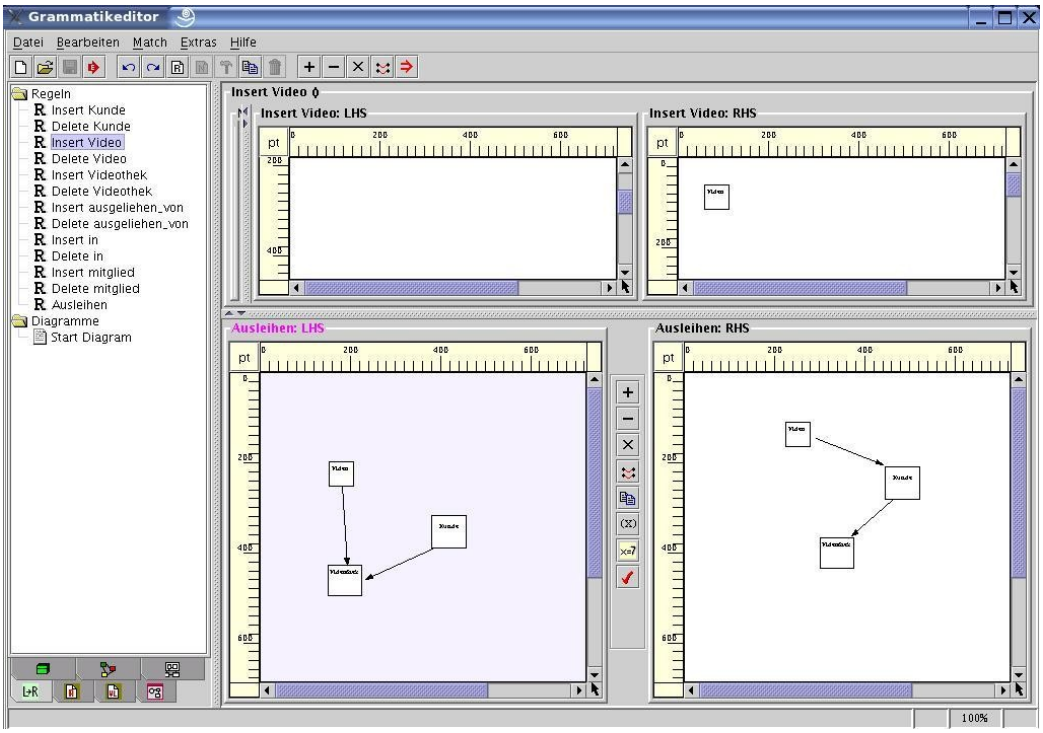

Abbildung 14: Ein Screenshot des Grammatikeditors

diagramms als auch das erstellen neuer Regeln geschieht syntaxgesteuert, daher nur unter Verwendung der bereits vorhandenen Regeln. In der Abbildung wird eine neue Regel AUSLEIHEN für das Videotheksbeispiel definiert. Dafür werden beide Working Displays benutzt für beide Seiten der Regel. In den oberen beiden Feldern ist die aktuell verwendete Regel INSERT VIDEO angezeigt, welche bei der Definition des Alphabets vordefiniert wurde.

Die Regeln werden zur Ausführung an den Interpreter AGG übergeben. Darum unterscheiden sich die Regeln im wesentlichen nicht von denen, die in Abschnitt 2.2 definiert wurden: Es gibt eine linke Seite, eine rechte Seite und beliebig viele negative Anwendungsbedingungen. Außerdem wird AGG das aktuelle Laufzeitdiagramm als Startgraph übergeben. Die Diagrammstruktur wird in die Graphstruktur von AGG übersetzt. Gegebene Morphismen zwischen Regel und Laufzeitgraph werden durch Markierungen in GenGED angegeben und ebenfalls an AGG übergeben. Wird nur ein Teil eines Morphismus markiert, komplettiert AGG ihn automatisch.

Wird für ein Diagramm ein Symbol gewählt, für das Attribute deklariert wurden, fragt GenGED den Wert ab (wie in Abbildung 12). Für die linke Regelseite können Bedingungen an die Attributwerte des Laufzeitgraphen gestellt werden, z.B. $x > 0$. Dies ist in AGG nicht möglich und stellt also eine Erweiterung dar. Die Bedingung an den Laufzeitgraphen wird nicht in AGG, sondern in GenGED, beziehungsweise der visuellen Umgebung, abgefragt, allein die Transformation geschieht in AGG.

AGG in GenGED: In GenGED wird AGG als visueller Interpreter genutzt. AGGs eigene Editoren und die GUI werden nicht genutzt. AGG wird nur zur Transformation der Diagramme benutzt. GenGED benutzt zwar die Konzepte von AGG, jedoch ist die benutzte Graphstruktur anders, da die Diagramme anders dargestellt sind. Um AGG zu nutzen, müssen die Graphstrukturen von GenGED zu AGG kompatiblen Strukturen konvertiert werden. Die Symbole werden zu Knoten in AGG und die Verbindungen zwischen den Symbolen zu Kanten. Es werden die attributierten Regelgraphen, ein Laufzeitgraph und die Morphismen übergeben. Nachdem das Diagramm transformiert wurde, wird es wieder in die GenGED-Struktur zurück konvertiert. GenGED stellt hier eine Erweiterung von AGG dar, da es mit Layoutconstraints das strukturierte Editieren von Diagrammen ermöglicht.

3.3 Die VL Umgebung

Die VL Umgebung wird im *VL Spezifizierungseditor* konfiguriert. Es werden Regeln ausgewählt, die im Grammatikeditor definiert wurde, und ein dazu passendes visuelles Alphabet, das im Alphabeteditor definiert wurde. Sowohl AGG, das für die Graphtransformation genutzt wird, als auch ParCon sind Bestandteile der VL Umgebung.

Die VL Umgebung kann als eine von GenGED unabhängige Anwendung genutzt werden. Sie stellt zwei graphische Editoren für Diagramme bereit, einen Syntaxeditor und einen Freihandeditor. In beiden Editoren wird mit dem visuellen Alphabet gearbeitet. Mit dem Syntaxeditor werden die ausgewählten Regeln benutzt um syntaxgesteuert Diagramme aufzubauen. Im Freihandeditor werden die Verbindungen zwischen den Symbolen manuell eingefügt. Falls eine Parsepezifikation für die VL Umgebung gewählt wurde, kann ein Diagramm zu jedem beliebigen Zeitpunkt geparst und so überprüft werden. Wenn ein Simulator existiert, kann in einer anderen Ansicht ein beliebig erstelltes Diagramm simuliert werden.

3.4 ParCon

Das ParCon Projekt [4; 1] wurde an der Universität Paderborn von 1992 bis 1996 entwickelt. Parcon wurde für das Lösen von graphischen Constraints entwickelt. Es zeichnet sich durch eine hohe Geschwindigkeit (Effizienz) und Parallelisierbarkeit zur Arbeit auf Rechnerclustern aus. ParCon kann Gleichungen und Ungleichungen der Constraints lösen. Die Constraints können durch logische Operatoren miteinander verknüpft werden. Sie können interaktiv hinzugefügt oder gelöscht werden. ParCon bekommt alle Constraints und eine initiale Variablenbelegung. Darauf versucht es das *Constraint Satisfaction Problem* zu lösen, also eine Variablenbelegung zu finden, die alle Constraints gültig macht. Dabei versucht ParCon, so viele der zuerst übergebenden Variablen wie möglich zu erhalten. Werden die zunächst ungültigen graphischen Contraints mit einer anderen Variablenbelegung gelöst, wird gewährleistet, daß das Diagramm so wenig wie möglich verändert wird.

Zum Beispiel soll das Gleichungssystem der beiden Quadrate aus Abbildung 15 gelöst werden. Die beiden Quadrate überlappen sich. X,y und k stellen Variablen dar, die die Position und die Größe der Quadrate definieren. ParCon wird eine neue Variablenbelegung als Koordinaten oder Längen der Quadrate finden, so daß sie sich nach der Lösung nicht mehr überlappen. Gleichzeitig werden sie sich so wenig wie möglich verändert haben, nur so weit, bis die Bedingung erfüllt ist. So wird das Diagramm so wenig wie möglich von der Eingabe des Benutzers abwei-

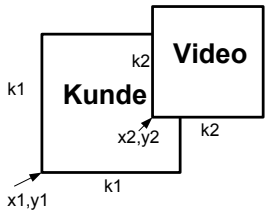

Abbildung 15: Zwei quadratische Symbole sollen sich nicht überschneiden

chen.
ParCon ist die einzige Komponente in GenGED, die nur auf den Betriebssystemen Linux und Sun läuft. Darum ist auch GenGED nicht für Windows verfügbar.

3.5 Vorteile und Nachteile von GenGED

GenGED ist komplett in Java implementiert und aus den spezifizierten Grammatiken wird die VL Umgebung ebenfalls in Java-Code erzeugt. Der Vorteil der Java-Implementierung ist, daß GenGED leicht erweitert werden kann. Mittlerweile gibt es für GenGED unter anderem einen Generator für Animationsumgebungen, in dem die Transformationen durch fließende Übergänge dargestellt sind. Die Strukturänderung des Diagramms sieht wie ein kurzer Zeichentrickfilm aus.

Durch ParCon wird GenGED eingeschränkt. Je umfangreicher das Alphabet ist, bzw. die Diagramme sind, desto mehr schießt die Anzahl der Constraints in die Höhe und das Lösen mit ParCon wird ineffektiv. So wird es ab einer bestimmten Graphengröße unmöglich, die Constraints zu lösen und so die Spezifizierung der Sprache fortzuführen. ParCon ist die einzige Komponente von GenGED, die nicht in Java implementiert wurde und leider läuft ParCon nur unter Linux und Sun und nicht unter Windows.

4 Spezifizierung der Umgebung und des Simulators

In der VL Spezifizierung [2; 8] wird aus einem Satz Graphregeln und einem dazu gehörigen Alphabet die Grammatik für die zu generierende Umgebung zusammengesetzt. Soll in der VL Umgebung ein Diagramm nach genau der definierten VL Sprache erstellt werden, wird dazu die Syntaxgrammatik benutzt (Abschnitt 4.1). Soll für ein beliebiges Diagramm überprüft werden, ob es zu der definierten Sprache gehört, muß dafür eine Parsegrammatik erstellt werden (Abschnitt 4.2). Soll auf einem Startgraphen ein Ablauf mehrerer Regeln erfolgen, müssen diese in einer Simulationsgrammatik festgelegt werden (Abschnitt 4.3).

4.1 Syntaxgrammatik

Syntaxgesteuertes Editieren von Diagrammen erlaubt die Erstellung streng nach vordefinierten Graphregeln. Auf diese Weise können keine Diagramme entstehen, die nicht zu der definierten Sprache gehören. Begonnen wird mit einem Startdiagramm, das auch leer sein kann. Auf diesem können nun Regeln angewendet werden, die Knoten und Kanten hinzufügen, Attributen neue Werte zuweisen, Kanten umbiegen, Kanten löschen, Knoten miteinander verschmelzen und so weiter. Jede Regel kann beliebig oft und in beliebiger Reihenfolge gewählt werden, solange das aktuelle Diagramm den Anwendungsbedingungen der Regel entspricht. Das Diagramm wird also nicht per Hand gezeichnet, sondern ein Startausdruck wird mit Hilfe der Grammatikregeln und AGG transformiert. Das Ergebnis wird weiter transformiert, bis das gewünschte Diagramm erstellt ist.

Beispielsweise wird mit einem leeren Diagramm begonnen. Die erste Regel besagt: ”Füge VIDEO-objekt hinzu.” Die zweite Regel sagt: ”Füge KUNDEN-objekt hinzu”. Die dritte Regel:”Wenn ein Objekt VIDEO und ein Objekt KUNDE existieren, füge Objekt AUSGELIEHEN VON dazwischen ein.” Es ist nicht möglich, ein Diagramm zu erstellen, in dem das Objekt AUSGELIEHEN VON ohne die beiden anderen Objekte existiert.

4.2 Parsegrammatik

Mit einer Parsegrammatik kann für ein beliebiges Diagramm überprüft werden, ob es Teil der definierten Sprache ist.
Um zu überprüfen, ob ein Diagramm gültig ist, werden auf das Diagramm so lange Regeln angewendet, bis es den Zustand eines vorher definierten *Stopdiagrammes* erreicht. Dann ist der Graph gültig. Kann das Diagramm den Zustand

nicht erreichen, ist das Diagramm ungültig. In dieser Variante wird also ein einfacher Backtrackingalgorithmus ausgeführt. Diese Auswertung geschieht intern, der Benuztzer erhält nur das Ergebnis.
Beispielsweise soll ein Diagramm geparst werden, in dem es Kanten- und Knotensymbole gibt (Abbildung 16). In diesem Diagramm darf keine Kante existieren, die nicht zwischen zwei Knoten verläuft. Als Stopdiagramm wird ein Diagramm gewählt, in dem nur Knotensymbole existieren. Mit jeder Regel wird eine Kante gelöscht, allerdings nur dann, wenn sie zwischen zwei Knoten verläuft. Existieren in dem Diagramm Kanten, die nicht an einem Knoten hängen, werden sie nicht durch die Regeln gelöscht und das Stopdiagramm wird nicht erreicht. Das Diagramm ist ungültig.
Im Allgemeinen werden für eine Parsegrammatik Regeln erstellt, die die Regeln der zugehörigen Syntaxgrammatik rückgängig machen. Eine Syntaxgrammatik für das Beispiel würde keine Regel beinhalten, die eine Kante einfügt, wenn sie nicht zwischen zwei Knoten eingefügt wird.

Die Ebenenfunktion: Eine Parsegrammatik kann durch die Ebenenfunktion (*layer*) erweitert werden. Das ist optional, erleichtert aber das Parsen. Die Re-

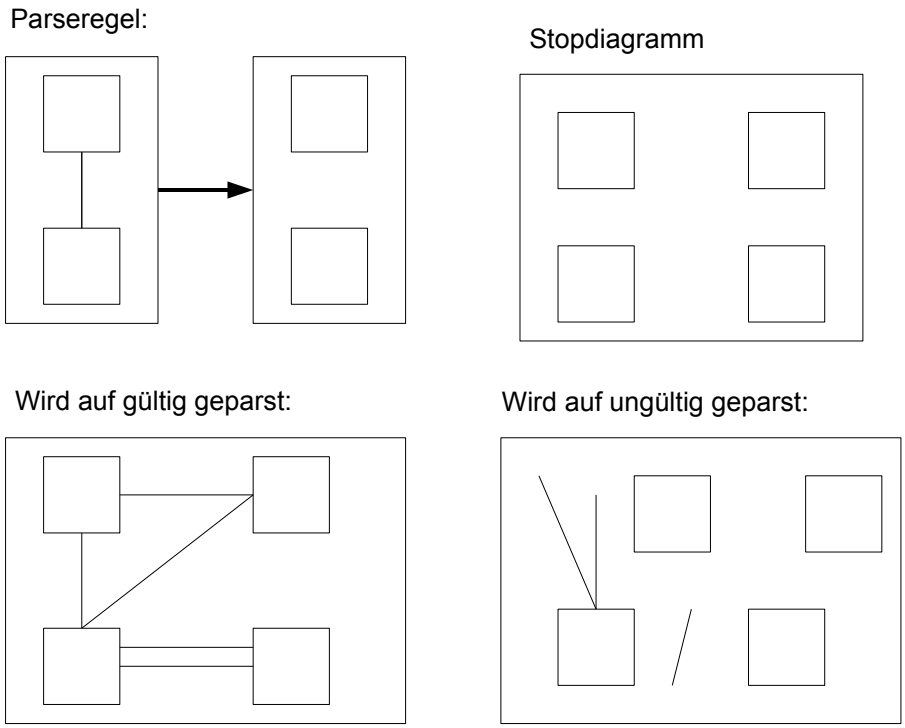

Abbildung 16: Parseregel, Stopdiagramm und zwei Laufzeitgraphen

geln werden in verschiedenen Ebenen aufgeteilt. Jeder Regel wird eine Integerzahl zugeteilt, die für die Ebene steht. Beim Parsen werden die Ebenen von 0 bis n durchgegangen. Innerhalb einer Ebene ist die Reihenfolge der Regelanwendung jedoch nichtdeterministisch. Immer werden nur die Regeln der aktuellen Ebene so oft angewendet wie möglich. Kann in einer Ebene keine Regel mehr angewendet werden, werden die Regeln der nächst höheren Ebene ausgeführt. Es kann danach nicht mehr auf eine niedrigere Ebene zurück gegiffen werden.

Die Kritisches-Paar-Analyse: Es kann passieren, daß für eine Regel eine Übereinstimmung im Diagramm gefunden wird, welche durch Anwendung einer anderen Regel zerstört wird. Beispielsweise soll ein Diagramm geparst werden, in dem es zwei Symbole vom Typ KNOTEN und ein Symbol vom Typ KANTE gibt. Eine Regel INSERTZWEITEKANTE könnte dieser Konstellation ein weiteres Symbol KANTE hinzufügen, aber nur, wenn bereits eine KANTE EXISTIERT. Eine andere Regel $DeleteKante$ findet auch eine Übereinstimmung und löscht die KANTE (Abbildung 17). Wird $DeleteKante$ als erstes angewendet, kann $InsertKante$ danach nicht mehr angewendet werden. Auf dasselbe Diagramm angewendet, könnte der Parsealgorithmus bei mehrfachem Durchführen verschiedene Verläufe annehmen. Ein Durchlauf könnte ein positives Ergebnis ergeben, ein anderer Durchgang ein negatives. $InsertZweiteKante$ und $DeleteKante$ bilden ein kritisches Paar.

Die Kritische-Paar-Analyse sucht nach diesen Überschneidungen bei der Ausführung von Regeln. Wird kein kritisches Paar gefunden, ist die Reihenfolge der Regelausführung irrelevant. Wurde eines gefunden, so kann der Benutzer die Regeln durch die Ebenenfunktion trennen und so die Reihenfolge festlegen. Der Sinn dieser Analyse ist, dem Benutzer bei der Regelwahl zu helfen, damit trotz aufgrund des Nichtdeterminismus immer das richtige Ergebnis erhält.

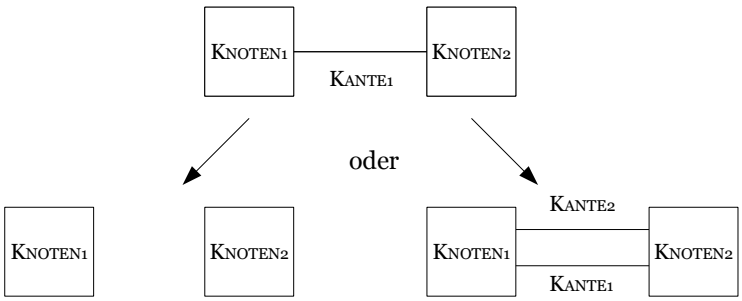

Abbildung 17: Zwei Möglichkeiten den Parsevorgang fortzusetzen

4.3 Simulator

Der Simulator wird im VL Spezifizierungseditor konfiguriert. Es wird dort bestimmt, welche Regeln in welcher Reihenfolge und wie oft angewendet werden sollen. Es wird eine Simulationsgrammatik aus den Graphregeln erstellt.

Ein Simulationsschritt ist eine atomare Operation, der während der Simulation ausgeführt wird. Atomar bedeutet, daß er unteilbar ist, nicht unterbrochen werden kann und keine Zwischenergebnisse liefert. Während eines Simulationsschrittes wird ein Simulationsausdruck SE (*Simulation Expression*) über Regeln ausgewertet. Mit Hilfe dieser Ausdrücke kann dynamisch auf verschiedene Diagramme eingegangen werden. Ein Simulationsausdruck ist eine Verknüpfung von Regeln und boolschen Ausdrücken *false* und *true*. Eine Regel wird mit *true* bewertet, wenn sie auf dem aktuellen Laufzeitdiagramm angewendet werden kann. Anderenfalls wird sie mit *false* bewertet. Ist ein Simulationsausdruck $false$, wird die Simulation abgebrochen und es erscheint eine Fehlermeldung.

Ein Simulationsausdruck ist eine Verknüpfung zwischen Regeln und boolschen Ausdrücken. Sie können mit folgenden logischen Operatoren und einfachen imperativen Konstrukten miteinander kombiniert werden: Es seien e, f und g Simulationsausdrücke, *true*, *false* oder eine beliebige Regel:

- $(not\ e)$ ist *true*, wenn e *false* ist und *false*, wenn e *true* ist
- Für $(e\ and\ f)$ werden zunächst die Ausdrücke e und f jeweils für sich ausgewertet. Der Gesamtausdruck ist nur dann *true*, wenn sowohl e als auch f *true* sind.
  $(e\ andc\ f)$ ist ein bedingter (*conditional*) Ausdruck. Die Ausdrücke e und f werden nacheinander ausgewertet. Sobald einer der Ausdrücke $false$ ist, ist klar, daß der Gesamtausdruck $false$ ist und die übrigen Ausdrücke werden nicht mehr ausgewertet.
- $(e\ or\ f)$ ist dann *true*, wenn wenigstens eines von beiden, e oder f, *true* ist. $(e\ orc\ f)$ ist die bedingte Variante. Sobald einer der Ausdrücke $true$ ist, ist der Gesamtausdruck $true$.
- $if\ e\ then\ f\ else\ g\ fi$: Zunächst wird e ausgewertet. Wenn e *true* ist, dann wird f ausgewertet, wenn e *false* ist, wird g ausgewertet.
- $while\ e\ do\ f\ done$ ist dann *true*, wenn der Schleifenkörper f mindestens einmal aus ausgewertet wird und sonst *false*.

Zum Beispiel:

while rule(SUCHEKUNDE) *do*{

if rule(HATAUSGELIEHEN)

then rule(ZAHLELEIHGEBÜHR)

}

Jedesmal, wenn die Regel SUCHEKUNDE ausgeführt wird, wird auch die if-Anweisung ausgeführt. Kann die Regel HATAUSGELIEHEN ausgeführt werden, wird auch die Regel ZAHLELEIHGEBÜHR ausgeführt.

In Abbildung 18 ist gezeigt, wie ein neuer Simulationsschritt im Spezifizierungseditor definiert wird. Dieser Simulationsschritt baut ein Diagramm des Videotheksbeispiels auf. Die Regeln INSERT KUNDE, INSERT VIDEO und INSERT VIDEOTHEK werden alle drei ausgeführt. Könnte eine der Regeln nicht angewendet werden, wäre der ganze Simulationsausdruck ungültig. Die Simulationsausdrücke

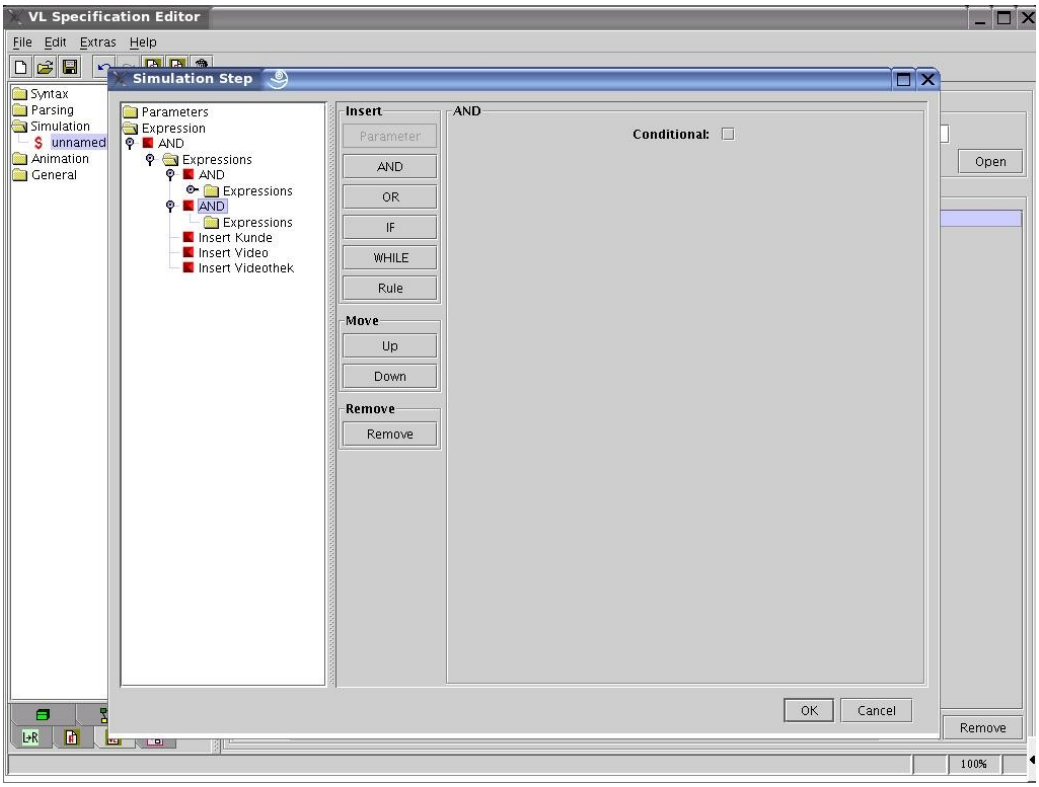

Abbildung 18: Eine neuer Simulationsschritt

werden beliebig tief ineinander verschachtelt. Mit einem Operator werden beliebig viele weitere Ausdrücke und Regeln miteinander verbunden.

In AGG ist nur eine direkte Hintereinanderausführung der Regeln möglich. Mit den Kontrollstrukturen des vorliegenden GenGED Simulators können einerseits Regelanwendung voneinander abhängig gemacht werden. Zum Beispiel kann eine Regel nur dann ausgeführt wird, wenn eine bestimmte andere Regel ebenfalls ausgeführt werden konnte. Andererseits können mit Ausdrücken wie *oder* und *if* auf verschiedene Startdiagramme flexibel eingegangen werden.

5 Zusammenfassung und Ausblick

GenGED generiert eine VL Umgebung, in der Diagramme einer visuellen Sprache syntaxgesteuert oder freihand erstellt werden können. Hierfür wird in dem Alphabeteditor ein visuelles Alphabet erstellt. Das Alphabet legt durch Constraints das Layout der Diagramme fest. Die Constraints werden durch das integrierte Programm ParCon gelöst. In dem Grammatikeditor werden visuelle Grammatikregeln erstellt. Die Transformation der Diagramme mit Hilfe der Regeln wird mit der integrierten Komponente AGG durchgeführt. Dafür werden die Diagramme in die von AGG unterstützte Graphstruktur übersetzt.

Die VL Umgebung ist eine eigenständige Komponente, die in Java-Code generiert wird. Frei erstellte Diagramme können auf ihre Kompatibilität zu der visuellen Sprache überprüft werden. Auf einem Startdiagramm können nacheinander verschiedene Graphtransformationen ausgeführt werden und so wird ein Lauf von Veränderungen simuliert.

Eindruck: GenGED ist eine interessante Anwendung von Graphersetzungen. Da Graphen auf rein textuellem Weg nur sehr umständlich definiert werden können, ist es eine Erleichterung, einige Schritte mit visuellen Mitteln zu realisieren. Um eine visuelle Sprache zu erstellen, müssen jedoch viele Zwischenschritte gemacht werden. Auch für einfache Diagramme ist die Definition mit GenGED mühselig und langwierig. Besonders bei der Definition der Constraints verliert sich schnell der Überblick. Bei zunehmender Größe der Diagramme gibt es aufgrund des ineffizienten Constraintlösers längere Wartezeiten und Unterbrechungen.
GenGED hat Potential, da der Java-Code eine stete Weiterentwicklung erlaubt. Wünschenswert sind unter anderem Kompatibilität zu anderen Betriebssystemen, effizientere Lösung der Constraints, eine Vereinfachung bei der Erstellung komplexer visueller Sprachen. Im momentanen Entwicklungsstand ist zu bezweifeln, daß GenGED in der Praxis Anwendung findet.

Ausblick: Für GenGED wurde ein Generator für Animationsumgebungen entwickelt [3], der auf Basis der Graphregeln Animationen erstellt. Bei der normalen Simulation durchläuft der Laufzeitgraph diskrete Zustände. In einem Animationssimulator werden die Zuständsänderungen zu kontinuierlichen Abläufen. Das wirkt wie ein kurzer Zeichentrickfilm. Dazu wurde GenGED unter anderem um einen Animationsregeleditor erweitert.

Literatur

[1] *ParCon Homepage*. `http://wwwcs.uni-paderborn.de/cs/ag-szwillus/forschung/projekte/Parcon2/index.html`. Letzter Zugriff: 28.1.2007.

[2] BARDOHL, ROSWITHA, CLAUDIA ERMEL und INGO WEINHOLD: *Specification and Analysis Techniques for Visual Languages with GenGED*. Technischer Bericht, TU Berlin, 2002. `http://www.cs.tu-berlin.de/cs/ifb/Ahmed/RoteReihe/2002/TR_02_13.ps.gz`.

[3] EHRIG, KARSTEN: *Konzeption und Implementierung eines Generators für Animationsumgebungen für visuelle Modellierungssprachen*, 2003. `http://www.cs.tu-berlin.de/cs/ifb/Ahmed/RoteReihe/2003/TR2003_17.pdf`.

[4] GRIEBEL, PEER: *ParCon - Dokumentation*. `http://wwwcs.uni-paderborn.de/cs/ag-szwillus/dokumente/papers/Parcon_Handbuch.ps.gz`. Letzter Zugriff: 28.1.2007.

[5] RUDOLF, MICHAEL und GABRIELE TAENTZER: *Introduction to the Language Concepts of AGG*. TU Berlin, 1999. `http://tfs.cs.tu-berlin.de/agg/`.

[6] TU BERLIN: *The AGG 1.5.0 Development Environment; The User Manual*. `http://tfs.cs.tu-berlin.de/agg/`. Letzter Zugriff: 28.1.2007.

[7] TU BERLIN: *AGG Homepage*. `http://tfs.cs.tu-berlin.de/agg/`. Letzter Zugriff: 28.1.2007.

[8] TU BERLIN: *GenGED Homepage*. `http://tfs.cs.tu-berlin.de/~genged/`. Letzter Zugriff: 28.1.2007.