PROGRES

PROgramming with
Graph
REwriting
Systems

PROgrammierte
GRaph-
Ersetzungs-
Systeme

Agenda

PROGRES

- Motivation
- Entwicklung
- Komponenten des Systems
  - Umgebung
  - Sprache
- Anwendungsbereiche
- Vergleich
- Fazit: Stärken/Schwächen

Motivation I

PROGRES

- Bau komplexer interaktiver Systeme
- graphartige Datenstrukturen + Zugriffsoperationen
- Sprachen, Methoden, Werkzeuge
  - Modellierung der Datenstrukturen
  - Beschreibung der Prozesse
  - Validierung der Beschreibungen
  - Umsetzung in korrekt & effizient arbeitende Endprodukte

Motivation II

PROGRES

- Formale Spezifikationssprachen (CIP-L, Larch, Z)
- hohe Programmier-/Prototypsprachen (Smalltalk, Prolog)
- Polymorphe funktionale Sprachen (ML, SETL)
- grafische Spezifikationssprachen (PROGRES)

Motivation III

PROGRES

- Modellierung komplex strukturierter Sachverhalte
- Datenmodell: diane (directed, attributed node & edge) Graph
- regelorientierte und imperative Formulierung der Zugriffsoperationen
- polymorphes & im Wesentlichen statisches Typsystem mit Deklarationspflicht
  (über 300 Konsistenzregeln)
- vollständige formale Definition der statischen und dynamischen Semantik
- imperativ und kompilativ ausführbar

Entwicklung

PROGRES

- Projekt, 1986 - 2006
- Uni BW München, RWTH Aachen
- Prof. Dr. Andy Schürr (plus ca. 50 weitere)
- PROGRESS vs. PROGRES (frz. „Fortschritt“)
- Gesamtsystem: 700.000 Zeilen Code (Modula-3, C, C++, tcl/tk)

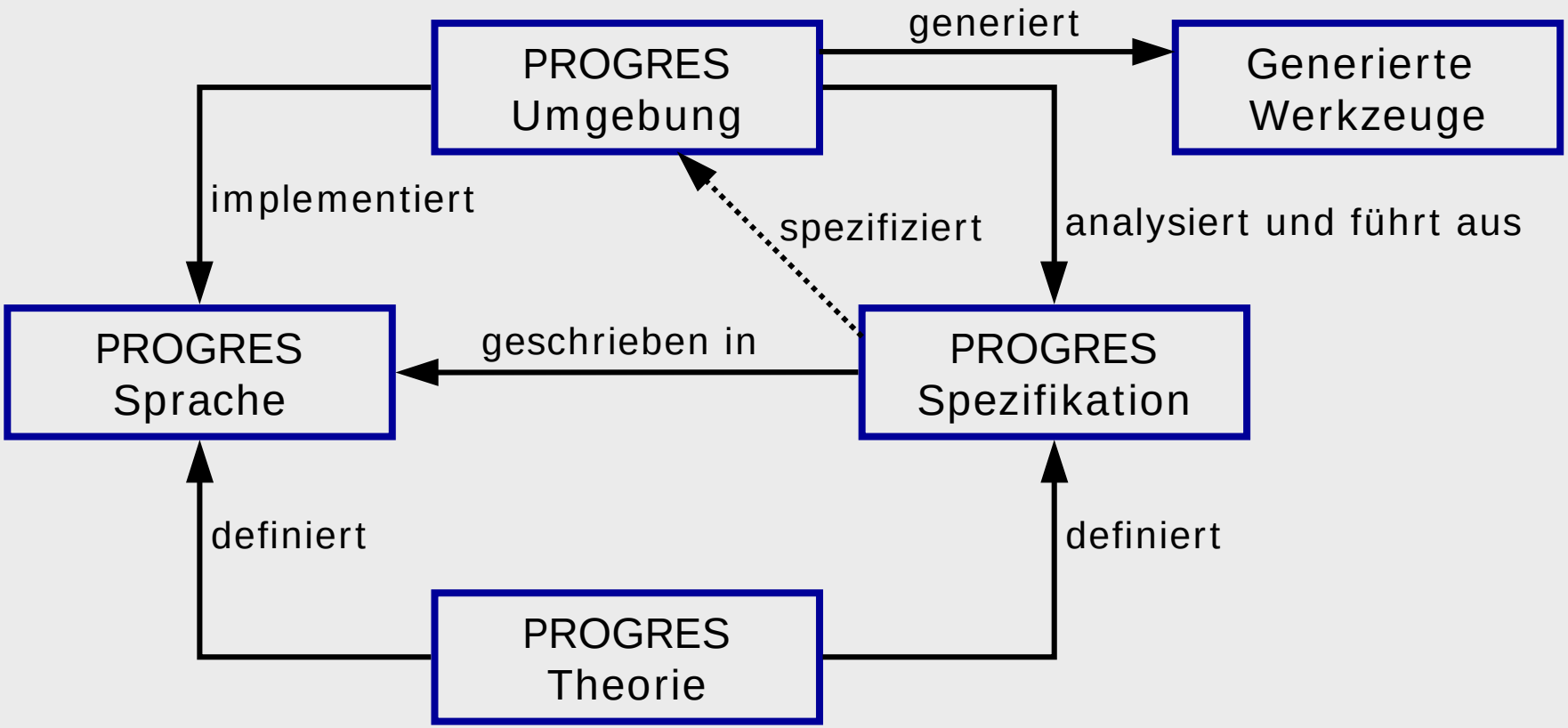
Entwicklungsziele

PROGRES

- Verwendung grafischer Syntax nur wenn angebracht
- basiert auf graphorientiertem DB-System GRAS
- Unterscheidung:
  - Datendefinition
  - Datenmanipulation
- Graph-Klassen-Deklaration für Typprüfungen
- Erkennen von Konflikten in Ersetzungsregeln
- Unterstützung der Programmierparadigmen:
  - regelorientiert
  - imperativ (Definition von Strategien zur Regelanwendung)

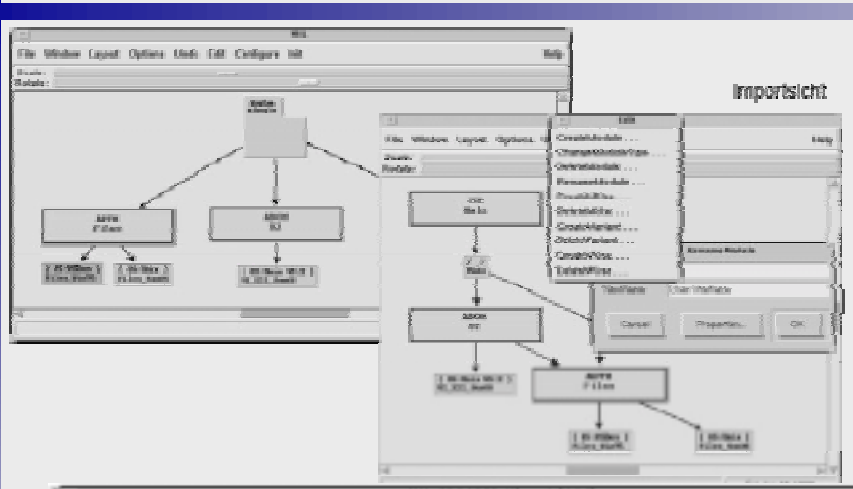
Komponenten des Systems

PROGRES



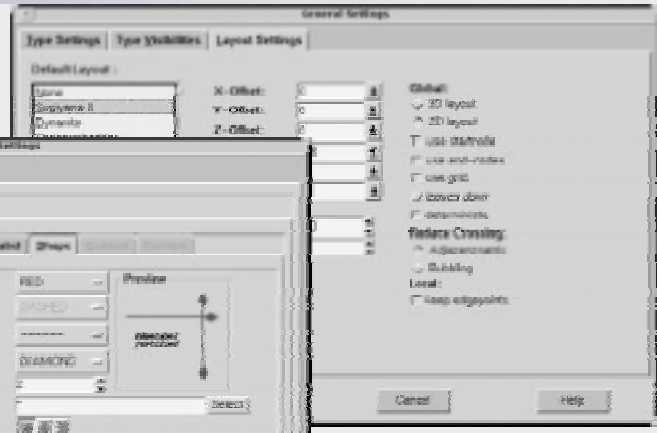
Umgebung

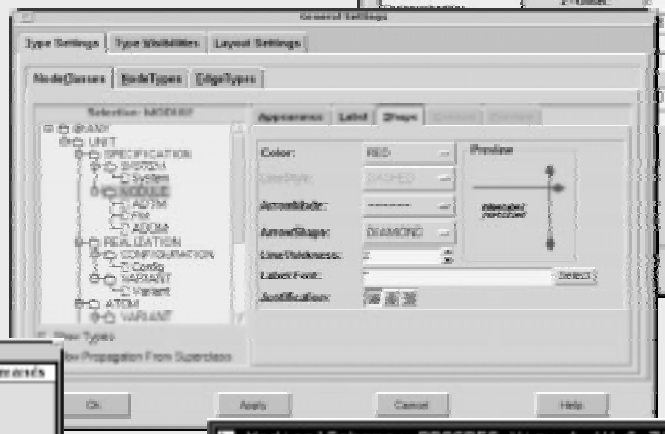
PROGRES

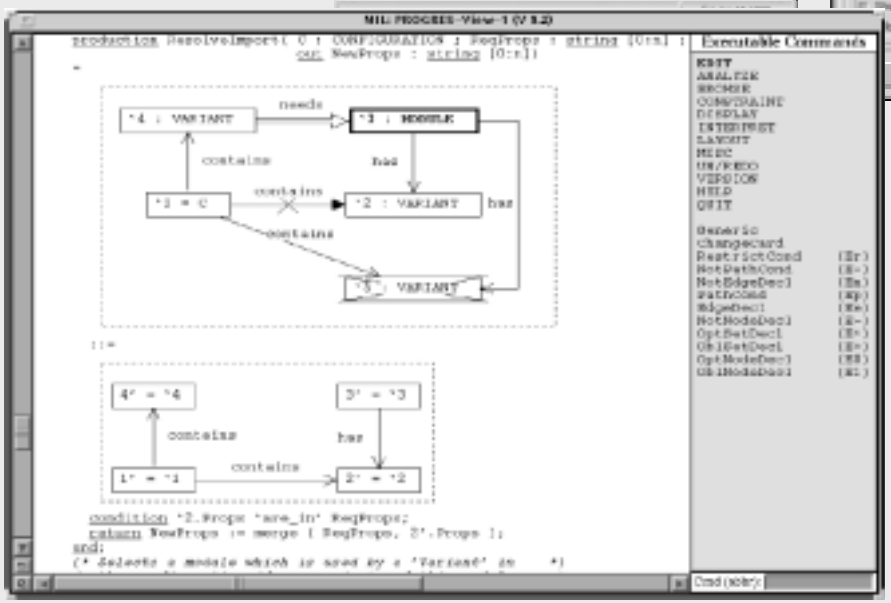


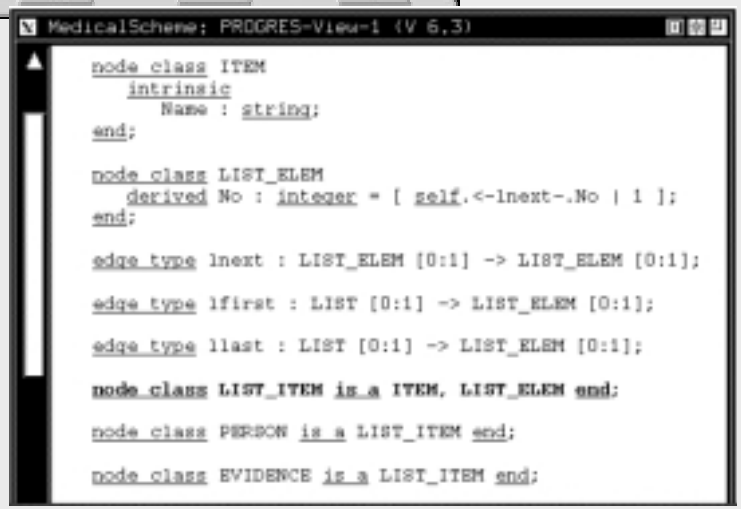
Importsicht

Aussehen dargestellter Kanten:





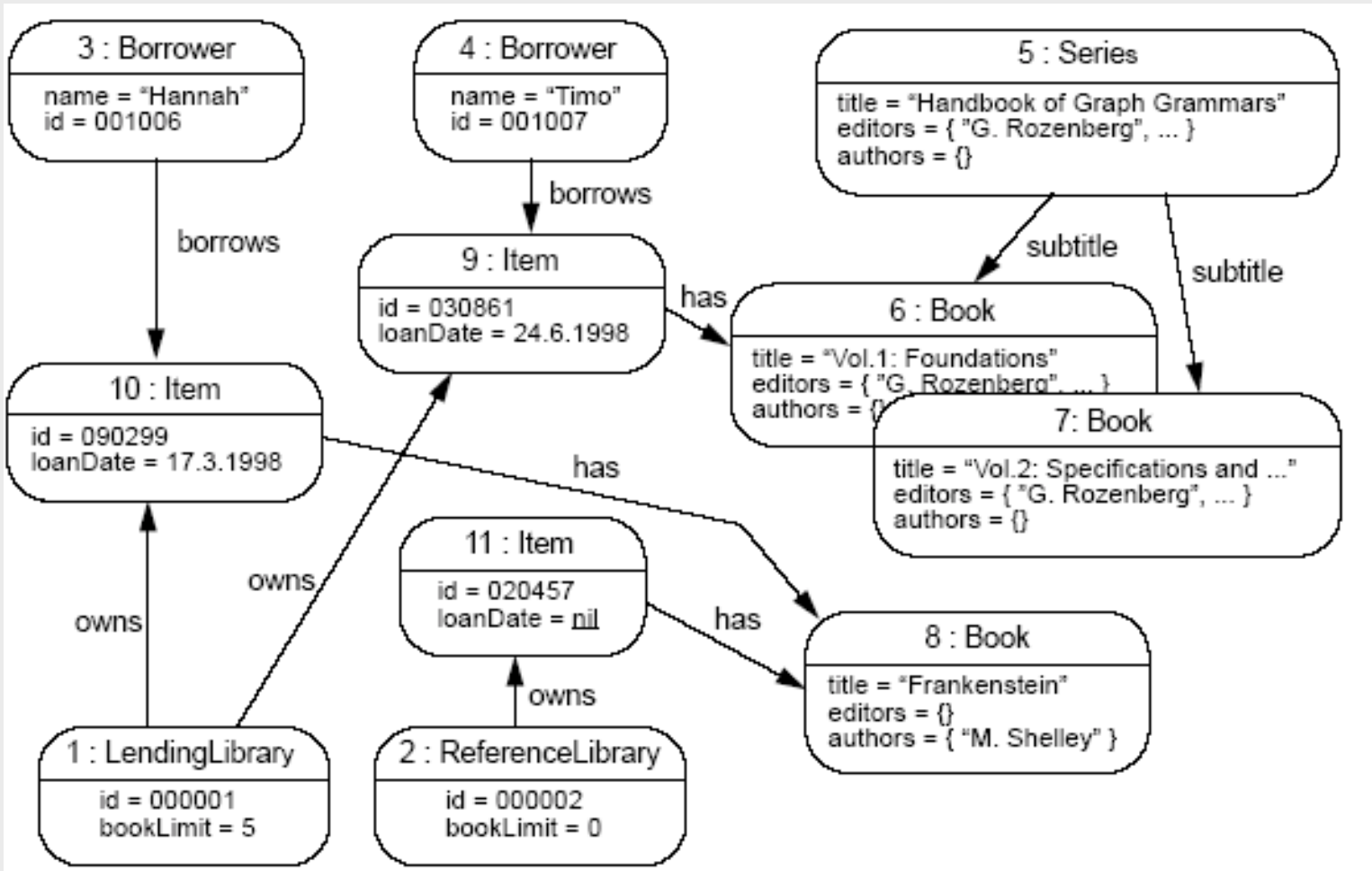




Beispielanwendung

PROGRES

- Bibliotheksverwaltung (Library Information System - LIS)



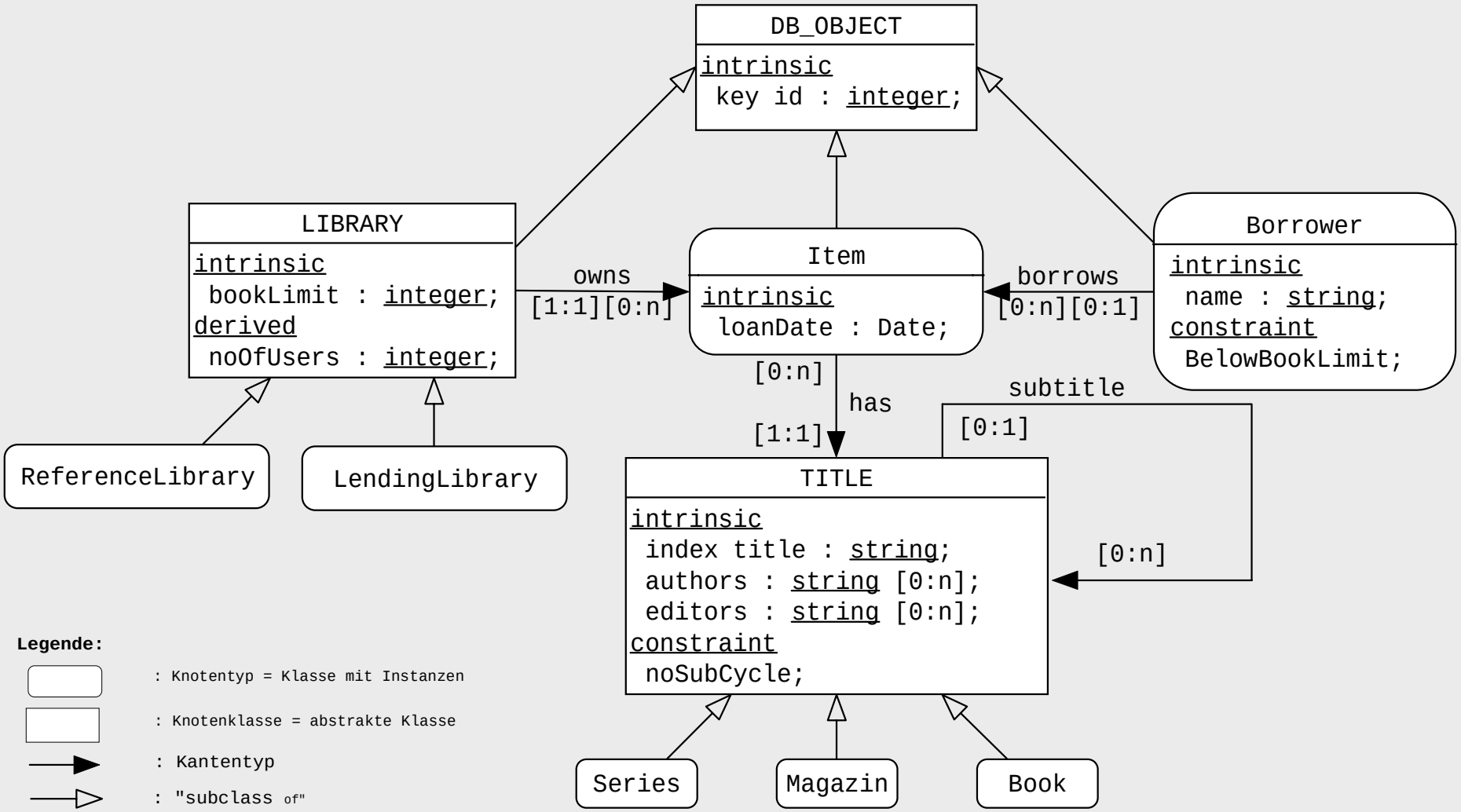
Funktionalität

PROGRES

- Syntaxgesteuerter Text- und Grafikeditor
- micro-emacs Texteditor
- inkrementeller Parser für Anbindung des Texteditors
- inkrementeller Analysator (prüft ca. 300 Konsistenzregeln)
- Graphbrowser
- Übersetzer nach C und Modula-2 (für Werkzeuggenerierung)
- tcl/tk-basierterter Benutzeroberflächengenerator
- „grauenhaft altmodische Benutzeroberfläche“

Graphschema

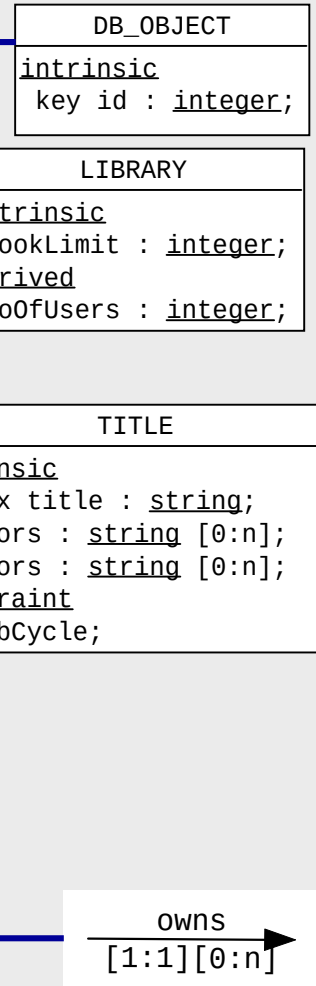
PROGRES



Textuelle Darstellung

PROGRES

```
node class DB_OBJECT end;
node class LIBRARY is a DB_OBJECT
   intrinsic bookLimit : integer := 10;
end;
node type LendingLibrary : LIBRARY end;
node type ReferenceLibrary : LIBRARY end;
...
node class TITLE
   intrinsic index title : string;
                  authors : string [0:n] := nil;
                  editors : string [0:n] := nil;
end;
node type Item : DB_OBJECT
   intrinsic loanDate : Date [0:1];
end;
edge type owns : LIBRARY [1:1] -> Item [0:n];
edge type has : Item [0:n] -> TITLE [1:1];
```



Graph- und Typmodell

PROGRES

- Datenmodell: diane (directed, attributed node & edge)-Graph
- Knoten: 2-stufiges Typsystem
  - Knotenklasse (abstrakte Klasse)

```
node class DB_OBJECT end;
```

    - gemeinsame Attribute verschiedener Knotentypen
    - (Mehrfach-)Vererbung zwischen Klassen möglich
  - Knotentyp

```
node type Item : DB_OBJECT end;
```

    - Klasse mit Instanzen
    - Name, Attribute (intrinsic/derived), Einschränkungen
- Kantentyp

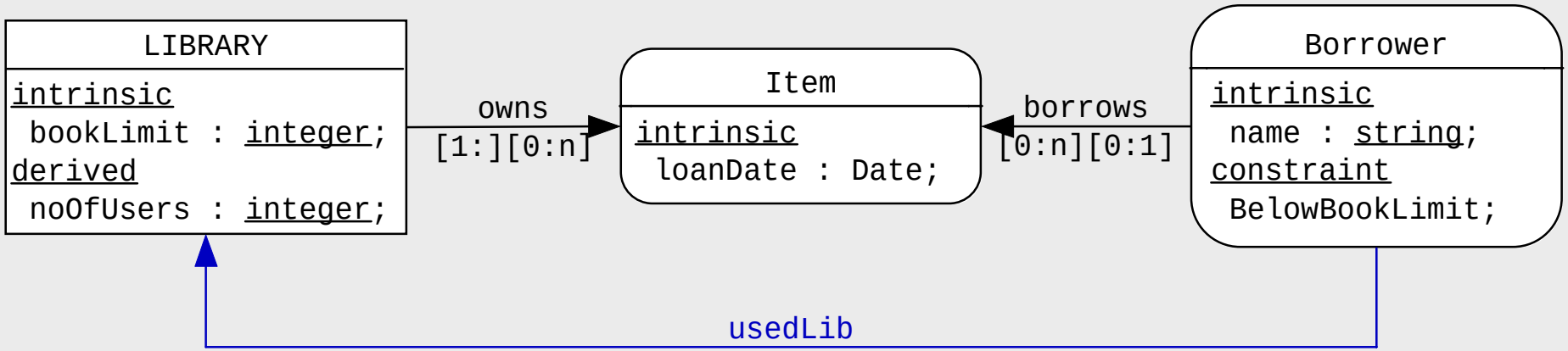
```
edge type owns : LIBRARY [1:1] -> Item [0:n];
```

    - Beziehung zwischen zwei Knotenklassen
    - Typname, Quell-, Zielknotenklasse, Kardinalitäten

Pfadausdrücke

PROGRES

- Relationen zwischen Knoten
- Funktionen auf Knotenmengen
  - vorwärts traversieren: `-edge->`
  - rückwärts traversieren: `<-edge-`
- Pfadverknüpfung mit Operatoren: `&, or, and, but not`



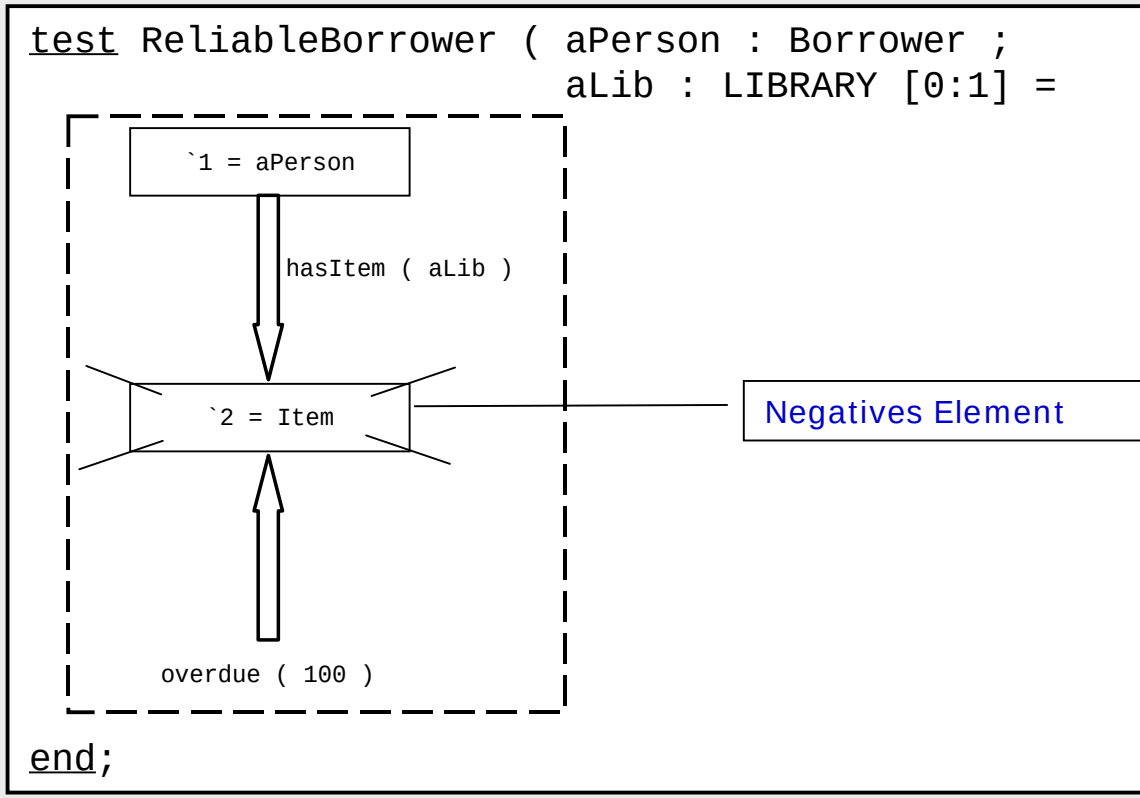

```
path usedLib : Borrower [0:n] -> LIBRARY [0:n] =
-borrows->
& <-owns-end;
```

Abfragen - Teilgraphensuche

PROGRES

- (subgraph test)
  - Existenz
  - Nichtexistenz

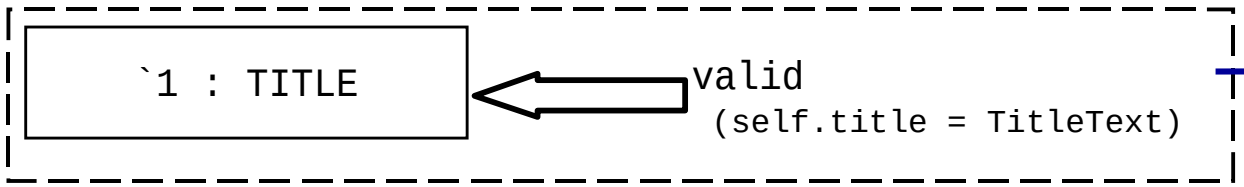
  von Teilgraphen im Arbeitsgraphen
- Query
  - Zusammenbau von Tests zu Abfragen



Ersetzungen - Produktionen

PROGRES

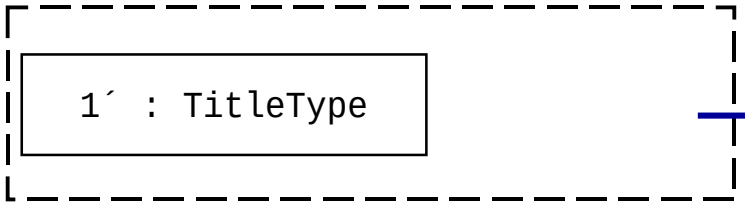
```
production AddTitle( TitleType : type in TITLE;
           TitleText : string
           EditorSet : string [0:n];
           AuthorSet : string [0:n] ) [0:1] =
```



Zwei Graph-Muster:

LHS (left hand side)
linke Seite

```
::=
```



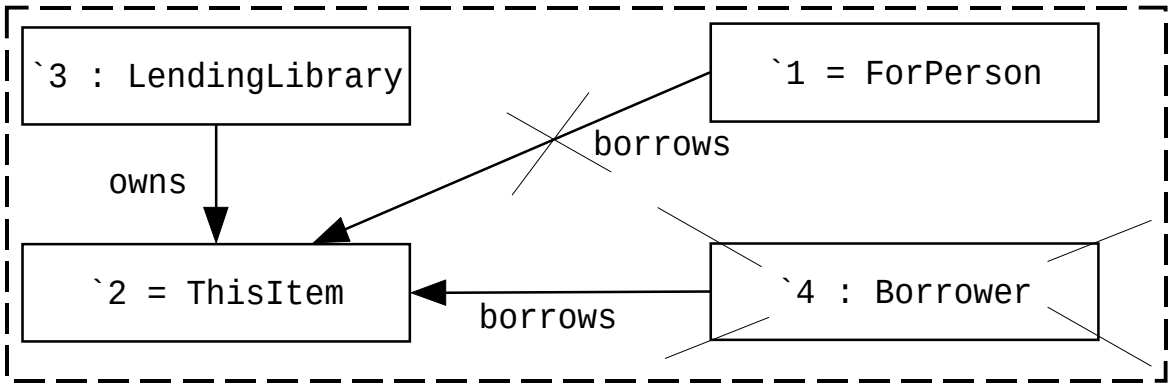
RHS (right hand side)
rechte Seite

```
transfer 1'.title := TitleText;
         1'.editors := Editors; 1'.authors := Authors;
end;
```

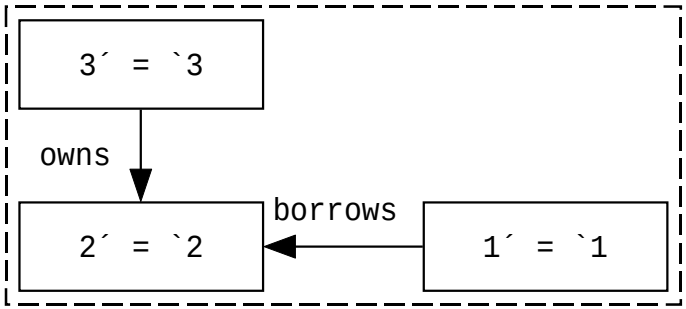
Produktionen

PROGRES

```
production LendItem( ForPerson : Borrower ; ThisItem : Item) [0:1] =
```




```
::=
```




```
  condition card ( ‘1.hasItem ( ‘3 ) ) <= ‘3.bookLimit;
  transfer 2’.loanDate := currentDate;
end;
```

Ersetzungen - Transaktionen

PROGRES

- Zusammenbau von Produktionen
- ACID-Eigenschaften von Datenbanktransaktionen
  - Atomicy
  - Consistency
  - Isolation
  - Duration
- anders als bei Datenbanktransaktionen:
  - Backtracking kann erfolgreich beendete Transaktionen wieder zurücksetzen
  - Effekt einer nichtdeterministischen Transaktion nicht vorhersagbar

Transaktionen I

PROGRES

```
transaction CleanDB [1:1] =
  use Name : string;
      LostItems : Item [1:n] do
   loop
     RemoveOverdueBorrower( out Name, out LostItems )
   & write( Name )
   & for all i := elem ( LostItems ) do
        RemoveItem( i )
     end
   end : [1:1] (* loop as a whole is deterministic *)
  end
end;
```

Wiederholung solange wie möglich

Für jedes Element ausführen

Transaktionen II

PROGRES

```
transaction FindAndLend( InLibraries : LendingLibrary [1:n];
                         Keyword : string;
                         ForPerson : Borrower ) [0:n] =
  use Match : Item do
    choose
        FindItem( InLibraries, Keyword, out Match )
      & LendItem( ForPerson, Match )
    else
        FindItem( InLibraries, Keyword, out Match )
      & ReserveItem( ForPerson, Match )
    end
  end
end;
```

Weg auswählen
(oben -> unten)

Kontrollstrukturen

PROGRES

- Queries
  - Kontrollstrukturen zum Zusammenbau von Tests zu komplizierten Abfragen
- Transaktion
  - Zusammenbau von Produktionen zu komplizierten Graphtransformationen
- ähnlich den Operatoren für Pfadausdrücke

Die Sprache PROGRES

PROGRES

- Spezifikationssprache basierend auf Graphersetzung
- Multiparadigmatisch
  - regelorientierte und visuelle Beschreibung von Graphqueries und Graphtransformationen
  - objektorientierte Modellierung von Graphschemata
  - deklarative Definition und inkrementelle Berechnung abgeleiteter Attribute
  - imperative Programmierung
  - nichtdeterministische Programmierung mit Tiefensuche und Backtracking

Anwendungsbereiche

PROGRES

- Spezifikation integrierter Software-Engineering-Umgebungen (CASE)
  - Werkzeuge
  - Datenstrukturen
- Beschreibung von Werkzeugen in CIM-Umgebungen
  - Prozessmodellierung
  - Versionsverwaltung
  - Konfigurationsmanagement
- Definition
  - Semantik der Abfragesprache visueller Datenbanken

Vergleich

PROGRES

	PROGRES [18]	SETL [7]	Prolog [4]	OPS-5 [13]	ML [11]	Smalltalk [8]
Datenmodell	Attributierte Graphen	Mengen	n-stellige Relationen	Attribut-listen	Terme u. Funktionen	Attributierte Objekte
Programmier-Paradigma	regelorientiert und imperativ	imperativ	logikorientiert	regelorientiert	funktional	objekt-orientiert
Typkonzept	polymorph und obligate Dekl.	Laufzeittests u. opt. Dekl.	kein Typkonzept	kein Typkonzept	polymorph u. Typinferenz	Laufzeittests
Zustandsraum	Graph	Variablen	Faktenbasis	Faktenbasis	—	Objekthalde
Mustersuche	Heuristiken für eine Regel	—	—	RETE-Netz für Regelmenge	—	—
Abgeleitete Fakten	inkr. berechn. Attr. und Relationen	—	Batch-Inferenz von Relationen	—	—	—
Backtracking	implizit kontrolliert	explizit kontrolliert	implizit und ¬ Faktenbasis	—	—	—
Konkr. Syntax	Grafik/Text	Text	Text	Text	Text	Text

Schwächen von PROGRES

PROGRES

- Gewöhnungsbedürftig
  - eigene Syntax für Kontrollstrukturen und Attributausdrücke
- Effizienzprobleme
  - Backtracking
  - abgeleitete Attribute
- Modul- bzw. Paketkonzept erst in Entwicklung
- viel zu viele Sprachelemente
- Nicht unterstützt:
  - Erstellung von Korrektheitsbeweisen

Stärken

PROGRES

- Multiparadigmatisch
  - insbesondere imperative mit visueller regelbasierter Programmierung
  - Entwurf von Klassendiagrammen
- Hybrid
  - ausgewogene Mischung aus grafischen und textuellen Sprachelementen
- statische Konsistenzbedingungen inkrementell überprüfbar
- Unterstützung funktionaler Abstraktion

...wird nicht nur von seinen Erfindern benutzt

Vielen Dank für die Aufmerksamkeit!

Für Weiterleser:

google „Andy Schürr“ PROGRES Graphersetzung