Graph-basierte Modellierung in Software-Werkzeugen

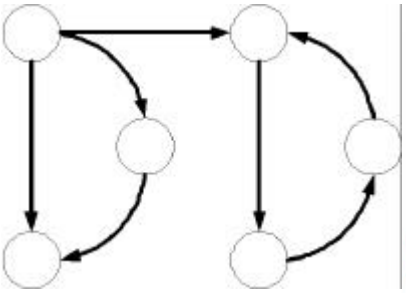
Jürgen Ebert
Universität Koblenz-Landau

Inhalt

- Einleitung
- Modellierung mit Graphen
- Modellierung mit Graphklassen (Schemata und Constraints)
- Implementation von Graphen (Algorithmen und Anfragen)
- Software-Werkzeuge (Interoperabilität und GUPRO)
- Zusammenfassung

Einleitung
(Was sind Graphen?)

Graphen



Arten von Graphen

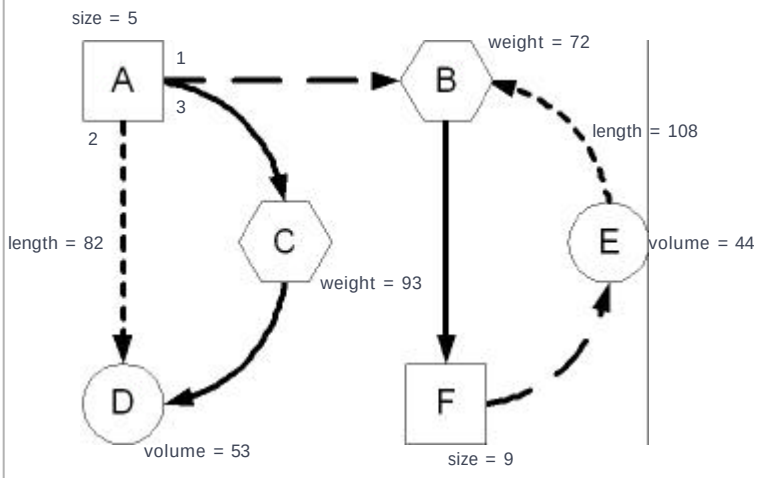
- gerichtet/ungerichtet
- mit/ohne Mehrfachkanten
- mit/ohne Schlingen
- markiert/unmarkiert
- azyklisch/planar/vollständig/…

Definition

Ein (endlicher) gerichteter **Graph** $G = (V, E, \phi)$ ist besteht aus durch

- einer endliche nichtleere Menge V (engl. vertex set) von **Knoten**,
- eine endliche Menge E (engl. edgeset) von **Kanten** mit $V \cap E = \varnothing$, und
- einer **Inzidenzabbildung** $\phi : E \rightarrow V \times V$

typisiert, attributiert, geordnet



TGraphen

Typisierte, attributierte und angeordnete gerichtete Graphen heißen **TGraphen**.

Modellierung mit Graphen

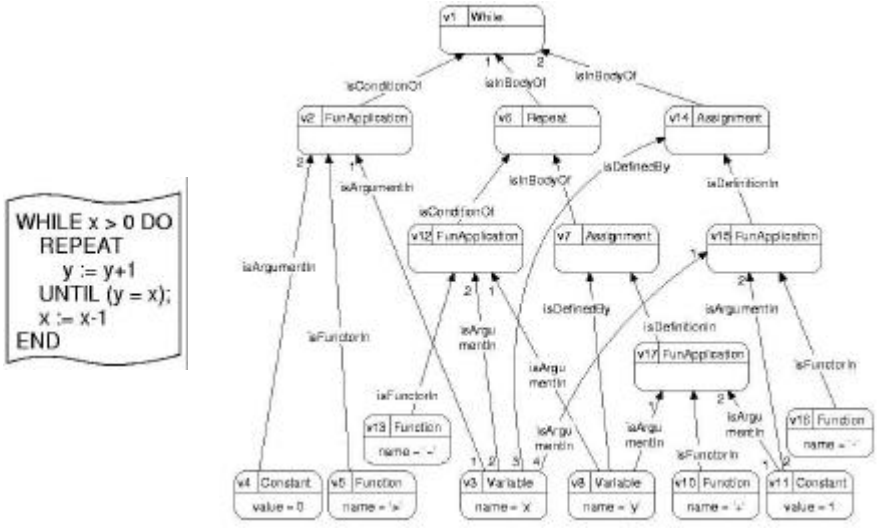
Graphen als Modelle

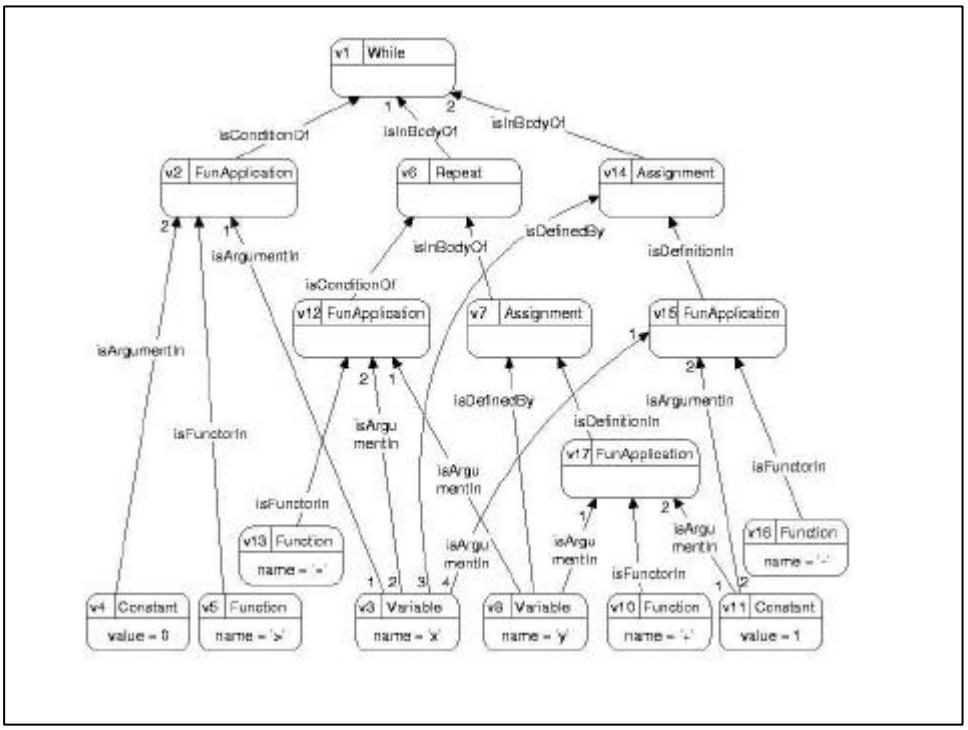
Ein **Modell** ist ein ~~zielgerichtetes~~ Abbild eines Systems (i.w.S.), das die Realität durch ~~Abstraktion~~ auf die problemrelevanten Aspekte vereinfacht und hierzu ähnliche Beobachtungen und Aussagen ermöglicht wie das Ausgangssystem.

zielgerichtete
~~Abstraktion~~

Source Code Fragment

```
WHILE x > 0 DO
   REPEAT
      y := y+1
   UNTIL (y = x);
   x := x-1
END
```



Vorteile der Graphenmodellierung

Graphen sind gleichzeitig

- mächtig
  d.h. sie sind sinnvolle Abstraktionen
  für viele Sachverhalte
- formal
  d.h. sie sind mathematische Objekte,
  über die man räsonnieren kann
- anschaulich
  d.h. man kann sie visualisieren
  und somit auch intuitiv wahrnehmen
- effizient
  d.h. man kennt zahlreiche schnelle
  Verfahren

Modellierung mit Graphklassen

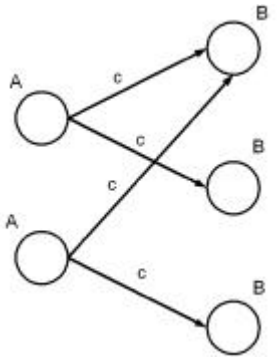
Graphklassen

Eine Klasse von Graphen wird definiert durch

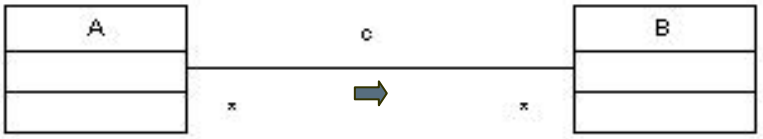
- ein Schema
  (meta model)
- zusätzliche Integritätsbedingungen
  (constraints)

Beispiel

Bipartite Graphen sind Graphen mit zwei Knotentypen, deren Kanten nur zwischen Knoten verschiedenen Typs verlaufen.



Schema

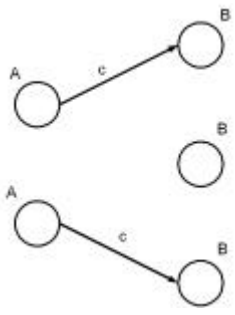


Integritätsbedingungen

Ein Matching ist ein bipartiter Graph, bei dem jeder Knoten nur mit höchstens einer Kante inzident ist.

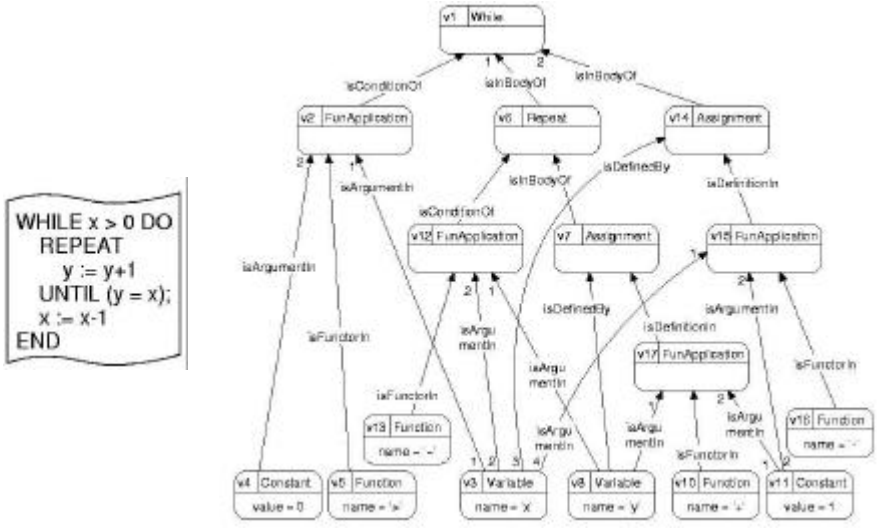
$$\forall a : A \bullet \delta_c^+(a) \leq 1 \wedge \forall b : B \bullet \delta_c^-(b) \leq 1$$

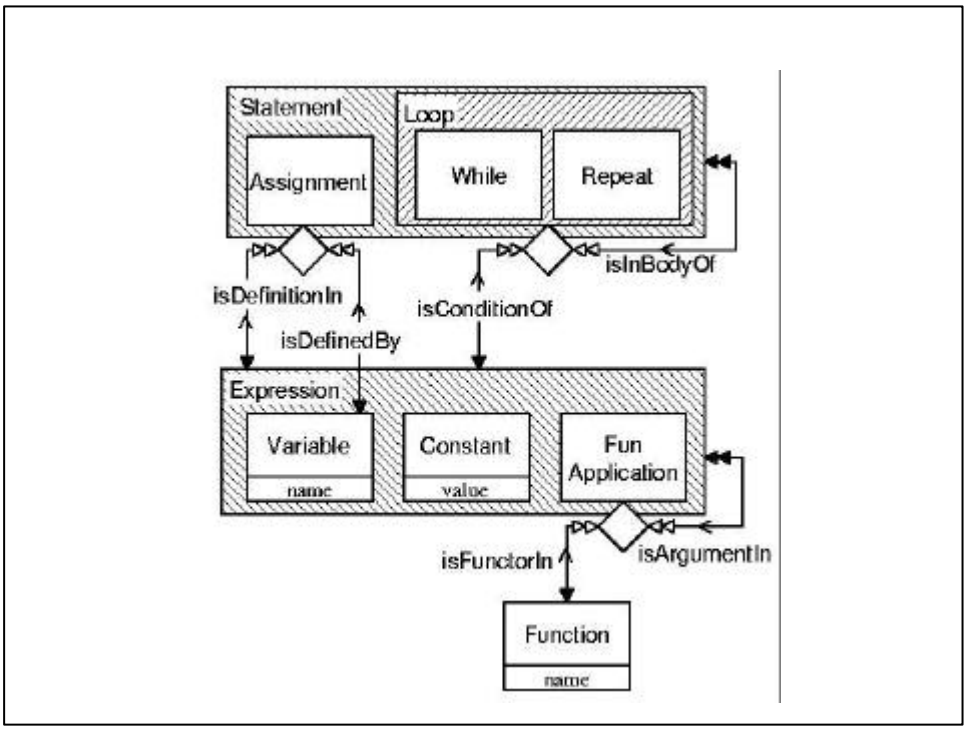
Matching

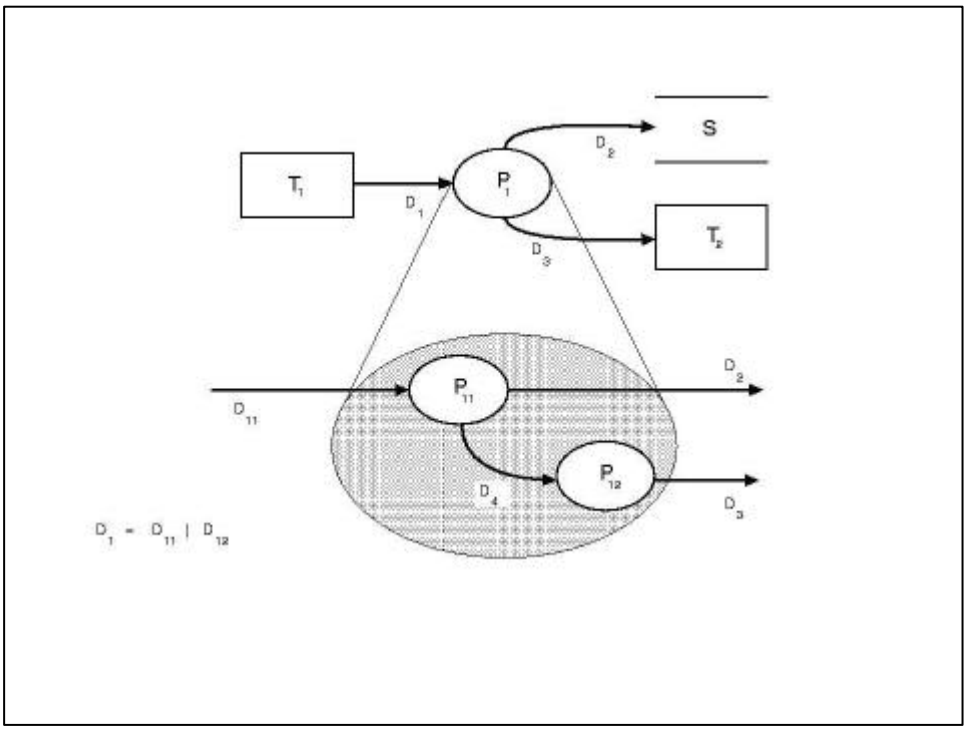


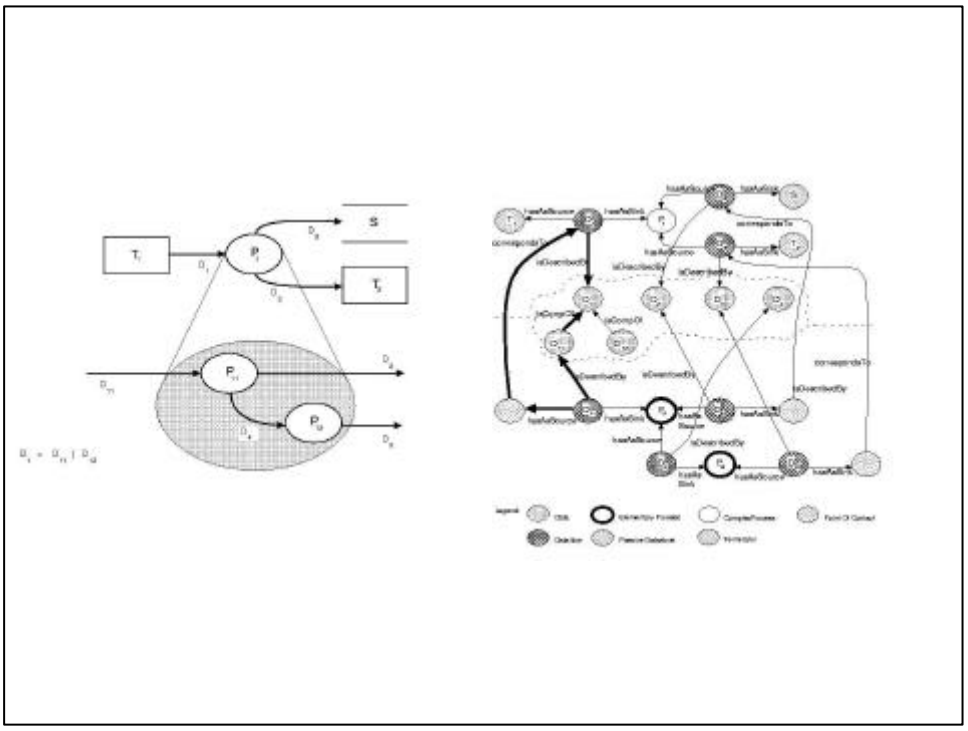
$$\forall a : A \bullet \delta_c^+(a) \leq 1 \wedge \forall b : B \bullet \delta_c^-(b) \leq 1$$

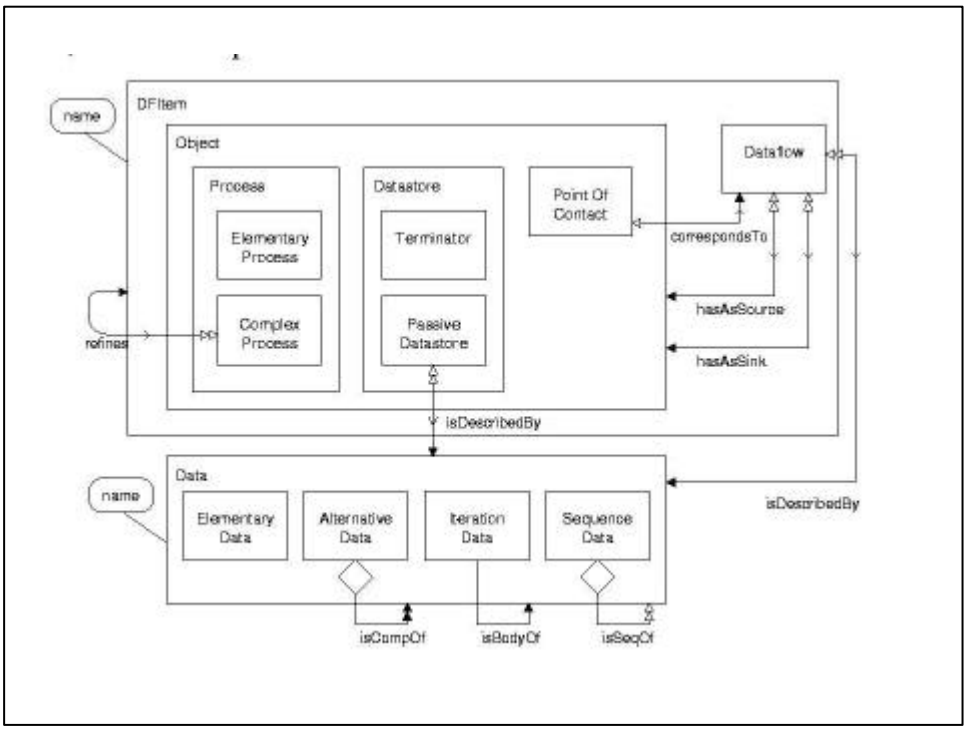
Modellierung mit Graphklassen (Schemata)




Statement
Loop
Assignment
While
Repeat
isInBodyOf
isDefinitionIn
isConditionOf
isDefinedBy
Expression
Variable
name
Constant
value
Fun
Application
isFunctorIn
isArgumentIn
Function
name


S
T1
P1
D1
D2
D3
T2
P11
D11
D2
P12
D4
D3
D1 = D11 | D12




name
DFItem
Object
Process
Elementary Process
Complex Process
Datastore
Terminator
Passive Datastore
Point Of Contact
Dataflow
corresponsdsTo
hasAsSource
hasAsSink
refines
isDescribedBy
Data
name
Elementary Data
Alternative Data
Iteration Data
Sequence Data
isDescribedBy
isCompOf
isBodyOf
isSeqOf

Prinzipien der Modellierung (1)

Die Modellierung geschieht objekt-basiert:

- Jedes identifizierbare relevante Objekt wird durch genau einen Knoten repräsentiert.
- Jede Beziehung zwischen zwei Objekten wird durch eine Kante repräsentiert.

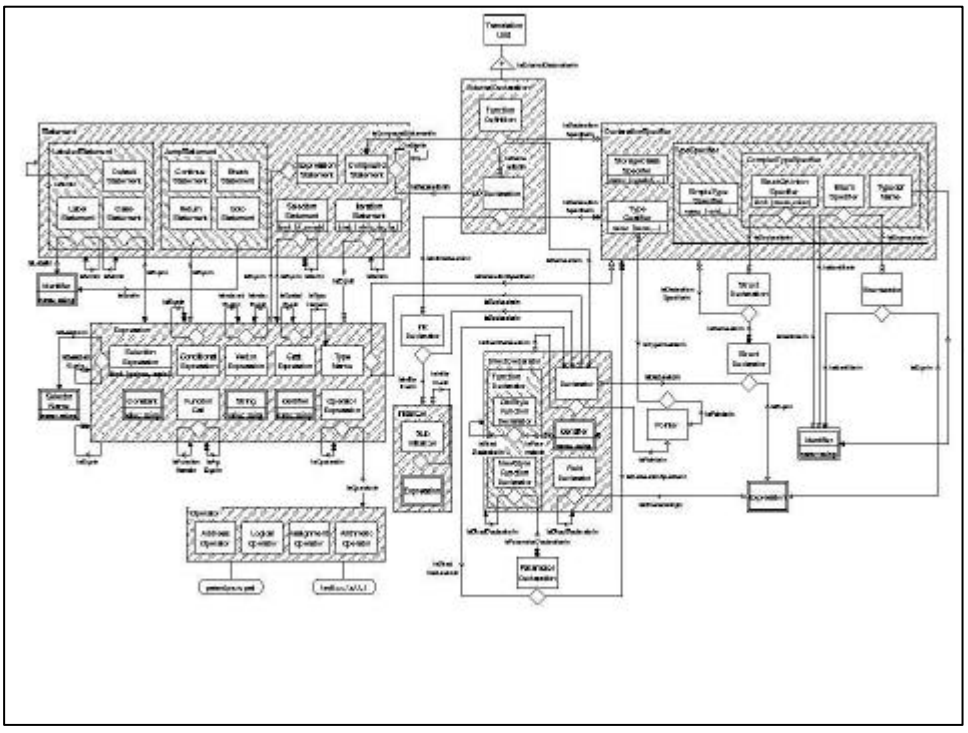
Prinzipien der Modellierung (2)

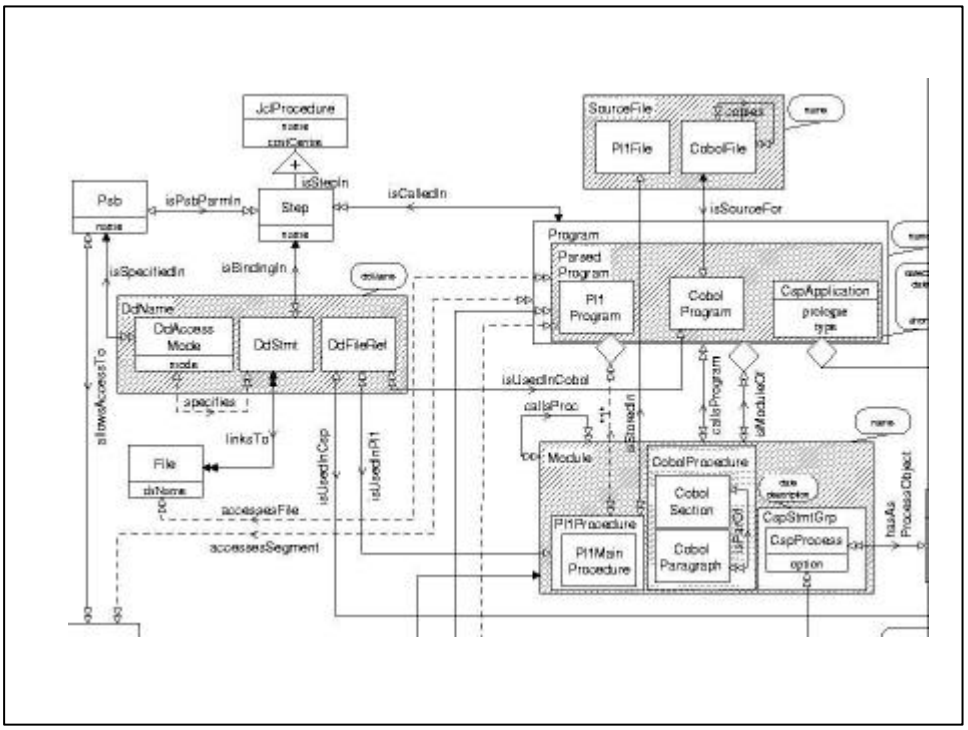
- Objekte mit gemeinsamen Eigenschaften erhalten denselben Typ.
- Eigenschaften von Objekten werden durch Attribute beschrieben.
- Die Vorkommensreihenfolgen von Beziehungen werden durch Kantenreihenfolgen abgebildet.

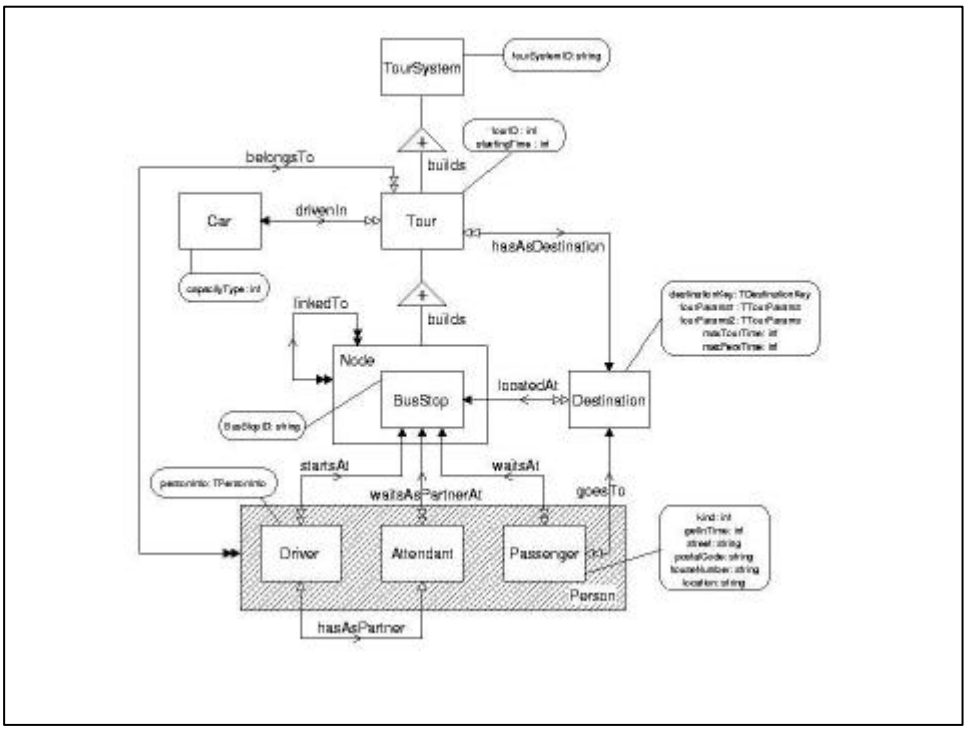
Sichten auf das Schema

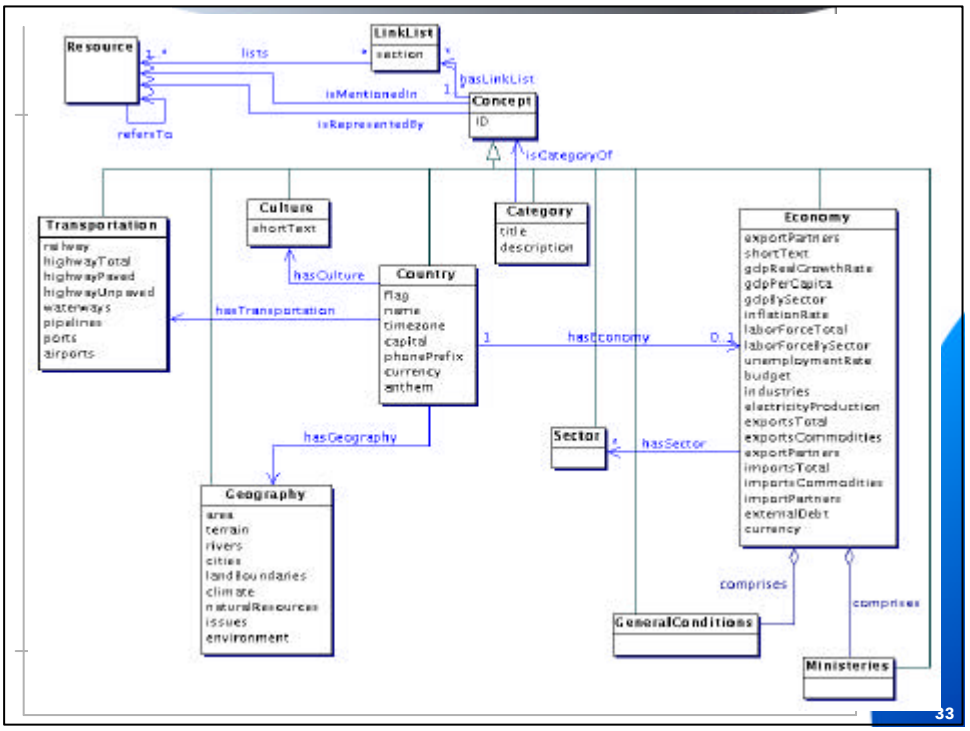
Graph-Schemata sind gleichzeitig

- Konzeptuelle Beschreibungen des Anwendungsbereichs
- Formale Beschreibungen der Daten
- Anleitungen zur Implementation




JclProcedure
Psb
isPsbParmIn
Step
isStepIn
isCalledIn
SourceFile
Pl1File
CobolFile
isSourceFor
Program
Parsed Program
Pl1 Program
Cobol Program
CapApplication
prologue
isSpecifiedIn
isBindingIn
DdName
DdAccess Mode
DdStmt
DdFileRel
specifies
isUsedInCobol
callsProc
callsProgram
isModuleOf
Module
CobolProcedure
Cobol Section
Cobol Paragraph
Pl1Procedure
Pl1Main Procedure
CapStmtGrp
CapProcess
option
hasAs ProcessObject
allowsAccessTo
linksTo
File
dbName
accessesFile
accessesSegment
isUsedInCap
isUsedInPl1


TourSystem
builds
belongsTo
Car
drivenIn
Tour
hasAsDestination
linkedTo
Node
BusStop
locatedAt
Destination
startsAt
waitsAt
waitsAsPartnerAt
goesTo
Driver
Attendant
Passenger
Person
hasAsPartner



Arten der Modellierung

Die Modellierung von (textuellen oder visuellen) Artefaktsprachen kann erfolgen

- aus verschiedenen Sichten
- auf verschiedenen Granularitätsstufen
- für verschiedene Sprachen
- integriert

Modellierung mit Graphklassen (Integritätsbedingungen)

Schema Constraints

Häufig sind nicht alle durch das Schema beschriebenen TGraphen zulässig. Sie müssen evtl. weitere Integritäts-bedingungen erfüllen.

Diese Bedingungen können durch eine Einschränkungssprache erfasst werden:

GRAL

for G **in** DFD **assert**

p_5: $isDag(refines)$;

p_6: $\{s_1, s_2 : Datastore \mid s_1 \leftarrow_{hasAsSink} \rightarrow_{hasAsSource} s_2\} = \emptyset$;

p_7: $\forall p : ComplexProcess \bullet p(\leftarrow_{hasAsSource} | \leftarrow_{hasAsSink}) =$
$p \leftarrow_{refines} \odot PointOfContact \rightarrow_{correspondsTo}$;

p_8: $\forall c : PointOfContact \bullet$
$c \rightarrow_{correspondsTo} \rightarrow_{isDescribedBy}$
$(\leftarrow_{IsCompOf} | \leftarrow_{isBodyOf} | \leftarrow_{isSeqOf})^*$
$\leftarrow_{isDescribedBy} (\rightarrow_{hasAsSink} | \rightarrow_{hasAsSource}) c$;

p_9: $\forall d : Dataflow \bullet$
$d(\rightarrow_{hasAsSource} | \rightarrow_{hasAsSink}) \odot PassiveDatastore \rightarrow_{isDescribedBy}$
$(\leftarrow_{IsCompOf} | \leftarrow_{isBodyOf} | \leftarrow_{isSeqOf})^* \leftarrow_{isDescribedBy} d$

GRAL

GRAL (GRaph Specification Language) ist eine Z-*Erweiterung* zur Beschreibung von Grapheigenschaften.

GRAL ist so *eingeschränkt*, dass es algorithmisch zugänglich ist, d.h. das Testen von GRAL-Prädikaten auf einem konkreten TGraphen ist polynomial.

GRAL-Erweiterungen

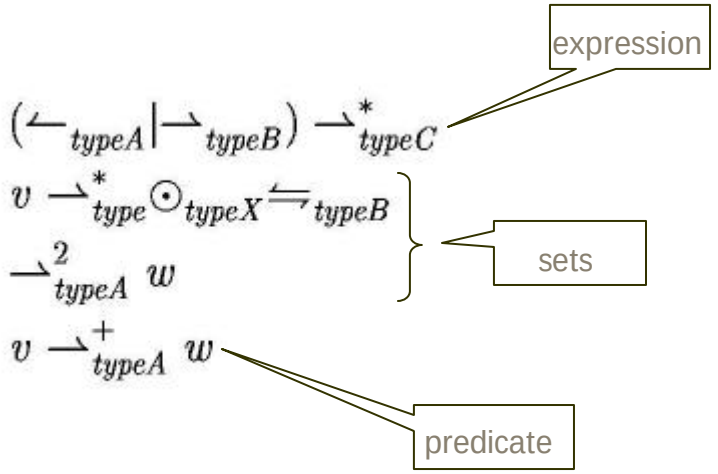
- Basisprädikate über Graphelemente (*isIsolated*, *isLoop*) und Graphen (*isTree*, *isAcyclic*, *isConnected*)
- Funktionen von Typbezeichnern auf Knotenmengen (*vertexClass*), Kantenbezeichner (*edgeClass*) or induzierte Teilgraphen (*eGraph*, *vGraph*)
- Reguläre Pfadausdrücke

Pfadausdrücke

Pfadausdrücke sind reguläre Ausdrücke über Knoten und Kantentypen:

- Konkatenation,
- Fallunterscheidung und
- Iteration.

Pfadausdrücke



GRAL-Einschränkungen

- Quantoren haben nur endliche Bereiche
- Alle Prädikate und Funktion sind polynomial berechenbar

Implementation mit Graphen

Implementation mit Graphen (Algorithmen)

GraLab

TGraphen können als Datenstruktur (mit allen ihren Eigenschaften) effizient implementiert werden.

Es gibt Klassenbibliotheken für TGraphen:

- GraLab 4.0 (C++)
- JGraLab (Java)

Properties of (J)Gralab

Eigenschaften

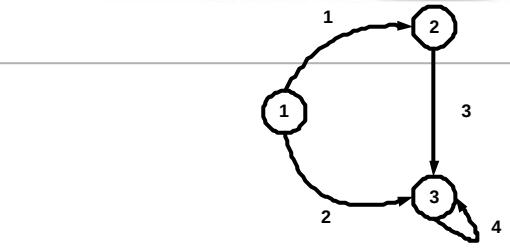
- Kanten sind Objekte erster Ordnung
- Kanten können in beide Richtungen traversiert werden
- Mehrfachkanten sind erlaubt
- Persistente und temporäre Attribute sind möglich
- Gerichtete und ungerichtete Sichten existieren nebeneinander
- Traversierung ist effizient

```
procedure dfs (v: vertex);
    num := num + 1;
    v.number := num;
    for all e in Λ+ (v) do
        w := omega (e);
        if w.number = 0 then
            w.parent := e;
            dfs (w)
        end
    end
end .
```

temporary attribute

control structure

function



	-4	-3	-2	-1		1	2	3	4
node	3	2	1	1		2	3	3	3
next	0	4	-3	3		2	0	0	-4

first	1	-1	-2
	1	2	3

Algorithmenentwicklung

Algorithmen auf TGraphen können

- in TGraph-orientiertem Pseudocode entwickelt und
- direkt implementiert werden (optimiert für Traversierungen).

Implementation mit Graphen (Anfragen)

Graphanfragen

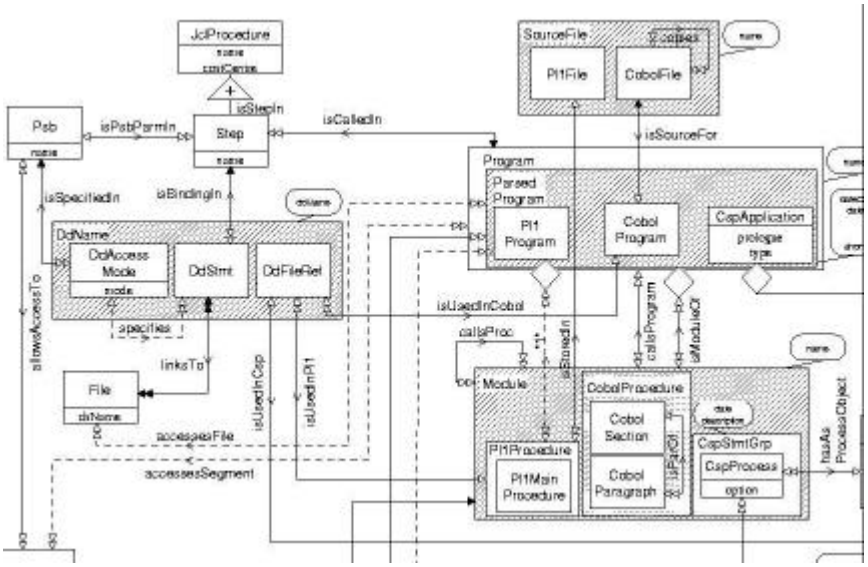
Die Extraktion von Information aus einem Graphen kann durch Anfragen an den Graphen erfolgen

GReQL (GRaph Query Language)

FROM *<variable declarations>*

WITH *<GRAL-condition>*

REPORT *<expression list>* END



GReQL Example

```
FROM jcl:V{JclProcedure}
REPORT jcl.name AS JCL_Proc,
       FROM called:V{Program}
       WITH jcl <--{isStepIn}<--{isCalledIn}
                (<--{isModuleOf}-->{callsProgram})*
                called
       REPORT called.name END
       AS recursivelyCalledProg
END
```


Software-Werkzeuge

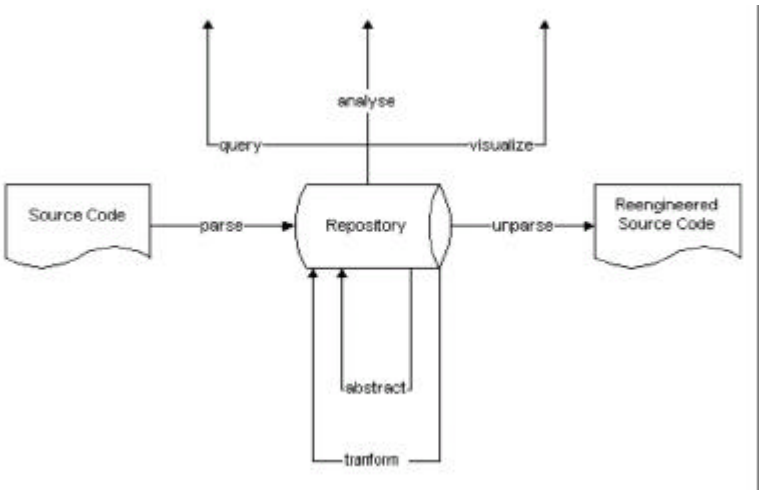
Umgebungen

Die Bandbreite softwaretechnischer Werkzeuge ist sehr breit und erstreckt sich über alle Phasen

- CASE-Tools
- Programmier-Tools
- Reengineering-Tools
- …

Moderne Tools werden zu Umgebungen integriert.

Repository-Basierung



Repository-Technologien

Für die praktische Implementation werden verschiedenen Technologien verwendet, z.B.

- TGraphen (Gupro)
- Relationen (grok)
- relationale Datenbanken (Sneed)
- oo-Datenbanken
- XML-Text
- Prolog (shore)

Software-Werkzeuge (Interoperabilität)

GXL

Konzeptionell sind Repository-Daten spezielle TGraphen.

GXL (Graph Exchange Language)
ist eine XML-Sprache zum Austausch von Graphschemata und Graphen.


```
<?xml version = "1.0" ?>
<!DOCTYPE gxl SYSTEM "gxl.dtd" >
<gxl>
<graph id = "simpleGraph"
            edgeids = "true" >
   <type xlink:href = "schema.gxl" />
   <node id = "v1" >
      <type xlink:href =
        "schema.gxl#Function" />
      <attr name = "name" >
        <string>main</string>
      </attr>
   <node id = "v2" >
      <type xlink:href =
        "schema.gxl#FunctionCall" />
   </node>
   <node id = "v3" >
      <type xlink:href =
        "schema.gxl#FunctionCall" />
   </node>
   ...
```

```
   ...
   <edge id = "e8"
      from = "v7" to = v3"
      toorder = "1"/>
      <type xlink:href = "schema.
        gxl#isInput" />
   </edge>
   <edge id = "e9"
      from = "v6" to = v2"
      <type xlink:href = "schema.
        gxl#isOutput" />
   </edge>
   <edge id = "e10"
      from = "v7" to = v3"
      <type xlink:href = "schema.
        gxl#isOutput" />
   </edge>
</graph>
</gxl>
```

GXL erlaubt die Beschreibung von

- TGraphen
- Hypergraphen
- Hierarchischen Graphen

GXL ist in der Reengineering-Welt weit verbreitet.

Software-Werkzeuge (GUPRO)

In der Werkzeug-Umgebung

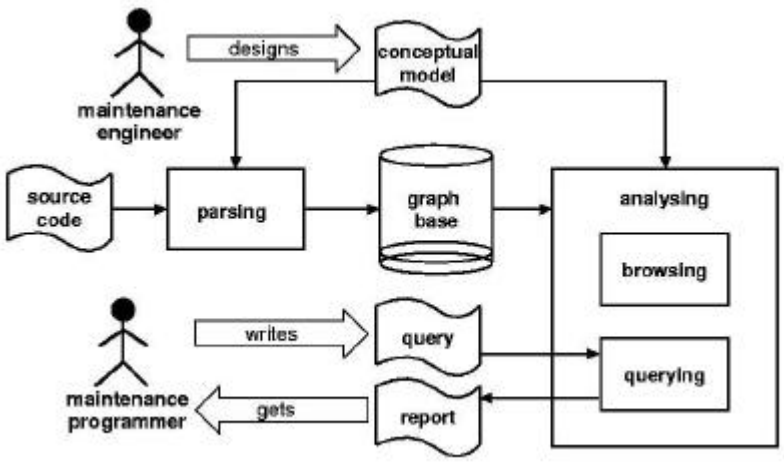
GUPRO

(Generische Umgebung zum Programmverstehen)

werden alle relevanten Informationen in Graphen gehalten.

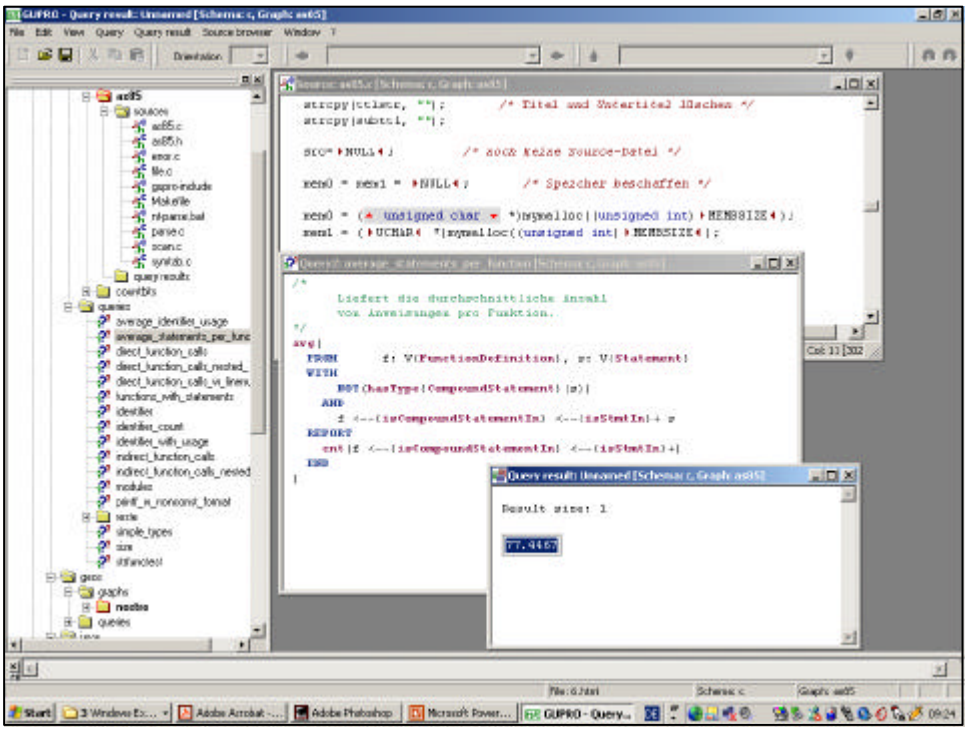
Dadurch, dass die Graphklassen durch Schemata spezifiziert sind, wird die Generizität erreicht.

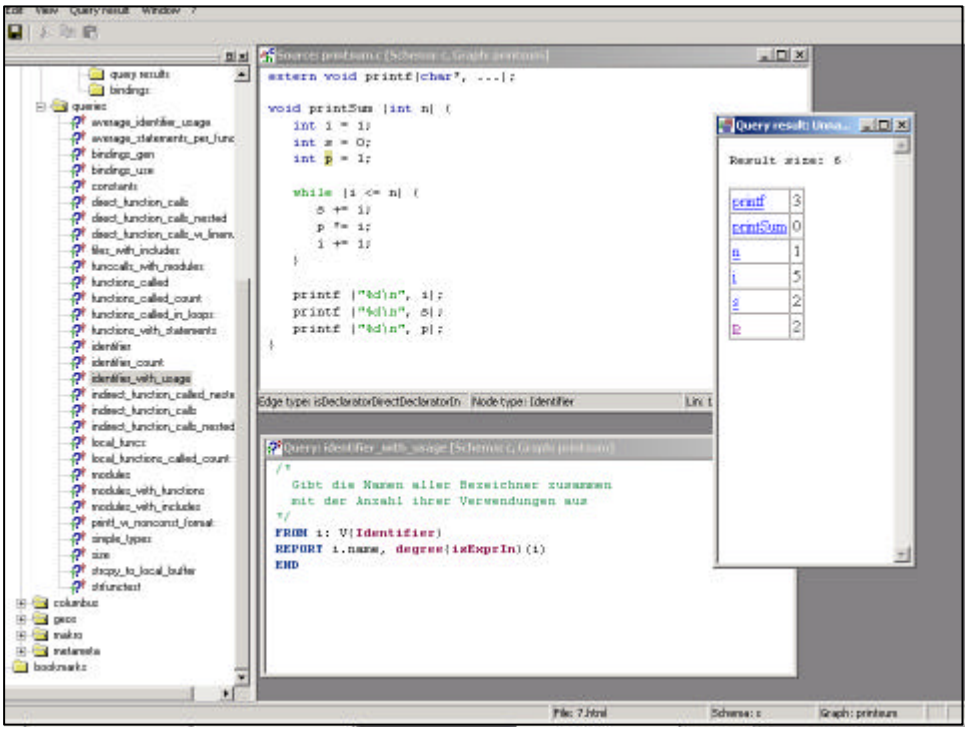
GUPRO: Datenfluss-Architektur



GUPRO unterstützt das Reengineering und Programmverstehen durch

- Anfragen an Graphen (querying)
- Schema-basiertes Navigieren durch Graphen (browsing)
- Visualisierung des Graphen durch Programm-text unter Berücksichtigung des Präprozessors (folding)





Zusammenfassung

Inhalt

- Modellierung mit Graphklassen (Schemata und Constraints)
- Implementation von Graphen (Algorithmen und Anfragen)
- Software-Werkzeuge (Interoperabilität und GUPRO)

- TGraphen sind ein mächtiges Mittel zur Modellierung.
- Sie eignen sich besonders gut zur Modellierung von Artefakten in Software-Werkzeugen.
- Sie sind gleichzeitig anschaulich, formal und effizient.

- Sie bieten eine günstige Abstraktionsebene für Interoperabilität von Tools (abstrakt genug und inhaltsreich).
- Sie bieten eine gute Basis für generische Werkzeuge, da Graphklassen durch Schemata und Bedingung definiert werden können.
- Es gibt Ansätze für eine zusammenhängende Technologie (grUML, GRAL, GReQL, GXL).