

Jürgen Ebert
University Koblenz-Landau
<http://www.uni-koblenz.de/~ebert>

METAMODELS TAKEN SERIOUSLY THE TGRAPH APPROACH



GOALS OF THIS TALK

- × lightweight tutorial on metamodeling
- × consistent basis for generic tools/services
- × powerful and rigorous approach
- × seamless and efficient implementation

OVERVIEW

- + Reverse Engineering Tools/Services
- + Artifacts (Modeling)
- + Artifact Languages (Metamodeling)
- + TGraphs and their Formalization
- + TGraphs and their Implementation
- + Applications
- + Conclusion

REVERSE ENGINEERING TOOLS/SERVICES

REVERSE ENGINEERING

... the reconstruction of (more) abstract knowledge from given software engineering artifacts

REVERSE ENGINEERING TASKS

The CSMR community is dealing with several concrete reverse engineering tasks:

- × recognition of class candidates
- × identification of subsystems
- × detection of clones
- × slicing
- × pattern detection
- × ...

ARTIFACTS

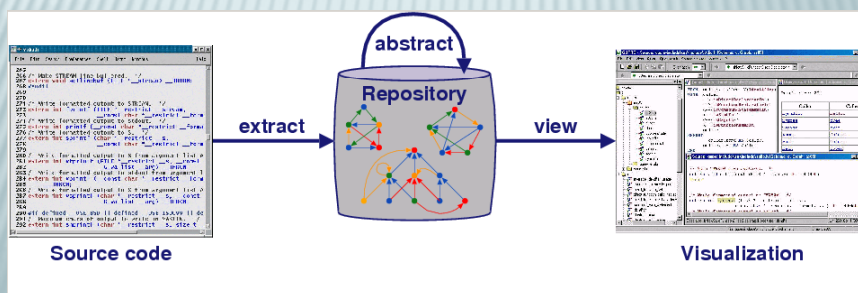
Reverse engineering services work on concrete artifacts:

- | | |
|------------------|------------------------|
| × source code | comments |
| × class diagrams | requirement statements |
| × shell scripts | database schemas |
| × svn scripts | ... |

Integrated into ONE system description

EXTRACT-ABSTRACT-VIEW

Reverse Engineering Tools follow the
extract-abstract-view-metaphor



REPOSITORY SERVICES

Reverse Engineering Tasks are supported by
repository services

- | | |
|-----------------|------------------|
| × analysis | concept analysis |
| × querying | cluster analysis |
| × abstraction | transformation |
| × visualization | ... |

REPOSITORY TECHNOLOGIES

Representation of facts can be done in different ways:

- * relational databases
- * logical fact bases
- * sets and relations
- * XML files
- * graphs

GRAPHS

Graphs are a versatile means for representing facts in tool repositories

Graphs are the „most general representation“

All other representations can be mapped on graphs (see GXL)

ARTIFACTS

12

ARTIFACT LANGUAGES

All software engineering artifacts are written in their respective language:

- × textual, visual, or natural languages
- × formal, semiformal, or informal languages

The artifacts are instances of their language

MODELING

The fact repository contains models of the artifacts, i.e. abstractions suitable to the reverse engineering task to be solved, depending on

- × the language of the artifact
- × the viewpoint addressed by the task
- × the granularity needed

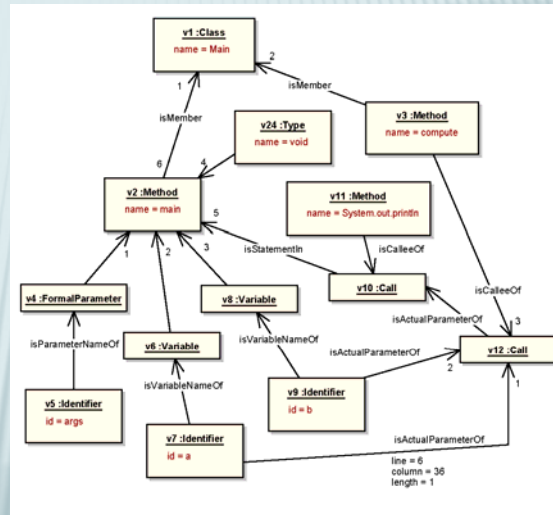
EXAMPLE

```
1 public class Main {
2     public static void main(String[] args) {
3         int a = 26;
4         int b = -5;
5         System.out.println(compute(a, b));
6     }
7     private static int compute(int a, int b) {
8         return Utility.twice(Utility.add(a, b));
9     }
10 }
11
12 public class Utility {
13     public static int add(int x, int y) {
14         return x + y;
15     }
16     public static int twice(int x) {
17         return 2 * x;
18     }
19 }
```

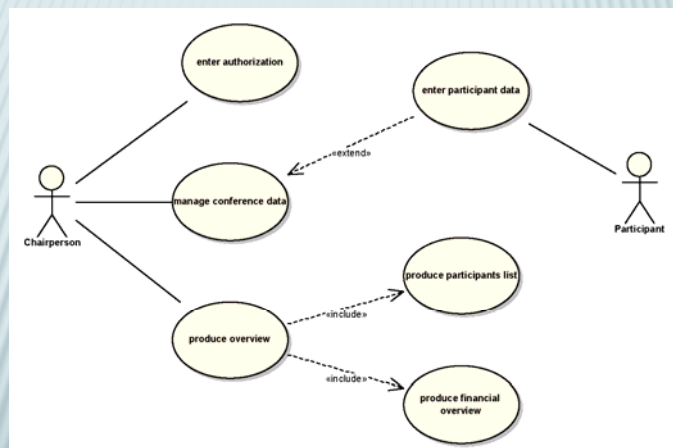
EXAMPLE

- ✗ language: Java
- ✗ viewpoint: syntax
- ✗ granularity: fine

```
public static void main(String[] args) {
    int a = 26;
    int b = -5;
    System.out.println(compute(a, b));
}
```

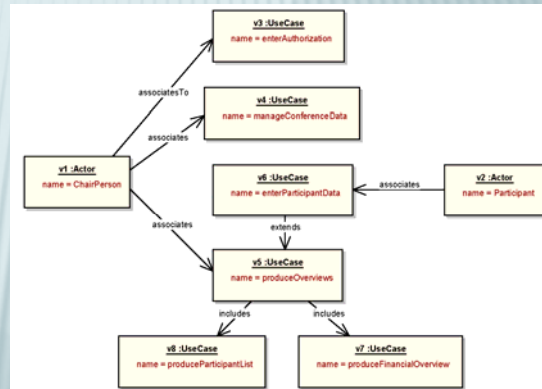


USE CASE DIAGRAM



EXAMPLE

- × language: Use Case Diagram
- × viewpoint: syntax
- × granularity: fine



ASPECTS

Languages

Java, C, C++, ..., RSL, Class Diagrams,
Logs, Traces, ...

Viewpoints (Perspectives)

Syntax, Controlflow, Dataflow, Usage,
Containment, ...

Granularity

fine, medium, coarse

ARTIFACT LANGUAGES

20

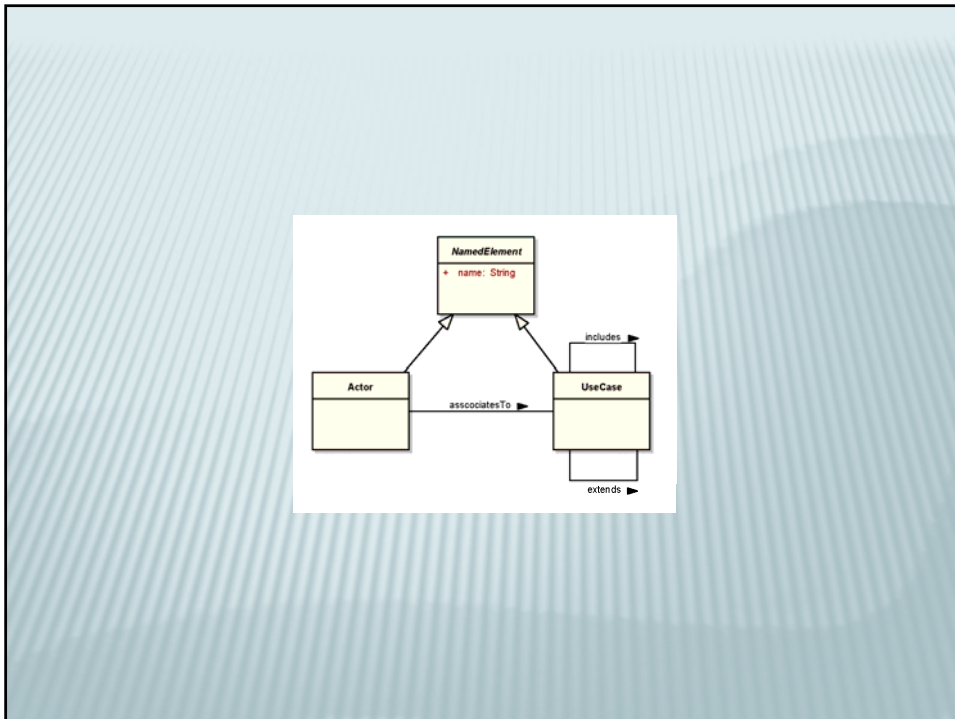
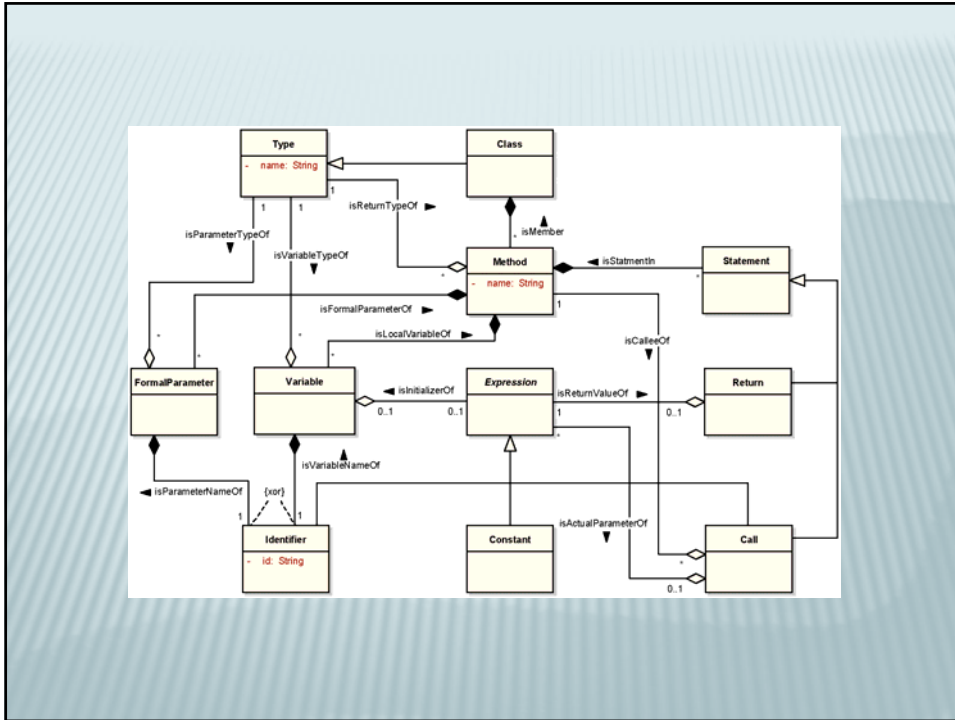
SCHEMAS

A program is an instance of a grammar

A diagram is an instance of a metamodel

A graph is an instance of a schema

Schemas are (slightly restricted) metamodels



SCHEMA-INSTANCE-DICHOTOMY

× Grammar and Program

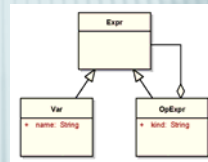
```

Stmt := Var ':=' Expr ';'
Expr := Expr ('+' | '-') Expr
      | '(' Expr ')'
      | Var
  
```

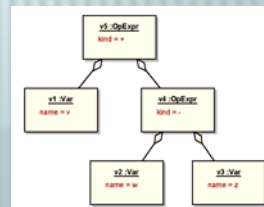
↑ isInstanceOf

```
x := v + (w - z);
```

× Schema and Graph



↑ isInstanceOf



METAMODEL ENGINEERING

It is helpful to develop (and agree upon) reference metamodels for reverse engineering tasks which are widely usable wrt

- × Language
- × Viewpoint
- × Granularity

TGRAPHS AND THEIR FORMALIZATION

25

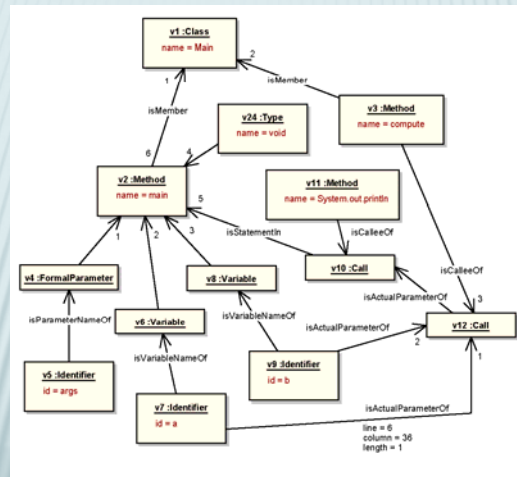
TGRAPHS

TGraphs are

- ×typed: vertices and edges have a type
- ×attributed: (depending on the type)
vertices and edges have valued attributes
- ×ordered: the vertices, the edges, and the incidences are ordered
- ×directed: edges have a start and an end vertex

Multiple type inheritance is allowed

EXAMPLE



PROPERTIES

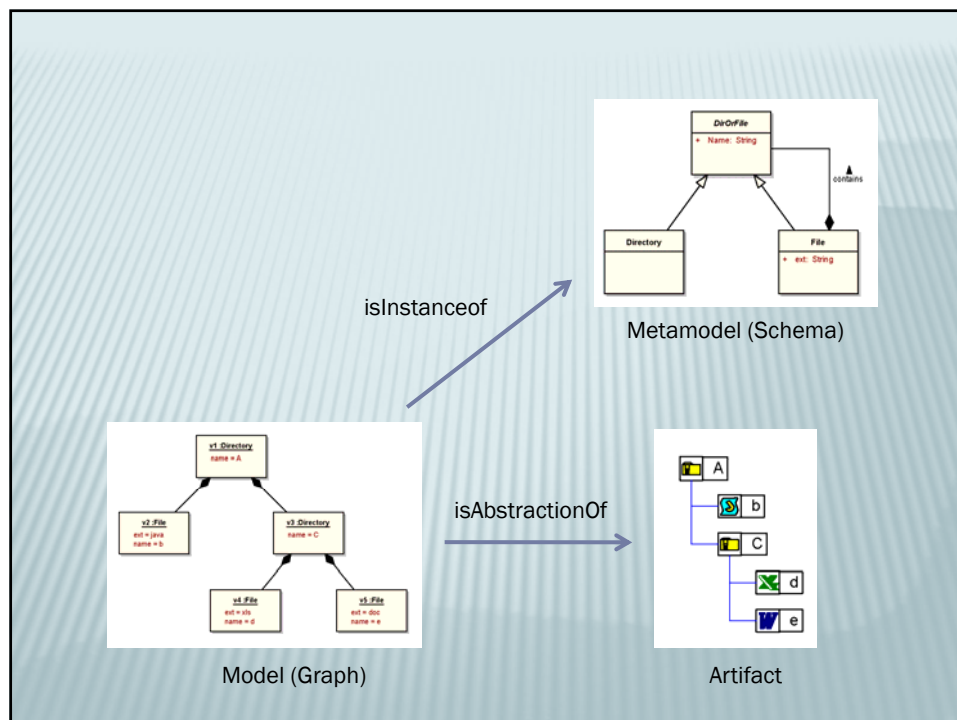
TGraphs have powerful properties:

- × Edges are first class citizens
- × Traversal is supported in both directions
- × Graphs as a whole are subject to algorithms
- × ...
- × Entities are modeled by vertices
- × Occurrences are modeled by edges
- × Sequences are expressed by edge order

TGRAPHS AND SCHEMAS

A schema is a metamodel whose instances are graphs

Schemas describe TGraph classes
TGraphs are models of artifacts

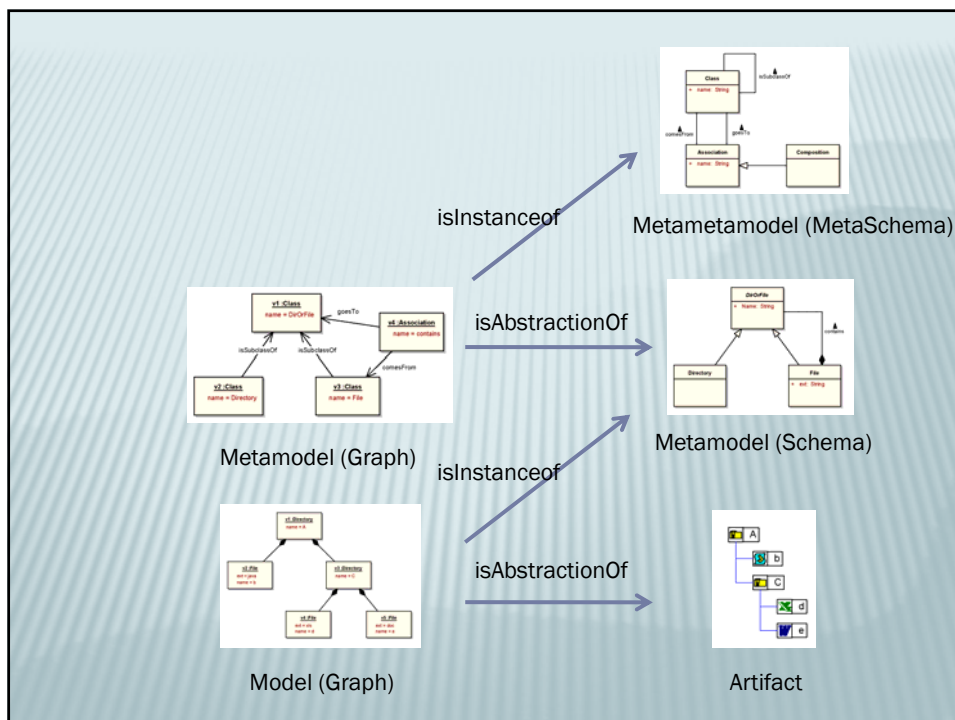


METAMODELING

The `isInstanceOf` relation is precisely defined

The `isAbstractionOf` relation is formalized by

- × Extractors (Parsers, Scanners)
 - that generate the graph from the artifact
- × Editors
 - that keep the graph as the internal model of the artifact
- × Renderers
 - that generate the artifact from the graph



METAMODELING HIERARCHY

This metamodeling hierarchy has several properties:

- × It is conceptually identical to MOF (M1-M3)
- × Everything is abstracted to the same formalism

FORMALIZATION

The TGraph-based metamodeling approach can be formalized, thus leading to a rigorous technology

Generic TGraph tools can be applied without additional effort (as long as a schema is given)

MATHEMATICAL DEFINITION

A *TGraph* $G = (Vseq, Eseq, \Lambda seq, type, value)$ consists of

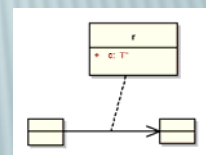
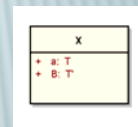
- an injective sequence $Vseq$ of a set $V \subseteq VERTEX$ of vertices
- an injective sequence $Eseq$ of a set $E \subseteq EDGE$ of edges
- a total incidence function $\Lambda seq : V \rightarrow seq(E \times \{in, out\})$
such that for each $e \in E$
there is exactly one vertex v with $(e, out) \in \Lambda seq(v)$
and exactly one vertex w with $(e, in) \in \Lambda seq(w)$
- a total type function $type : V \cup E \rightarrow TYPE$
- a total value function $value : V \cup E \rightarrow (ATTR \leftrightarrow VALUE)$

GRUML DIAGRAMS

TGraphs can be specified by a subset of UML-class diagrams (grUML):

- × Classes: Vertex Types
- × Associations: Edge Types
- × Attributes: Element Attributes
- × Specialization: Type Inheritance
- × Multiplicities: Degree Restrictions

(slightly more than EMOF)

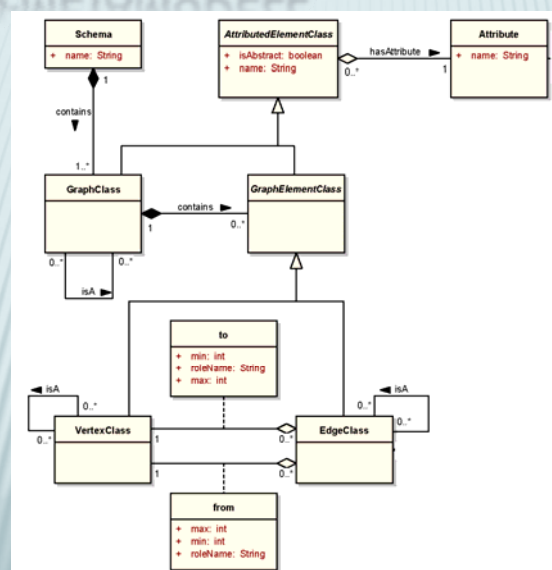


COMPATIBILITY

A TGraph compatible to a schema

- × if the element types and the attribute assignments in the graph respect the schema,
- × if the incidences of the edges respect the schema, and
- × if the vertex degrees respect the multiplicities (under inheritance)

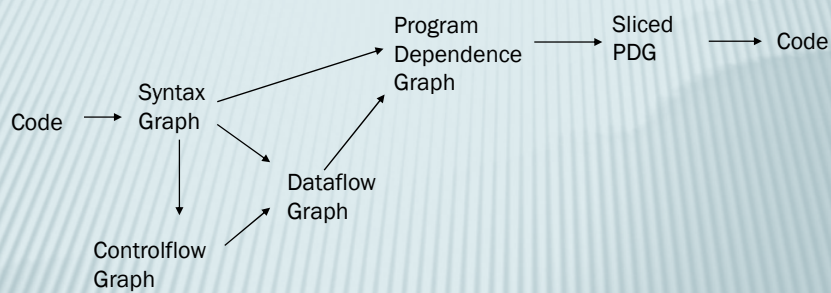
GRALAB-METAMODELL



GENERIC ALGORITHMS

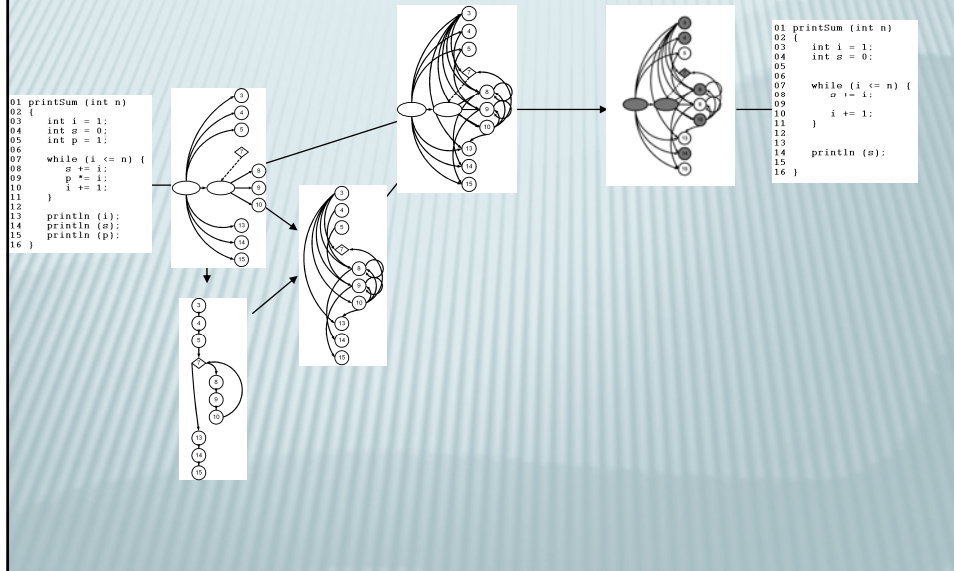
Given a set of graph classes, transformation algorithms can be defined and implemented generally on a reference schema

SLICING



Slicing can be refined into several subtasks transforming one graph class into another

SLICING



TGRAPHS AND THEIR IMPLEMENTATION

GRALAB

There is an elaborated API (JGraLab)
which supplies all operations for

- × creating,
- × manipulating, and
- × traversing

TGraph schemas and TGraphs (wrt schema)

<http://userpages.uni-koblenz.de/~ist/JGraLab>

JGRALAB

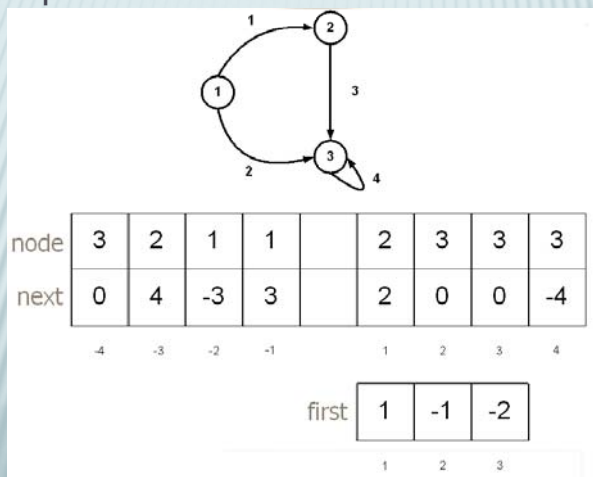
JGraLab is

- × based on symmetric incidence lists,
- × supplies all operations derivable from its
metametamodel, and
- × allows pseudocode-like programming of graph
algorithms

Graphs may contain several millions of elements

SYMMETRIC INCIDENCE LISTS

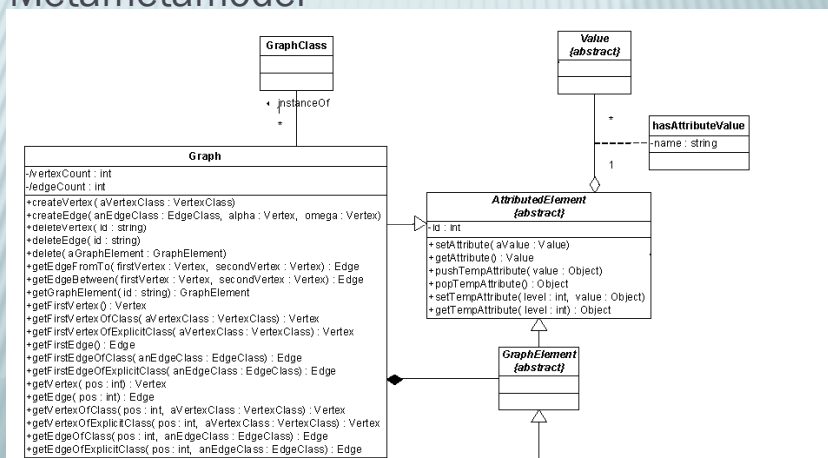
The implementation is efficient wrt traversal



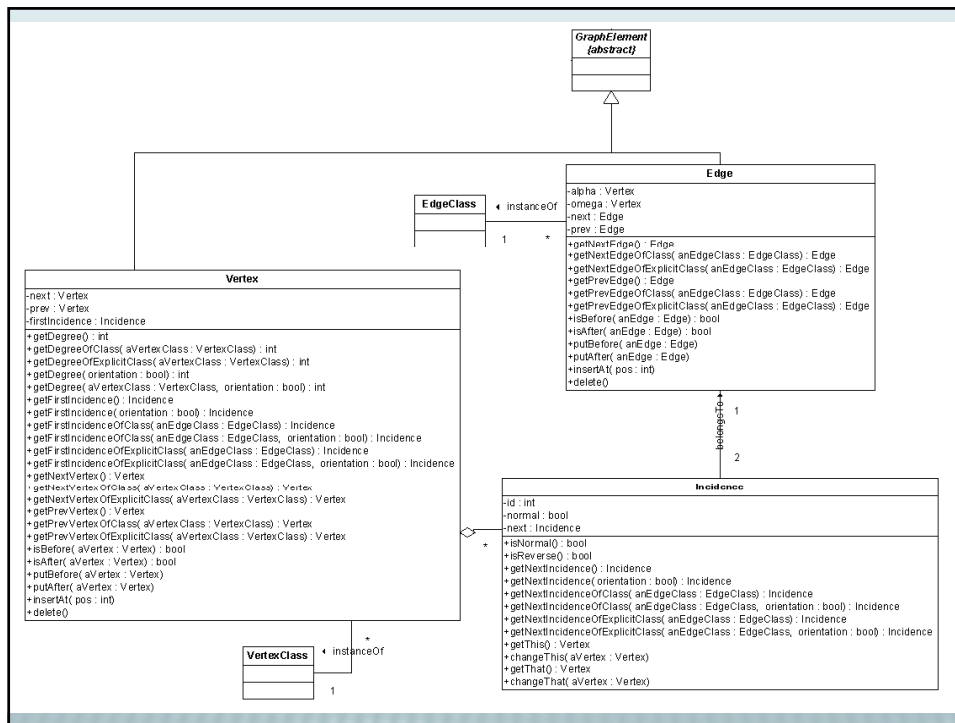
JE37

API

The JGraLab API is derived from the JGraLab Metamodel



JE37 Ich weiß nicht, ob das gut, dass ich die Implementation schon hier anspreche
Jürgen Ebert; 28.03.2008



SAMPLE GRALAB CODE

The GraLab-API that allows creation, manipulation and traversal of schemas and graphs according to the metamodel

```

GraphClass scClass = schema.getGraphClass("SoftwareCase");
Graph sc = new Graph(scClass);
VertexClass reqClass = sc.getVertexClass("Requirement");
Vertex req = sc.createVertex(reqClass);
req.setAttribute("name", "Create.bills");
  
```

ADVANCED FEATURES (ACCESS LAYER)

JGraLab contains a tool that also generates Java classes from grUML diagrams

```
SoftwareCase sc = SoftwareCase.create();  
Requirement req = softwareCase.createRequirement();  
req.setName("Create.Bills");
```

Thus graph-algorithms and object-oriented programming are combined

QUERYING

Many reverse engineering tasks rely heavily on information extraction, i.e. querying the fact repository:

- × Software metrics
- × Cross referencing
- × Program Slicing
- × Impact Analysis
- × Traceability determination

JGraLab comes with a schema sensitive graph query language: GReQL

GREQL EXAMPLE

```

1  from caller, callee: V{Method}
2  with caller {
3      <-- {isStatementIn}
4      [ <-- {isReturnValueOf} ]
5      <-- {isActualParameterOf} *
6      <-- {isCalleeOf}
7  }+ callee
8  report
9  caller.name as "Caller",
10 callee.name as "Callee"
11 end

```

Caller	Callee
main	System.out.println
main	compute
main	twice
main	add
compute	twice
compute	add

GRAPH-BASED TOOLS

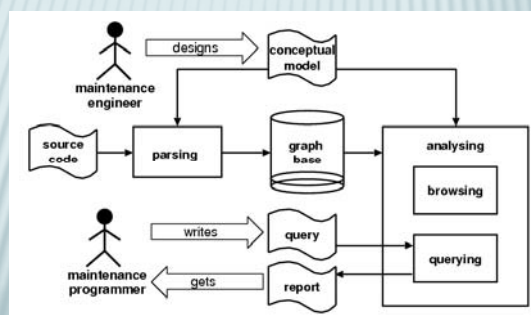
APPLICATIONS

The TGraph approach has been applied in several tools

- × Kogge (MetaCase)
- × Gupro (Program Understanding)
- × ReDSeeDS (Requirements-based ReUse)
- × Planning of Transports

GUPRO

GUPRO = Generic Understanding of Programs



The screenshot shows the QGupro IDE interface. The main window is titled "QGupro" and has a menu bar with "File", "Edit", "Project", "Query", "Window", "Help", and "Settings". Below the menu bar is a toolbar with several icons. The interface is divided into three main panes:

- Workspace:** Shows a tree view of the project structure. The "cosmos-client" folder is selected, containing files like "client.c", "comm.c", "ident.c", "debug.c", and "server.c".
- Query Editor:** Contains a query written in a GUPRO-like language. The query starts with "FROM caller, callee: V(Identifier)" and "WITH caller". It includes several pattern-matching rules like "{isDirectDeclaratorIn}", "{isFunctionDeclaratorIn}", "{isCompoundStatementIn}", "{isStmntIn}", "{isDeclarationIn}", "{isInitDeclaratorIn}", "{isInitializationIn}", "{isExprIn}", and "{isFunctionNameIn}". The query ends with "REPORT caller.name AS Caller, callee.name AS Callee" and "END".
- Query Result:** Displays the results of the query. It shows "Graph id = E08340cddb0cabe86370f7", "Computation time = 30 ms", and "Result size = 409". A table with two columns, "Caller" and "Callee", lists the results: "message", "error_nsq", and "info_msg" under "Caller", and "_builtin_va_start" under "Callee". Below the table, a snippet of C code from "debug.c" is shown, highlighting the "void message" function signature.

GUPRO

GUPRO works on heterogenous software on different levels on abstraction:

- × C
- × C++ via Columbus
- × Java
- × UML via XMI

JE34

(including preprocessor support for C/C++)

JE34 vtl. mehr zu Gupro
Jürgen Ebert; 02.04.2008

CONCLUSION

55

„MESSAGE“ OF THIS TALK

Use a rigorous approach (seamlessly formal and efficiently implemented) to support:

- × engineering of metamodels
- × representation and handling of facts
- × implementation of reverse engineering services (by coding and/or querying)

STEPS

- × Identify the reverse engineering task
- × Build a metamodel M of the relevant information (depending on language, viewpoint, granularity)
- × Write (or generate) extractors from artifact languages to M
- × Design and implement the solution using algorithms and or queries

METAMODEL ENGINEERING

Metamodelling becomes the central activity.
A metamodeling style guide is needed.

- × Which concepts are entities/attributes?
- × When to use inheritance?
- × When to use associations/aggregations?
- × Which cardinality constraints are enforced?
- × ...
- × Avoid cloning and duplication!
- × Use edge order!

TGRAPH APPROACH

The TGraph Approach consists of

- + A powerful graph type: TGraph
 - + A UML sublanguage with TGraph semantics: gruML
 - + A Java API with Java Access Layer: GraLab
 - + A query language: GReQL
 - + Several utility tools:
 - scripting language, GXL export, ...
- ... and is being developed further

JE27



JE27

URL eintragen

Jürgen Ebert; 31.03.2008