Universität Koblenz-Landau
Institut für Softwaretechnik
GUPRO

Components of Interchange Formats

(Metaschemas and Typed Graphs)

presented by

Andreas Winter

Included Papers

M. van den Brand, H. de Jong, P. Olivier:
A Common Exchange Format for Reengineering Tools based on ATerms

M. W. Godfrey:
Practical Data Exchange for Reverse Reengineering Frameworks: Some Requirements, Some Experiences, Some Headaches

J. Ebert, B. Kullbach, A. Winter:
GraX – Graph Exchange Format

Contents

Structure

Problem

- various structures are used in current tools
  - Abstract Syntax Trees: *ATerms*, ...
  - Graphs: Datrix-TA, *TA, TGraphs*, PROGRES, RSF, ...
  - Relations: relational Databases, RPA, ...

Requirements

- exchange of data requires a *common superset*, enclosing these structures

Proposed Solutions

- terms
- directed, typed, attributed graphs

Format

Problem

- Programming Languages
  - single language systems (Cobol, C, C++, ...)
  - multi language systems
- Level of Abstraction
  - AST level
  - architectural level

Requirements

- *adaptable* and *extensible* interchange format

Proposed Solutions

- exchange of *schema data* and
- exchange of *instance data*

Data Access

Problem

- large amount of data
- import from/export to various tools

Requirements

- *efficient* and *standardized* access to interchanged data

Proposed Solutions

- application programming interface
- set of standardized, predefined tools

Transformation

Problem

- different tasks require different representations
- seamless data exchange between different representations

Requirements

- *efficient transformation* between different representations

Proposed Solutions

- set of *(reference) schemata* and definition of *transformation* between them

Metaschema

Problem

- mechanism to describe schema information for different applications
- foundation to define schema transformations

Requirements

- common mechanism for *describing schemata*

Proposed Solutions

- *metaschema* (UML class diagram)

Efficient Annotated Terms (ATerms)

M. G. J. van den Brand,
H. A. de Jong,
P. Klint,
P. Olivier, et al.

Approach

- exchange format for *tree-like data structures* like parse trees, abstract syntax trees, generated code, formatted source text
- comprehensive *procedural interface* for manipulation in C and Java

ATerms - Structure

- Software systems are represented by *annotated Terms*

ATerm Examples

```
constants   : abc
numerals    : 42
literals    : "asdf"
lists       : [], [1, "abc" 2],
              [1, 2, [3, 4]]
functions   : f("a"), g(1,[])
annotations : f("a") {"remark"}
```

ATerm Grammar (SDF)

```
ATerm                 -> ATerms
ATerm "," Aterms      -> ATerms
"[" "]"               -> ATermList
"[" ATerms "]"        -> ATermList
AInt                  -> AFun
Literal               -> AFun
ATermList             -> ATerm
AFun                  -> ATerm
AFun "(" ATerms ")"   -> ATerm
"{" ATerms "}"        -> Annotation
ATermList Annotation  -> ATerm
AFun Annotation       -> ATerm
AFun "(" ATerms ")"
     Annotation       -> ATerm
```


ATerms - Structure

- Software systems are represented by *annotated Terms*

ATerm Examples

```
constants   : abc
numerals    : 42
literals    : "asdf"
lists       : [], [1, "abc" 2],
              [1, 2, [3, 4]]
functions   : f("a"), g(1,[])
annotations : f("a") {"remark"}
```

ATerm Grammar (EBNF)

```
ATerm   ::= AFun
              ["(" ATerms ")"]
              [Annotation]
          | "[" [ATerms] "]"
              [Annotation]
ATerms  ::= ATerm ("," ATerm)*
AFun    ::= AInt | Literal
Annotation ::= "{" ATerms "}"
```


ATerms - Format

Instance Level

- focuses (only) on tree like structures
- Notation
  - terms over *variables*, *constants* and a given set of *function symbols*

Schema Level

- schemata implicitly represented
  - defined by an appropriate set of functions e.g. for representing abstract syntax trees (AsFix)
  - further schema information can be represented by annotations with ATerms

ATerms – Example (1)

AsFix (ASF+SDF Fixed Format)

- parse tree format for ASF/SDF
  - SDF (Syntax Definition Formalism)
  - ASF (Algebraic Specification Formalism)
- each term represents a parse tree including
  - syntax rules
  - original layout
  - comments
- AsFix representation is self-contained
  - term includes all grammar information needed for interpretation

ATerms – Example (2)

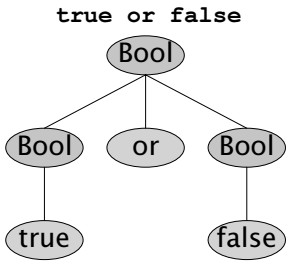
AsFiX Functions

- `prod(T)` : production rule
- `appl(`T_1`,`T_2`)` : applying production rule T_1 to arguments T_2
- `l(T)` : literal `T`
- `sort(T)` : sort `T`
- `w(T)` : white space `T`
- `attr(T)` : single attribute
- `attrs(T)` : lost of attributes
- `no-attrs` : empty list of attributes

ATerms – Example (3)

Grammar (SDF)

```
true          -> Bool
false         -> Bool
Bool or Bool -> Bool {left}
```



AsFix Term

```
appl(
  prod([sort("Bool"),l("or"),sort("Bool")],sort("Bool"),
       attrs([attr("left")])),
  [appl(prod([l("true")],sort("Bool"),no-attrs)
        [l("true")]),
   w(" "), l("or"), w(" "),
  [appl(prod([l("false")],sort("Bool"),no-attrs)
        [l("false")])])
```


ATerms – Data Access

ATerms API

- Making and Matching ATerms (4)
  - make new ATerms
  - match ATerms
  - compare ATerms
- Reading and Writing ATerms (6)
  - write/read ATerms to/from text files
  - write/read ATerms to/from *binary files*
  - write/read ATerms to/from strings
- Annotating ATerms (3)
  - set/get/remove annotations

TA Exchange Format (TAXForm)

I. T. Bowman,
M. W. Godfrey,
R. C. Holt, et al.

University of Waterloo

Approach

- exchange format for *graph-like structures* based on TA (Tuple Attribute Language)
- *efficient transformation* between different graph-like structures

TAXForm - Structure

- Software systems are represented by *directed, attributed, typed graphs*

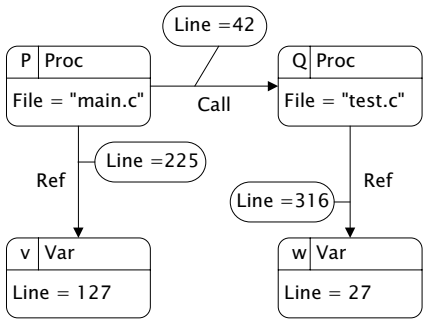
main.c

```
function P {
42    q();
127   T1 v;
225   x = v++;
}
```

test.c

```
function Q {
27    T2 w;
316   z = w;
}
```

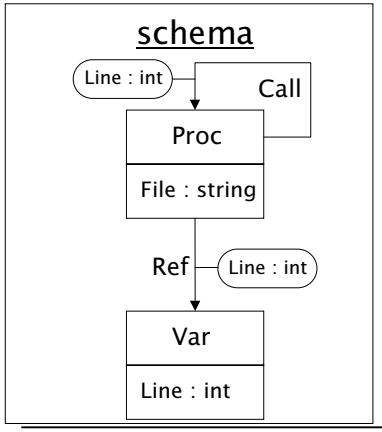
typed graph



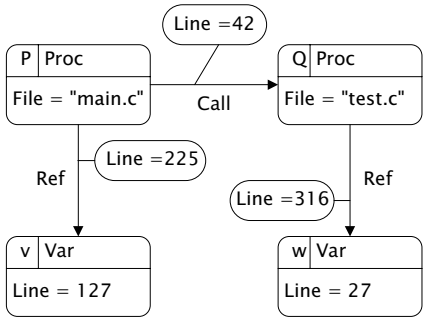
TAXForm - Structure

- Software systems are represented by *directed, attributed, typed graphs*

schema

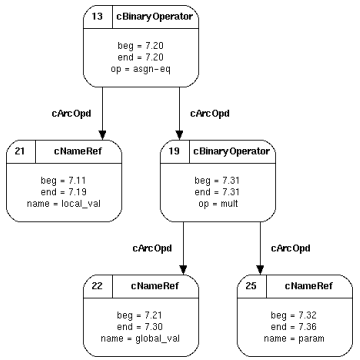


typed graph



TAXForm - Level of Abstraction

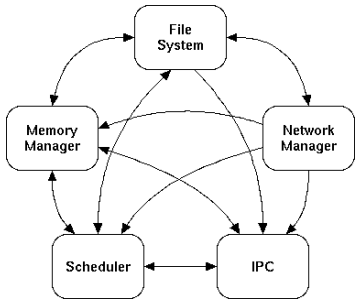
AST Level



Datrix AST for C++ statement

```
local_var = global_var * param
```

Architectural Level



top level subsystems of LINUX kernel

TAXForm – Format

Instance Level

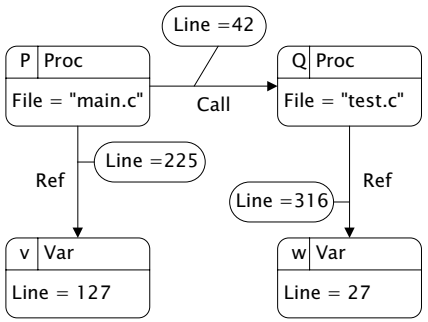
- focuses on arbitrary graph-like structures
  - on AST level
  - on architectural level
- notation:
  - *fact tuples* and *fact attributes*

Schema Level

- schemata explicitly represented
- notation:
  - *schema tuples* and *schema attributes*

TAXForm – Example (1)

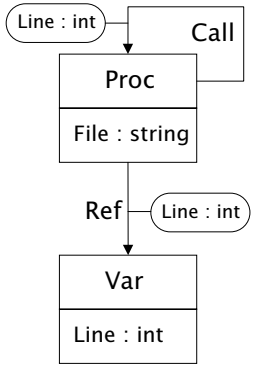
instance level




```
FACT TUPLE
$INSTANCE P Proc
$INSTANCE Q Proc
$INSTANCE v Var
$INSTANCE w Var
Call P Q
Ref P v
Ref Q w
FACT ATTRIBUTE
P {File = "main.c"}
Q {File = "test.c"}
v {Line = 127}
w {Line = 27}
(Call P Q){Line = 42}
(Ref P v){Line = 225}
(Ref q w){Line = 316}
```


TAXForm – Example (2)

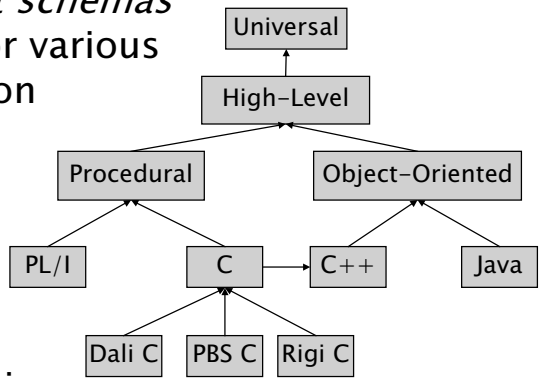
schema level




```
SCHEMA TUPLE
Call Proc Proc
Ref Proc Var
SCHEME ATTRIBUTE
Proc {File}
Var {Line}
(Call) {Line}
(Ref) {Line}
```


TAXForm – Transformation (1)

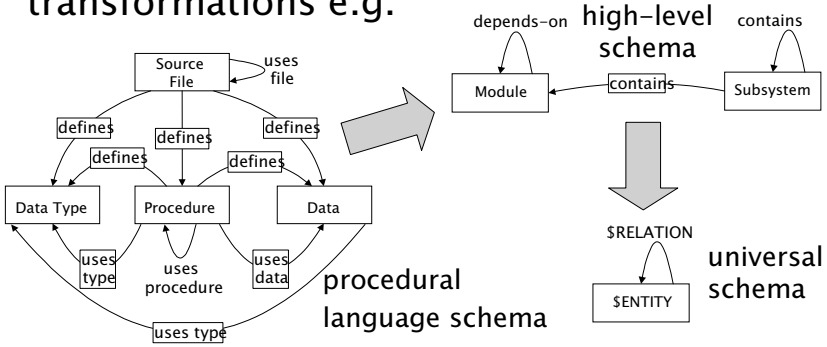
- lots of *different schemas* were defined for various purposes (e.g. on architectural level)



- TAXForm Utopia:
  - *Transformation between schemata*

TAXForm – Transformation (2)

- TAXForm offers a *fixed set* of schema transformations e.g.



- Work to do:
  - *very general schema transformation mechanism*

Graph Exchange Language (GXL)

J. Ebert,
R. C. Holt,
B. Kullbach,
A. Schürr,
A. Winter

joint graph exchange format
GraX, TA, PROGRES, RSF, RPA

Approach

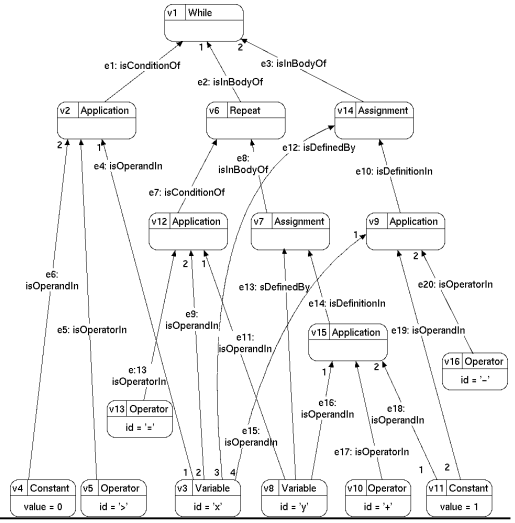
- XML exchange format for *graph-like structures* based on *TA* and *TGraphs*
- *metamodel for defining schemata*

GXL – Structure

Software systems are represented by

- *ordered,*
- *directed,*
- *attributed,*
- *typed graphs*

AST Level

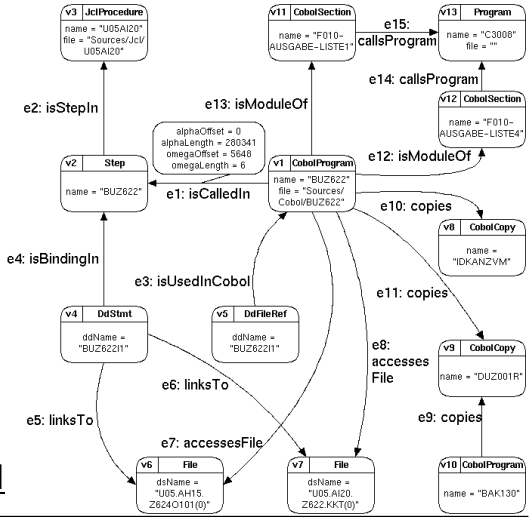


GXL – Structure

Software systems are represented by

- *ordered,*
- *directed,*
- *attributed,*
- *typed graphs*

architectural Level



GXL – Foundation

Instance Level

- focuses on arbitrary graph-like structures
  - on AST level
  - on architectural level
- notation:
  - *XML document*

Schema Level

- schemata explicitly represented
- notation:
  - *XML document*

GXL – Document Type Definition

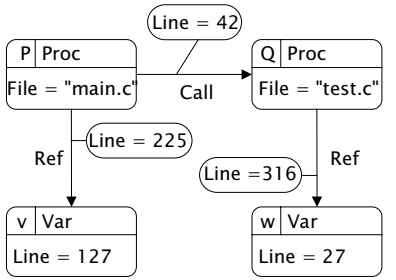
- Exchange of *instance and schema* data by attributed graphs
- XML based markup language for representing *attributed graphs*

GXL DTD (simplified)

```
<!ELEMENT gxl (node | edge)* >
<!ATTLIST gxl
     schema      CDATA    #REQUIRED
     identifiededges (true|false)
                          #REQUIRED >
<!ELEMENT node (attr)* >
<!ATTLIST node
     id          ID       #REQUIRED
     type        CDATA    #IMPLIED
     edgeorder   IDREFS   #IMPLIED >
<!ELEMENT edge (attr)* >
<!ATTLIST edge
     id          ID       #IMPLIED
     type        CDATA    #IMPLIED
     begin       IDREF    #REQUIRED
     end         IDREF    #REQUIRED >
<!ELEMENT attr EMPTY >
<!ATTLIST attr
     name        CDATA    #REQUIRED
     value       CDATA    #IMPLIED >
```


GXL Example (1)

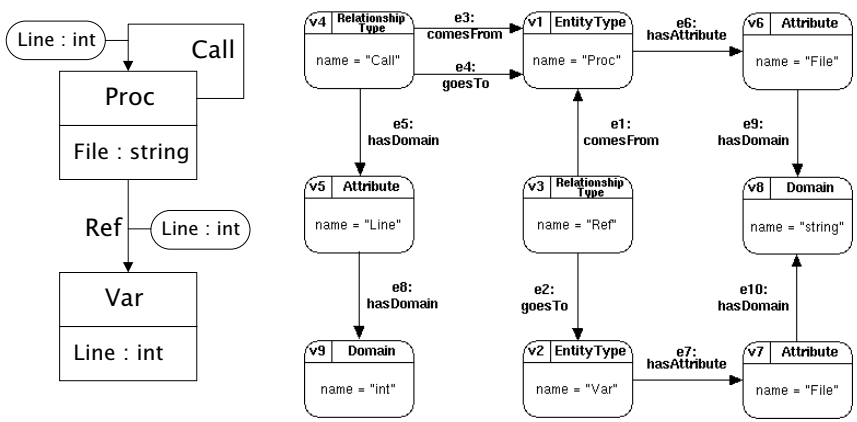
instance level




```
<gxl>
<node id="P" type="Proc">
     <attr name="File" value="main.c" />
</node>
<node id="Q" type="Proc">
     <attr name="File" value="test.c" />
</node>
<node id="v" type="Var">
     <attr name="Line" value="127" />
</node>
<node id="w" type="Var">
     <attr name="Line" value="27" />
</node>
<edge begin="P" end="Q" type="Call">
     <attr name="Line" value="42" />
</edge>
<edge begin="P" end="v" type="Ref">
     <attr name="Line" value="225" />
</edge>
<edge begin="Q" end="w" type="Ref">
     <attr name="Line" value="316" />
</edge>
</gxl>
```

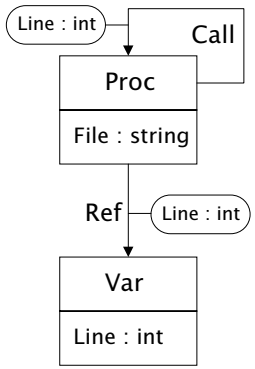

GXL – Example (2)

schema level



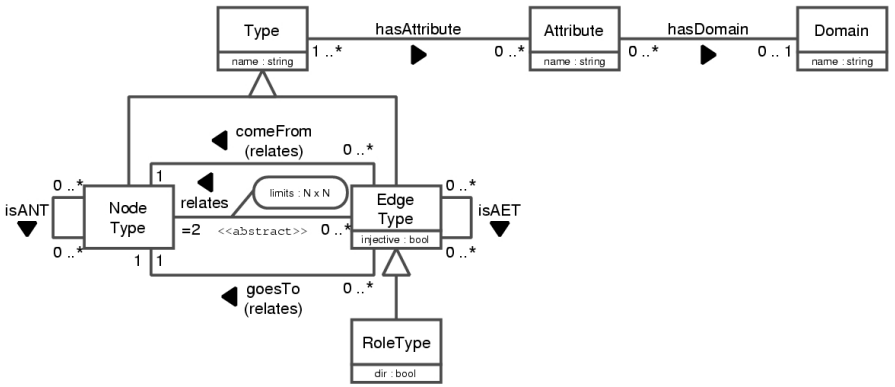
GXL – Example (2)

schema level

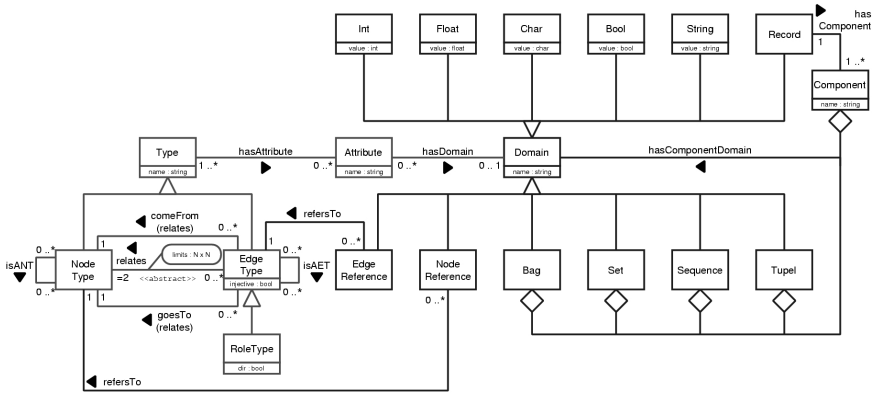



```
<?xml version="1.0" ?>
<!DOCTYPE grax SYSTEM "gxl042.dtd">
<gxl schema = "meta.1.0.scx">
<node id = "v1" type = "EntityType">
     <attr name = "name" value = "Proc"/>
</node>
<node id = "v2" type = "EntityType">
     <attr name = "name" value = "Var"/>
...
<node id = "v3" type = "RelationshipType">
     <attr name = "name" value = "s"/>
</node>
...
<edge id = "e1" type = "comesFrom">
    begin = "v3" end = "v1">
</edge>
<edge id = "e2" type = "goesTo"
    begin = "v3" end = "v2">
</edge>
...
</gxl>
```


GXL – Metaschema (1)



GXL – Metaschema (2)



with attribute structure

GXL – Data Access

GXL Tools

- parsers
  - parse GXL instance and schema data
- filters
  - transfer GXL documents into TA, TGraphs, PROGRES, RSF, RPA, ...
  - transfer TA, TGraphs, PROGRES, RSF, RPA, ... into GXL documents
- checkers
  - check if instance matches ist schema
- helpers
  - recover schema information

Comparison

	ATerms	TAXFORM	GXL
Format	Schema and Instance		
Structure	Terms (AST)	directed, attributed, typed Graphs	
Data Access	API	suitable Tools	
Metaschema			UML Class Diagram
Transformation		Transformation Tools	

Conclusion

Agreement
- exchange of *schema and instance data*
- by *directed, attributed, typed graphs*
- with a *XML-language*
- based on a common *Metamodel*

Work to do
- definition of a suitable *API*
- definition of a *set of tools*
- definition of powerful *schema transformation*

Extensions
- consideration of *hierarchical graphs*, hypergraphs, and *graph transformation rules*

GXL

- Report:
  - R. C. Holt, A. Schürr, A. Winter: *GXL: Toward a Standard Exchange Format*, Fachberichte Informatik, Uni Koblenz
    http://www.gupro.de/techreports/RR-1-2000
- URL: (coming soon)
  - **http://www.gupro.de/GXL**

you are all invited to participate in the further development of GXL