

Looking for a Graph eXchange Language

Presentation at the APPLIGRAPH-Subgroup Meeting
on Exchange Formats for Graph Transformation
September 5-6, 2000, Paderborn University, Germany

Ric Holt
Software Architecture Group
University of Waterloo
Canada

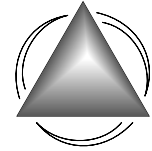


Andy Schürr
Institute for Software Technology
University of the Federal Armed Forces
Munich, Germany



Andreas Winter
Institute for Software Technology
University of Koblenz/Landau
Germany





Contents of Presentation:

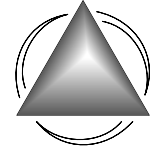
1. Short introduction to basic GXL features (graphs and graph schemata)
2. Discussion of pros and cons of all suggested graph exchange formats
3. Short introduction to advanced GXL features (meta schemata, ...)
4. Discussion of various (meta) schema representation approaches (UML, ...)
5. Summary

GXL Home Page (with pointer to latest GXL DTD):

<http://www.gupro.de/GXL/>

Technical Report (with 25 applications of GXL predecessors):

<http://www.uni-koblenz.de/fb4/publikationen/gelbereihe/RR-1-2000/RR-1-2000.pdf>



1. Introduction to GXL

GXL is based on (reengineering) exchange formats

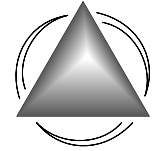
- ⇒ TA (Tuple Attribute Language) from University of Waterloo, CA
- ⇒ GraX (Graph Exchange Format) from University of Koblenz/Landau, D

GXL replaces/incorporates exchange formats of

- ⇒ PROGRES graphs from RWTH Aachen/University Bw Munich, D
- ⇒ RPA (Relation Partition Algebra) from Philips Research Eindhoven, NL
- ⇒ RSF (Rigi Standard Format) from University of Victoria, CA.

GXL has no intention to support

- ⇒ exchange of graph transformations
 - Berlin proposal, Grrr proposal, ...
- ⇒ exchange of graph layout data
 - standardization efforts of graph drawing community

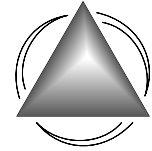


GXL design goals

- ⇒ based on standards XML and UML as a graphical notation
- ⇒ MOF compliant (has to be checked in detail)
- ⇒ simple, compact representation of simple graphs
- ⇒ suitable for broad spectrum of graph models (including hypergraphs)
- ⇒ uniform representation of graphs and graph schemata
- ⇒ limited support for exchange of meta models
- ⇒ support for representation of complex attribute values
- ⇒ extensible wrt. to used meta models and attribute types

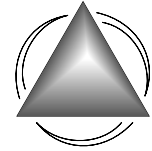
We need your assistance and

- ⇒ your opinion concerning a number of open problems (representation of graph schema, inheritance concepts, representation of attribute values, ...)
- ⇒ your input to incorporate still missing features and fix design flaws
- ⇒ and your help to establish GXL as the graph representation kernel of forthcoming graph transformation and graph layout exchange formats



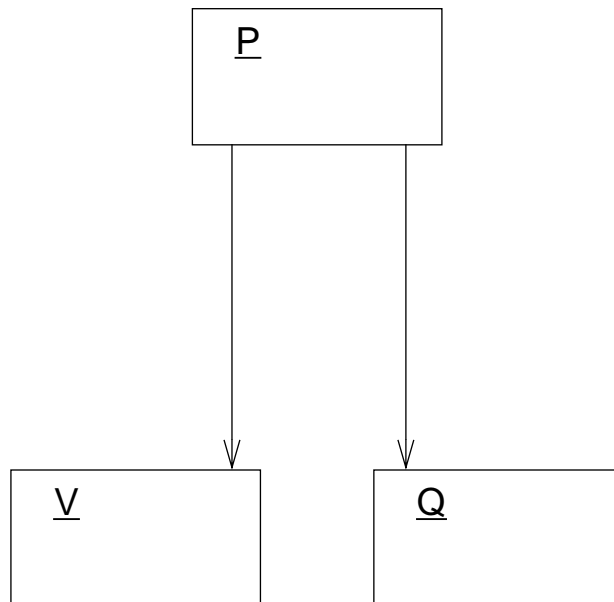
GXL is already supported by

- ⇒ Bell Canada (Datrix Group)
- ⇒ IBM Centre for Advanced Studies, Canada
- ⇒ Mahindra British Telecom, India
- ⇒ Nokia Research Center, Finland (Software Technology Laboratory)
- ⇒ Philips Research, The Netherlands (Software Architecture Group)
- ⇒ RWTH Aachen, Germany (Department of Computer Science III)
- ⇒ University of Berne, Switzerland (Software Composition Group)
- ⇒ University Bw München, Germany (Institute for Software Technology)
- ⇒ University of Koblenz, Germany (GUPRO Group)
- ⇒ University of Oregon, USA (Department of Computer Science)
- ⇒ University of Paderborn, Germany (AG Softwaretechnik)
- ⇒ University of Stuttgart, Germany (BAUHAUS Group)
- ⇒ University of Victoria, Canada (RIGI Group)
- ⇒ University of Waterloo, Canada (Software Architecture Group)

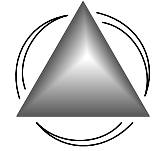


A Program Graph and its GXL Representation:

UML Notation for Graph:

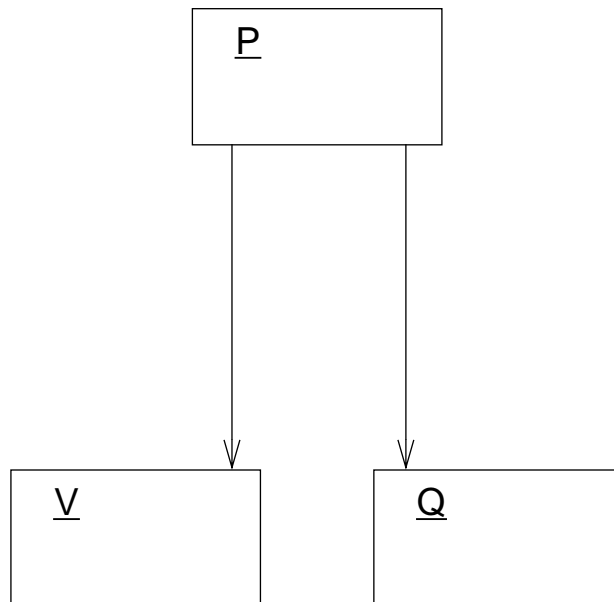


simple directed graph



A Program Graph and its GXL Representation:

UML Notation for Graph:



simple directed graph

GXL Exchange Format:

```
<graph>
  <node id="P" />

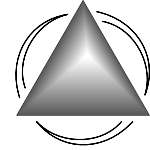
  <node id="V" />

  <node id="Q" />

  <edge from="P" to="V" />

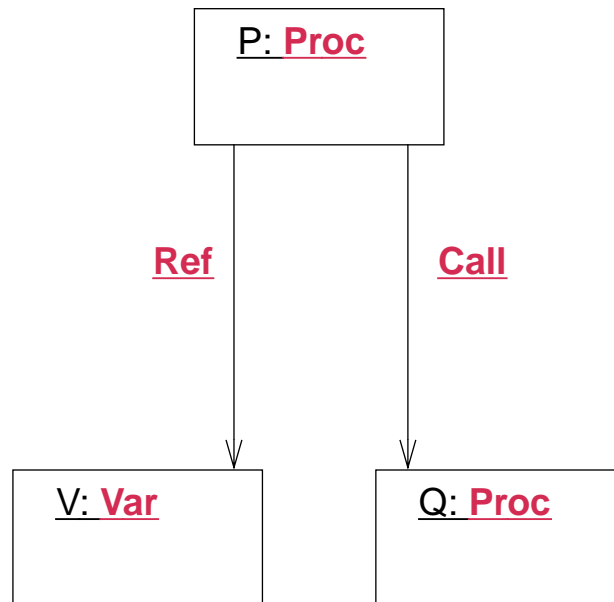
  <edge from="P" to="Q" />

</graph>
```



A Program Graph and its GXL Representation:

UML Notation for Graph:



labeled directed graph

GXL Exchange Format:

```
<graph>
  <node id="P" type="Proc" />

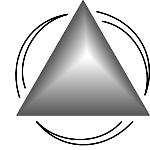
  <node id="V" type="Var" />

  <node id="Q" type="Proc" />

  <edge from="P" to="V" type="Ref" />

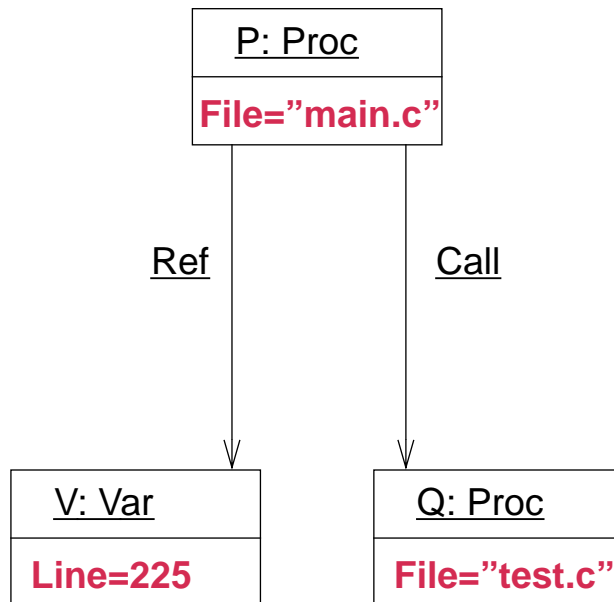
  <edge from="P" to="Q" type="Call" />

</graph>
```



A Program Graph and its GXL Representation:

UML Notation for Graph:



node-attributed graph

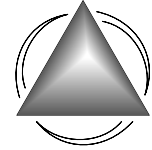
GXL Exchange Format:

```

<graph>
  <node id="P" type="Proc">
    <attr name="File"><str>main.c</str></attr>
  </node>
  <node id="V" type="Var">
    <attr name="Line"><int>225</int></attr>
  </node>
  <node id="Q" type="Proc">
    <attr name="File"><str>test.c</str></attr>
  </node>
  <edge id="r" from="P" to="V" type="Ref" />

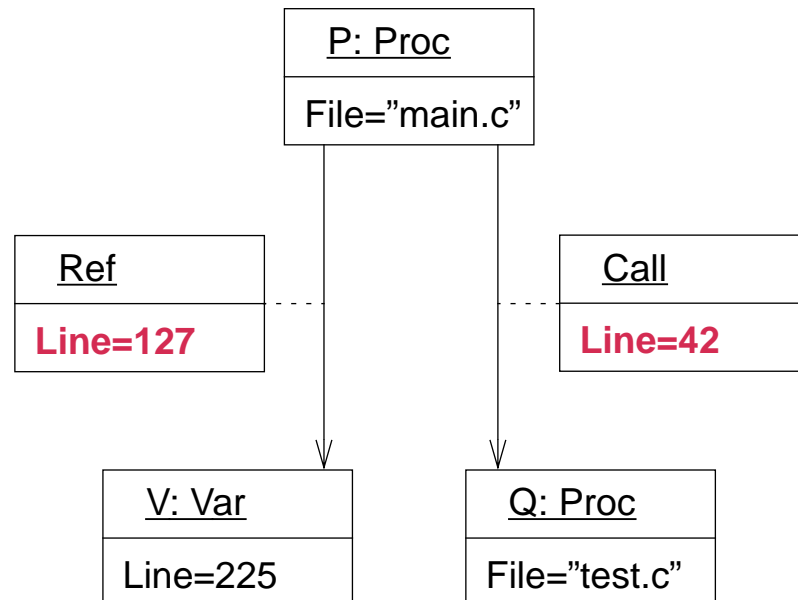
  <edge id="c" from="P" to="Q" type="Call" />

</graph>
  
```



A Program Graph and its GXL Representation:

UML Notation for Graph:

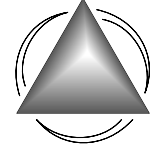


edge-attributed graph

GXL Exchange Format:

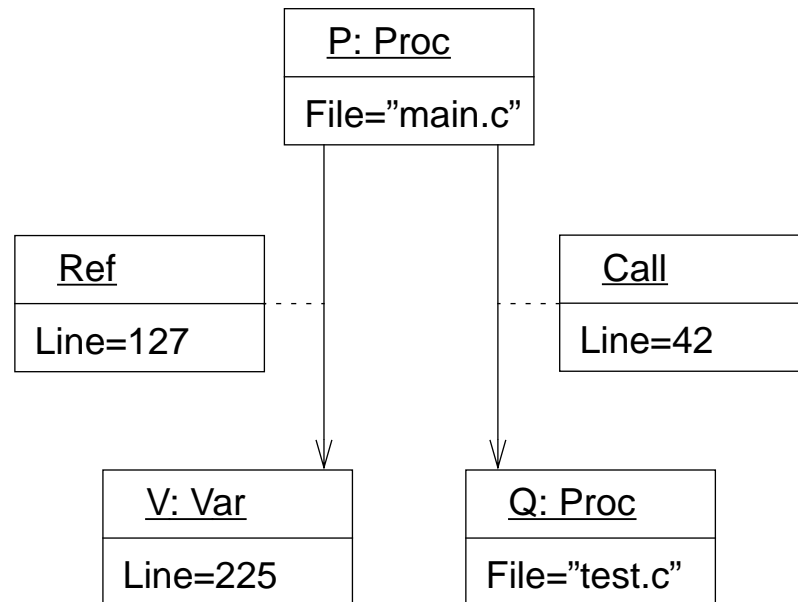
```

<graph>
  <node id="P" type="Proc">
    <attr name="File"><str>main.c</str></attr>
  </node>
  <node id="V" type="Var">
    <attr name="Line"><int>225</int></attr>
  </node>
  <node id="Q" type="Proc">
    <attr name="File"><str>test.c</str></attr>
  </node>
  <edge from="P" to="V" type="Ref">
    <attr name="Line"><int>127</int></attr>
  </edge>
  <edge from="P" to="Q" type="Call">
    <attr name="Line"><int>42</int></attr>
  </edge>
</graph>
  
```



A Program Graph and its GXL Representation:

UML Notation for Graph:

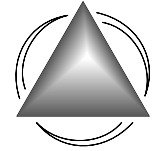


graph with schema

GXL Exchange Format:

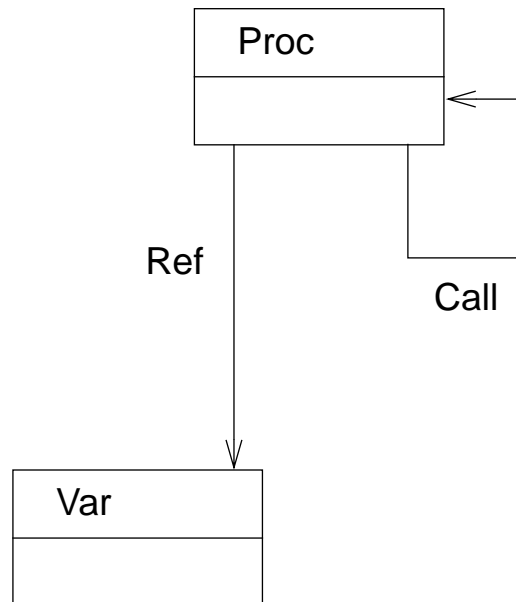
```

<graph schema="ProgSchema">
  <node id="P" type="Proc">
    <attr name="File"><str>main.c</str></attr>
  </node>
  <node id="V" type="Var">
    <attr name="Line"><int>225</int></attr>
  </node>
  <node id="Q" type="Proc">
    <attr name="File"><str>test.c</str></attr>
  </node>
  <edge from="P" to="V" type="Ref">
    <attr name="Line"><int>127</int></attr>
  </edge>
  <edge from="P" to="Q" type="Call">
    <attr name="Line"><int>42</int></attr>
  </edge>
</graph>
  
```

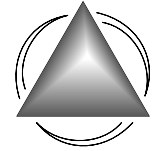


A Program Graph Schema and its GXL Representation:

UML Notation for Graph Schema:

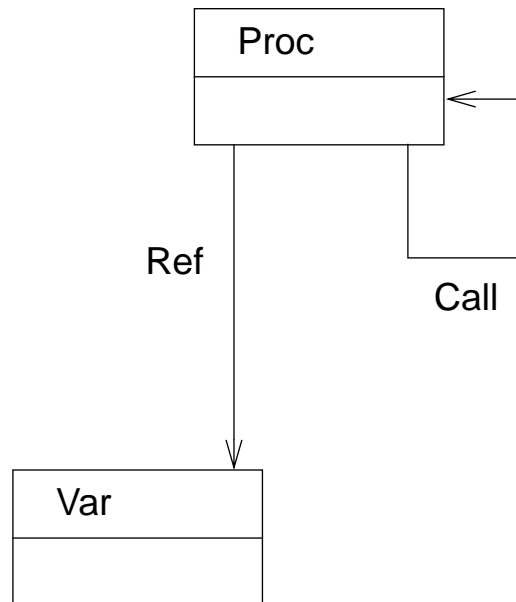


simple graph schema



A Program Graph Schema and its GXL Representation:

UML Notation for Graph Schema:



simple graph schema

GXL Exchange Format:

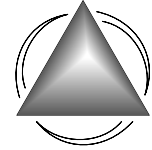
```
<graph id="ProgSchema">
  <node id="Proc" />

  <node id="Var" />

  <edge from="P" to="V" type="Ref" />

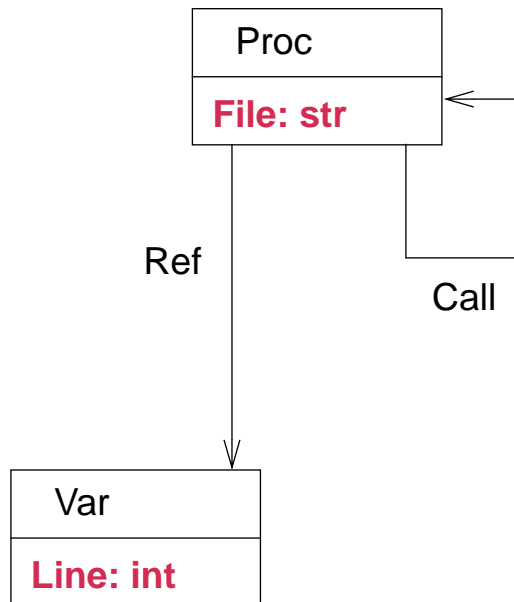
  <edge from="P" to="Q" type="Call" />

</graph>
```



A Program Graph Schema and its GXL Representation:

UML Notation for Graph Schema:



graph schema with
node attributes

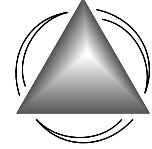
GXL Exchange Format:

```

<graph id="ProgSchema">
  <node id="Proc">
    <attr name="File"><str/></attr>
  </node>
  <node id="Var">
    <attr name="Line"><int/></attr>
  </node>
  <edge from="P" to="V" type="Ref" />

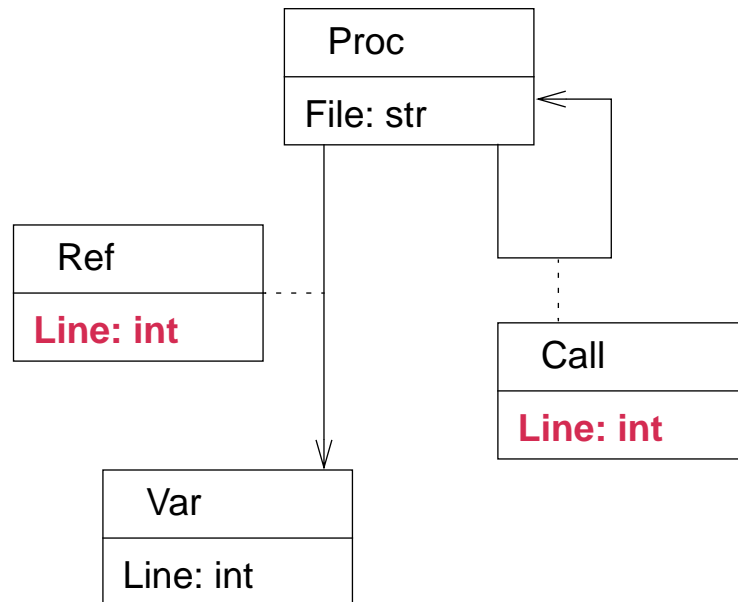
  <edge from="P" to="Q" type="Call" />

</graph>
  
```



A Program Graph Schema and its GXL Representation:

UML Notation for Graph Schema:

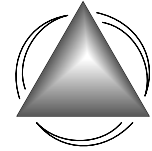


graph schema with
edge attributes

GXL Exchange Format:

```

<graph id="ProgSchema">
  <node id="Proc">
    <attr name="File"><str/></attr>
  </node>
  <node id="Var">
    <attr name="Line"><int/></attr>
  </node>
  <edge from="P" to="V" type="Ref">
    <attr name="Line"><int/></attr>
  </edge>
  <edge from="P" to="Q" type="Call">
    <attr name="Line"><int/></attr>
  </edge>
</graph>
  
```



2. Comparison with Other Graph Exchange Formats

Exchange formats presented at the meeting:

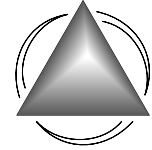
- ⇒ Berlin proposal
- ⇒ Kent proposal (GRXL)
- ⇒ Catalonia proposal

Related exchange formats (not considered):

- ⇒ GML (University of Passau)
- ⇒ GraphXML (University of Amsterdam)

Compared features:

- ⇒ Nodes and node types
- ⇒ Edges and edge types
- ⇒ attribute instances and declarations



Representation of Nodes:

Berlin Proposal:

```

<!ELEMENT Node (TaggedValue?, Attribute*, NodeLayout?) >
<!ATTLIST Node
  id          ID          #REQUIRED  <!-- defines node id -->
  type       IDREF       #IMPLIED  <!-- type id must be defined in same document -->
>

```

Catalonia Proposal:

```

<!ELEMENT Node EMPTY>
<!ATTLIST Node
  id          ID          #REQUIRED  <!-- defines node id -->
  label      CDATA       #IMPLIED

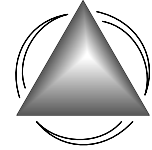
```

GXL Proposal (simplified):

```

<!ELEMENT node (attr*) >
<!ATTLIST node
  id          ID          #REQUIRED  <!-- defines node id -->
  type       NMTOKEN     #IMPLIED  <!-- type id may be defined in other document -->
>

```



Kent Proposal:

```
<!ELEMENT node (attr*) >
```

```
<!ATTLIST node
```

```
  id          ID          #REQUIRED  <!-- defines node id -->
```

```
  type       IDREF       #IMPLIED  <!-- type id must be defined in same document -->
```

```
  match      IDREF       #IMPLIED
```

```
  label      CDATA       #IMPLIED
```

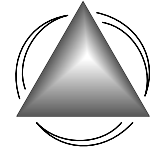
```
  ... <!-- layout attributes + lhs/rhs attributes -->
```

```
>
```

```
>
```

Questions:

- ⇒ explicit support for definition of layout data ?
- ⇒ “type” and/or “label” attribute ?
- ⇒ “type” is IDREF or NMTOKEN or CDATA ?
- ⇒ “TaggedValue” and “Attribute” ?
- ⇒ purpose of “match” attribute ?



Representation of Directed Edges:

Berlin Proposal:

```

<!ELEMENT Edge (TaggedValue?, Attribute*, EdgeLayout?) >
<!ATTLIST Edge
  id      ID          #REQUIRED  <!-- defines edge id -->
  type    IDREF       #IMPLIED  <!-- type id must be defined in same document -->
  source  IDREFS      #IMPLIED  <!-- references nodes -->
  target  IDREFS      #IMPLIED  <!-- references nodes -->
>

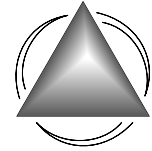
```

Catalonia Proposal:

```

<!ELEMENT edge EMPTY>
<!ATTLIST edge
  id      ID          #REQUIRED  <!-- defines edge id -->
  source  IDREF       #REQUIRED  <!-- references nodes -->
  target  IDREF       #REQUIRED  <!-- references nodes -->
  label   CDATA      #IMPLIED

```



GXL Proposal:

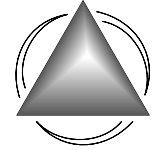
```

<!ELEMENT edge (attr*) >
<!ATTLIST edge
  id      ID          #IMPLIED  <!-- defines edge id -->
  type    NMTOKEN    #IMPLIED  <!-- type id may be defined in other document -->
  from    IDREF       #REQUIRED <!-- references nodes or edges -->
  to      IDREF       #REQUIRED <!-- references nodes or edges -->
>
    
```

Kent Proposal:

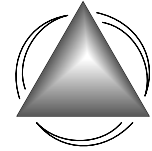
```

<!ELEMENT edge (attr*) >
<!ATTLIST edge
  id      ID          #REQUIRED <!-- defines edge id -->
  type    IDREF       #IMPLIED  <!-- type id must be defined in same document -->
  match   IDREF       #IMPLIED
  begin   IDREF       #REQUIRED <!-- references nodes -->
  end     IDREF       #REQUIRED <!-- references nodes -->
  label   CDATA       #IMPLIED
  ... <!-- + lhs/rhs attributes -->
>
    
```



Questions:

- ⇒ explicit support for definition of layout data ?
- ⇒ “id” of edge REQUIRED ?
- ⇒ “type” and/or “label” attribute ?
- ⇒ “type” is IDREF or NMTOKEN or CDATA ?
- ⇒ “TaggedValue” and “Attribute” ?
- ⇒ edges between edges supported ?
- ⇒ purpose of “match” attribute ?
- ⇒ “begin”/“end” versus “source”/“target” versus “from”/“to” ?
- ⇒ one construct for directed edges and simple form of hyperedges ?



Representation of Attribute Instances:

Berlin Proposal:

```

<!ELEMENT Attribute (AttrLayout?, attrval)>
<!ATTLIST Attribute
  type      IDREF      #REQUIRED
  name      CDATA      #REQUIRED
>
<!ELEMENT attrval ( PrimitiveValue | Objectref | Object )>

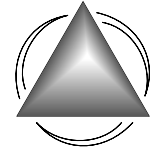
```

GXL Proposal (simplified):

```

<!ELEMENT attr ((%val;)? , attr*) >
<!ATTLIST attr
  name      NMTOKEN  #REQUIRED  <!-- uses name of attribute declaration -->
>
<!ENTITY % val " value                                <!-- for values of not predefined standard type -->
                | locator                               <!-- pointer to node, object, constant, ... -->
                | bool | int | float | string          <!-- standard types -->
                | seq | bag | set | struct | union
                | foreign                               <!-- DTD extension point -->
                " >

```

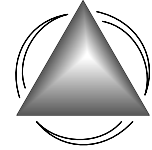


Kent Proposal:

```
<!ELEMENT attr (attrelement)*>
<!ATTLIST attr
  name      CDATA      #REQUIRED
  value     CDATA      #IMPLIED
>
<!ELEMENT attrelement EMPTY>
<!ATTLIST attrelement
  name      CDATA      #REQUIRED
  value     CDATA      #IMPLIED
>
```

Questions:

- ⇒ explicit support for definition of layout data ?
- ⇒ attributes of attributes ?
- ⇒ attribute value type is IDREF or part of value definition or omitted or ... ?
- ⇒ complex values such as seq, struct, ... , object needed ?
- ⇒ attribute value is attribute and/or child of tag ?
- ⇒ explicit support for attribute “kind” ?



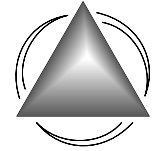
Representation of Node Types:

Berlin Proposal:

```
<!ELEMENT NodeType (TaggedValue?, AttrType*, NodeTypeLayout?) >
<!ATTLIST NodeType
  id      ID      #REQUIRED
  name    NMTOKEN #REQUIRED
>
```

GXL Proposal (simplified):

```
<!ELEMENT node (attr*) >
<!ATTLIST node
  id      ID      #REQUIRED  <!-- defines node type id -->
  type    NMTOKEN #IMPLIED  <!-- references meta type -->
>
```

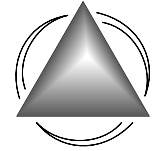


Kent Proposal:

```
<!ELEMENT nodetype (attr*) >
<!ATTLIST nodetype
  id      ID      #REQUIRED  <!-- defines node type id -->
  parent  IDREF   #IMPLIED   <!--inheritance ? -->
  ... <!-- layout attributes + lhs/rhs attributes -->
>
```

Questions:

- ⇒ explicit support for definition of layout data ?
- ⇒ different syntax for nodes and node types ?
- ⇒ meta modeling support required ?
- ⇒ distinction between “id” and “name” ?
- ⇒ inheritance via “parent” attribute (versus isa-edges) ?
- ⇒ “TaggedValue” and “Attribute” ?



Representation of Edge Types:

Berlin Proposal:

```

<!ELEMENT EdgeType (TaggedValue?, AttachName*, AttrType*, EdgeTypeLayout?) >
<!ATTLIST Edge
  id      ID          #REQUIRED
  name    NMTOKEN    #REQUIRED
  source  IDREFS     #IMPLIED    <!-- references node types -->
  target  IDREFS     #IMPLIED    <!-- references node types -->
>
<!ELEMENT AttachName EMPTY>
<!ATTLIST AttachName
  name    NMTOKEN    #REQUIRED
>

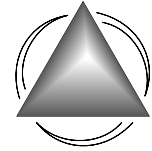
```

Kent Proposal:

```

<!ELEMENT edgetype (attr*) >
<!ATTLIST edgetype
  id      ID          #REQUIRED    <!-- defines edge type id -->
  parent  IDREF       #IMPLIED    <!-- type id must be defined in same document -->
  directed (true|false) "true"

```



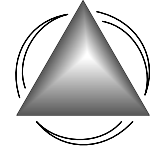
GXL Proposal:

```

<!ELEMENT edge (attr*) >
<!ATTLIST edge
  id      ID          #IMPLIED    <!-- defines edge type id -->
  type    NMTOKEN    #IMPLIED    <!-- references meta edge type id -->
  from    IDREF      #REQUIRED   <!-- references node or edge type -->
  to      IDREF      #REQUIRED   <!-- references node or edge type -->
>
    
```

Questions:

- ⇒ explicit support for definition of layout data ?
- ⇒ different syntax for edges and edge types (meta modeling support)?
- ⇒ distinction between “id” and “name” ?
- ⇒ inheritance via “parent” attribute (versus isa-edges) ?
- ⇒ same construct for directed edges and hyperedges ?
- ⇒ “TaggedValue” and “Attribute” ?
- ⇒ “directed” flag needed (instead of “directed” flag for graph)
- ⇒ purpose of “attach” construct (not used by edge instances) ?



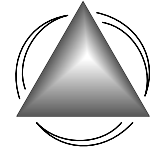
Representation of Attribute Declarations:

Berlin Proposal:

```
<!ELEMENT AttrType (TaggedValue?, AttrTypeLayout?)>
<!ATTLIST Attribute
  id          ID          #REQUIRED
  typename   NMTOKEN    #REQUIRED
  attrname   NMTOKEN    #REQUIRED
>
```

GXL Proposal (simplified):

```
<!ELEMENT attr ((%val;)? , attr*) >
<!ATTLIST attr
  name       NMTOKEN    #REQUIRED  <!-- uses name of attribute declaration -->
>
```

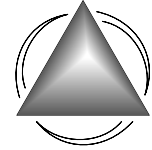


Kent Proposal:

```
<!ELEMENT attr (attrelement)*>
<!ATTLIST attr
  name      CDATA      #REQUIRED
  value     CDATA      #IMPLIED
>
<!ELEMENT attrelement EMPTY>
<!ATTLIST attrelement
  name      CDATA      #REQUIRED
  value     CDATA      #IMPLIED
>
```

Questions:

- ⇒ explicit support for definition of layout data ?
- ⇒ attributes of attributes ?
- ⇒ distinction between “id” and “name” needed ?
- ⇒ attribute type is IDREF or part of (default) value definition or omitted ?
- ⇒ complex value types such as seq, struct, ... , object needed ?



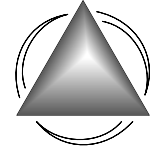
3. Advanced GXL Features

Support of other graph models:

- ⇒ multigraphs with edge identifiers
- ⇒ multigraphs with ordered edges (including trees)
- ⇒ rather general form of hypergraphs
- ⇒ graphs with complex attribute values

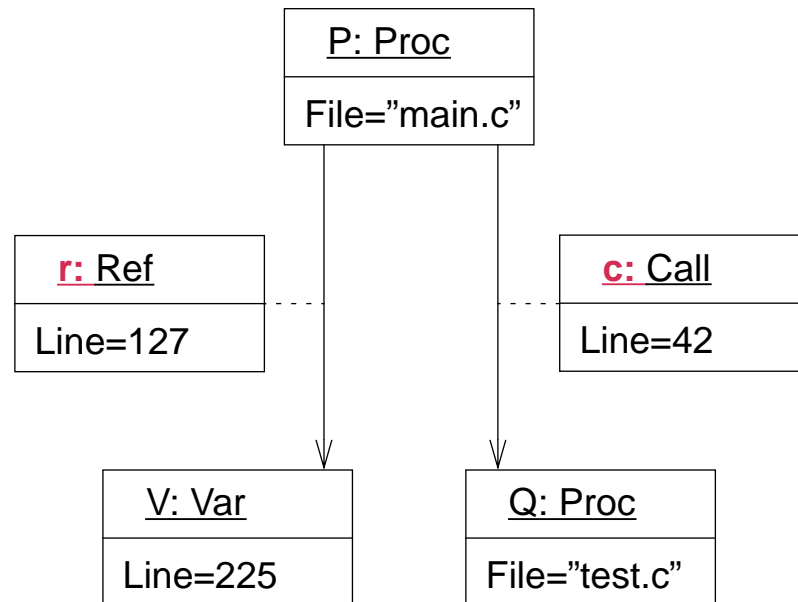
Advanced schema modeling features:

- ⇒ representation of derived attributes
- ⇒ schema graphs with meta schema
- ⇒ meta schema (being its own meta meta schema)



A Multigraph with Edge Identifiers and its GXL Representation:

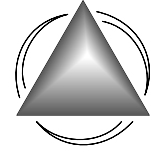
UML Notation:



GXL Exchange Format:

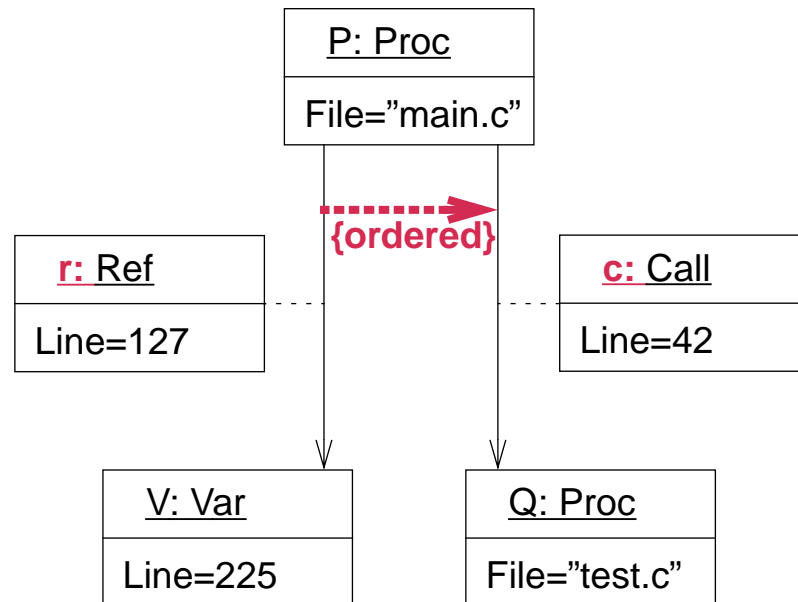
```

<graph id="Program" schema="ProgSchema">
  <node id="P" type="Proc">
    <attr name="File"><str>main.c</str></attr>
  </node>
  <node id="V" type="Var">
    <attr name="Line"><int>225</int></attr>
  </node>
  <node id="Q" type="Proc">
    <attr name="File"><str>test.c</str></attr>
  </node>
  <edge id="r" from="P" to="V" type="Ref">
    <attr name="Line"><int>127</int></attr>
  </edge>
  <edge id="c" from="P" to="Q" type="Call">
    <attr name="Line"><int>42</int></attr>
  </edge>
</graph>
  
```



An Ordered Multigraph Graph and its GXL Representation:

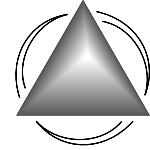
UML-like Notation:



GXL Exchange Format:

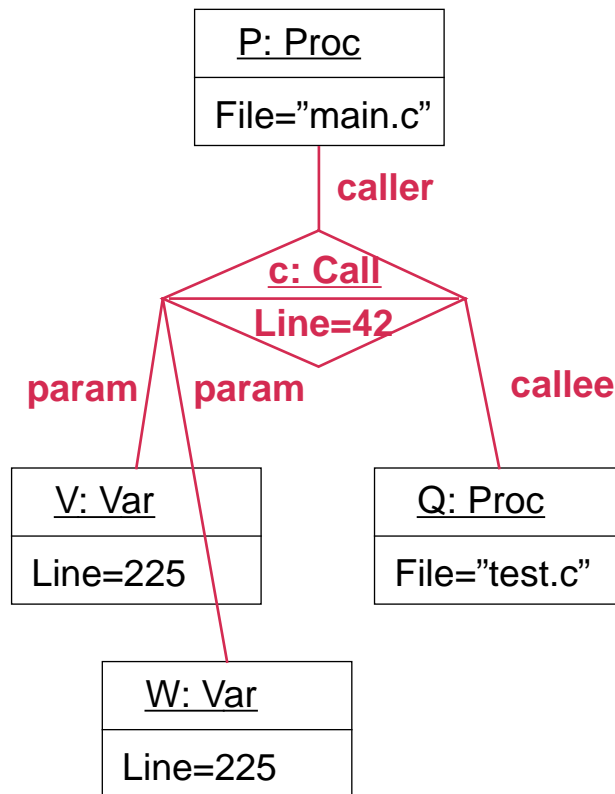
```

<graph id="Program" schema="ProgSchema">
  <node id="P" type="Proc">
    <attr name="File"><str>main.c</str></attr>
    <order role="out" edges="r c">
  </node>
  <node id="V" type="Var">
    <attr name="Line"><int>225</int></attr>
  </node>
  <node id="Q" type="Proc">
    <attr name="File"><str>test.c</str></attr>
  </node>
  <edge id="r" from="P" to="V" type="Ref">
    <attr name="Line"><int>127</int></attr>
  </edge>
  <edge id="c" from="P" to="Q" type="Call">
    <attr name="Line"><int>42</int></attr>
  </edge>
</graph>
  
```



A Hypergraph and its GXL Representation:

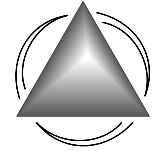
UML-like Notation:



GXL Exchange Format:

```

<graph id="Program" schema="ProgSchema">
  <node id="P" type="Proc">
    <attr name="File"><str>main.c</str></attr>
  </node>
  <node id="V" type="Var">
    <attr name="Line"><int>225</int></attr>
  </node>
  <node id="Q" type="Proc">
    <attr name="File"><str>test.c</str></attr>
  </node>
  <rel id="c" type="Call">
    <attr name="Line"><int>42</int></attr>
    <link role="caller" ref="P" />
    <link role="callee" ref="Q" />
    <link role="param" ref="V W" />
  </rel>
</graph>
  
```



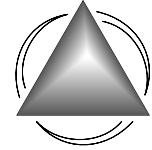
Complex Attribute Values and their GXL Representations:

List of Coordinates representation with standard GXL DTD:

```
<attr name="pointlist">
  <seq>
    <struct> <attr name="x"><float>1.0</float></attr>
      <attr name="y"><float>2.0</float></attr>
    </struct>
    <struct> <attr name="x"><float>2.0</float></attr>
      <attr name="y"><float>3.0</float></attr>
    </struct>
  </seq>
</attr>
```

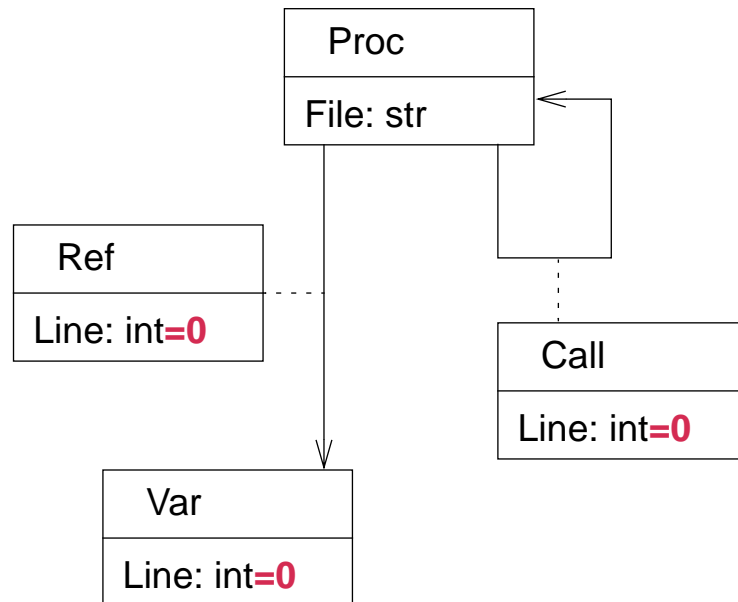
List of Coordinates representation with extended GXL DTD:

```
<attr name="pointlist">
  <seq>
    <coordinate x="1.0" y="2.0" />
    <coordinate x="2.0" y="3.0" />
  </seq>
</attr>
```



A Graph Schema with Default Attribute Values and its GXL Representation:

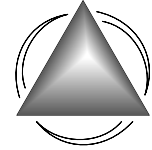
UML Notation for Graph Schema:



GXL Exchange Format:

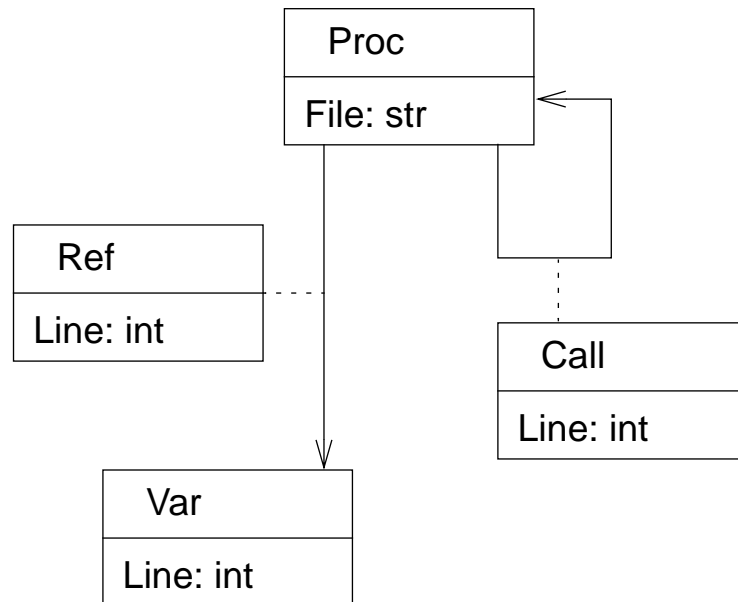
```

<graph id="ProgSchema">
  <node id="Proc">
    <attr name="File"><str/></attr>
  </node>
  <node id="Var">
    <attr name="Line"><int>0</int></attr>
  </node>
  <edge from="P" to="V" type="Ref">
    <attr name="Line"><int>0</int></attr>
  </edge>
  <edge from="P" to="Q" type="Call">
    <attr name="Line"><int>0</int></attr>
  </edge>
</graph>
  
```



A Graph Schema with Edge Type Identifiers and its GXL Representation:

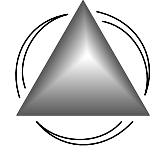
UML Notation for Graph Schema:



GXL Exchange Format:

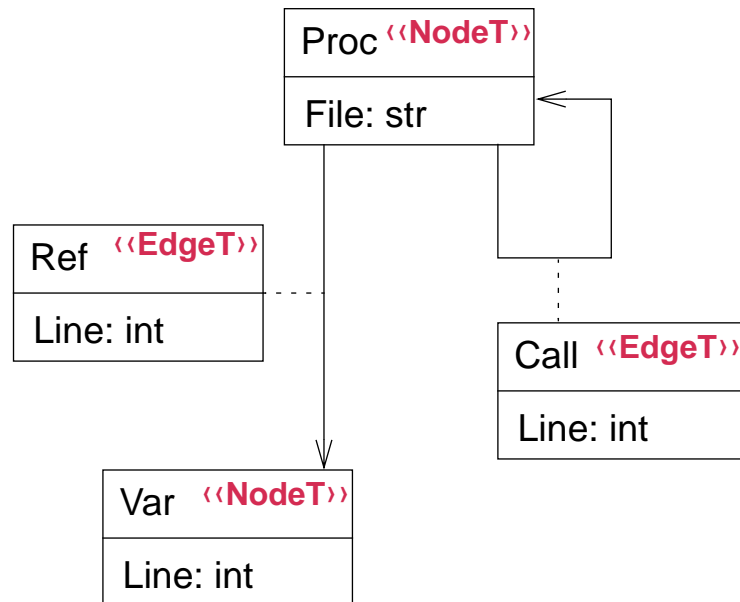
```

<graph id="ProgSchema">
  <node id="Proc">
    <attr name="File"><str/></attr>
  </node>
  <node id="Var">
    <attr name="Line"><int/></attr>
  </node>
  <edge id="Ref" from="P" to="V" type="Ref">
    <attr name="Line"><int/></attr>
  </edge>
  <edge id="Call" from="P" to="Q" type="Call">
    <attr name="Line"><int/></attr>
  </edge>
</graph>
  
```



A Graph Schema with Meta Schema and its GXL Representation:

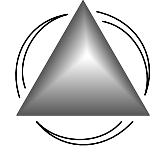
UML Notation for Graph Schema:



GXL Exchange Format:

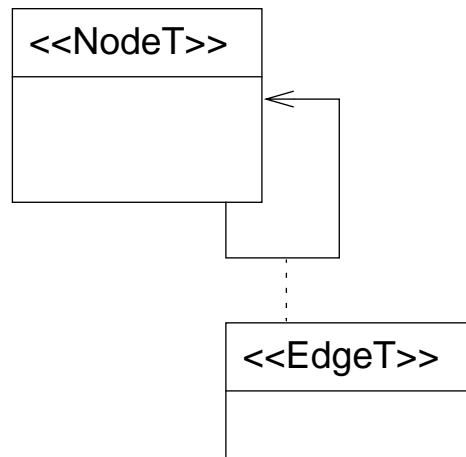
```

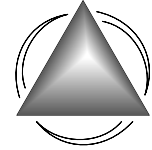
<graph id="ProgSchema" schema="Meta">
  <node id="Proc" type="NodeT">
    <attr name="File"><str/></attr>
  </node>
  <node id="Var" type="NodeT">
    <attr name="Line"><int/></attr>
  </node>
  <edge id="Ref" from="P" to="V" type="EdgeT">
    <attr name="Line"><int/></attr>
  </edge>
  <edge id="Call" from="P" to="Q" type="EdgeT">
    <attr name="Line"><int/></attr>
  </edge>
</graph>
  
```



A Program Graph Meta Schema and its GXL Representation:

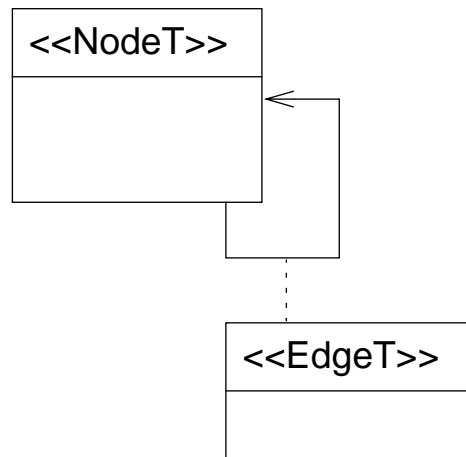
UML-like Notation for Meta Schema:





A Program Graph Meta Schema and its GXL Representation:

UML-like Notation for Meta Schema:

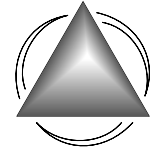


GXL Exchange Format:

```
<graph id="Meta">
  <node id="NodeT" />

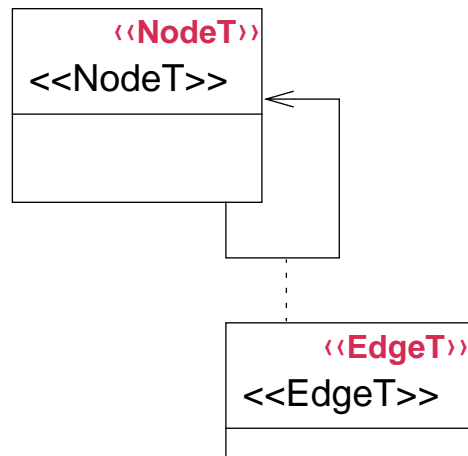
  <edge id="EdgeT" from="NodeT" to="NodeT" />

</graph>
```



A Self-Referencing Graph Meta Schema and its GXL Representation:

UML-like Notation for Meta Schema

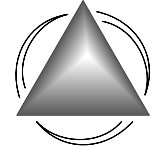


GXL Exchange Format:

```
<graph id="Meta" schema="Meta">
  <node id="NodeT" type="NodeT" />
```

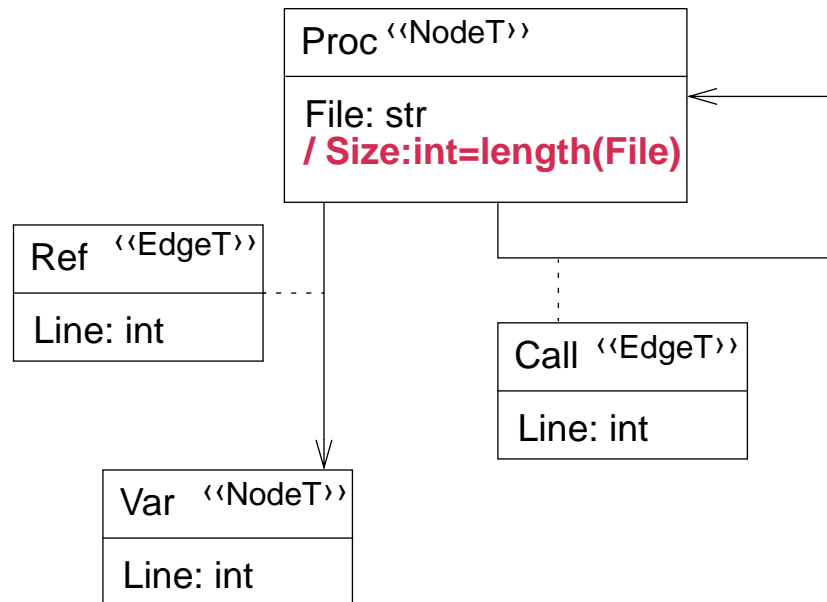
```
  <edge id="EdgeT" from="NodeT" to="NodeT"
    type="NodeT" />
```

```
</graph>
```



A Graph Schema with Derived Attribute and its GXL Representation:

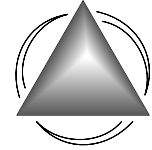
UML Notation for Graph Schema:



GXL Exchange Format:

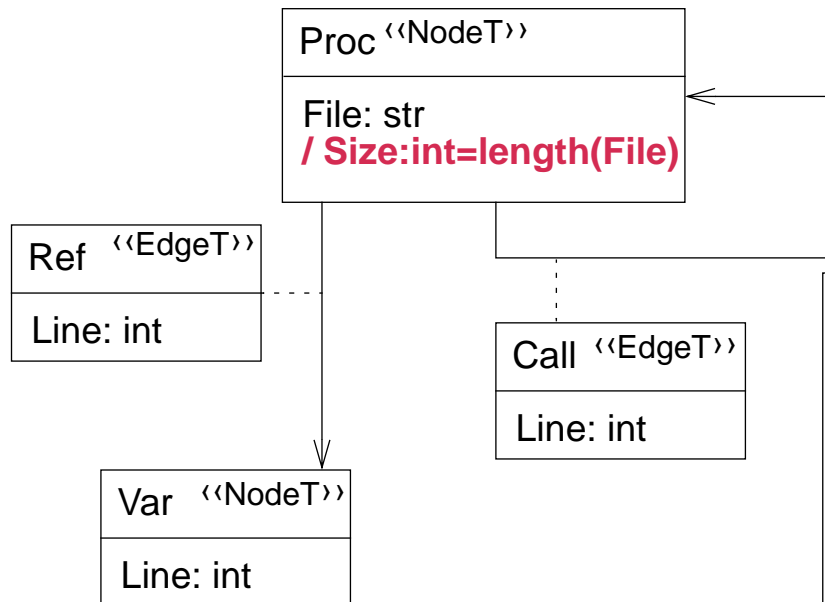
```

<graph id="ProgSchema" schema="Meta">
  <node id="Proc" type="NodeT">
    <attr name="File" kind="Normal"><str/></attr>
    <attr name="Size" kind="Derived">
      <int>length(File)</int>
    </attr>
  </node>
  <node id="Var" type="NodeT">
    <attr name="Line" kind="Normal"><int/></attr>
  </node>
  <edge id="Ref" from="P" to="V" type="EdgeT">
    <attr name="Line" kind="Normal"><int/></attr>
  </edge>
  <edge id="Call" from="P" to="Q" type="EdgeT">
    <attr name="Line" kind="Normal"><int/></attr>
  </edge>
</graph>
  
```



A Graph Schema with Derived Attribute and its GXL Representation:

UML Notation for Graph Schema:



graph schema with
derived attribute

GXL Exchange Format:

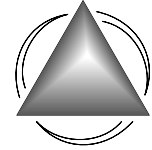
```

<graph id="ProgSchema" schema="Meta">
  <node id="Proc" type="NodeT">
    <attr name="File" kind="Normal"><str/></attr>
    <attr name="Size" kind="Derived">
      <int>length(File)</int>
    </attr>
  </node>
  <edge id="Ref" from="Proc" to="Ref" type="EdgeT">
    <attr name="Line" kind="Normal"><int/></attr>
  </edge>
  <edge id="Call" from="Proc" to="Call" type="EdgeT">
    <attr name="Line" kind="Normal"><int/></attr>
  </edge>
  <edge id="Var" from="Proc" to="Var" type="EdgeT">
    <attr name="Line" kind="Normal"><int/></attr>
  </edge>
</graph>
  
```

or with evaluation code as attribute:

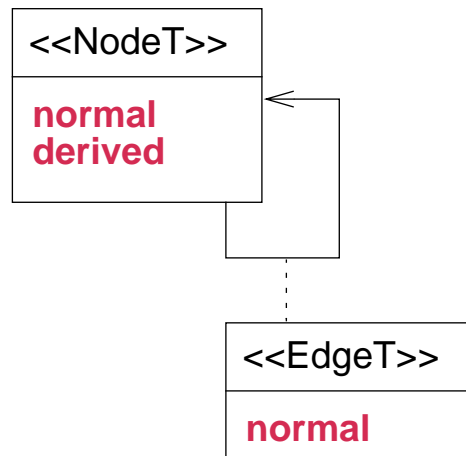
```

<attr name="Size" kind="Derived">
  <int/>
  <attr name="Expr">
    <str>length(File)</str>
  </attr>
</attr>
  
```



A Graph Meta Schema with Attributes and its GXL Representation:

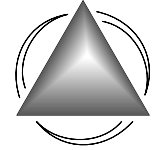
UML Notation for Meta Schema?



GXL Exchange Format:

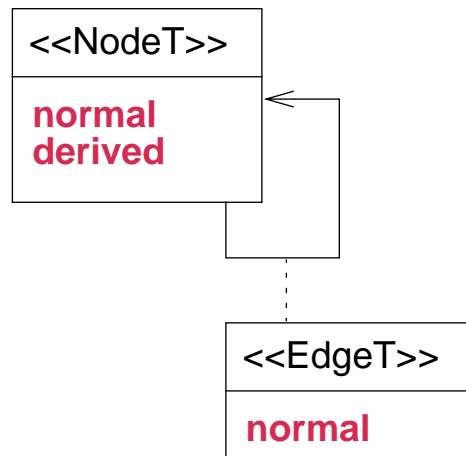
```

<graph id="Meta">
  <node id="NodeT">
    <attr name="Normal"></attr>
    <attr name="Derived"></attr>
  </node>
  <edge id="EdgeT" from="P" to="V">
    <attr name="Normal"></attr>
  </edge>
</graph>
  
```



A Graph Meta Schema with Attributes and its GXL Representation:

UML Notation for Meta Schema?



GXL Exchange Format:

```

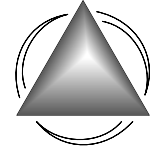
<graph id="Meta">
  <node id="NodeT">
    <attr name="Normal"></attr>
    <attr name="Derived"></attr>
  
```

or with declaration of attribute of attribute:

```

<attr name="Derived">
  <attr name="Expr">
    <str/>
  </attr>
</attr>

```



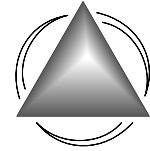
4. Schema Representations and Meta Modeling

Different syntax for node/edge instances and types:

- ✓ clear distinction between instance and type level
- ▼ schema graphs are not ordinary graphs
- ▼ no direct support for meta schema graphs

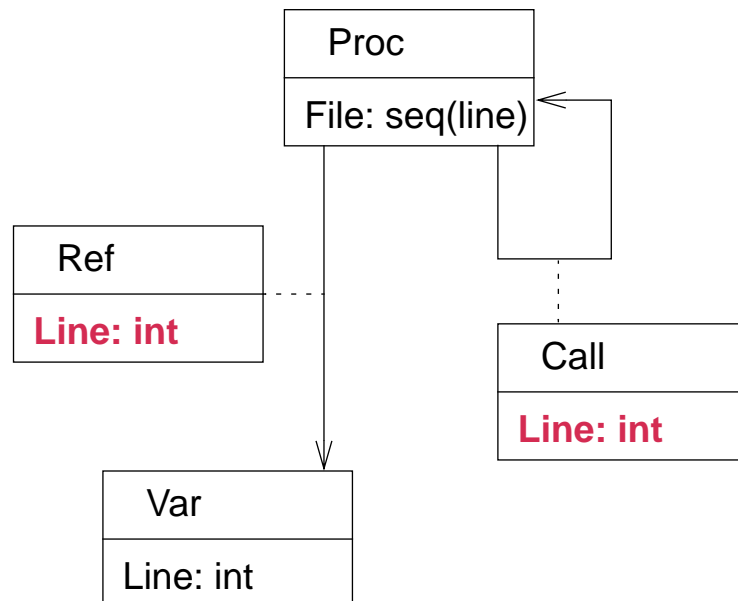
Same syntax for node/edge instances and types:

- ⇒ Variant 1 (TA approach - used until now):
node or edge types are ordinary nodes or edges, attribute declarations are ordinary attributes
- ⇒ Variant 2 (GraX and UML approach):
node or edge types and attribute declarations are nodes of the schema graph connected via edges



Variant 1 (TA variant) of a Program Graph Schema:

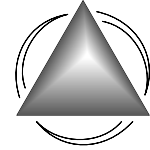
UML Notation for Graph Schema:



GXL Exchange Format:

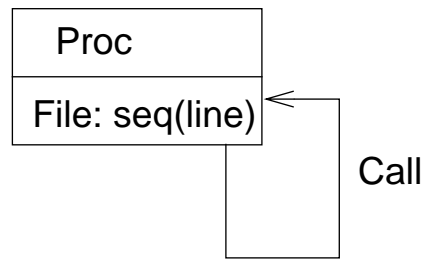
```

<graph id="ProgSchema">
  <node id="Proc">
    <attr name="File">
      <seq><value type="line" /></seq>
    </attr>
  </node>
  <node id="Var">
    <attr name="Line"><int/></attr>
  </node>
  <edge from="P" to="V" type="Ref">
    <attr name="Line"><int/></attr>
  </edge>
  <edge from="P" to="Q" type="Call">
    <attr name="Line"><int/></attr>
  </edge>
</graph>
    
```

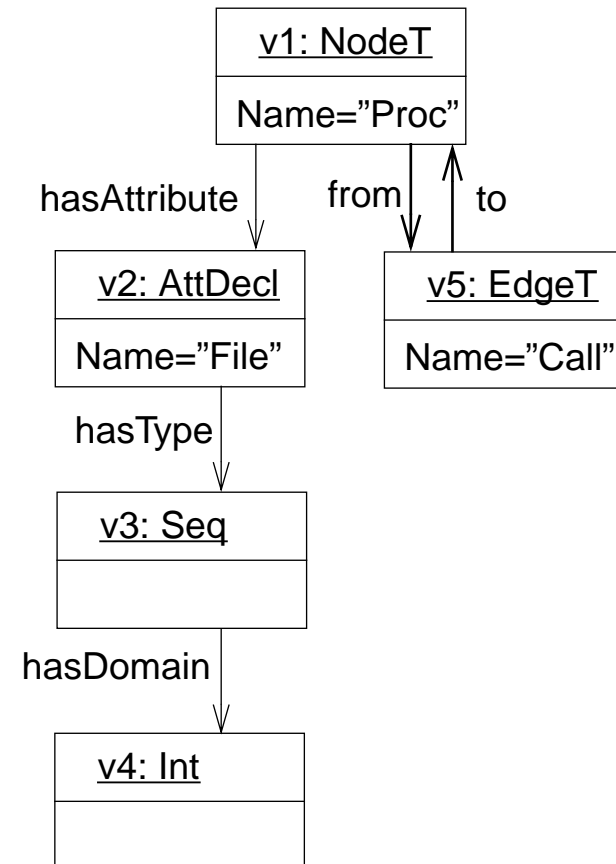


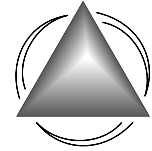
Variant 2 (GraX and UML variant) of a Program Graph Schema:

UML Notation for Graph Schema:



Internal Graph Representation:





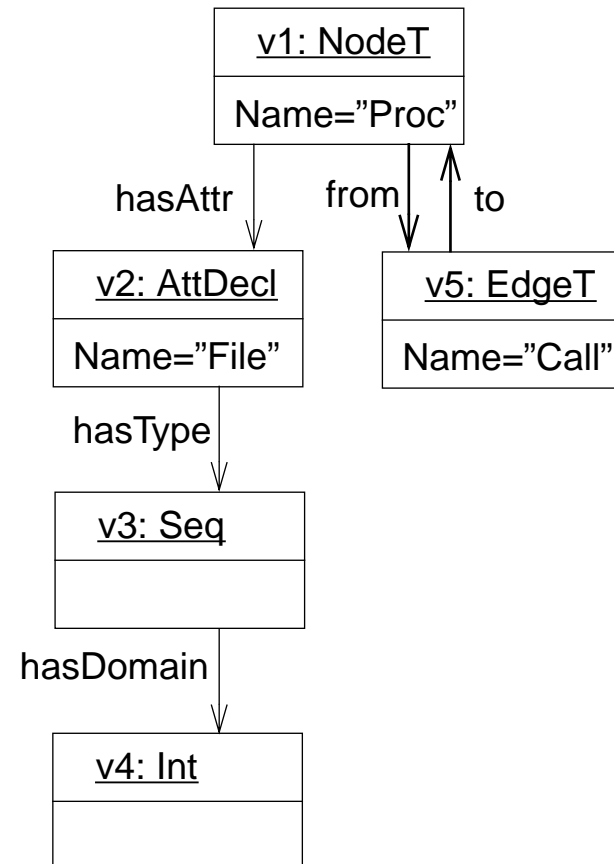
Variant 2 (GraX and UML variant) of a Program Graph Schema:

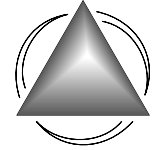
GXL Text Representation:

```

<node id="v1" type="NodeT">
  <attr name="Name"><str>Proc</str></attr>
</node>
<edge from="v1" to="v2" type="hasAttr" />
<node id="v2" type="AttDecl">
  <attr name="Name"><str>File</str></attr>
</node>
<edge from="v2" to="v3" type="hasType" />
<node id="v3" type="Seq" />
<edge from="v3" to="v4" type="hasDomain" />
<node id="v4" type="Int" />
<node id="v5" type="EdgeT">
  <attr name="Name"><str>Call</str></attr>
</node>
<edge from="v1" to="v5" type="from" />
<edge from="v5" to="v1" type="to" />
    
```

Internal Graph Representation:





5. Summary

(Our) goals for this meeting:

- ⇒ find a standard graph exchange format for the graph transformation and the software reengineering community
- ⇒ define extension points for adding layout data, transformation rules, ...

Future work:

- ⇒ merge our standard graph exchange format with the forthcoming graph exchange format of the graph drawing community
- ⇒ realize portable graph exchange format processing tools
- ⇒ add inheritance and graph extension concepts
- ⇒ make graph exchange format “really” MOF compliant (if possible)
- ⇒ use standard link and (attribute) value representation mechanisms of XML (if existent)