An Open Visualization Framework for Metamodel-Based Modeling Languages

Péter Domokos, Dániel Varró
{domokos, varro}@mit.bme.hu

Dept. of Measurement and Information Systems
Budapest Univ. of Technology and Economics
GraBaTs 2002
Outline

• Visual modeling languages and standardization
• An SVG-based visualization framework
  – Conceptual overview
  – Drawing metamodel
  – Automated transformations
• Running example: Petri Nets
• Conclusions and Future work
Defining Visual Modeling Languages

• Abstract syntax
  – visual definition by metamodels
  – UML notation

• Concrete syntax
  – visual representation for models

• Static well-formedness constraints
  – textual constraints
  – graph patterns as constraints

• Dynamic behavior
  – meta-level specification: graph transformation
Objectives

- Automated derivation of graphical representation
  - from elementary AS to CS mappings
  - automated calculation of graphical layout
- Standard, XML-based environment
  - XMI (XML Metadata Interchange) for models
  - XSLT (XSL Transformations) for transformations
  - SVG (Scalable Vector Graphics) for graphics
- Evaluation for GTXL Standardization
  - default visualization for GXL / GTXL?
  - standard mappings?
Characteristics

• Abstract syntax: Metamodeling
  – MOF approach
• Concrete syntax:
  SVG (Scalable Vector Graphics)
  – XML standard
  – visualized by web browsers
• Derivation process:
  – mapping abstract syntax to concrete syntax by model transformations
  – off-line approach (for documentation/evaluation)
Conceptual Overview

- **Domain metamodel**
- **Transformation rules**
- **Drawing metamodel**

- **Domain model**
- **Logical drawing**

- **model transformation (VIATRA)**

- **Inputs**
- **Output**

- **SVG file**
- **XML model**

- **XSLT (Java)**

- **conforms to**

- **layout calculation**
Conceptual Overview

Domain metamodel:
- defines the abstract syntax
- Petri nets: concepts of places, transitions

Domain model:
- a concrete model instance
- well-formedness assumed
- modeled by typed and attributed graphs
- XMI representation

Transformation rules:
- mapping abstract syntax into concrete syntax
- graph transformation rules with spec. structure

Drawing metamodel:
- drawing primitives (circle, line, box, text)
- logical relations (e.g. containment)

Logical drawing

model transformation (VIATRA)

SVG file:
- easy generation from logical drawing model
- automatically rendered by web browsers

SVG generation:
- XSLT technology
- model independent!!!

Layout calculation:
- generating physical coordinates
- off-the-shelf toolkit (IBM GFC)
- exporting in XMI

Budapest University of Technology and Economics
Department of Measurement and Information Systems
Example:
SVG Visualization of Petri Nets
A Metamodel of Petri Nets

- **Class**: Net
  - +outputArcs
  - InputArc
  - +inputArcs
- **Association**: Place
  - +places
  - +fromPlace
  - +toPlace
- **Aggregation**: Token
  - +tokens
- **Aggregation**: Transition
  - +transitions
  - +toTransition
  - +fromTransition
- **Aggregation**: OutputArc
  - +outputArcs
  - +inputArcs

- **Arrows**:
  - from Transition to OutputArc
  - from Place to InputArc
  - from Place to Transition
  - from Transition to Place
  - from Transition to OutputArc
  - from Place to OutputArc
  - from Place to InputArc

- **Elements**:
  - Transition
  - OutputArc
  - InputArc
  - Place
  - Token
Graph Representation of a Petri Net Model
A Metamodel of Drawing Primitives

- **GrObject**
  - +contains
  - +target
  - +source

- **DrawingElement**
  - +start
  - +consist_of
  - +next

- **DrawingPrimitives**
  - stroke
  - stroke-width
  - x
  - y

- **GrNode**
  - +source
  - +target

- **GrEdge**
  - +source
  - +target

- **Text**
  - x
  - y
  - txt

- **Rectangle**
  - width
  - height
  - fill

- **Line**
  - x2
  - y2

- **Ellipse**
  - rx
  - ry
  - fill

- **Circle**
  - r
  - fill
The Drawing Model of Sample Petri Net
Graph Transformation rules:
- Mapping from Petri Net objects to Drawing objects
- Additional control structures
- Defining paintbrush attributes

LHS

RHS

|x : String = '25'|
|y : String = '25'|
|r : String = '25'|
|stroke : String = 'black'|
|stroke-width : String = '2'|
|fill : String = 'white'|

Model Transformation Rules
Checkpoint: What Comes Next?

- Domain metamodel
- Transformation rules
- Drawing metamodel
- Logical drawing
- Model transformation (VIATRA)
- SVG file
- Final XMI model
- XSLT (Java)

Conforms to

Layout calculation
XMI Representation of Drawing Model

XML Element

Current phase:
• paintbrush parameters, BUT
• no physical coordinates
Graph Layout Calculation

• Method:
  – abstracting from drawing primitives
  – calculating coordinates for GrObjects

• Off-the-shelf tool support:
  IBM Graph Foundation Classes (GFC)
  + Java class library for graph
    • creation
    • manipulation
    • visualization
  – problems in visualizing non-planar graphs
Result of Graph Layout Calculation

<GrNode xmi.id="13">
    <contains>
        <GrNode xmi.idref="24"/>
    </contains>
    <start>
        <Circle xmi.idref="14"/>
    </start>
    <consist_of>
        <Circle xmi.id="14">
            <cx>25</cx>
            <cy>25</cy>
            <r>25</r>
            <stroke>black</stroke>
            <stroke_width>2</stroke_width>
            <fill>white</fill>
            <next/>
        </Circle>
    </consist_of>
</GrNode>

<GrNode xmi.id="24">
    <contains/>
    <start>
        <Circle xmi.idref="25"/>
    </start>
    <consist_of>
        <Circle xmi.id="25">
            <cx>25</cx>
            <cy>25</cy>
            <r>10</r>
            <stroke>black</stroke>
            <stroke_width>2</stroke_width>
            <fill>black</fill>
            <next/>
        </Circle>
    </consist_of>
</GrNode>

Current phase:
• paintbrush parameters, AND
• concrete physical coordinates

Java + GFC:
• easy XML handling
• platform independence
XSL Transformation

• XSL Transformations
  – XML standard for syntactic manipulation of XML documents

• In Our Framework:
  – From: Drawing XMI
  – To: SVG format

• General solution
  – one XSLT sheet
  – any instance of any modeling language
  – another transformation technique, BUT
    *hidden from the user*
Scalable Vector Graphics (SVG)

- SVG = an XML-based language to describe vector graphical images
  - Drawing primitives: circle, rectangle, line, etc.
  - Grouping: for composite drawing objects
  - Basic manipulations: rotate, scale, translate,
  - User interaction...

```xml
<svg width="100%" height="100%" viewBox="0 0 200 600">
  <g transform="translate(125 125)">
    <!-- place -->
    <circle cx='25' cy='25' r='25'
      stroke='black' stroke-width='2'
      fill='white'/>
  </g>
  <!-- token -->
  <circle cx='25' cy='25' r='10'
    stroke='black' stroke-width='2'
    fill='black'/>
</g>
</svg>
```
Conclusions

• Open visualization framework for VLs
  + highly automated
    • model transformations
    • graph layout calculation
  + heavily relies on XML technologies
  + tool independent architecture

– off-line (no user interaction)
– GFC: not suitable for complex graph layout
– lack of graphical constraints
Future Work (Suggestions)

• More sophisticated graph drawing tools
  – IBM GFC: not strong enough

• Drawing Constraints: CSVG (Constraint SVG)
  http://www.cs.washington.edu/research/constraints/web/csvg-www10
  – using constraints as further graphical restrictions

• User Interaction in SVG
  – handling of mouse events, etc.
  – Vision: SVG-based UI for visual language tools?

• Default SVG representation for GXL/GTXL