Grailog KS Viz: A Grailog Visualizer for Datalog RuleML Using an XSLT Translator to SVG
Agenda

- Introduction
  - Graph inscribed logic (Grailog)
  - Approach overview
  - Supported Grailog elements
  - Example

- SVG elements
  - Used elements

- Transformation using XSLT
  - Basic structure of Grailog KS Viz

- Demo

- Conclusion & Future Work
Graph inscribed logic (Grailog)

- Systematic combination of generalized graph constructs for **data & knowledge representation**
- 2-dimensional graph-logic **visualization for humans** in the loop of data & knowledge elicitation, specification, validation and reasoning

**Grailog:**

For further information, please see: [http://wiki.ruleml.org/index.php/Grailog](http://wiki.ruleml.org/index.php/Grailog)

**Datalog RuleML:**

```
<Implies>
  <Atom>
    <Rel>sell</Rel>
    <Var>merchant</Var>
    <Var>person</Var>
    <Var>object</Var>
  </Atom>
</Implies>
```

```
<Implies>
  <Atom>
    <Rel>buy</Rel>
    <Var>person</Var>
    <Var>merchant</Var>
    <Var>object</Var>
  </Atom>
</Implies>
```
Approach overview

- Initial version of a "Grailog Knowledge-Source Visualizer"
- Mapping from Datalog RuleML/XML, to Scalable Vector Graphics (SVG)/XML:
  - subset to subset
  - target Grailog's "fully node copied" normal form
- Realized by using eXtensible Stylesheet Language Transformations 2.0 (XSLT 2.0)

**Methodology:**
- create Grailog representation in SVG
- create transformation from Datalog RuleML XML to SVG Grailog representation
Supported Grailog elements

- Binary relations:
  - inst1 \(\text{binrel}\) inst2
  - var1 \(\text{binrel}\) var2

- N-ary relations (n>2):
  - inst1 \(\text{rel}\) inst2 \(\text{inst3}\) \(\text{inst4}\)

- Single-premise rules:
  - inst1 \(\text{rel1}\) inst2 inst3 inst4
  - inst5 \(\text{rel2}\) inst6 inst7 inst8

- Multi-premise rules:
  - term11 \(\text{rel1}\) term12 term13 term14
  - term21 \(\text{rel2}\) term22 term23 term24
  - term31 \(\text{rel3}\) term32 term33 term34
Example

If George knows a player and an arena is a hockey rink, then George plays with the player in the arena.

```
<RuleML>
  <Assert mapClosure="universal">
    <Implies>
      <And>
        <Atom>
          <Rel>know</Rel>
          <Ind>George</Ind>
          <Var>player</Var>
        </Atom>
        <Atom>
          <Rel>be</Rel>
          <Var>arena</Var>
          <Ind>Hockey rink</Ind>
        </Atom>
      </And>
      <Atom>
        <Rel>play</Rel>
        <Ind>George</Ind>
        <Var>player</Var>
        <Var>arena</Var>
      </Atom>
    </Implies>
  </Assert>
</RuleML>
```
Individual constants and variables

- **text**
  - id, x, y

- **rect**
  - id, x, y, rx, ry, height, width, stroke (color, width)

- **polygon**
  - id, points, stroke (color, width), style

- **pattern**
  - id, patternTransform, x, y, width, height

- **path**
  - id, d, stroke (color, width)

Lengths of the text elements and positions and scales of the rectangles / polygons were calculated and assigned with JavaScript.
Arrows, double-arrows and rectangles with rounded corners

- **marker**
  - `id`, `markerWidth`, `markerHeight`

- The arrow is an SVG path element with an arrow head, which is created with a marker element and another path element, on top.

- The double-arrow is created with a path on a particular track.

- Positions and scales of these elements were also calculated and assigned with JavaScript.
Basic structure of Grailog KS Viz (1)

- Start the SVG file:
  - definitions (id, version, namespace, marker, pattern)

- **Differentiate** between rules and facts:
  - searching for `<Implies>`, `<And>` and `<Atom>`
  - using parent and child relationships and positions for determining the type of the considered atom
    - fact
    - single-premise rule: head, single premise
    - multi-premise rule: head, multi premise

- **Differentiate** between binary and n-ary (n>2):
  - using the number of children of an atom
  - used for determining if a simple arrow or an (intermediate-node-cutting) hyperarc arrow is needed
Basic structure of Grailog KS Viz (2)

- **Create unique variable names** for the used JavaScript code:
  - concatenation of the type and the position in the XML tree

- **Create** the needed **SVG elements**:
  - text (with its value from the source file), rect, polygon, path

- **Create** the needed **JavaScript code**:
  - using `<xsl:if>` for alternating the code
  - keeping track of the maximum height and width of the viewBox of the resulting SVG file

- **End** the SVG file
Demo
Conclusion

- Successful implementation of a **reliable, fast and easy-to-use** tool for transforming Datalog RuleML/XML rules and facts into their corresponding SVG/XML Grailog visualization(s)

Future Work

- **Merging** of the individual SVG elements of rules and facts to one graph
  - User could directly see connectivity
  - High computational complexity could lead to response-time issues

- **Support** of:
  - unary relations, (positional-)slotted variants and typed variants
  - Grailog visualizers for other rule and ontology languages

- Realization of **inverse translators** parsing Grailog SVG/XML diagrams into RuleML/XML trees
  - Authoring tool that allows users to visually design rule bases in the graphically rendered SVG representation
  - Could ultimately lead to a complete Grailog IDE


http://ruleml.org/papers/Primer/RuleMLPrimer2012-08-09/RuleMLPrimer-p0-2012-08-09.html, visited on October 19th, 2012

http://ruleml.org/1.0/, visited on October 19th, 2012


http://www.ruleml.org/1.0/exa/Datalog, visited on June 12th, 2013
Thank you!

Project website:  http://www2.unb.ca/~mkoch/cs6795swt/index.html

(The project website provides detailed documentation of the project, concrete examples, the Grailog KS Viz tool to download and further links and information to all related topics)

Authors:

Martin Koch:  
(University of New Brunswick)  

martin.koch[AT]unb.ca

Sven Schmidt:  
(University of New Brunswick)  

sven.schmidt[AT]unb.ca

Harold Boley:  
(University of New Brunswick)  

harold.boley[AT]ruleml.org

Rainer Herpers:  
(Bonn-Rhein-Sieg University of Applied Sciences)  

rainer.herpers[AT]h-brs.de
Appendix
SVG

- Features of SVG used to create the Grailog elements:
  - static graphics
  - text and vector graphic shapes as graphical objects
  - assigning styles and transformations
- All described in XML

JavaScript

- Was essential...
  - to get the lengths of the different texts
  - to scale the elements
  - to position the elements
  - to assign other dynamic features
- The importance of JavaScript was not obvious in the first considerations of the project
Example: individual constant

```html
<svg version="1.1" xmlns="http://www.w3.org/2000/svg">

  <text id="text1" x="50" y="50">inst1</text>
  <rect id="rect1" style="stroke:#000000; fill: none; stroke-width:1;"/>

  <script>
    document.getElementById("rect1").setAttribute("width",
      parseFloat(document.getElementById("text1").getComputedTextLength())
    + 20);
    document.getElementById("rect1").setAttribute("height",40);
    document.getElementById("rect1").setAttribute("x",
      parseFloat(document.getElementById("text1").getAttribute("x")) - 10);
    document.getElementById("rect1").setAttribute("y",
      parseFloat(document.getElementById("text1").getAttribute("y")) - 25);
  </script>

</svg>
```
XSLT

- Features of XSLT used to create the SVG Grailog representation:
  - XSLT Elements
    - stylesheet, output
    - apply-templates, template
    - value-of, for-each
    - variable, if, text
  - XSLT Functions
    - current()
    - concat(string, string, ...)
    - not(arg)
    - count((item, item, ...))
    - position()
    - last()
Example: individual constant

```xml
<xsl:if test="parent::r:Assert">
  <xsl:variable name="countRelations" select="count(preceding-sibling::*) + 1"/>
  <xsl:for-each select="."/>
  <xsl:if test="position()=2"><!-- First term -->
    <xsl:if test="(ancestor-or-self::r:Ind) or (ancestor-or-self::r:Data)"/>
    <xsl:element name="rect">
      <xsl:attribute name="id">
        <xsl:value-of select="concat('rect','Relation',$countRelations,position())"/>
      </xsl:attribute>
      <xsl:attribute name="style">
        stroke:#000000; fill: none; stroke-width:1;
      </xsl:attribute>
    </xsl:element>
    <xsl:element name="text">
      <xsl:attribute name="id">
        <xsl:value-of select="concat('text','Relation',$countRelations,position())"/>
      </xsl:attribute>
      <xsl:value-of select="."/>
    </xsl:element>
  </xsl:if>
  <script type="text/javascript" language="JavaScript">
    … (relevant JavaScript code with adjusted variable names etc.) …
  </script>
</xsl:if>
… (second term, next terms, last term; each for „Ind“, „Data“, „Var“ and „Rel“) …
```
Difficulties and their solutions

- Support of two different source document versions:
  - Datalog RuleML file without and with RuleML namespace and schema

- Support of all recent major webbrowsers:
  - two different methods for computing the length of the text elements
    - needed for determining the width and position of the graphical elements
  - normal version of the Grailog Visualizer
    - uses `getComputedTextLength()`
    - does not work in Firefox
  - monospaced font version of the Grailog Visualizer
    - uses `XMLSerializer()` and `serializeToString()`
    - does work in Firefox, but not in Safari or Chrome
Supported Web Browsers

- Supported (green) and unsupported (red) Web browsers of the normal version of Grailog KS Viz:

<table>
<thead>
<tr>
<th>Representation in Browser</th>
<th>Firefox (16.0.2)</th>
<th>Google Chrome (23.0.1271.64 m)</th>
<th>Internet Explorer (9.0.6112.16421)</th>
<th>Opera (12.10)</th>
<th>Safari (5.1.7)</th>
</tr>
</thead>
<tbody>
<tr>
<td>After on-the-fly transformation</td>
<td>not supported</td>
<td>not supported</td>
<td>supported</td>
<td>supported</td>
<td>supported</td>
</tr>
<tr>
<td>Retrospectively saved as SVG</td>
<td>supported</td>
<td>supported</td>
<td>supported</td>
<td>supported</td>
<td>supported</td>
</tr>
</tbody>
</table>

- Supported (green) and unsupported (red) Web browsers of the monospaced font version of Grailog KS Viz:

<table>
<thead>
<tr>
<th>Representation in Browser</th>
<th>Firefox (16.0.2)</th>
<th>Google Chrome (23.0.1271.64 m)</th>
<th>Internet Explorer (9.0.6112.16421)</th>
<th>Opera (12.10)</th>
<th>Safari (5.1.7)</th>
</tr>
</thead>
<tbody>
<tr>
<td>After on-the-fly transformation</td>
<td>supported</td>
<td>not supported</td>
<td>supported</td>
<td>supported</td>
<td>not supported</td>
</tr>
<tr>
<td>Retrospectively saved as SVG</td>
<td>supported</td>
<td>not supported</td>
<td>supported</td>
<td>supported</td>
<td>not supported</td>
</tr>
</tbody>
</table>