Obvious: A Meta-Toolkit to Encapsulate Information Visualization Toolkits — One Toolkit to Bind Them All

By Fekete et al.

Original paper presented at the IEEE Conference on Visual Analytics Science and Technology

Presented by Michael Shah
October 31, 2012
Duration: ~10-15 min.
"[One Toolkit to Rule them], One toolkit to Bind Them All"
Software Engineering Problem for VA applications

- What limitations will we have? (Tradeoffs)
  - What data structures do we have
  - What data structures do we have to add
  - Performance (Can we do things interactively)
  - Can we predict these when we start a project?

- What kinds of tools can we leverage for our research?
Obvious: meta-toolkit

- Targeted to Visual Analytics developers
- Provides lots of visualizations and interaction techniques
- Extensible choice of toolkits to interact with
- Encapsulates InfoVis toolkit
  - Obviousx - Java library provide utility classes to Obvious data model(common formats, CSV, GraphML, etc.)
Benefits of Obvious Toolkit

- Improves the reusability of code and components;
- Improves the interoperability of code, data models and visualizations;
- Defers the choice of which concrete toolkits to use to a later stage of the VA development;
- It enforces a better separation of concerns in VA applications so that the data models can be specified independently of the visualizations and views;
- It allows toolkit and library developers to easily integrate their tool into the rich environment of Obvious-compatible systems;
- It clarifies issues with notification and allows VA to scale up using a standard architecture; and
- It specifies a set of interfaces and a stable vocabulary which simplifies learning.
Related Work

- **Visualization Toolkits**
  - InfoVis Toolkit (IVTK) - in-memory database manager
  - Prefuse
  - Improvise
  - Discovery

- **Graphic Libraries**
  - Scene-Graph toolkits (Piccolo, Jazz, Tulip, Jazz)
    - InvoVis generally favors for flexibility
  - Direct-Graphics(OpenGL, Processing?)

- **Graph Libraries**
  - JUNG, Boost Graph Library, Cytoscape (graph library)

- **Standardization** (ISO to VTK to community)
InfoVis Reference Model

Figure 1: The Information Visualization Reference Model [24]

Polyithic Design - Separation between data, structures, Visualization, View, etc.
(Obvious) Data Model

- Cross-Toolkit interoperability
- Introspection (Reflection, full schema exposed)
- Batch Editing
- Notification (See Figure)
  - Observer Design Pattern
  - Several Strategies for how to send changes
Visualization and View Model

- No agreed upon abstraction (polyithic vs. monolithic)
- Code below can be wrapped and translated into other toolkits (black box provided)
- Another Example:
  - new IvtkTimeSeriesVis( table, null, "timeseries", params)
- An Obvious visualization works with any Obvious data model.

Listing 1: Creating a visualization using a Factory

```java
Map params = new HashMap();
params.put("x", "id");
params.put("y", "age");
Visualization vis = VisualizationFactory.getInstance()
    .createViewVisualization (table, null, "scatterplot", params);
```
Figure 2: Class diagram of the Obvious data model
Proxy Tuple

```
Table
+ get(Row, Field) : Value
+ tuples() : Iterator<Tuple>

Graph
+ tuples() : Iterator<Tuple>
+ nodes() : Iterator<Node>
+ edges() : Iterator<Edge>

Tuple
- row : int
+ get(Field) : Value
+ isValid() : boolean

Node
+ inlinks() : Iterator<Edge>
+ outlinks() : Iterator<Edge>

Edge
+ getSource() : Node
+ getTarget() : Node
```

**Figure 6. The Proxy Tuple Pattern.** Tuples provide an object-oriented proxy for accessing a row of table data. The Node and Edge subclasses play a similar role for graphs, also enabling traversal of the network structure.

## Implementations

<table>
<thead>
<tr>
<th>Bindings</th>
<th>obvious .data</th>
<th>obvious .vis</th>
<th>obvious .view</th>
<th>Binding Size</th>
<th>Toolkit Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prefuse</td>
<td>10 classes 1235 lines</td>
<td>7 classes 446 lines</td>
<td>2 classes 94 lines</td>
<td>120KB</td>
<td>3.3MB</td>
</tr>
<tr>
<td>IVTK</td>
<td>6 classes 1404 lines</td>
<td>5 classes 460 lines</td>
<td>1 class 105 lines</td>
<td>115KB</td>
<td>10MB</td>
</tr>
<tr>
<td>Improvise</td>
<td>3 classes 519 lines</td>
<td></td>
<td></td>
<td>27KB</td>
<td>5.7 MB</td>
</tr>
<tr>
<td>JDBC</td>
<td>8 classes 1936 lines</td>
<td></td>
<td></td>
<td>75KB</td>
<td>3MB</td>
</tr>
<tr>
<td>JUNG</td>
<td>3 classes 714 lines</td>
<td>1 class 139 lines</td>
<td>2 classes 114 lines</td>
<td>93KB</td>
<td>4.3MB</td>
</tr>
</tbody>
</table>

Table 1: Footprint of Obvious bindings.
Evaluation/Usage

- Step 1: Create Obvious data structure
- Step 2: Create Obvious Visualization
- Step 3: Create a View
- See Listings 2,3,4 for code examples
- Can wrap or copy tables from Weka or RapidMiner into Obvious
  - Can
Listing 2: Visualizing a graph with Obvious

```java
// Creates the graph structure. First, set the factory to use (ivtk).
// Then loads the native data structure, and get a factory instance.
// Finally, calls the convenient method of the factory.
System.setProperty("obvious.DataFactory",
        "obvious.ivtk.data.IvtkDataFactory");
infovis.Graph g = Algorithms.getGridGraph(10, 10);
DataFactory factory = DataFactory.getInstance();
Network network = factory.createGraph(g);

// Creates the associated visualization using the
// factory for visualization. No predicates and extra
// parameters are given to the constructor.
Visualization vis = new IvtkVisualizationFactory()
        .createVisualization(network, null, "network", null);

// Creates the view. No predicates and extra parameters are given to
// the constructor.
View view = new IvtkObviousView(vis, null, "graphview", null);
// Standard Java window creation
JFrame frame = new JFrame();
JScrollPane panel = new JScrollPane(view.getViewJComponent());
```
Use Cases

- EdiDuplicate - 1 week to implement
- DBMS Caching Tables - Used Obvious's implementation
- WILD - Network Visualization
Thoughts

- Will multi-language (C++, C#, etc.) support really help this project?
- VTK dominates in scientific visualization, if prefuse or some other tool dominates, how useful is this?
- Meta-toolkit? Meta-data for sure, but what more can be done with Visualization and View models (hard problem I think)
- Curious to see where this has been applied in VA now--overall seems useful
- Would open-source or a plug-in environment beat this system?
Mixed-Initiative Systems

- 3 Pieces
  - visualization
    - Yes
  - interaction analysis
    - ? (Maybe with machine-learning support? New Libraries added)
  - backend/db
    - Yes
Source Code and Project Page

http://code.google.com/p/obvious/wiki/Visualization
Obvious: A Meta-Toolkit to Encapsulate Information Visualization Toolkits — One Toolkit to Bind Them All

By Fekete et al.

Original paper presented at the IEEE Conference on Visual Analytics Science and Technology

Presented by Michael Shah

October 31, 2012

Duration: ~10-15 min.
Factories

A *factory* is an object for creating other objects. It is an abstraction of a *constructor*, and can be used to implement various allocation schemes. For example, using this definition, singletons implemented by the singleton pattern are formal factories.

A factory object typically has a *method* for every kind of object it is capable of creating. These methods optionally accept *parameters* defining how the object is created, and then return the created object.

Factory objects are used in situations where getting hold of an object of a particular kind is a more complex process than simply creating a new object. The factory object might decide to create the object's *class* (if applicable) dynamically, return it from an *object pool*, do complex configuration on the object, or other things.
What is a Design Pattern?

- "It is a description or template for how to solve a problem that can be used in many different situations. Patterns are formalized best practices that the programmer must implement themselves in the application" (Source: http://en.wikipedia.org/wiki/Software_design_pattern)