A COMMON INTEGRATED FRAMEWORK FOR HETEROGENEOUS MODELING SERVICES

ANASTASIA MAVRIDOU, TAMAS KECSKES, QISHEN ZHANG, AND JANOS SZTIPANOVITS

VANDERBILT UNIVERSITY, NASHVILLE, TN, USA
WHY?

- WebGME – easy visual modeling with added features and integration capabilities for a smooth user experience
- Formula – strong formal base and concise form for expressing constraints
- Gremlin – strong querying capabilities and optimized performance due to graph based data representation
Typed graph $T = \langle L, M, \tau_v, \tau_e \rangle$

- $L$ is a labeled graph that represents the domain with its structural semantics
- $M$ is a model graph that implements the actual model that has to follow the rules of $L$
- $\tau_v$ inheritance relationship among vertices of $M$ and $L$
- $\tau_e$ inheritance relationship among edges of $M$ and $L$

This type of graph description can be effectively modeled in all three representations
HOW? – FEATURES OF THE TYPED GRAPH SYSTEM

• Pros
  • Each representation understands this graph system
  • It is enough to define the well formedness rules once and then they can be applied to any DSML (not even tied to WebGME meta-modeling language)

• Cons
  • The system can be extended with special constraints if need be, though due to the lack of generic translation between Formula and Gremlin the rules need to be described in both in an equivalent way
  • Due to the nature of the typed graph system, the additional rules either has to be written on a domain agnostic abstraction level or they became cumbersome and complex
<table>
<thead>
<tr>
<th>Objects</th>
<th>WebGME meta</th>
<th>FORMULA translation</th>
<th>Gremlin translation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concept</td>
<td><a href="image1.png">Diagram</a></td>
<td><code>StateBase is MetaNode('StateBase');</code></td>
<td><code>StateBase = graph.addVertex('class', 'MetaNode', 'name', 'StateBase');</code></td>
</tr>
<tr>
<td>Containment</td>
<td><a href="image2.png">Diagram</a></td>
<td><code>metaContainment1 is MetaEdge('metaContainment1', ArchitectureStylesLibrary, ArchitectureStyle, exactlyOneMultiplicity, starMultiplicity);</code></td>
<td><code>metaContainment1 = graph.addVertex('class', 'MetaEdge', 'name', 'metaContainment1'); MContainment1.addEdge('src', ArchitectureStylesLibrary, 'min',1, 'max',1); MContainment1.addEdge('dst', ArchitectureStyle, 'min',0);</code></td>
</tr>
<tr>
<td>Attribute</td>
<td><a href="image3.png">Diagram</a></td>
<td><code>Guard_has_guardMethod is MetaEdge('guardMethod', Guard, String, exactlyOneMultiplicity, starMultiplicity);</code></td>
<td><code>Guard_has_guardMethod = graph.addVertex('class', 'MetaEdge', 'name', 'guardMethod'); Guard_has_guardMethod.addEdge('src', Guard, 'min',0); TransitionHasGuard.addEdge('dst', String, 'min',1, 'max',1);</code></td>
</tr>
<tr>
<td>Pointer (one to one association)</td>
<td><a href="image4.png">Diagram</a></td>
<td><code>Connection_point_src_ConnectorEnd is MetaEdge('src', Connection, ConnectorEnd, exactlyOneMultiplicity, exactlyOneMultiplicity);</code></td>
<td><code>Connection_point_src_ConnectorEnd = graph.addVertex('class', 'MetaEdge', 'name', 'src'); Connection_point_src_ConnectorEnd.addEdge('src', Connection, 'min',0); Connection_point_src_ConnectorEnd.addEdge('dst', ConnectorEnd, 'min',1);</code></td>
</tr>
<tr>
<td>Set (many to many association)</td>
<td><a href="image5.png">Diagram</a></td>
<td><code>ComponentType.collects_ComponentType is MetaEdge('associatedWith', ComponentType, ComponentType, exactlyOneMultiplicity, starMultiplicity);</code></td>
<td><code>ComponentType.collects_ComponentType = graph.addVertex('class', 'MetaEdge', 'name', 'associatedWith'); ComponentType.collects_ComponentType.addEdge('src', ComponentType, 'min',0); ComponentType.collects_ComponentType.addEdge('dst', ComponentType, 'min',0);</code></td>
</tr>
<tr>
<td>Inheritance (identical to Mixin)</td>
<td><a href="image6.png">Diagram</a></td>
<td><code>NodeInheritance(StateBase, State);</code></td>
<td><code>State.addEdge('type', StateBase);</code></td>
</tr>
</tbody>
</table>
The rules basically govern that every ‘model’ element should have a corresponding ‘language’ element and that every relationship in the ‘model’ has a definition that allows it. They also control that the number of relationships has to fit into the requirements of the cardinalities of the language definitions.
FUTURE

• Creating a bi-directional translation among Formula rules and Gremlin queries so that the user would only need to use the more concise Formula language

• Allow the use of a domain specific formula depiction which would let users create their constraints on a more manageable abstraction level (the one they defined for their domain instead of the common typed graph one)
THANK YOU!

- Questions?