Yet Another Three QVT Languages

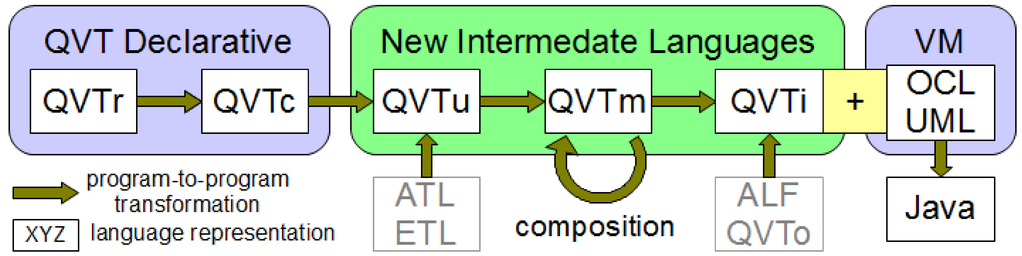
Edward Willink, Willink Transformations Ltd,
Horacio Hoyos, The University of York,
Dimitris Kolovos The University of York

(Old) QVT Languages and Tools

QVTo (Operational mappings)
- Eclipse QVT Operational (moving again)
- SmartQVT (mature)

QVTr (Relational)
- Eclipse QVT Declarative (editors only)
  - (execution next year, this poster)
- Medini QVT (mature, disappointing)
- ModelMorf (beta/proprietary)

QVTc Core)
- Eclipse QVT Declarative (editors only)
  - (execution next year, this poster)



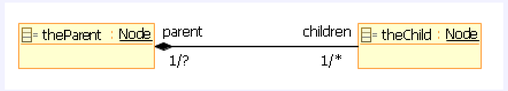
New QVTc subset Languages

QVTu (Unidirectional)
- Eliminates multi-directional bloat
- Interchange for declarative M2M languages

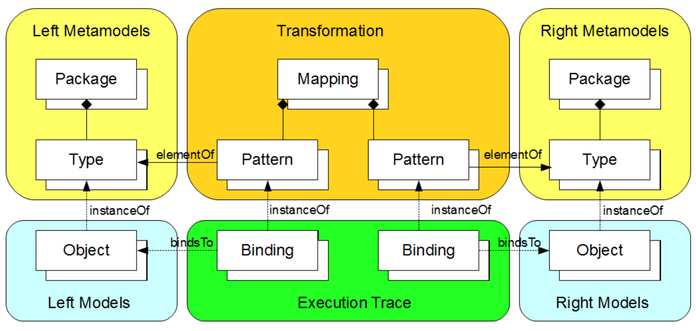
QVTm (Minimal)
- Normalized form, discards syntactic sugar
- Interchange for composition, optimization

QVTi (Imperative)
- Imperative semantics, practical schedule
- Interchange for imperative M2M languages

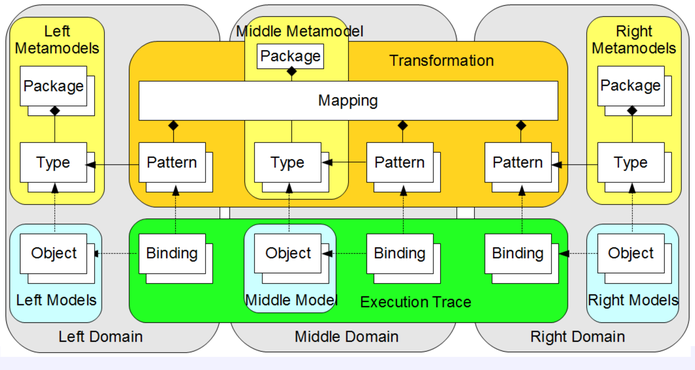
QVTc Principles



Model Transformation: recognize patterns on left hand side
create corresponding patterns on right hand side

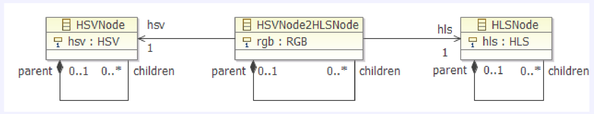


Typical Model Transformation Language:
patterns and bindings for left and right hand side
hidden traceability to support overlapping output patterns

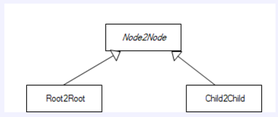


QVT Core Model Transformation Language:
patterns and bindings for left, middle and right hand side
middle model makes traceability explicit and customizable

QVTc and QVTi

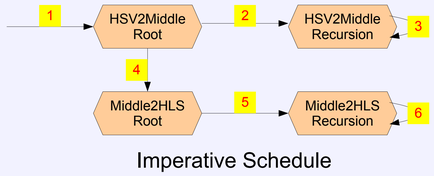


Left Metamodel Middle Metamodel Right Metamodel



Declarative, No Schedule

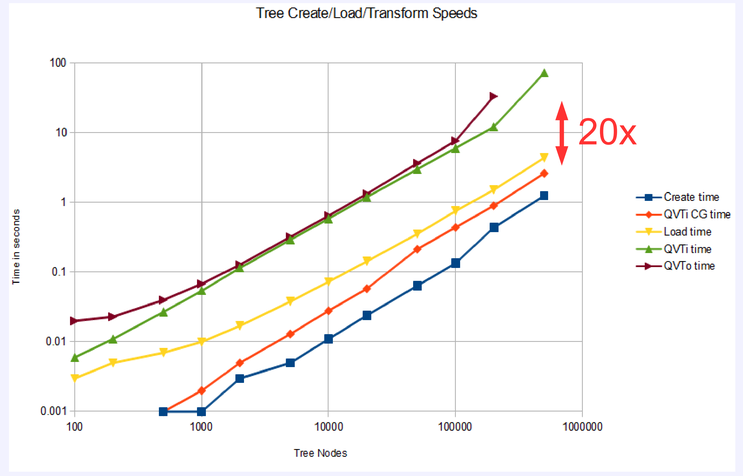
```
import 'HSV2HLS.ecore';          -- import the HSV2HLS package

transformation ColorChange {     -- declare the ColorChange transformation
    hsv imports HSVTree;         -- hsv TypedModel uses HSVTree package
    hls imports HLSTree;         -- hls TypedModel uses HLSTree package
    imports HSV2HLS;             -- middle TypedModel uses HSV2HLS package
}
                                 -- utility queries for color conversions
query ColorChange::hls2rgb(color : HLSTree::HLS) : HSV2HLS::RGB;
query ColorChange::hsv2rgb(color : HSVTree::HSV) : HSV2HLS::RGB;
query ColorChange::rgb2hls(color : HSV2HLS::RGB) : HLSTree::HLS;
query ColorChange::rgb2hsv(color : HSV2HLS::RGB) : HSVTree::HSV;

map Node2Node in ColorChange {      -- abstract mapping of a Node
    enforce hsv() { realize hsvNode : HSVNode | } -- hsv node exists
    enforce hls() { realize hlsNode : HLSNode | } -- hls node exists
    where() {
        realize middleNode : HSVNode2HLSNode |  -- middle node exists
        middleNode.hsv := hsvNode;              -- middle to hsv trace
        middleNode.hls := hlsNode;              -- middle to hls trace
        middleNode.rgb := hsv2rgb(hsvNode.hsv); -- hsv to middle color
        middleNode.rgb := hls2rgb(hlsNode.hls); -- hls to middle color
        hsvNode.hsv := rgb2hsv(middleNode.rgb); -- middle to hsv color
        hlsNode.hls := rgb2hls(middleNode.rgb); -- middle to hls color
    }
}

map Root2Root in ColorChange refines Node2Node { -- refined for a root
    enforce hsv() {                 -- hsv can be an output
        hsvNode.parent := null; }   -- parent must be null
    enforce hls() {                 -- hls can be an output
        hlsNode.parent := null; }   -- parent must be null
    where() {                       -- middle is always the middle
        middleNode.parent := null;  -- parent must be null
    }
}

map Child2Child in ColorChange refines Node2Node { -- refined for a child
    enforce hsv(hsvParent : HSVNode | ) {   -- additional parent variable
        hsvNode.parent := hsvParent; }      -- node's parent is parent
    enforce hls(hlsParent : HLSNode | ) {   -- additional parent variable
        hlsNode.parent := hlsParent; }      -- node's parent is parent
    where(middleParent : HSVNode2HLSNode | ) { -- additional parent var
        middleNode.parent := middleParent; } -- node's parent is parent
}
```



Imperative Schedule

```
import 'HSV2HLS.ecore';

transformation hsv2hls {
    hsv imports HSVTree;  -- Should specify the correct package
    hls imports HLSTree;  --
    imports HSV2HLS;
}
query hsv2hls::hls2rgb(color : HLSTree::HLS) : HSV2HLS::RGB;
query hsv2hls::hsv2rgb(color : HSVTree::HSV) : HSV2HLS::RGB;
query hsv2hls::rgb2hls(color : HSV2HLS::RGB) : HLSTree::HLS;
query hsv2hls::rgb2hsv(color : HSV2HLS::RGB) : HSVTree::HSV;

1 map HSV2MiddleRoot in hsv2hls { -- Mapping root nodes L to M
    hsv(hsvRoot : HSVNode | hsvRoot.parent = null;) { }
    where() {
        realize middleRoot : HSVNode2HLSNode |
        middleRoot.hsv := hsvRoot;
        middleRoot.rgb := hsv2rgb(hsvRoot.hsv);
    }
    map HSV2MiddleRecursion {       -- recursive call to visit children
2       hsvNode <= hsvRoot.children;
        middleParent := middleRoot;
    }
4   map Middle2HLSRoot {        -- invoke middle to output mapping
        middleNode := middleRoot;
    }
}

map HSV2MiddleRecursion in hsv2hls { -- Mapping child nodes L to M
    hsv(hsvNode : HSVNode | ) {}
    where(middleParent : HSVNode2HLSNode | ) {
        realize middleNode : HSVNode2HLSNode |
        middleNode.parent := middleParent;
        middleNode.hsv := hsvNode;
        middleNode.rgb := hsv2rgb(hsvNode.hsv);
    }
    map HSV2MiddleRecursion {       -- recursive call to visit children
3       hsvNode <= hsvNode.children;
        middleParent := middleNode;
    }
}

map Middle2HLSRoot in hsv2hls { -- Mapping root nodes M to R
    enforce hls() {
        realize hlsNode : HLSNode |
    }
    where(middleNode : HSVNode2HLSNode |) {
        hlsNode.parent := null;
        middleNode.hls := hlsNode;
        hlsNode.hls := rgb2hls(middleNode.rgb);
    }
    map Middle2HLSRecursion {       -- recursive call to visit children
5       middleNode <= middleNode.children;
    }
}

map Middle2HLSRecursion in hsv2hls { -- Mapping child nodes M to R
    enforce hls() {
        realize hlsNode : HLSNode |
    }
    where(middleNode : HSVNode2HLSNode |) {
        hlsNode.parent := middleNode.parent.hls;
        middleNode.hls := hlsNode;
        hlsNode.hls := rgb2hls(middleNode.rgb);
    }
6   map Middle2HLSRecursion {       -- recursive call to visit children
        middleNode <= middleNode.children;
    }
}
```

QVTi (CG) Performance



"Create Time" = time to create N nodes in memory.
"Load time" = time to load N nodes from XMI on disk to memory.
"QVTo time" = time for QVTo to transform N nodes in memory
"QVTi time" = time for interpreted QVTi to transform N nodes in memory
"QVTi CG time" = time for compiled QVTi to transform N nodes in memory

Not shown since they are almost identical to QVTi CG time:
"Copy time" = time for EcoreUtil.copyAll to copy N nodes in memory
"Save time" = time to save N nodes from memory to XMI on disk

QVTo fails for 500,000 nodes on a default VM.
Copy and QVTi fail for 1,000,000 nodes on a default VM.

QVTi (interpreted) similar in speed to QVTo (interpreted)

QVTi (code generated) 20 times faster

WILLINK transformations

THE UNIVERSITY of York

EPSRC Pioneering research and skills