Introduction

This document specifies the metamodel for IFML Diagram Interchange (IFML DI). The IFML DI is meant to facilitate interchange of IFML diagrams between tools rather than being used for internal diagram representation by the tools.

The IFML DI metamodel is defined as a MOF-based metamodel. The IFML DI classes only define the visual properties used for depiction. All other properties that are required for the unambiguous depiction of IFML diagram elements are derived from the referenced IFML model elements.

Multiple depictions of a specific IFML Element in a single diagram are not allowed.

Architecture

The IFML language specification provides three normative artifacts at M2 (shown with shaded boxes in Figure 1): the abstract syntax model (IFML), the IFML diagram interchange model (IFML DI), and the mapping specification between the IFML DI and the graphics model (IFML Mapping Specification).

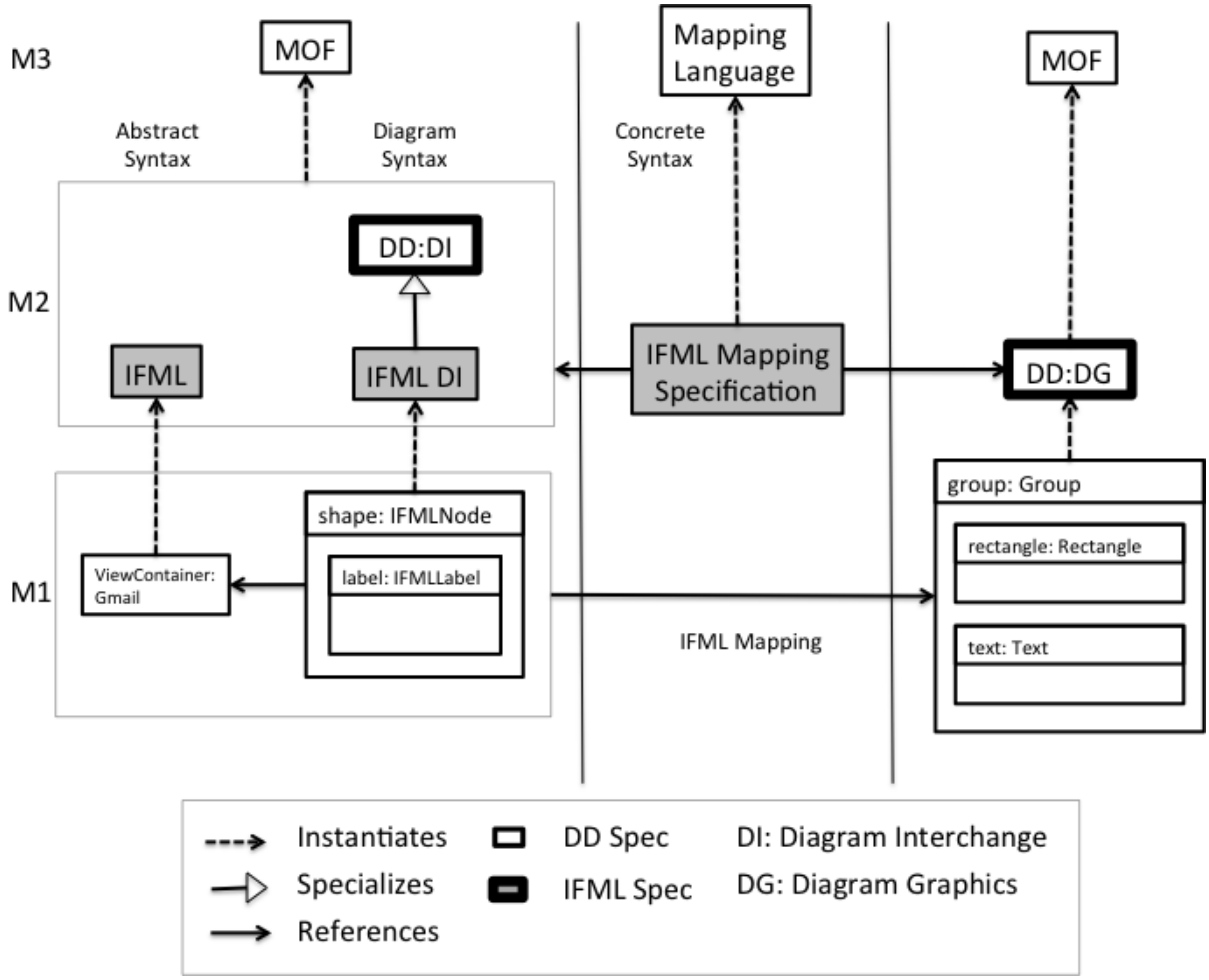

Figure 1: Diagram Definition Architecture for IFML

At M1 (left), Figure 1 shows an instance of IFML::Core::ViewContainer as a model element. Next to it, on the right, the figure shows an instance of IFMLDI::IFMLNode referencing the ViewContainer element, indicating that the ViewContainer is depicted as a node on the diagram.

The node also contains an instance of IFMLDI::IFMLLabel representing the textual label of the ViewContainer on the diagram. On the right of M1, the figure shows an instance of DG::Group containing instances of DG::Rectangle and DG::Text

IFML DI specializes DD:DI, which specifies the graphics the user has control over, such as the position of nodes and line routing points. This information is what is captured for interchange between tools.

DD:DG represents the graphics that the user has no control over, such as shape and line styles, because they are the same in all languages conforming to the DD specification. DD:DG is derived by executing the mapping specification, in the middle, between IFML DI and DG.

IFML Diagram Interchange (DI) Meta-model

The IFML DI metamodel extends the DI metamodel, where appropriate. The class IFMLDiagram represents the diagram, which composes IFMLDiagramElements. An IFMLDiagram is an IFMLNode because it may be rendered as a figure and be connected to other figures. IFMLDiagramElements optionally reference elements of an IFML model, the latter denoted by the IFML:Core:Element class. IFMLDiagramElements that do not reference elements of an IFML model are purely notational diagram elements such as notes and the link that connects the note with the model element. IFMLDiagramElements may also be styled with instances of class IFMLStyle (e.g. font type and size).

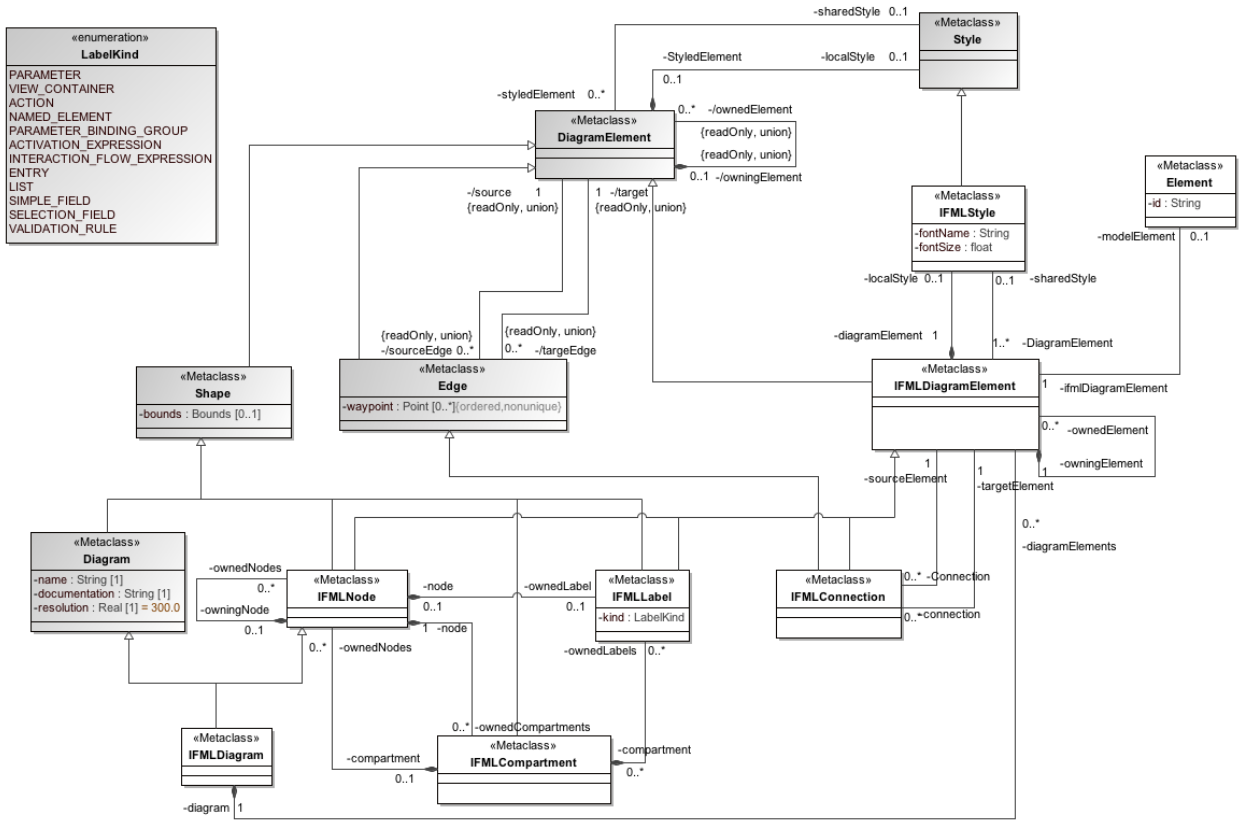

Figure 2: IFML Diagram Interchange (DI) Meta-model

Classes are defined for interchanging shapes and edges of the interaction flow diagram and the content diagram, based on the following notational patterns (see Figure 3):

- Pattern (a): A shape that has a label and an optional list of compartments, each of which having an optional list of labels or other shapes (e.g., the ViewContainer box, ViewComponentPart box, Form ViewComponent rounded box or the classes of the

ContentModel).

- Pattern (b): A shape that has a label only (e.g., the Event ball or Action hexagon notation)
- Pattern (c): An edge that may be dashed or solid (e.g., NavigationFlows and DataFlows)

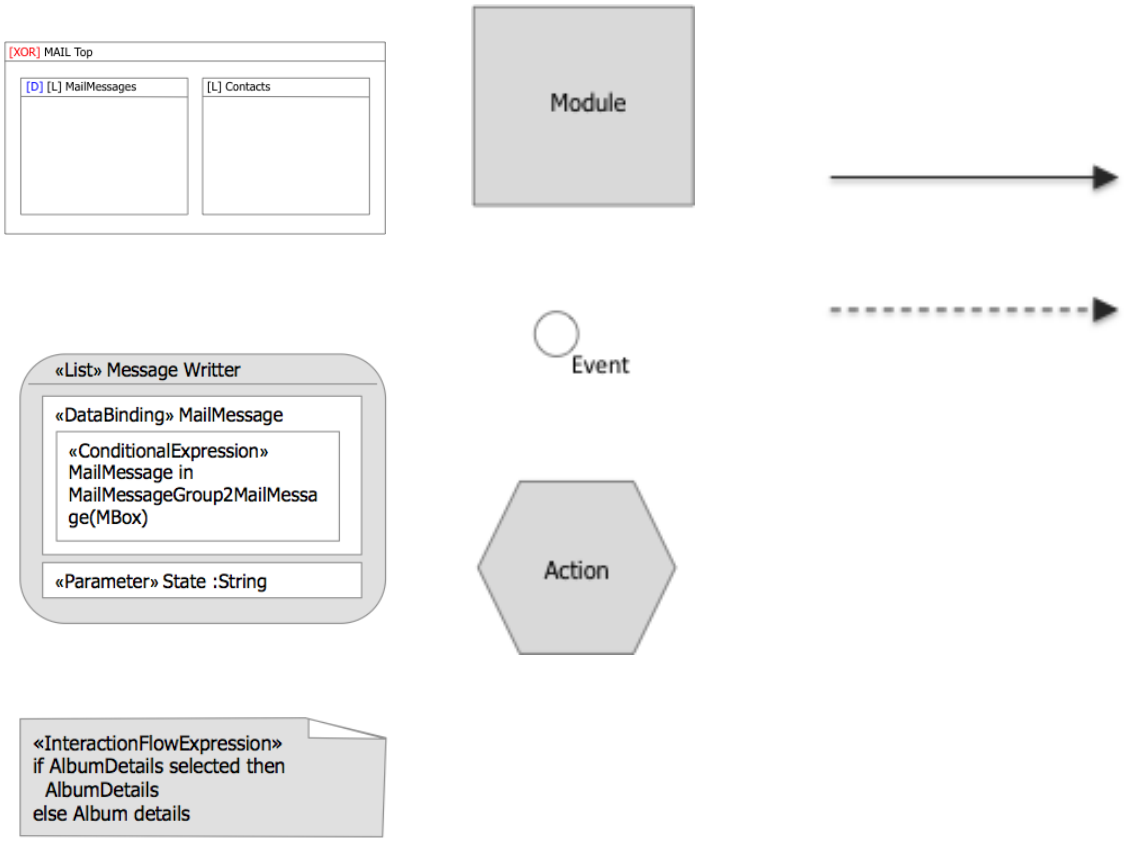

Figure 3: Notational patterns

Based on these patterns, three shape classes (IFMLNode, IFMLLabel and IFMLCompartment) and one edge class (IFMLConnection) are defined and related to realize the patterns. These classes (except IFMLCompartment) are subclasses of IFMLDiagramElement to allow them to be styled independently and to reference their own IFML Element.

Some classes have properties to disambiguate the notation and a corresponding enumeration. For instance labels may be of different kinds such as Parameter, ViewContainer, etc., which will determine how the text decoration will be rendered.

IFML DI to DG Mapping Specification

The DD architecture expects language specifications to define mappings between interchanged and non-interchanged graphical information, but does not restrict how it is done. The IFML DI to DG mapping is accomplished in this specification by means of the following QVT mapping.

```
1  transformation IFMLDItoDG(in ifmldi: IFMLDI, in ifml: IFML, out DG)
2
3  main() {
4    ifmldi.objectsOfType(IFMLDiagram)->map toGraphics();
5  }
```

```
6
7   mapping IFMLDiagram::toGraphics(): Canvas {
8     member += self.diagramElements->map toGraphics();
9   }
10
11  mapping IFMLDiagramElement::toGraphics(): Group {
12    localStyle  := copyStyle(self.localStyle);
13    sharedStyle := copyStyle(self.sharedStyle);
14  }
15
16  mapping IFMLNode::toGraphics(): Group inherits IFMLDiagramElement::toGraphics() {
17    member += self.modelElement.map toGraphics(self);
18    member += self.ownedCompartments->map toGraphics();
19    member += self.ownedLabel.map toGraphics();
20  }
21
22  mapping IFMLLabel::toGraphics(): Text inherits IFMLDiagramElement::toGraphics() {
23    var e := self.modelElement;
24    bounds := self.bounds;
25    data := switch {
26      case (self.kind = LabelKind::NAMED_ELEMENT)
27        e.name;
28      case (self.kind = LabelKind::VIEW_CONTAINER)
29        e.oclAsType(ViewContainer).getLabelText();
30      case (self.kind = LabelKind::ACTION)
31        e.oclAsType(Action).getLabelText();
32      case (self.kind = LabelKind::PARAMETER)
33        "«Parameter» " + e.name + ": " + e.type.name;
34      case (self.kind = LabelKind::ENTRY)
35        "«Entry» " + e.name;
36
37      case (self.kind = LabelKind::LIST)
38        "«List» " + e.name;
39
40      case (self.kind = LabelKind::SIMPLE_FIELD)
41        "«SimpleField» " + e.name;
42      case (self.kind = LabelKind::SELECTION_FIELD)
43        "«SelectionField» " + e.name;
44      case (self.kind = LabelKind::PARAMETER_BINDING_GROUP)
45        "«ParameterBindingGroup»";
46      case (self.kind = LabelKind::ACTIVATION_EXPRESSION)
47        "«ActivationExpression»";
48      case (self.kind = LabelKind::INTERACTION_FLOW_EXPRESSION)
49        "«InteractionFlowExpression»";
50      case (self.kind = LabelKind::VALIDATION_RULE)
51        "«ValidationRule»";
52      default
53        "";
54    };
55  }
56
57  query ViewContainer::getLabelText(): String {
58
59    var text += if self.isXOR then "[XOR] " endif;
60
61    text += if self.isLandmark then "[L] " endif;
62
63    text += if self.isDefault then "[D] " endif;
64
65    return text + self.name;
66
67  }
68
69  query Window::getLabelText(): String {
70
71    var text := if self.isNewWindow and self.isModal then "[Modal] " endif;
72
73    text += if self.isNewWindow and not self.isModal then "[Modeless] " endif;
74
75    text += if self.isLandmark then "[L] " endif;
76    text += if self.isDefault then "[D] " endif;
77
78    return text + self.name;
79  }
80  query Action::getLabelText(): String {
81
82    var text := if self.isClientSide then "[ClientSide]\n" endif;
83
84    return text + self.name;
85
86  }
87
88
89  mapping Element::toGraphics(n: IFMLNode): GraphicalElement
90    disjuncts ViewContainer::toRectangle, ViewComponent::toRectangle,
91              Module::toRectangle, ViewComponentPart::toRectangle, Event::toCircle,
92              Action::toPolygon, ViewPoint::toPolygon {
93  }
94
95  mapping ViewContainer::toRectangle(n: IFMLNode): Rectangle {
    bounds := n.bounds;
  }

  mapping ViewComponent::toRectangle(n: IFMLNode): Rectangle {
```

```
96    bounds := n.bounds;
97    cornerRadius := 15;
98  }
99
100 mapping Module::toRectangle(n: IFMLNode): Rectangle {
101   bounds := n.bounds;
102 }
103
104 mapping ViewComponentPart::toRectangle(n: IFMLNode): Rectangle {
105   bounds := n.bounds;
106 }
107
108 mapping Event::toCircle(n: IFMLNode): Circle {
109   var b := n.bounds;
110   center := object Point{b.x + b.width / 2; b.y + b.height / 2};
111   radius := if b.width < b.height then
112               b.width / 2
113             else
114               b.height / 2
115             endif;
116 }
117
118 mapping Action::toPolygon(n: IFMLNode): Polygon {
119   var b := n.bounds;
120   point += object Point {b.width * (1/4); y := 0};
121   point += object Point {b.width * (3/4); y := 0};
122   point += object Point {b.width; b.height * (1/4)};
123   point += object Point {b.width; b.height * (3/4)};
124   point += object Point {b.width * (3/4); b.height};
125   point += object Point {b.width * (1/4); b.height};
126   point += object Point {0; b.height * (3/4)};
127   point += object Point {0; b.height * (1/4)};
128 }
129
130 mapping ViewPoint::toPolygon(n: IFMLNode): Polygon {
131   var b := n.bounds;
132   point += object Point {b.width * (1/2); y := 0};
133   point += object Point {b.width; b.height};
134   point += object Point {0; b.height};
135 }
136
137 mapping ParameterBindingGroup::toPolygon(n: IFMLNode): Polygon {
138   var b := n.bounds;
139   point += object Point {x:=0,y:=0};
140   point += object Point {b.width*3/4,y:=0};
141   point += object Point {b.width,b.height};
142   point += object Point {b.width*1/4,b.height};
143 }
144
145 mapping IFMLCompartment::toGraphics(): Group {
146   member += object Rectangle {bounds:= self.bounds};
147   member += self.ownedNodes.map toGraphics();
148   member += self.ownedLabels.map toGraphics();
149 }
150
151 mapping IFMLConnection::toGraphics(): Group inherits IFMLDiagramElement::toGraphics()
152 {
153   member += self.modelElement.map toGraphics(self);
154 }
155
156 mapping Element::toGraphics(c: IFMLConnection): GraphicalElement
157   disjuncts NavigationFlow::toPolyline, DataFlow::toPolyline {
158 }
159
160
161 mapping NavigationFlow::toPolyline(c: IFMLConnection): Polyline {
162   point := c.waypoint;
163   sharedStyle := solidStyleProp;
164   endMarker := arrowMarkerProp;
165 }
166
167 property solidStyleProp = object DG::Style {
168   strokeDashLength := Sequence{};
169 }
170
171 property arrowMarkerProp = object Marker {
172   size := object Dimension {width := 2; height := 2};
173   reference := object Point {x := 2; y := 1};
174   member += object Polygon {
175     point += object Point {x := 0; y := 0};
176     point += object Point {x := 2; y := 1};
177     point += object Point {x := 0; y := 2};
178   }
179 }
180
181 mapping DataFlow::toPolyline(c: IFMLConnection): Polyline {
182   point := c.waypoint;
183   sharedStyle := dashedStyleProp;
184   endMarker := arrowMarkerProp;
185 }
```

```
property dashedStyleProp = object DG::Style {
  strokeDashLength := Sequence{2, 2};
}

helper copyStyle(s: IFMLStyle): DG::Style {
  fontName := s.fontName;
  fontSize := s.fontSize;
  fillColor := s.fillColor;
}
```