Abstract: It has not been studied about modeling methods for SVG documents to represent animation in the web that has been recently increased in interest. In this paper, I propose component modeling for SVG documents by CBD methodology. First, I propose conceptual modeling by UML class diagrams for converting SVG document into component diagram. And then, I propose rules to convert the UML class diagram into component diagram. Thus, main contribution of this paper is that it can generate a component diagram for a SVG document using Component-Based Development methodology.

Keywords: component diagram, class diagram, SVG animation, UML, CBD

1. INTRODUCTION

SVG (Scalable Vector Graphics) a language for describing two-dimensional graphics in XML. SVG allows three types of graphic objects: vector graphic shapes (e.g., paths consisting of straight lines and curves), images and text. Graphical objects can be grouped, styled, transformed and composited into previously rendered objects. Text can be in any XML namespace suitable to the application, which enhances search ability and accessibility of the SVG graphics. The feature set includes nested transformations, clipping paths, alpha masks, filter effects, template objects and extensibility. SVG drawings can be dynamic and interactive.

SVG is a host language in terms of SMIL (Synchronize Multimedia Integration Language) Animation and especially introduces additional constraints and features as permitted by that specification. Except for any SVG-specific rules explicitly mentioned in this specification the normative definition for SVG's animation elements and attributes is the SMIL animation specification. The normative definition for SVG's animation elements and attributes is the animation specification.

SVG supports the following four animation elements (animate, set, animateMotion, animateColor) which are defined in the SMIL Animation specification.

Every day more organizations are being embraced CBD (component-based development) for its promise of code reusability and, as a result, reduced development effort and faster time-to-market.

CBD promises a more efficient way to develop applications with commercial, vendor-provided or internally developed objects. Using a visual development environment to test and assemble these objects significantly enhances productivity for all the members of the development team.

In this paper, we propose CBD method to figure component specification for SVG animation. It includes business type modeling, interface responsibility modeling, interface type modeling and component architecture modeling.

2. RELATED WORKS

DOM (Document Object Model) assigns various XML objects to variables. The variables can thus be accessed and manipulated by any number of applications. It is XML's tree structure that enables you to quickly retrieve the information or XML component you need. But it doesn't represents information about attribute and aggregation relationship. Besides that, there are XOMT diagram and UML class diagram in the area of DTD modeling method. XOMT, on the basis of OMT object-oriented access, has too many parts that cannot be expressed by OMT itself, so extends OMT needs an extension. This requires that you learn another notation, and you will be not mapping class or the generalization concept used in the object model to semantic DTD. The modeling method using UML class diagram propose mapping rules from XML Document with links UML class diagram. It describes general mapping rules for XML DTD and Document Instance. It is not mapped to elements and documents for a special purpose. In case of RDF (Resource Description Framework), it doesn't map every element to class. According to kinds of RDF resource, it maps to relationships. In case of SMIL, it doesn't map synchronized tag to class. It represents synchronization through message between classes.

But the effect of SVG animations is variable according to time. We can not represent them using static class diagram. It cab be used class diagram in step of conceptual modeling and component diagram for dynamic modeling of animation.

3. SVG Animation and UML

3.1 SVG Animation

3.1.1 Animation element

SVG is a host language in terms of SMIL Animation and therefore introduces additional constraints and features as permitted by that specification. SVG supports the following four animation elements which are defined in the SMIL Animation specification.

(1) animate
It allows scalar attributes and properties to be assigned different values over time.

(2) set
A convenient shorthand for 'animate', which is useful for assigning animation values to non-numeric attributes and properties, such as the 'visibility' property.

(3) animateMotion
It moves an element along a motion path.

(4) animateColor
It modifies the color value of particular attributes or properties over time.

(5) animateTransform
It modifies one of SVG's transformation attributes over time, such as the 'transform' attribute.

3.1.2 Extended attribute

Additionally, SVG includes the following compatible extensions to SMIL Animation.
(1) path attribute
SVG allows any feature from SVG’s path data syntax to be specified in a path attribute to the animationMotion element.

<table>
<thead>
<tr>
<th>Name</th>
<th>Command</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>moveto</td>
<td>M or m</td>
<td>(x y)+</td>
<td>Start a new sub-path at the given(x,y) coordinate.</td>
</tr>
<tr>
<td>closepath</td>
<td>Z or z</td>
<td>none</td>
<td>Close the current subpath by drawing a straight line from the current point to current subpath’s initial point.</td>
</tr>
<tr>
<td>lineto</td>
<td>L or l</td>
<td>(x y)+</td>
<td>Draw a line from the current point to the given (x,y) coordinate which becomes the new current point.</td>
</tr>
<tr>
<td>H or h</td>
<td></td>
<td>x+</td>
<td>Draw a horizontal line from the current point(cpx, cpy) to (x, cpy).</td>
</tr>
<tr>
<td>V or v</td>
<td></td>
<td>y+</td>
<td>Draw a vertical line from the current point (cpx, cpy) to (cpx, y).</td>
</tr>
<tr>
<td>curve</td>
<td>C or c</td>
<td>(x1 y1 x2 y2 x y)+</td>
<td>Draw a cubic Bezier curve from the current point to (x,y) using (x1,y1) as the control point at the beginning of the curve and (x2, y2) as the control point at the end of the curve.</td>
</tr>
<tr>
<td></td>
<td>Q or q</td>
<td>(x1 y1 x y y)+</td>
<td>Draw a quadratic Bezier curve from the current point to (x,y) using (x1,y1) as the control point.</td>
</tr>
</tbody>
</table>

(2) keypoints attribute
SVG adds a keyPoints attribute to the animateMotion to provide precise control of velocity of motion path animations.

(3) rotate attribute
SVG adds a rotate attribute to the animateMotion to control whether an object is automatically rotated so that its X-axis points in the same direction (or opposite direction) as the directional tangent vector of the motion path.

3.2 UML class diagram

3.2.1 class and stereotype
A class is an abstraction of the common properties from a set containing many similar objects. UML classes are being shown to use rectangles with the name of the class inside the rectangle. A variation uses a three-segment box; the top segment has the name of the class, the middle segment contains a list of attributes, and the bottom segment contains a list of operations.

A stereotype is the class of an entity in the UML metamodel. The UML metamodel is the model of UML itself, expresses in UML. Stereotypes provide an important extension mechanism to UML, allowing users to extend the modeling language to better address their needs. The usual notation for stereotypes is to enclose the stereotype in guillemets preceding the name of the entity; for example "(type) stack", which is the name of a class providing a stack interface for another implementation.

3.2.2 association and composition
Associations are logically bidirectional unless explicitly constrained. A binary association is drawn as a solid path connecting two classifier symbols. An aggregation is a special type of association that implies logical or physical containment. Composition is a strong form of aggregation. Composition means that part objects are solely the responsibility of the composite class. Composition is shown by graphical inclusions of the components within the composite or with a filled in aggregation diamond.

3.3 UML component diagram
The component diagram's main purpose is to show the structural relationships between the components of a system.

3.3.1 component and Interface
A component represents a modular, deployable, and replaceable part of a system that encapsulates implementation and exposes a set of interfaces. A component is drawn as a rectangle with two smaller rectangles protruding from its left side.

An interface is a specifier for the externally-visible operations of a class, component, or other classifier (including subsystems) without specification of internal structure. An interface may also be displayed as a small circle with the name of the interface placed below the symbol. The circle may be attached by a solid line to classifiers that support it.

3.3.1 dependency and realization
A dependency indicates a semantic relationship between two model elements. It is shown as a dashed arrow between two model elements. A realization is a kind of dependency relationship. It connects model element like interface to each component. An interface supply behavioral specification not structure and implementation.

4. SYSTEM

4.1 System architecture
The following figure is a system architecture that generates a component Diagram from SVG Document.

(1) Parser
XML parser checks syntax error and generates from SVG Document to DOM tree format.

(2) Business Type Modeller
It extracts business objects and defines attributes. And it represents composition relation with multiplicity.

We call ‘(core)’ business object in case of type to have business information that is continuously tracking and has independent identifier.

It needs to separate core business type and represents composition relations with multiplicity.
(3) Interface Responsibility Modeller
   It allocates one interface to core business object. And it is marked ‘1..n’ multiplicity at composition relation. It has association relations using stereotype for animation tag.

(4) Interface Type Modeller
   Interface type to manage core business object is inserted automatically get(), set() operations. In case of interface type for animation, it is inserted SVG attributes that has been transferred into operations.

(5) Component Architecture Modeller
   It generates each component for each core business object. The component has '《componentSpecification》' stereotype. And interface type has other notation and can be attached to component.

4.1.1 Generating of Business Type Model
   Parsed SVG document generates class diagram by business type modeller. The rules are following.
   [Rule 1] Element name to be start tag is class name and attribute is private attribute.
   [Rule 2] It is attached ‘《core》’ stereotype to SVG, path, text and basic shapes element. SVG class become class.
   [Rule 3] There is a composition relation between SVG class and another class.
   [Rule 4] When it occurs one more same classes, it represents only one class with occurrence times.

   The following algorithm is that it generates from SVG Document to business type model using mapping rules

   for ( Number of start tags )
   {
     if (Path or Text or Basic shape)
       then
         if (The ‘《core》’ class)
           update multiplicity
         else
           generate ‘《core》’ class
           for (Number of attribute list)
             insert attributes and values
             Composition relation
         }
     else
       make class
       Composition relation
   }

4.1.2 Generating of Interface Responsibility Model
   It converts UML class diagram into interface responsibility model by algorithm for generating business type model
   [Rule 5] It generates interface manager class of composition relation with 1..n multiplicity for ‘《core》’ class. Interface manager class has ’《interface Type》’ stereotype.
   [Rule 6] Animation elements is to be class with ‘《interface Type》’ stereotype and has association relations.

   The following algorithm is that it converts business type model to interface type model by using above rules.

   Input: Business type model, SVG Document
   Output: Interface Responsibility Model(UML class diagram)

   begin
   {
     for (number of ‘《core》’ classes
       { // from Business type model
         generate ‘《interface Type》’ class
         make composition relations
         represent multiplicity
       }
     for (animation element tags)
       { // from SVG document
         generate ‘《interface Type》’ class
         make association relations
         represent multiplicity
       }
   }

   It generates UML class diagram composed of classes and relations that is extracted from SVG document. It inserts attribute and value to class and multiplicity to composition.

4.1.3 Generating of Interface Type Model
   It converts UML class diagram into interface responsibility model by algorithm for generating business type model
   [Rule 5] It generates interface manager class of composition relation with 1..n multiplicity for ‘《core》’ class. Interface manager class has ’《interface Type》’ stereotype.
   [Rule 6] Animation elements is to be class with ‘《interface Type》’ stereotype and has association relations.

   The following algorithm is that it generates from SVG Document to business type model using mapping rules

   Input: SVG Document
   Output: Business Type Model

   begin
   {
     Make a root class ‘《core》’
     // Make a SVG class
     for ( Number of attribute list )
       insert attributes and values

   The following algorithm is that it generates interface responsibility model from business type model. It generates interface manager classes and attaches association relations.
The following table 2 is summarized items in UML class diagram that is mapped to items in SVG Document.

<table>
<thead>
<tr>
<th>SVG document items</th>
<th>kinds of element tag</th>
<th>UML class diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td>element</td>
<td>svg, path, text, basic shape element</td>
<td>class</td>
</tr>
<tr>
<td>animation</td>
<td>animation</td>
<td>public attribute</td>
</tr>
<tr>
<td>attribute</td>
<td>svg, path, text, basic shape element</td>
<td>private attribute</td>
</tr>
<tr>
<td>relation with element tags</td>
<td>animation</td>
<td>association relation</td>
</tr>
</tbody>
</table>

### 4.1.3 Generating of Interface Type Model

The following rules are inserted operations for ‘〈interface Type〉’ class in interface responsibility model.

- **Rule 7** Interface manager class generates get(), set() public operation.
- **Rule 8** It maps attributes of animation element class into operations.

The following algorithm is to insert operations to interface responsibility model.

```
begin
  for (〈interface Type〉 class if (Interface manager)
    insert get(), set() operations
  else
    for(number of attributes)
      map to operation
  }
end;
```

We complete interface responsibility model to insert operation into class in class diagram.

### 4.1.4 Generating of Component Architecture model

It converts interface responsibility model that is UML class diagram into component diagram by using following rules.

- **Rule 9** Class with ‘〈core〉’ stereotype is to be a component with ‘〈componentSpecification〉’ stereotype.
- **Rule 10** Class with ‘〈interface Type〉’ stereotype is to be interface.
- **Rule 11** It is dependency relations between is ‘〈componentSpecification〉’ component. And it is realization relations between component and interface.

The following algorithm generates component diagram by using above rules.

```
begin
  for (〈interface Type〉 class
    generate 〈componentSpecification〉 component
    for (〈interface Type〉 class)
      generate an interface
    for (relations with 〈core〉 classes)
      make dependency
    for (relations with 〈core〉 class and 〈interface Type〉 class)
      make a realization relation
  }
end;
```

In the Table 4, we shall describe mapping relations between class diagram and component diagram.

### 4.2 Results

It generates a component diagram from a SVG documents.

#### 4.2.1 SVG Document

It is an instance of SVG document that is input for this system.

```xml
<?xml version="1.0" standalone="no"?>
<svg width="5cm" height="3cm" viewBox="0 0 500 300">
  <desc>Example animMotion01 - demonstrate motion animation computations</desc>
  <!--Draw the outline of the motion path in blue, along with three small circles at the start, middle and end.-->
  <path d="M100,250 C 100,50 400,50 400,250">
```

A component represents a modular, deployable, and replaceable part of a system that encapsulates implementation and exposes a set of interfaces. A component is drawn as a rectangle with two smaller rectangles protruding from its left side.

### 4.2 Results

It generates a component diagram from a SVG documents.
4.2.2 UML Class Diagram
We generates from SVG document to UML class diagram and component diagram.

(1) Business Type model
It generates UML class diagram that is identified core business class defined relationships from SVG document.

(2) Interface Responsibility model
It inserts interface type class to business type model.

(3) Interface Type Model

4.2.3 UML Component Diagram
It is component diagram mapped to interface responsibility.

4.4 Comparison
The following table is comparison of modeling rules with XML application.

XOMT proposes new notations extended OMT. [5,6,7] Modeling interprets XML document structure using static diagram. It accord object-oriented concept (Inheritance etc) [8, 9] modeling uses synchronized modeling for multimedia document. It is focused on time. This study uses component and interface for modelling web animation.

Main contribution of most study is that it maps XML application to an UML class diagram. But the UML class diagram is focused on static structure for representing XML documents. We can not describe dynamic animations of SVG document using class diagram. This study separates static shape elements and dynamic animation elements. And it uses interface class for supporting animation.

It uses component based development method for generating component diagram. There are 4 steps : the generation of business type model, the generation of interface
Table 3 Comparison with modeling of XML applications

<table>
<thead>
<tr>
<th>Domain</th>
<th>XOMT [5, 6, 7] Modeling</th>
<th>[8, 9] Modeling</th>
<th>This study</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SGML/DTD</td>
<td>SGML/XML DTD, document instance (included RDF, WIDL, CDF, OSD)</td>
<td>SMIL document</td>
</tr>
<tr>
<td>Approach</td>
<td>Object-Oriented</td>
<td>Object-Oriented</td>
<td>Object-Oriented</td>
</tr>
<tr>
<td>Based modeling language</td>
<td>OMT</td>
<td>UML</td>
<td>UML</td>
</tr>
<tr>
<td>Diagram</td>
<td>class, component</td>
<td>class, component</td>
<td>class, component</td>
</tr>
<tr>
<td>Interface</td>
<td>none</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>Tool</td>
<td>XOMT Editor</td>
<td>Rational Rose, Plastic etc</td>
<td>Rational Rose, Plastic etc</td>
</tr>
</tbody>
</table>

5. Conclusion and future works

This paper proposes an component based development for web animation of SMIL, SVG animation. We need new method to support both static elements and dynamic elements. Main contribution of this paper is that it can make it easier to map SVG component to the object-oriented database scheme due to CBD methods. The CBD tools - Rational Rose or Together etc. - which supports various diagram including component diagrams and makes all kinds of database codes and object-oriented codes.

Research keeps studying how to integrate various XML application for web animation into component diagram and to manage using CBD methods.

REFERENCES