САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Математико-механический факультет

Кафедра Системного Программирования

Самойлов Антон Сергеевич

Реализация поддержки SVG в фреймворке JetBrains JetPad

Курсовая работа

Научный руководитель:
Соломатов К. В.

Санкт-Петербург
2014

Оглавление

Введение **3**

1. Фреймворк JetBrains JetPad **4**

2. Постановка задачи **5**

3. Описание объектной модели **6**

3.1. Структура классов 6

3.2. Работа с атрибутами 8

3.3. Обработка пользовательских событий 9

3.4. Отображение в модели представления 11

Заключение **13**

Введение

Во время работы с данными возникает необходимость отображения данных в точной графической форме. Для изображения графиков и диаграмм лучше всего подходит формат векторной графики, который хранит геометрические представления объектов, а не их попиксельное представление (растр). Векторная графика позволяет отображать данные с геометрической точностью, и, в отличии от растровых изображений, не теряет качество при масштабировании и геометрических преобразованиях. Самым распространенным и поддерживаемым форматом векторной графики является Scalable Vector Graphics (SVG) [6].

Формат SVG позволяет отображать на экране различные геометрические элементы: текст, геометрические примитивы (прямоугольник, эллипс и т.п.), фигуры со сложной составной границей, градиенты; производить геометрические трансформации изображения, обрабатывать пользовательские события мыши. Он является реализацией стандартов XML [4], DOM [3], поддерживает кастомизацию стилей с помощью каскадных таблиц стилей (Cascade Style Sheets, CSS) [5].

Для отображения сложных форматов данных в рамках проекта JetBrains JetPad[11] возникла необходимость реализации поддержки формата SVG с последующей интеграцией в существующую систему классов фреймворка. Данная курсовая работа посвящена реализации этой поддержки.

1. Фреймворк JetBrains JetPad

JetBrains JetPad — фреймворк для создания проекционных и диаграммных редакторов, поддерживающий несколько графических платформ: Java Swing и HTML/Web. Отображениие в HTML происходит с помощью технологии Google Web Toolkit (GWT) [8], которая позволяет транслировать программный код Java в язык JavaScript и запускать его в браузере клиента. В будущем возможно добавление поддержки дополнительных платформ.

Фреймворк создан для облегчения написания программного кода, использующего паттерн проектирования (design pattern) Model-View-Controller [7], и использует парадигму объектно-ориентированного реактивного программирования (Object-Oriented Reactive Programming) [10].

Определим некоторые понятия, используемые во фреймворке JetPad.

Properties — это реактивные сущности, реализованные с помощью механизма Java generics. Property реализует шаблон Wrapper [7], который "оборачивает" объект произвольного класса, предоставляя свои методы для доступа к этому объекту. Это делается для того, чтобы изменение значения одной Property распространялось на все зависимые от нее сущности. Это позволяет пользователям фреймворка изолировать логику модели от логики обновления зависимостей и избавляет от необходимости самостоятельно реализовывать механизмы синхронизации значений.

```
interface Property<ValueT> {
  ValueT get();
  void set(ValueT value);
  Registration addHandler(EventHandler handler);
}
```

Listing 1: Интерфейс Property

Observable Collections — набор коллекций которые поддерживают инкрементальные изменения. Инкрементальность означает, что при изменении коллекции зависимые сущности не перестраиваются полностью, а обновляют только соответствующую часть. Пользователь может назначать произвольные обработчки на события изменения структуры коллекции, такие как добавление или удаление элементов.

JetBrains JetPad имеет большое количество реализованной функциональности, которая используется для нужд внутренних проектов. В частности иерархия классов-наследников View позволяет управлять относительным расположением элементов на странице (layout), и автоматически пересчитывать их координаты и размеры при изменении геометрических параметров других элементов.

2. Постановка задачи

В рамках данной работы были поставлены следующие задачи:

- Разработать модуль для работы с объектной моделью векторной графики в формате SVG.
- Реализовать набор элементов иерархии SVG, достаточный для решения задач отображения внутренних форматов хранения данных в виде интерактивных диаграмм и графиков.
- Предоставить возможность работы со списком потомков конкретного SVG элемента как с Observable Collection.
- Предоставить возможность работы с атрибутами элементов SVG, в том числе еще не инициализированными, как с Properties. Для описанных в спецификации атрибутов предоставить интерфейс типизированного доступа.
- Разработать модули отображения объектной модели SVG в различные графические системы: Java Swing и HTML/GWT.
- Предоставить возможность пользователям фреймворка назначать произвольные обработчики для событий мыши и событий изменения атрибутов.
- Осуществить интеграцию модуля с существующими классами фреймворка. В частности необходимо реализовать класс-коннектор для возможности использования объектов SVG в одном layout с другими View с сохранением всех свойств и динамической подстройкой под изменения на странице.

3. Описание объектной модели

3.1. Структура классов

Классы элементов SVG составляют сложную иерархию (рис. 1).

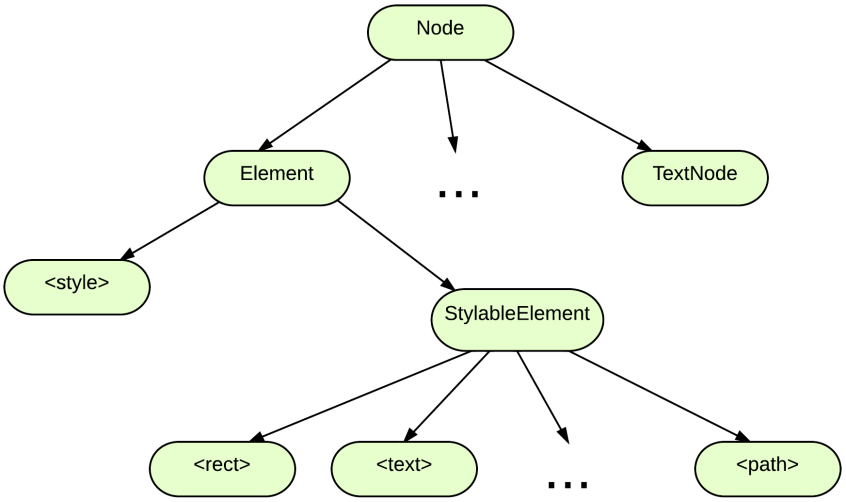

Рис. 1: Упрощенная схема иерархии классов модуля SVG

Для каждого элемента SVG существует набор заданных атрибутов, которые влияют на его представление. Большинство атрибутов задаются в текстовом виде (графические атрибуты, class, id), но некоторые типизированы (координаты, цвета). Объектная модель поддерживает оба класса атрибутов. Доступ к типизированным атрибутам предоставляется с помощью методов-аксессоров и их значения могут использоваться разработчиками в коде без необходимости парсинга строковых значений.

Для облегчения построения сложных элементов были разработаны дополнительные инструменты:

- PathBuilder — реализация паттерна проектирования Builder [7] для построения атрибута "*d*" графических элементов со сложной границей. Этот атрибут отвечает за последовательное построение границы из элементарных сегментов.

```
PathBuilder pathbBuilder = new PathBuilder(true);
pathBuilder
    .moveTo(125., 125.)
    .verticalLineTo(-100.)
    .ellipticalArc(100., 100., 0., false, false, -100., 100.)
    .closePath();
PathElement element = new PathElement(pathBuilder.build());
```

Listing 2: Пример использования PathBuilder

Code:

```
<svg>
  <path d="M125.0 125.0 v-
100.0 a100.0 100.0 0.0 0 0
-100.0 100.0 z"/>
</svg>
```

Result:

Рис. 2: Результат работы PathBuilder

- TransformBuilder — реализация паттерна проектирования Builder для построения атрибута "$transform$", который поддерживают большинство графических элементов. Атрибут "$transform$" отвечает за геометрические преобразования элемента, такие как сдвиги, повороты, сжатия и растяжения. Работа TransformBuilder аналогична PathBuilder.

- Для работы с цветами может использоваться существующий в фреймворке JetPad класс Color, который позволяет использовать набор заранее заданных цветов, создавать новые в пространстве RGB, работать с прозрачностью и отображать цвета в формат CSS.

3.2. Работа с атрибутами

Представления данных в формате SVG обычно содержат большое количество элементов. Кроме того каждый элемент SVG поддерживает неограниченное количество текстовых атрибутов. Так как существует требование доступа к произвольному атрибуту как к Property, в реализации модуля применяется механизм создания анонимной Property по требованию:

```
<ValueT> Property<ValueT> getAttr(Spec spec) {
    return new Property<ValueT>() {
      @Override
      public ValueT get() {
        return AttrStorage.get(spec);
      }

      @Override
      public void set(ValueT value) {
        AttrStorage.set(spec, value);
      }

      // other code
    };
  }
```

Listing 3: Создание Property on-demand

Это позволяет пользователям фреймворка работать с атрибутами единым образом вне зависимости от того, был ли атрибут инициализирован. Если атрибут будет инициализирован после создания соответствующей Property, сработают все назначенные обработчики и метод get() будет выдавать актуальное значение. В то же время память расходуется только на те атрибуты, которые были явно заданы.

Кроме того все коллекции атрибутов и обработчиков инициализируются "лениво" (lazy initialization) [2], и используют memory-efficient структуры данных (ListMap, EnumMap [2]).

3.3. Обработка пользовательских событий

Обработка пользовательских событий происходит по схеме ”снизу-вверх”. В такой схеме первыми событие получают ”листовые” члены иерархии, то есть элементы, не имеющие потомков. Далее событие обрабатывается, и, если во время обработки оно не было поглощено (consumed), передается на обработку предку элемента. Модуль должен предоставлять возможность назначать произвольные обработчики на события, в том числе после начальной инициализации модели, поэтому есть необходимость синхронизировать события происходящие в модели представления (Java Swing или GWT) с соответствующими событиями в объектной модели разрабатываемого модуля. С другой стороны, синхронизировать все события — плохой подход с точки зрения производительности, так как некоторые события, например движение мыши, происходят очень часто.

Задача была решена следующим образом: события представления не синхронизируются с моделью и поднимаются (bubbling) по иерархии представления, до тех пор, пока не достигнут элемента, имеющего обработчики соответствующего события. Далее обработка передается на сторону модели модуля JetPad SVG, а событие представления поглощается.

Рассмотрим простой пример — трехуровневая иерархия, обработчик события назначен только для элемента <g>:

```
<svg height="200" width="200">
    ...
    <g>
        <rect height="100" width="50" x="50" y="50">
        ...
    </g>
    ...
</svg>
```

Listing 4: Структура простого SVG изображения

Если пользователь совершает клик мышью по элементу <rect>, а обработчик для данного события назначен только на элемент <g>, то путь события будет пролегать, следующим образом: <rect> → <g> → jetpad.GElement → jetpad.SVGElement (рис. 3). На рисунке представлены как элементы объектной иерархии JetPad SVG (SVGElement, GElement, RectElement), так и соответствующие им объекты иерархии представления (<svg>, <g>, <rect>). Цветом выделен путь события по иерархии элементов.

Некоторые события, например передвижение курсора происходят очень часто, при этом лишь малая часть элементов назначает обработчики для этих событий. Таким

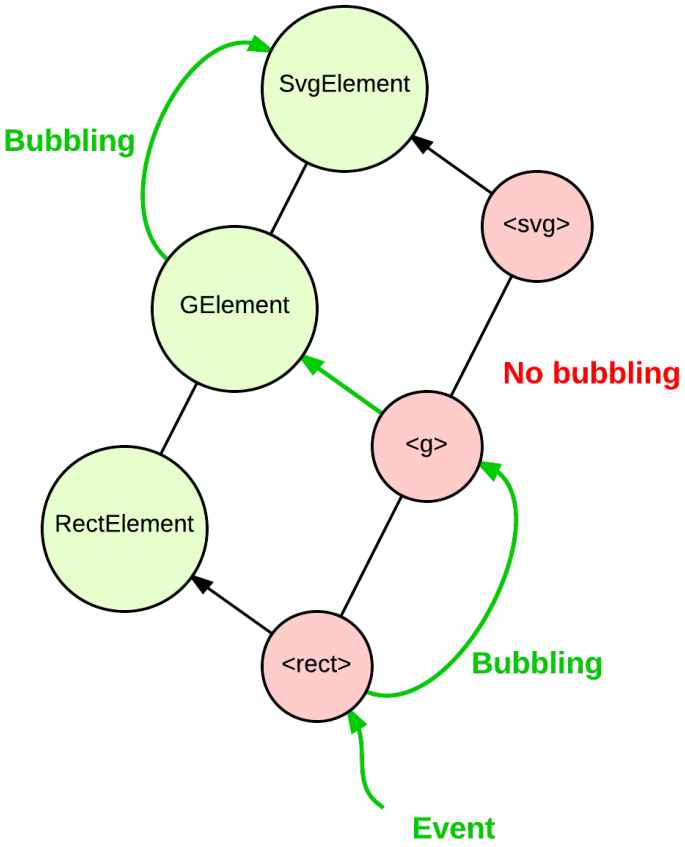

Рис. 3: Обработка пользовательского события

образом их синхронизация без необходимости может серьезно снижать производительность системы. Это особенно критично поскольку код исполняется на стороне клиента. Поэтому синхронизация происходит только для тех событий, для которых существует обработчик в иерархии модуля JetPad SVG.

3.4. Отображение в модели представления

В рамках работы была реализована возможность отображения объектной модели в различные представления.

Для отображения в HTML/GWT используется библиотека lib-gwt-svg [9]. Особенностью фреймворка является то, что код исполняется на стороне клиента. Это накладывает дополнительные ограничения на использование памяти и производительность. Кроме того, трансляция кода в JavaScript занимает значительно больше времени, чем компиляция Java кода, что замедляет скорость разработки.

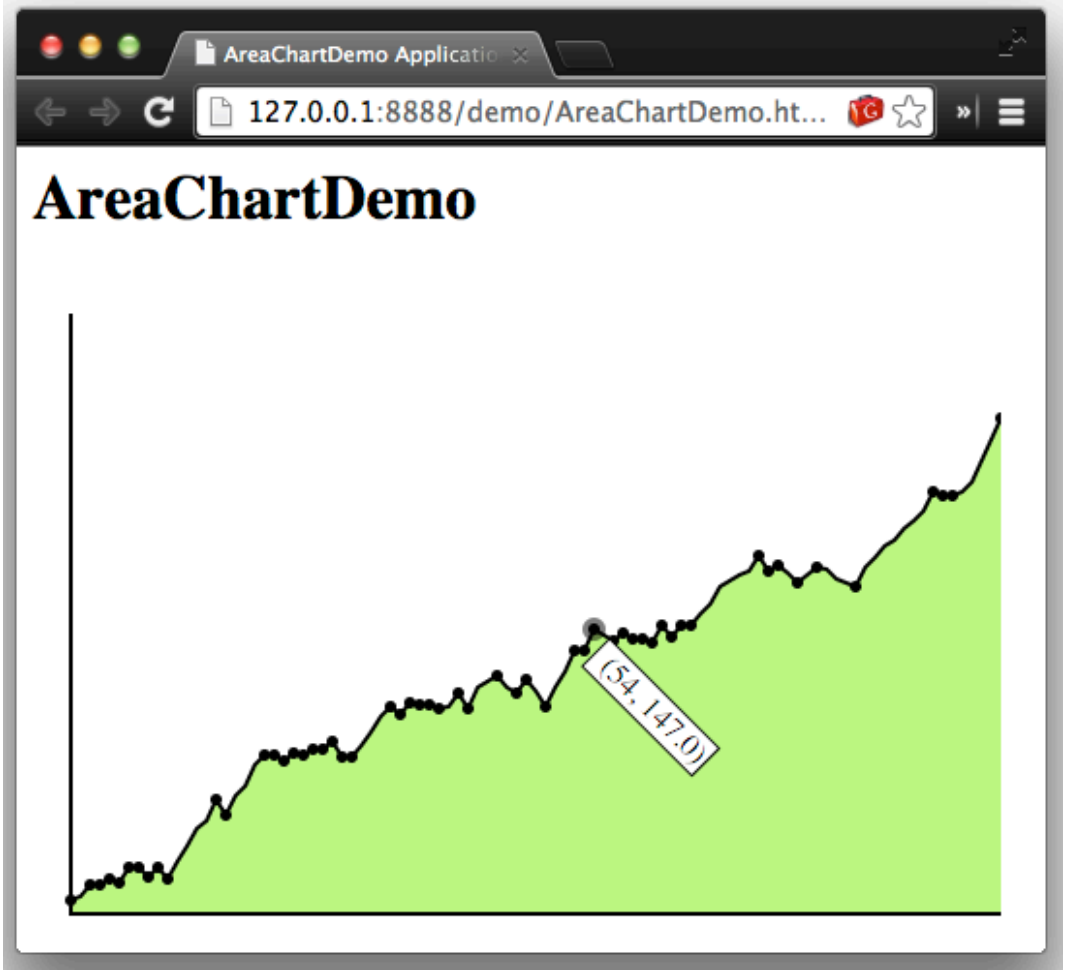

Рис. 4: Отображение модели в браузере с использованием GWT

Для отображения в Java Swing используется библиотека Apache Batik [1]. Одной из целей отображения в Java Swing является отладка графических компонент, поскольку компиляция модулей с использованием GWT занимает значительно больше времени и ресурсов. Таким образом существует требование идентичности представления и поведения модуля при отображении в обе системы.

Отображение происходит в соответствующие объектные модели библиотек, таким образом изменения в объектах JetPad SVG транслируются без перестройки самих объектов. Если же происходит изменение дерева элементов (добавление, удаление),

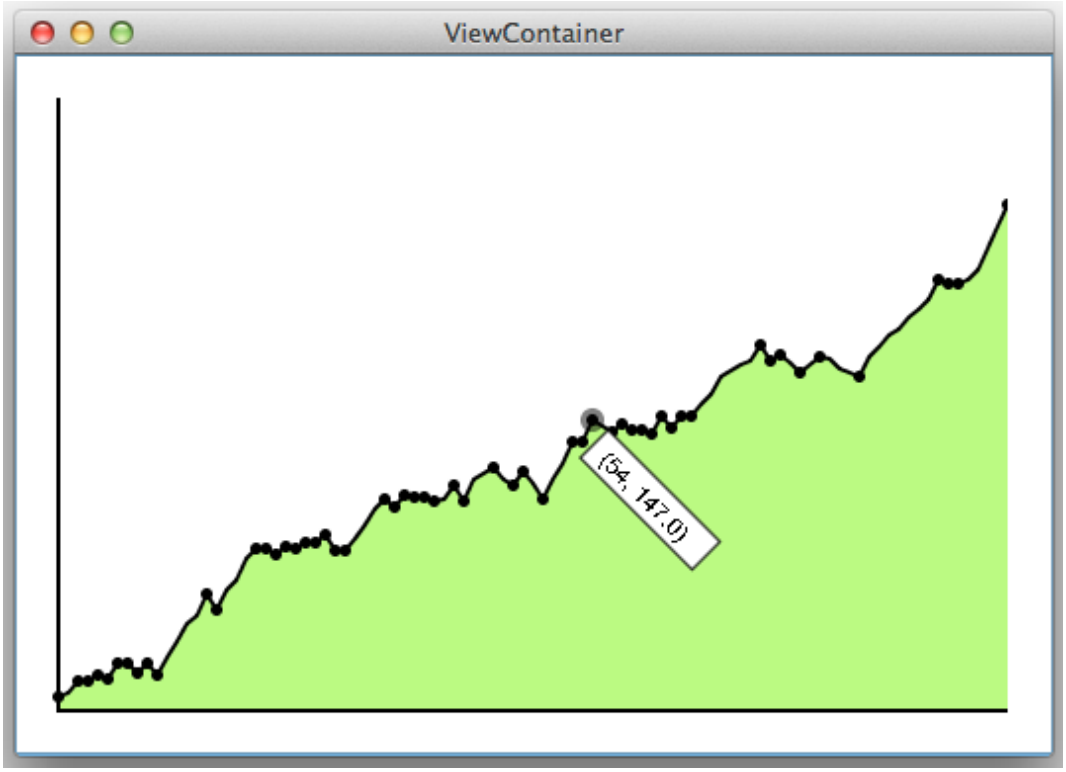

Рис. 5: Отображение модели в клиентском приложении с использованием Java Swing

то изменения происходят инкрементально, то есть перестраивается только затронутая часть.

Представления одной модели в обеих графических системах и их поведение полностью идентично, что способствует успешной отладке.

Заключение

В процессе работы были получены следущие результаты:

- Реализовано представление объектной модели SVG на языке Java в рамках фреймворка JetBrains JetPad.
- Реализован набор элементов SVG, достаточный для представления внутренних форматов данных в виде интерактивных диаграмм и графиков.
- Реализованы модули отображения объектной модели SVG в представления Java Swing и HTML/GWT.
- Реализованы инструменты для удобного создания комплексных атрибутов SVG из программного кода Java.
- Модуль интегрирован в существующую систему классов фреймворка JetBrains JetPad, и может использоваться в уже существующих приложениях без написания дополнительного кода.
- Созданы демонстрационные материалы и графические представления для внутренних форматов данных.

Список литературы

[1] Apache Batik SVG Toolkit. — 2014. — URL: http://xmlgraphics.apache.org/batik/.

[2] Bloch Joshua. Effective Java (2nd Edition). — Addison-Wesley, 2008. — ISBN: 0321356683.

[3] Consortium World Wide Web. Document Object Model (DOM) Level 2 Core Specification. — 2000. — URL: http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113/.

[4] Consortium World Wide Web. Extensible Markup Language (XML) 1.0 (Fifth Edition). — 2008. — URL: http://www.w3.org/TR/xml/.

[5] Consortium World Wide Web. Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification. — 2011. — URL: http://www.w3.org/TR/CSS2/.

[6] Consortium World Wide Web. Scalable Vector Graphics (SVG) 1.1 (Second Edition). — 2011. — URL: http://www.w3.org/TR/SVG/.

[7] Design Patterns, Elements of Reusable Object-Oriented Software / Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. — Addison-Wesley Professional, 1994. — ISBN: 0201633612.

[8] Google Web Toolkit. — 2014. — URL: http://www.gwtproject.org/.

[9] LIB-GWT-SVG. — 2014. — URL: http://www.vectomatic.org/libs/lib-gwt-svg.

[10] Salvaneschi Guido, Mezini Mira. Towards Reactive Programming for Object-oriented Applications. — 2014. — ISBN: 978-3-642-55099-7. — http://www.guidosalvaneschi.com/attachments/papers/2014_Towards-Reactive-Programming-for-Object-Oriented-Applications_pdf.pdf.

[11] Solomatov Konstantin. JetBrains Mappers MVC Framework. — 2014. — URL: https://github.com/JetBrains/jetpad-mapper/wiki/JetPad-Mappers.