Правительство Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Санкт-Петербургский государственный университет»

Кафедра системного программирования

Макулов Рустам Наилевич

Графическая библиотека на основе PostScript: механизм обработки событий

Дипломная работа

Допущена к защите.
Зав. кафедрой:
д. ф.-м. н., профессор Терехов А. Н.

Научный руководитель:
к. ф.-м. н., доц. Булычев Д. Ю.

Рецензент:
к. ф.-м. н., доц. Кознов Д. В.

Санкт-Петербург
2015

SAINT-PETERSBURG STATE UNIVERSITY

Chair of Software Engineering

Rustam Makulov

Event handling in a PostScript-based graphic library

Graduation Thesis

Admitted for defence.
Head of the chair:
Professor Andrey Terekhov

Scientific supervisor:
Dmitri Boulytchev

Reviewer:
Dmitry Koznov

Saint-Petersburg
2015

Оглавление

Введение

Задача разработки кроссплатформенных программных продуктов может быть решена по-разному. Например, создаются средства и инструменты, которые позволяют использовать один исходный код программ для работы в нескольких средах. Так виртуальная машина Java позволяет запускать программы на языке Java в различных системах, компилируя их в исполняемый код целевой платформы. На сегодняшний день JVM имеет реализации практически для всех операционных систем (ОС). Тем не менее, этого недостаточно для единой кроссплатформенной реализации графических интерфейсов программ, поскольку графическая оболочка конкретной операционной системы занимается отрисовкой примитивов и устанавливает свои ограничения на их поведение. Частично эту проблему решают графические библиотеки, например, Swing [1] и Qt [2]. Но оконный менеджер, встроенный в графическую оболочку ОС, всё равно регламентирует поведение и внешний вид графических интерфейсов.

Язык PostScript [3] является инструментом для создания высококачественной графики. В качестве графических примитивов в языке выступают отрезки и прямые, которые допускают произвольное масштабирование, вращение и другие преобразования. При этом PostScript является полноценным языком программирования, что, в отличие от декларативных языков, позволяет не только описывать конечное изображение, но и задавать способ его получения.

Значительную роль в графических пользовательских библиотеках играют события – действия или сигналы, которые могут быть идентифицированы и обработаны программой. В графических приложениях каждое действие пользователя порождает цепочку событий, которые

затем обрабатываются в приложении. Взаимодействие объектов также осуществляется при помощи событий: изменение состояния одного объекта приводит к изменению состояния другого, а событие оказывается средством связи между объектами.

Создание графических интерфейсов с помощью языка PostScript представляется невозможным до тех пор, пока в нем нет событий и механизма для их обработки. Добавление обработки событий позволит создать полноценную графическую библиотеку на PostScript. В свою очередь графическая библиотека на PostScript предоставит возможность создавать приложения, идентичные внешне в различных средах, и задавать новое нестандартное поведение интерфейсов в виде эффектов, например, размытие и волны, не предусмотренные оконным менеджером конкретной ОС и не зависящие от него.

В лаборатории JetBrains математико-механического факультета СПбГУ был разработан переносимый интерпретатор программ на языке PostScript, исполняемый в среде JVM [4, 5, 6]. Данный инструмент стал ядром проекта по разработке графической библиотеки на PostScript. Целью данной работы было расширить язык PostScript путём добавления в него событий и механизма для их корректной обработки. Параллельно в рамках других дипломных работ решались задачи отпимизированной интерпретации путём динамической компиляции, а также реализация графических примитивов и оконного менеджера.

1. Обзор предметной области

PostScript – это полнофункциональный язык программирования, спроектированный для создания графических изображений высокого качества. Интерпретатор языка PostScript – это инструмент для исполнения программ на языке PostScript. Он представляет собой JVM-реализацию языка. Swing – это популярная библиотека для создания графического интерфейса для программ на языке Java. Начиная с версии Java 1.2 (1998 год) Swing включён в Java Runtime Environment. Qt – это фреймворк для кроссплатформенной разработки на языке программирования C++.

Далее будет представлено краткое описание языка PostScript, интерпретатора языка, созданного в лаборатории JetBrains, и механизмов, обеспечивающих взаимодействие объектов в Qt и Swing.

1.1. PostScript

Язык PostScript используется для описания графических изображений и текстов с помощью подробных инструкций, отображаемых либо на экране компьютера, либо на печатной странице. В нем есть циклы, файловый ввод/вывод и определения функций. Программы на PostScript представляют собой набор комманд для рисования, выполнение которых позволяет создавать нетривиальные изображения.

Все данные, доступные программам на PostScript, представлены объектами. Операторы языка их создают и используют в качестве аргументов. Каждый объект имеет собственный тип, некоторый атрибут и значение. Объекты бывают простые, например: `boolean, integer, real, name, mark, null` и сложные, например, `array,`

`dictionary, file, string`.

Простые объекты являются атомарными: тип, атрибут и значение каждого объекта хранятся вместе, и их нельзя изменить независимо друг от друга. В отличие от значений простых объектов значения сложных объектов лежат в отдельной области памяти. Это позволяет во время копирования сложного объекта не дублировать, а просто передавать ссылку на значение исходного объекта. В результате объекты разделяют одно значение.

Специальная область памяти для хранения значений сложных объектов называется виртуальной памятью. Пространство виртуальной памяти, над которой работают операторы `save` и `restore`, называется локальной памятью. Оно используется для хранения информации, которая требуется в определенный период исполнения. Оставшееся пространство виртуальной памяти, необходимое для хранения информации, время жизни которой не зависит от хода программы, называется глобальной памятью.

Если значения объектов хранятся в них самих или в виртуальной памяти, то сами объекты хранятся на стеках. Всего среда исполнения использует пять стеков: стек операндов, стек словарей, стек исполнения, стек графических состояний, стек ограничивающих изображение путей. Стек исполнения хранит объекты, которые находятся в стадии исполнения. В любой точке исполнения программы он представляет собой стек вызовов программы.

Операторы `save` и `restore` используются для сохранения и восстановления значений объектов в локальной памяти. Первый из них сохраняет состояние локальной памяти и возвращает сохранённое состояние, а второй – восстанавливает состояние локальной памяти к со-

стоянию, сохранённому предшествующим оператором `save`.

Построение рисунка выполняется с помощью графических операторов, например, таких как `rlineto, stroke, fill`. Результат построения становится видимым на экране после выполнения оператора `showpage`. Интерпретатор выполняет программу на PostScript и отображает картинку в отдельном окне. После этого изменить получившееся изображение нельзя, потому что в языке отсутствуют поддержка взаимодействия пользователя и программы.

1.2. Интерпретатор языка PostScript

В 2013-2014 гг. в рамках проекта лаборатории JetBrains на основе спецификации языка PostScript был создан кроссплатформенный интерпретатор данного языка. Исходный код написан на языке Java, графическая часть была реализована с использованием библиотеки Swing, входящей в Java Runtime Environment.

Не вдаваясь в подробности, можно сказать, что проект состоял из трёх частей: библиотеки общей поддержки времени исполнения, библиотеки графической части и архитектуры, определяющей внутреннюю структуру и задающей связи между основными компонентами.

Промышленный интерпретатор ghostscript [7], написанный на языке C++, использовался в качестве эталонного интерпретатора. Его разработка продолжается по настоящее время.

1.3. Обработка событий в Swing

События – это важная часть в любой программе, использующей пользовательский интерфейс. Все подобные приложения управляются событиями. Приложение реагирует на различные типы событий, кото-

рые генерируются в процессе его работы. В любой модели событий присутствуют три основные компоненты: источник, приёмник и слушатель (listener).

Источник события – это объект, состояние которого подвергается воздействию пользователя или из других объектов. В результате подобного взаимодействия генерируется событие, заключающее в себе изменение состояния в источнике. Слушатель события – это объект, который уведомляется об изменении состояния. Источник события делегирует задачу обработки события своему слушателю.

Swing обеспечивают взаимодействие между объектами с помощью функций обратного вызова. Обратный вызов – это указатель на функцию. Таким образом, если необходим вызов функции обработки, чтобы уведомить о некотором событии, передается указатель на другую функцию (обратный вызов) функции обработки. Функция обработки вызовет функцию обратного вызова, когда это необходимо.

Когда в пользовательском интерфейсе что-то происходит, например, нажатие на кнопку или выбор элемента из списка, генерируется событие. Существует несколько типов событий, например, события действия, события текста, события переключения фокуса. События хранят информацию о произошедшем действии, например, если нажата кнопка, то информация об этом – это время, порядковый номер в списке графических элементов, источник события и изменённые ключи.

Обработка событий в Swing может быть реализована либо с использованием анонимных классов, либо внутренних классов, либо наследуемых классов.

1.4. Обработка событий в Qt

В Qt используется альтернатива техники обратного вызова: сигналы и слоты. Сигнал генерируется, когда происходит определённое событие. Виджеты Qt имеют множество определённых заранее сигналов, но всегда можно выделить подкласс виджетов, чтобы добавить свои сигналы на них. Слот – это функция, которая вызывается в ответ на определенный сигнал. Виджеты Qt имеют множество предопределённых слотов, но обычно выделяют подклассы виджетов и добавляют свои слоты для того, чтобы обрабатывать сигналы, в которых заинтересован разработчик.

Важным моментом является то, что события обрабатываются не мгновенно после возникновения; вместо этого они попадают в очередь событий, получают приоритеты и последовательно обрабатываются. Всем этим процессом управляет диспетчер событий. Диспетчер циклически обрабатывает очередь и отправляет события по месту назначения.

2. Механизм обработки событий

Поскольку язык PostScript изначально проектировался с целью описания графики, то в нем отсутствует обратная связь и, следовательно, какой бы то ни был механизм обработки событий.

2.1. Передача событий из Java в PostScript

Благодаря тому, что интерпретатор написан на Java, есть возможность обрабатывать события средствами Java: с помощью слушателей событий мыши и клавиатуры. На рис. 1 изображена диаграмма последовательности, на которой изображена передача событий из Java в PostScript.

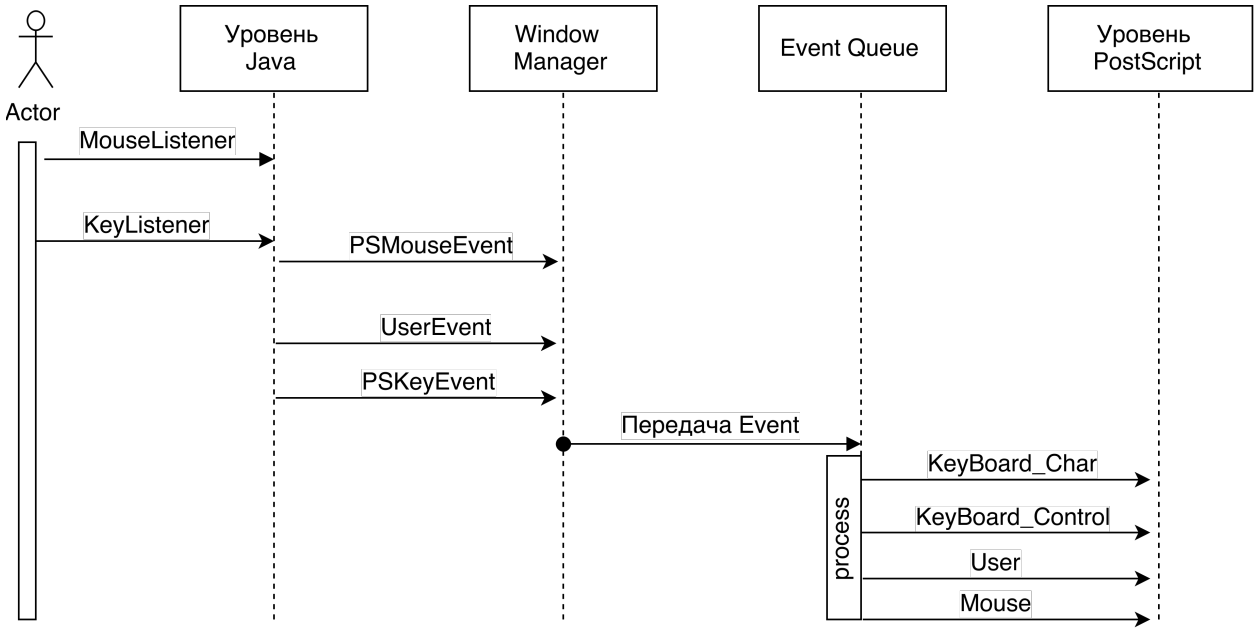

Рис. 1: Передача событий из Java в PostScript

На уровне Java слушатели мыши и клавиатуры отслеживают действия пользователя и генерируют события трёх типов: PSMouseEvent, UserEvent и PSKeyEvent. Затем эти события с соответствующей информацией передаются в оконный менеджер. Оконный менеджер добавляет

полученное событие в очередь обработки событий, и она последовательно исполняет события, поочерёдно вызывая операторы для каждого типа. На уровне PostScript события делятся на следующие типы: ввод печатаемого символа (KEYBOARD_CHAR), ввод управляющего символа (KEYBOARD_CONTROL), действие пользователя (USER) и действие мыши (MOUSE).

Абстрактный класс Event представляет собой события в PostScript, служит родителем для классов PSKeyEvent, PSMouseEvent, UserEvent и содержит в себе тип события EventType (см. рис. 2). В настоящее время реализована поддержка следующих типов событий: CLICK, DOUBLE_CLICK, RELEASE, PRESS, RESIZE, MOVE, DRAG, CLOSE, ENTER, EXIT, RIGHT_CLICK, PAINT, KEYBOARD_CHAR, USER, UPDATE_VALUE, KEYBOARD_CONTROL. Класс PSKeyEvent дополнительно содержит код нажатого символа, а PSMouseEvent – координаты клика. Класс UserEvent содержит строку – команду и предназначен для реализации механизма обратной связи: из PostScript в Java. Обработка событий мыши и клавиатуры производится в операторах keyevent и mouseevent. В результате работы этих операторов на стеке исполнения появляется процедура, задаваемая при описании события на уровне PostScript.

2.2. Передача событий из PostScript в Java

Графическую библиотеку предполагается использовать с помощью интерфейса программирования приложений (application programming interface). Для апробации библиотеки был выбран язык Java. С этой целью для каждого графического элемента в PostScript были созданы соответствующие классы на Java, например, PSButton, PSWindow и

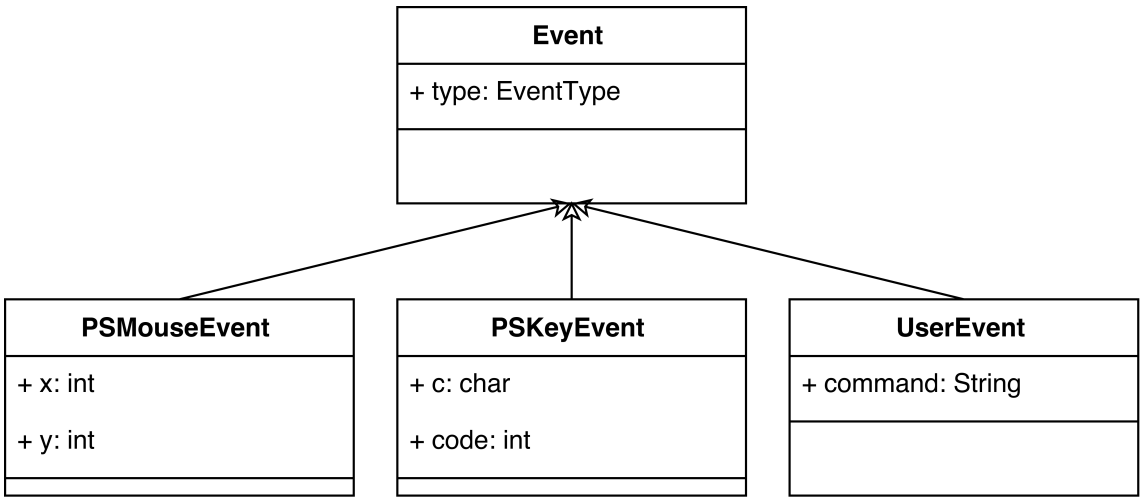

Рис. 2: Типы событий, передаваемых из Java в PostScript

так далее.

Главный класс, от которого наследуются все элементы графического интерфейса называется `PSComponent`. На рис. 3 изображена его структура.

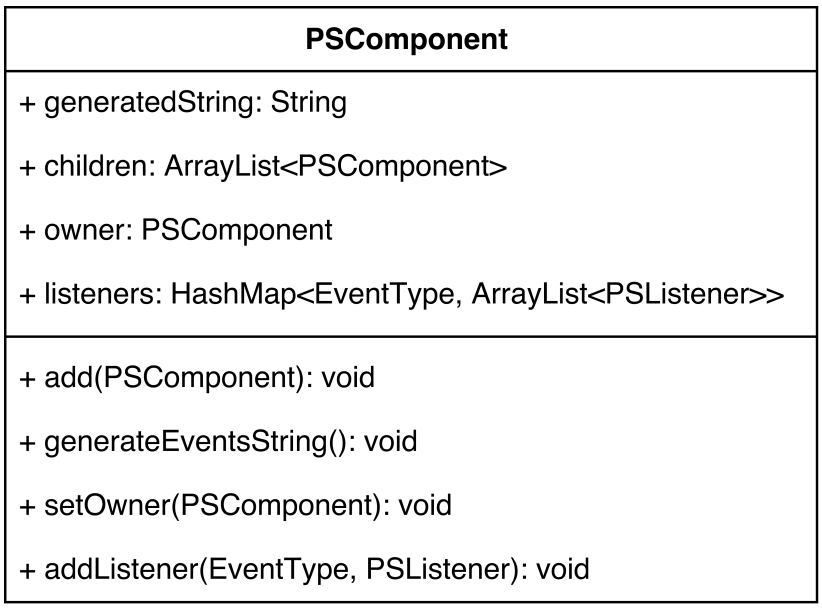

Рис. 3: Класс PSComponent

Поле `generatedString` представляет собой фрагмент программы на PostScript и отвечает за добавление компоненты в интерфейс. При этом данный фрагмент собирается последовательно в несколько

этапов. Первоначально он представляет собой шаблон, в который подставляются необходимые значения. Например, для кнопок он имеет следующий вид:

```
%s %d %d %d %d (%s) scene events button
```

За несколько этапов в шаблон заполняется имя, состоящее из уникального номера и префикса, например, `window1` для окна и `button2` для кнопки. Далее идут координаты левого нижнего угла, размеры по горизонтали и вертикали, надпись на кнопке (строки на PostScript указываются в скобках). Затем указывается имя родительского элемента, к которому прикрепляется элемент (по умолчанию это сцена (scene)). Ключевое слово `button`, необходимое для завершения создания кнопки.

Передача информации в обратную сторону, то есть из PostScript в Java была выполнена динамически, а именно, в тот момент когда добавляется слушатель на Java, как в данном примере

```
PSButton psButton = new PSButton(150, 150, 40, 30, "close");
psButton.addListener(EventType evType, PSListener listener){..}
```

неявно происходит добавление закладки – слушателя в поле `listeners`. Во время генерации конечной программы на PostScript, происходит вставка оператора `sendEventToJava`, отправляющего события на уровне PostScript. Данный оператор в качестве параметров принимает словарь и тип события. Словарь, который содержит в себе всю информацию об элементе графического интерфейса, передаётся автоматически во время исполнения.

Отдельный класс `Slots` хранит информацию об объектах в

PostScript. Она представляет объект класса HashMap. Паре из имени объекта и типа события соответствует процедура или действие, которое задано пользователем на языке Java при инициализации.

Каждому графическому объекту в PostScript соответствует объект Java. По сути происходит генерация программы на PostScript: каждый такой объект содержит строку-конструктор или команду, при выполнении которой мы получаем нужный элемент графического интерфейса. Команда передается в классе `UserEvent` в поле `command`. Данный класс содержит уникальный номер и префикс графического элемента, конкатенация которых в обратном порядке и есть имя объекта (которое теперь содержится внутри каждого), процедура, исполняемая при нажатии, родительский графический компонент.

3. Расширение языка PostScript

В стандартной реализации языка PostScript отсутствуют функциональные средства, наличие которых необходимо в библиотеке для создания пользовательских интерфейсов, например, события, которые поступают от мыши и клавиатуры, и их обработчики. В связи с этим возникла потребность расширить язык PostScript, добавив в него новые операторы. Это возможно благодаря тому, что интерпретатор, используемый в данном проекте, можно модифицировать в соответствии с возникающими потребностями. Модификация интерпретатора возможна, поскольку он проектировался и создавался участниками проекта.

Добавление новых функциональных возможностей в PostScript потребовало соответствующего расширения языка и реализующего его интерпретатора.

3.1. Добавление событий

Отслеживание событий в языке было выполнено средствами Java. Для обработки событий используются специальные синтаксические конструкции при создании каждого графического примитива. Графические примитивы – это кнопки, выпадающие списки, метки и так далее. Например, сигнатура метода, добавляющего кнопку, выглядит следующим образом:

```
450 600 100 70 (close)  window1 <</CLICK [{pop quit}[]] >>  button pop
```

В двойных угловых скобках передается словарь: пары из типов обрабатываемых событий и ассоциированных с ними процедур. Процедура может быть не одна, а целый список, поэтому также в язык были до-

бавлены списки, отсутствующие в нем. Реализация списков выполнена подобным образом, как реализованы списки на haskell.

3.2. Добавление потока

Для возможности выполнения графических эффектов был введен оператор `newThread`. Он используется для создания нового потока с новым контекстом в расширенном PostScript. В качестве входных параметров он использует исполняемые процедуры и число – количество операндов, которые нужно переместить на стек операндов в новом контексте из стека операндов контекста, в котором оператор был вызван. В самом конце происходит запуск нового потока с процедурой в получившемся контексте. По завершении выполнения поток удаляется автоматически.

Операторы `keyEvent` и `mouseEvent`, реализация которых выполнена на PostScript, выполняют поиск в глубину нужного элемента интерфейса. После нахождения элемента, который имеет должен реагировать на искомый тип события, выполняется хранящаяся в данном элементе по данному событию процедура. Если данный тип не отслеживается ни одним из элементов, то данное событие возвращается сцене.

Для отладки были введены операторы `print, pstack, debug, init`. Они позволяют контролировать внутренннее состояние интерпретатора во время исполнения, выводить информацию о загруженности стеков, верхнем элементе и вести логирование.

4. Поддержка выполнения фоновых задач

Поддержка выполнения фоновых задач представляется одной из основных характеристик графической библиотеки. Это связано с тем, что предполагается использование разного рода эффектов, например, повороты, растяжения и волны. Подобные эффекты, по определению, должны выполняться, не изменяя внутреннее состояние пользовательского интерфейса. С этим были связаны определенные трудности, поскольку в интерпретаторе изначально было реализовано только одно состояние графики. Состояние графики в PostScript – это текущая конфигурация графических настроек, например, текущая точка и построенный путь, матрица преобразования и установленный шрифт. При этом изменение состояния графики в процессе работы может также изменять состояние стека операндов, стека словарей и стека исполнения. Поэтому было предложено добавить в интерпретатор контексты исполнения.

Контекст представляет собой набор основных стеков: операндов, словарей, графики и исполнения и методы для работы с ними. При этом все контексты разделяют между собой локальную память и используют её для хранения значений сложных объектов языка. На рис. 4 изображены основные компоненты класса `Context`.

Если раньше все эти компоненты находились в библиотеке поддержке времени исполнения, то теперь они выделены в отдельную структуру. Таким образом, появилась возможность параллельного выполнения задач отрисовки с переключением между ними, а значит параллельной отрисовки интерфейса.

Возможность параллельной обработки интерфейса была реализована с помощью использования очереди отрисовки, в которой хранятся операторы рисования, реализованные в классе `PSPrimitive`. Класс

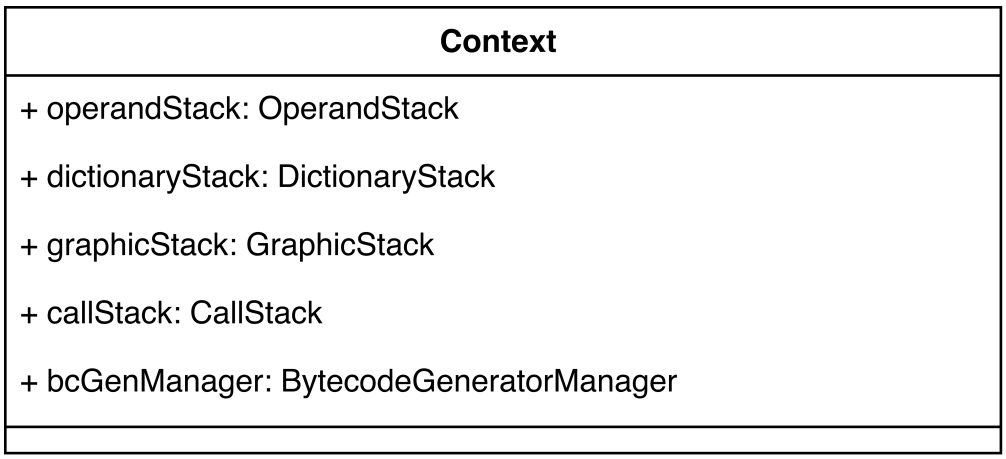

Рис. 4: Класс Context

`PSPrimitive` объединяет в себе все рисующие операторы языка: операторы заливки, рисования линии и так далее. В этом классе также хранится текущий графический путь и ссылка на тот контекст, в котором производится исполнение, из которого берутся параметры и в который кладутся результаты работы. Всё это позволило реализовать выполнение фоновых задач. В качестве примера были реализованы развороты диалогового окна по вертикали и горизонтали на 180 градусов, запуск волны по щелчку правой кнопкой мыши.

Заключение

В ходе выполненной работы были достигнуты следующие результаты.

- Спроектирован механизм обработки событий.
- Выполнено и реализовано расширение языка PostScript:
  - поддержка событий;
  - обработчики событий мыши и клавиатуры;
  - отладочные операторы.
- Выполнена поддержка выполнения фоновых задач:
  - реализована очередь выполнения графических примитивов;
  - реализована очередь выполнения событий;
  - реализована многопоточность.

Список литературы

[1] Creating a GUI with JFC/Swing. Oracle documentation. http://docs.oracle.com/javase/tutorial/uiswing/.

[2] Qt developer resources: documentation, guides, forums. http://www.qt.io/developers/.

[3] Adobe Systems Incorporated. Postscript language reference third edition. https://www.adobe.com/products/postscript/pdfs/PLRM.pdf.

[4] Р.Макулов. Архитектура интерпретатора для исполнения программ на языке postscript в JVM. Труды лаборатории языковых инструментов. Выпуск 2. 2014. с. 259-275.

[5] Д.Поздин. Реализация общей поддержки времени исполнения для интерпретатора языка PostScript. Труды лаборатории языковых инструментов. Выпуск 2. 2014. с. 276-296.

[6] А.Гудиев. Реализация графической части интерпретатора языка postscript. Труды лаборатории языковых инструментов. Выпуск 2. 2014. с. 297-312.

[7] Ghostscript an interpreter for the PostScript language and for PDF. http://www.ghostscript.com/.