Санкт-Петербургский государственный университет

Математическое обеспечение и администрирование информационных систем
Системное программирование

Юрьев Семен Юрьевич

Поддержка BPMN в проекте WMP

Выпускная квалификационная работа

Научный руководитель:
к.т.н., доц. Брыксин Т.А.

Рецензент:
Смирнов К.К.

Санкт-Петербург
2017

SAINT-PETERSBURG STATE UNIVERSITY

Software and Administration of Information Systems
Software Engineering

Semen Yuryev

BPMN support in WMP project

Graduation Project

Scientific supervisor:
Candidate of Engineering Sciences Timofey Bryksin

Reviewer:
Kirill Smirnov

Saint-Petersburg
2017

Оглавление

Введение

Визуальное моделирование — это метод, который применяется при разработке и эволюции программного обеспечения (ПО), а также в отдельных видах деятельности процесса разработки, и использует графовые модели для описания ПО с разных точек зрения. Визуальные модели, напоминающие чертежи в машиностроении, электротехнике, строительстве и иных инженерных сферах, оказываются удобными при работе с большими массивами информации, имеющими многочисленные внутренние связи. Визуальные спецификации точны, легко воспринимаются, обсуждаются и изменяются, они позволяют охватывать в сжатой и наглядной форме большое количество информации [13]. Одними из самых известных в индустрии разработки ПО моделей являются диаграммы UML.

Стандарт BPMN [2] (Business Process Model and Notation) — система условных обозначений (нотация) для моделирования бизнес-процессов. Спецификация BPMN описывает условные обозначения для отображения бизнес-процессов в виде диаграмм бизнес-процессов. BPMN ориентирована как на технических специалистов, так и на бизнес-пользователей. Для этого язык использует базовый набор элементов, понятный специалистам данной предметной области, которые позволяют определять сложные семантические конструкции.

В связи с развитием сети Интернет и популярностью мобильных сенсорных устройств (КПК, смартфоны, планшеты), появилась идея предоставить веб-инструмент для составления диаграмм. Доступность инструмента отовсюду и его независимость от платформы могут сделать его весьма удобным в использовании. Так появился проект WMP [7] (Web Modeling Project), посвященный реализации веб-платформы для создания редакторов диаграмм, разрабатываемый на кафедре системного программирования Санкт-Петербургского государственного университета. Приложение представляет собой объединение инструментов для поддержки диаграммных редакторов различных языков. Платформа позволяет расставлять элементы на сцене, проводить между

ними связи, задавать элементам свойства различных типов, а также корректно сохранять и загружать их с сервера. Сам проект делится на несколько частей: серверная часть, занимающаяся сохранением и загрузкой диаграмм для клиента, а также авторизацией пользователей; ядро проекта, содержащее основную функциональность, присущую всем редакторам; сами редакторы, которые используют функциональность ядра и с помощью которых можно создавать диаграммы того или иного языка. Благодаря такой структуре проекта становится проще создавать редакторы для диаграмм новых языков, поскольку функциональность ядра переиспользуется в новых редакторах.

До недавнего времени единственным поддерживаемым языком диаграмм в проекте WMP являлся язык диаграмм роботов, использующийся в TRIK Studio [3]. Язык программирования роботов довольно прост, и в связи с этим ядро проекта содержало лишь самые базовые для многих языков конструкции и элементы. Кроме того, для продвижения проекта WMP был необходим редактор, который мог бы быть полезен широкому кругу пользователей. Вышеописанный стандарт BPMN отлично подошел бы на роль языка диаграмм для нового редактора, но он весьма сложен, и создание редактора, поддерживающего этот формат полностью, не представлялось возможным в рамках текущих возможностей платформы.

Постановка задачи

Целью данной работы является усовершенствование инструментов платформы WMP посредством добавления новой функциональности и реализации BPMN-редактора на основе данной платформы. Для достижения этой цели были поставлены следующие задачи.

- Определить подмножество BPMN для дальнейшей реализации.
- Реализовать BPMN-редактор на базе платформы WMP, расширив её необходимыми для этого инструментами.

1. Обзор

1.1. Проект WMP

Проект WMP является распределённым приложением, построенным на основе микросервисной архтиектуры. Серверные компоненты написаны на языка Java, клиентские — на Typescript [8]. В качестве системы сборки используется Maven. Для организации сервисной архитектуры и клиент-серверного взаимодействия используется Apache Thrift [1]. Подробнее о сервисной архитектуре проекта WMP (в прошлом — QReal-web) можно узнать в курсовой работе Артемия Безгузикова [11].

Клиентская часть приложения состоит из двух частей — ядра платформы и конкретных редакторов. В ядре содержится базовая функциональность, необходимая каждому редактору. В редакторах реализуется специфичная для языка функциональность: веб-страницы редакторов в формате JSP [6], изображения и библиотеки, необходимые для корректной работы редакторов.

Основной класс в ядре платформы — DiagramEditorController. Он содержит информацию о всех других контроллерах и координирует их работу. Класс PaletteController отвечает за работу палитры элементов. Класс SceneController отвечает за работу сцены диаграммы, то есть за операции над элементами диаграммы: добавлением, удалением, выделением, изменением их параметров. Класс PropertyEditController синхронизирует изменения свойств элементов в редакторе свойств.

Особого внимания заслуживает класс UndoRedoController. Этот класс запоминает сделанные пользователем изменения и позволяет откатывать их назад (undo). Если откат был сделан ошибочно, можно вернуть сделанные ранее действия (redo). Сделанные пользователем действия запоминаются в виде команд. Интерфейс Command (рис. 1) имеет два метода — execute и revert, которые запускают и отменяют действие команды. MultiCommand имеет внутри список команд, и служит для запоминания действия, которое породило сразу несколько команд.

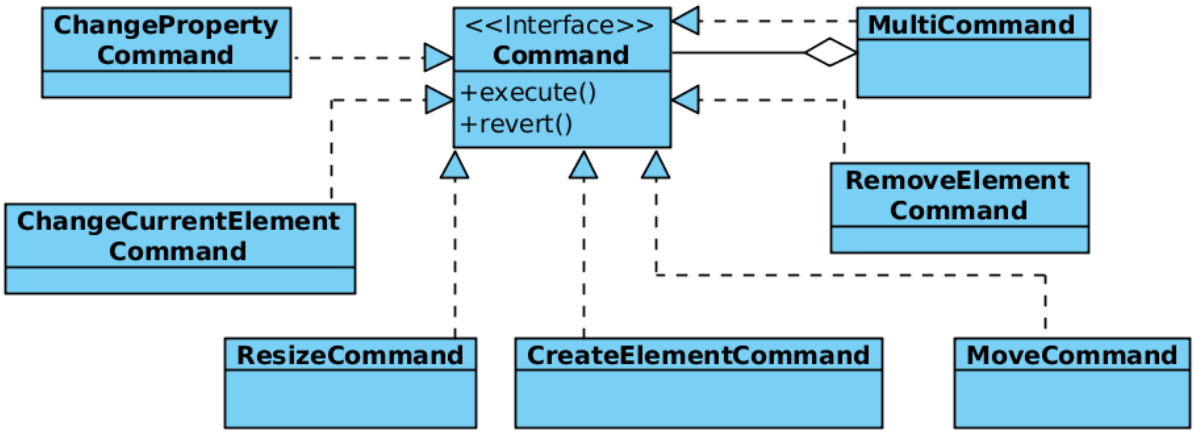

Рис. 1: Интерфейс Command и все его реализации

Проект WMP активно использует библиотеку JointJS [12]. Она написана на языке JavaScript [4], и предоставляет API для создания редакторов диаграмм. Она описывает базовые классы редактора диаграмм и взаимодействия между ними. Для создания объектов JointJS используется объект атрибутов, который задает внешний вид и поведение объекта на сцене. В проекте непосредственно используются три класса JointJS: joint.shapes.basic.Generic, описывающий обыкновенный элемент диаграммы; joint.dia.Link для связывающих объектов и joint.dia.Paper для сцены редактора.

Все элементы диаграмм реализуют интерфейс DiagramElement. Класс Link предназначен для реализации ассоциаций между объектами. Интерфейс DiagramNode описывает интерфейс узла диаграммы. Его единственная на данный момент реализация DefaultDiagramNode реализует базовую функциональность таких элементов (рис. 2).

Каждый из классов Link и DefaultDiagramNode является оболочкой для объекта из JointJS, который и находится на сцене. Для Link этот объект имеет тип joint.dia.Link, для DefaultDiagramNode — ImageWithPorts, который наследуется от класса joint.shapes.basic.Generic. ImageWithPorts расширяет этот класс, добавляя картинку внутрь элемента, которую видит пользователь при составлении диаграмм.

Класс DiagramScene наследует класс joint.dia.Paper из библиотеки JointJS и расширяет его функциональность для данного проекта. Он

ответственен за создание объектов JointJS на сцене редактора и их удаление при необходимости.

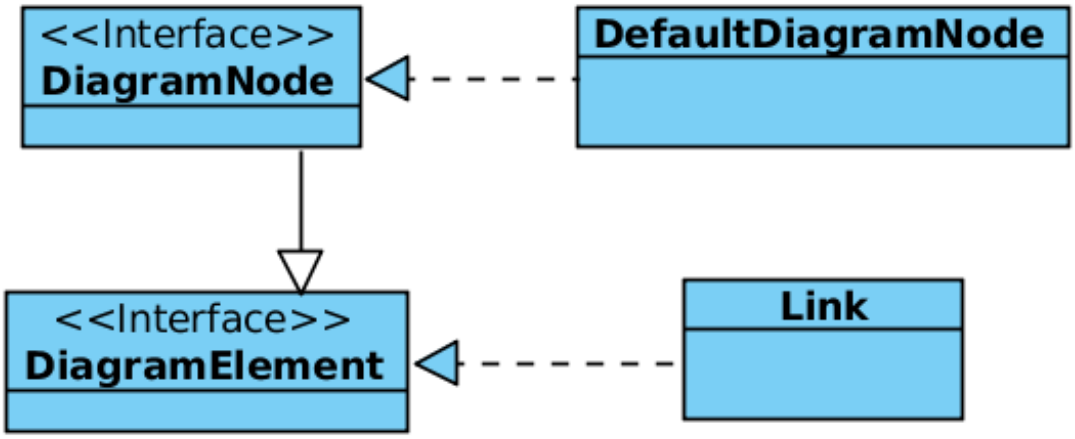

Рис. 2: Диаграмма классов для элементов на диаграмме

У каждого редактора существует конфигурационный файл, содержащий информацию о типах элементов, существующих в языке. При загрузке страницы редактора клиент первым делом посылает запрос серверу. Сервер в ответ посылает ему описания типов всех элементов данного редактора, содержащие базовую информацию о типах. Эти описания считываются в парсере типов (TypesParser). На их основе создаются объекты класса NodeType, хранящие информацию о типах, и впоследствии хранящиеся в DiagramEditorController. Имеено по этим объектам SceneController создает объекты DiagramElement.

Классы DiagramThriftExporter и DiagramThriftParser осуществляют сохранение и загрузку диаграмм. Посылая и принимая запросы от сервера с помощью Thrift, они переводят Thrift-структры в элементы DiagramElement при сохранении диаграммы и наоборот при загрузке.

1.2. Стандарт BPMN и его подмножество для реализации

В данном разделе приводится описание элементов BPMN, которые планировалось реализовать, и описание того, какую функциональность

необходимо было реализовать для корректного составления диаграмм с ними. Эти элементы являются базовыми, и их реализации хватит для описания большей части сценариев BPMN. Это не все элементы, которые описывает стандарт, но если корректно реализовать эти элементы, то в проекте появятся механизмы, которые сделают значительно проще реализацию оставшихся. Полное описание стандарта BPMN можно найти в спецификации [5].

1.2.1. События и логические операторы

События в стандарте BPMN означают какое-либо происшествие в мире. Изображаются окружностью с соответствующей иконкой внутри нее. События инициируют действия или являются их результатами. Согласно расположению в процессе события могут быть классифицированы на начальные, промежуточные и завершающие.

Логические операторы представляют точки принятия решений в процессе. Изображаются ромбом с иконкой внутри. С помощью логических операторов организуется ветвление и синхронизация потоков управления в модели процесса [10].

Рис. 3: В верхнем ряду все подтипы события «сообщение»; в нижнем ряду все логические операторы

Для редактора диаграмм оба этих типа являются обычными элементами, которые уже поддерживались в редакторе WMP (рис. 3). Для того, чтобы использовать их в диаграммах, необходимо лишь добавить их описания в конфигурационный файл редактора. Однако есть нюанс: у событий BPMN существует много подтипов, и задавать практически

идентичные элементы по одному будет нерационально. Следует доработать формат конфигурационного файла так, чтобы задание подтипов было как можно более коротким и простым. Кроме того, иногда пользователь может ошибаться при выборе конкретного подтипа при составлении своей диаграммы. Для этого было бы удобно иметь возможность поменять подтип для уже созданного и выставленного на диаграмму события.

1.2.2. Потоки

Потоки BPMN нужны для связывания элементов диаграммы между собой. Визуально они представляют собой направленную или ненаправленную ломаную линию. Всего в стандарте BPMN шесть потоков: поток управления, условный поток, поток по умолчанию, поток сообщений, направленная и ненаправленная ассоциация (рис. 4). До начала данной работы в редакторе было возможно соединять элементы только одним заданным типом связей. Для поддержки потоков BPMN необходимо было реализовать возможность выбора типа связи в редакторе. Для того, чтобы пользователь мог выбирать тип используемого потока, в палитре элементов необходимо было создать новую панель для выбора и создания связей.

1.2.3. Действия

Действия в BPMN делятся на задания и подпроцессы. Изображаются действия прямоугольниками со скругленными углами.

Задание в стандарте BPMN — единица работы, элементарное действие в процессе. Задания содержат внутри себя текст, описывающий это действие. Соответственно, для поддержки заданий необходимо поддержать возможность отображать текст внутри элемента, а также дать возможность пользователю ввести этот текст в редакторе.

Подпроцессы являются контейнерным элементом, то есть элементом, способным содержать в себе другие элементы диаграммы. Контейнерные элементы и элементы внутри них связаны, и при изменениях

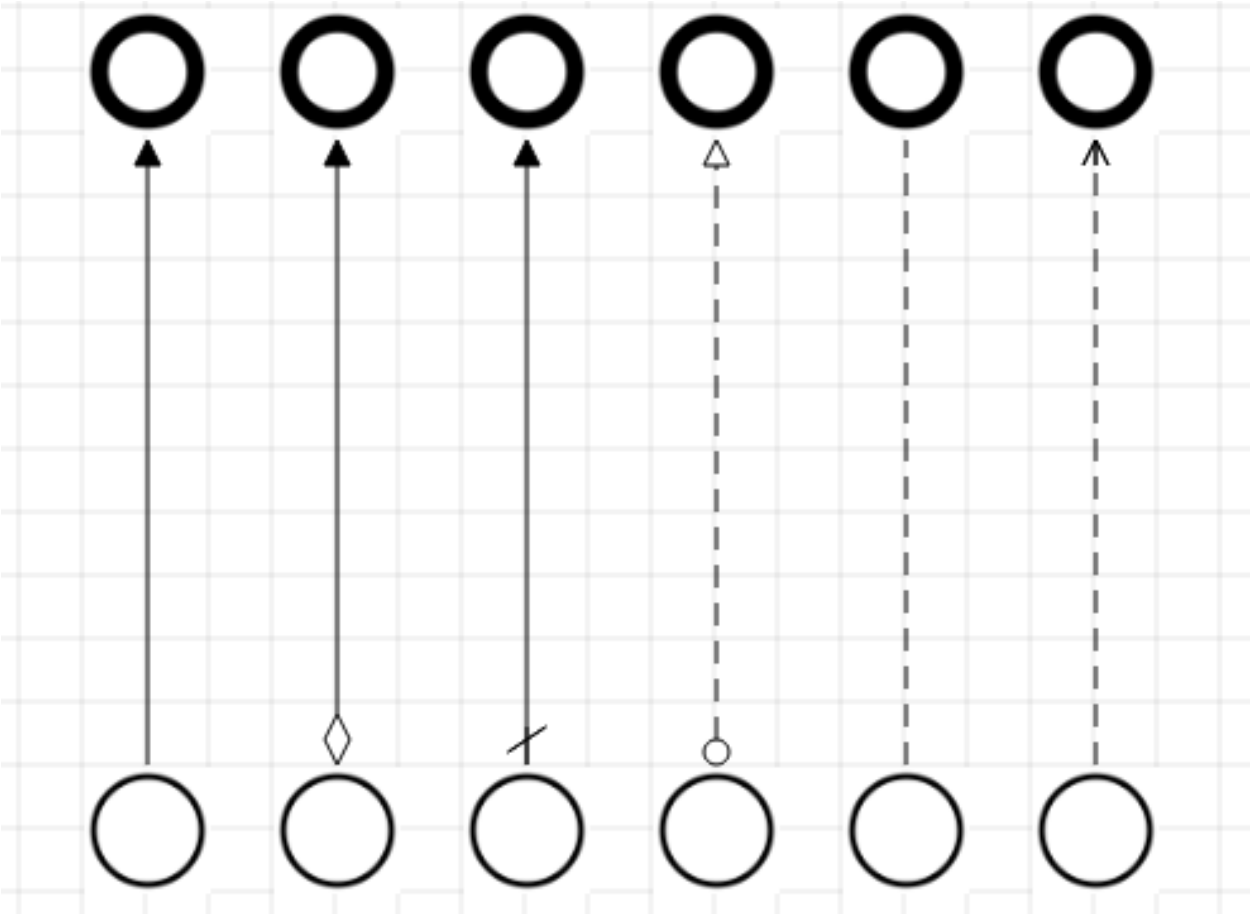

Рис. 4: Все потоки стандарта BPMN

родителя могут меняться дети, и наоборот. Например, если перемещается контейнерный элемент, то вместе с ним перемещаются и все его дочерние элементы и связанные с ними потоки. Кроме того, все эти взаимосвязи должны корректно работать с механизмами отката пользовательских действий и сохранения и загрузки диаграмм.

1.2.4. Пулы и дорожки

Пул является основным элементом диаграмм BPMN. Пул предназначен для отображения потока рассматриваемого процесса. Содержимое пула — это и есть тот процесс, диаграмма которого рассматривается. Пул разделен на части — дорожки, предназначенные для отображения организационных единиц (должности, отделы, роли, подразделения): исполнителей задач и подпроцессов процесса BPMN. Внутри блока помещается наименование организационной единицы [14]. Кроме

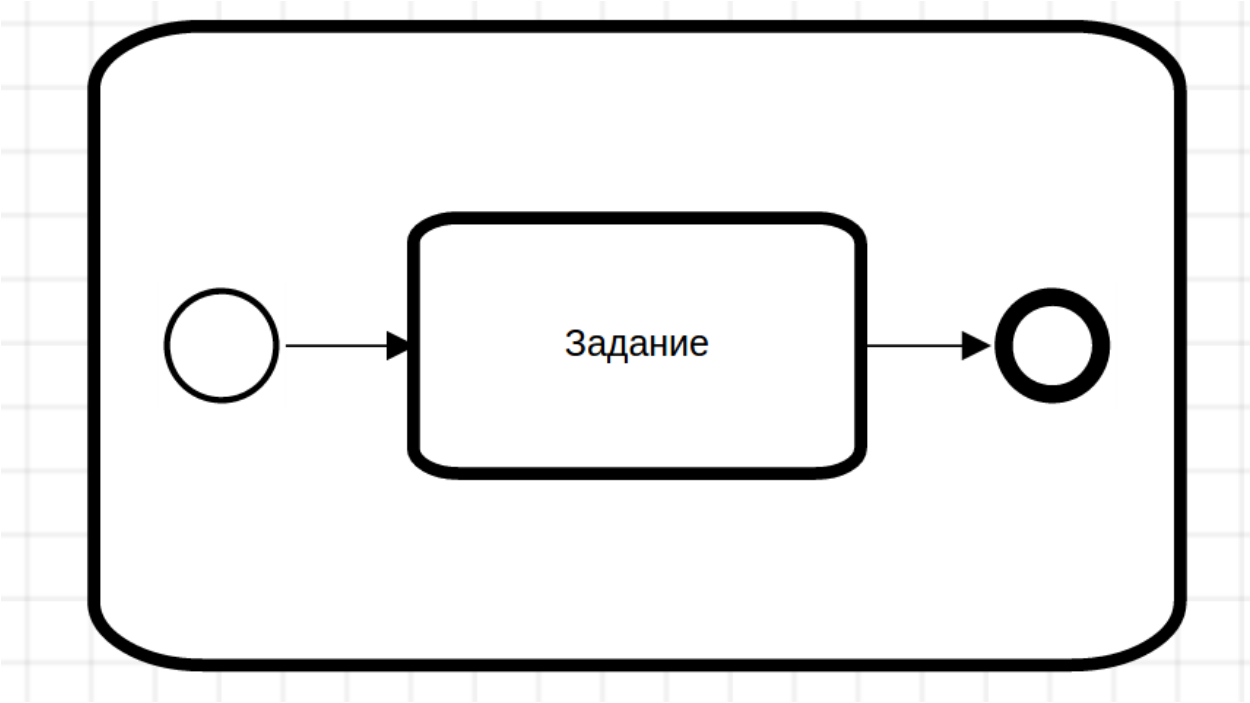

Рис. 5: Подпроцесс с собственной диаграммой и заданием внутри

того, дорожки могут сами делиться на дорожки, тем самым разделяя отделы на собственные подотделы (рис. 6).

В диаграмме и пул, и дорожка являются контейнерными элементами, но содержать они могут не все типы элементов, и содержаться они могут не в любых контейнерных элементах. Пулы могут содержать только дорожки; дорожки могут содержать все элементы, кроме пулов.

Стоит заметить, что пул является очень сложным для реализации элементом. В дополнение к требованиям, предъявляемым к контейнерным элементам, был составлен список новых. Во-первых, при изменении дорожек значительная часть пула должна перестраиваться. При растяжении по горизонтали все дорожки должны выравниваться по правому краю изменяемой дорожки. При растяжении по вертикали должны смещаться все элементы с той стороны, в которую идет растяжение, и рекурсивно должны растянуться соответствующим образом все элементы, которые содержат изменяемый элемент или его контейнер.

Во-вторых, было бы очень полезно иметь возможность удалять и вынимать дорожки из любого места пула, а также вставлять их в любое

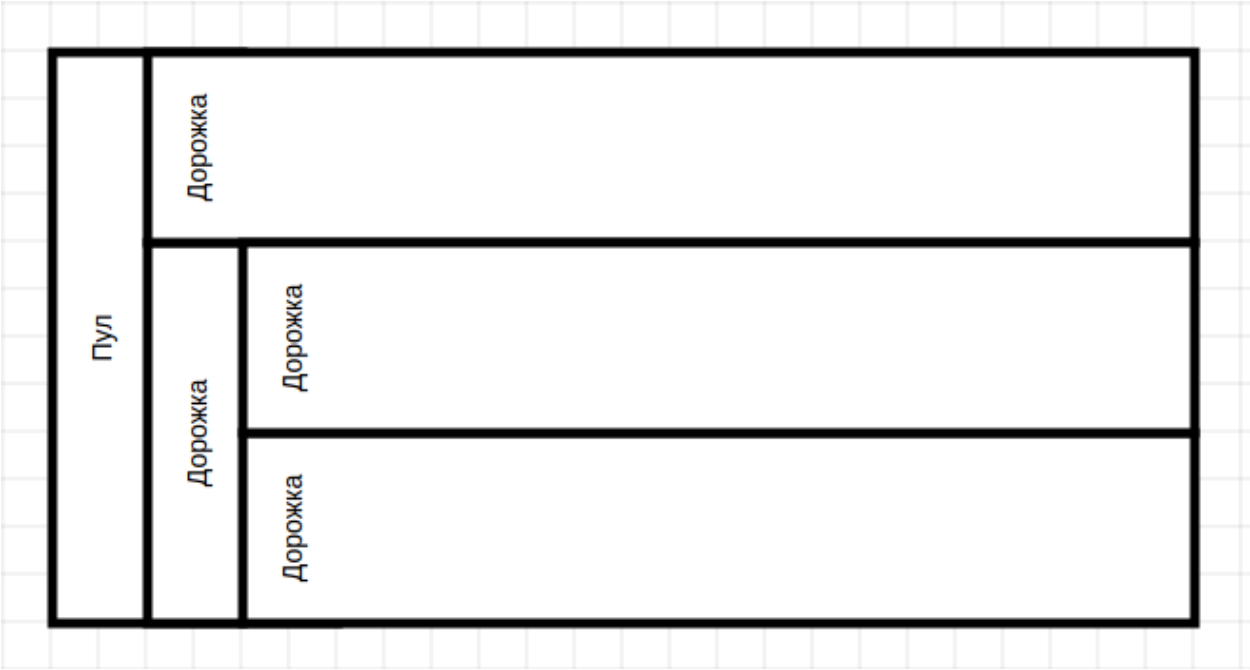

Рис. 6: Пулы и дорожки

место пула. Пул при всех этих изменениях должен перестраиваться, соответственно изменяя высоту и ширину дочерних элементов.

В-третьих, и пул, и дорожки имеют заголовок, отображаемый в их левой части. Этот заголовок повернут на 90 градусов против часовой стрелки. Таким образом, необходимо поддержать не только отображение текста в элементах, но и его отображение в заданном месте и под заданным углом.

В-четвертых, как и контейнерные элементы, пулы должны корректно работать с механизмом undo-redo. Эта задача является еще более сложной из-за того, как изменяется пул при изменениях любой из дорожек.

1.3. Существующие решения

Из существующих решений были рассмотрены самые популярные онлайн-редакторы BPMN, доступные бесплатно. Функциональность, описанная в пунктах 1.3.1-1.3.3 является базовой для любого редактора и поддерживается всеми популярными онлайн-редакторами BPMN. Однако функциональность пулов, описанная в пункте 1.3.4, не поддержи-

вается ни в одном из них целиком.

Самый популярный онлайн-редактор bpmn.io[1] поддерживает большинство из описанной ранее функциональности, но не позволяет вынимать дорожки из пула и вставлять их в нужное пользователю место пула.

Редактор Gliffy[2]. позволяет создавать только пулы с ограниченным количеством дорожек, которые не могут ни делиться, не перемещаться.

Редактор BPMN на веб-сайте draw.io[3] позволяет создать какую угодно диаграмму, но пул не всегда меняется при изменении дорожек внутри, и дорожки не разделяются на поддорожки.

Остальные просмотренные редакторы (GenMyModel, ModelWorld, Lucidchart) имеют те же самые или очень схожие проблемы. Как видно, основной проблемой онлайн-редакторов BPMN является статичность пулов в той или иной мере. В рамках данной работы была поставлена цель сделать пул как можно более изменяемым и подвижным элементом.

[1]Редактор BPMN bpmn.io, URL: http://bpmn.io/ (дата обращения: 25.04.17)

[2]Редактор диаграмм Gliffy, URL: https://www.gliffy.com/ (дата обращения: 25.04.17)

[3]Редактор диаграмм draw.io, URL: https://www.draw.io/ (дата обращения: 25.04.17)

2. Поддержка BPMN в ядре проекта WMP

2.1. Добавление необходимой функциональности

2.1.1. Поддержка подтипов

Прежде всего для поддержки подтипов событий нужно было выбрать формат, в которым подтипы будут заданы в конфигурационном файле. Был расширен уже имеющийся JSON-формат для описания типов. Подтипы событий BPMN в зависимости от возможного положения на диаграмме делятся на начальные (start), промежуточные (intermediate) и конечные (end). Эти группы делятся на конкретные подтипы. Так, например, выглядит описание подтипов события «сообщение»:

```
''subtypes'': {
    ''Start'' : [
        ''Regular'',
        ''Non-interrupting''
    ],
    ''Intermediate'': [
        ''Catch'',
        ''Non-interrupting'',
        ''Throw''
    ],
    ''End'': [
        ''Regular''
    ]
}
```

Заметим, что при таком формате в будущем можно будет задавать свойства для подгрупп элементов. Сейчас такая иерархия позволяет генерировать путь до соответствующей подтипу иконки по пути в дереве, а не прописывать его вручную.

Далее этот формат необходимо было прочесть и создать соответствующие описывающие подтипы объекты NodeType. Для этого была создана структура PaletteTree, которая хранит это дерево подтипов. Поскольку во всем проекте для хранения типов элементов до этого использовался контейнер Map, то для обратной совместимости добавлена возможность сгенерировать его из PaletteTree. Также стоит заметить, что разработанная система подтипов будет работать не только для событий BPMN, но и для сколь угодно глубокого дерева. Это может пригодиться при разработке других редакторов.

Также при наличии у типа подтипов каждому из подтипов добавляют свойство ChangeType. Это свойство отображается в редакторе слева, в редакторе свойств. Оно позволяет выбрать подтип уже выбранного элемента из выпадающего меню (рис. 7). При замене подтипа у элемента меняются присущие подтипу свойства, а также иконка для отображения в редакторе. Измененный подтип корректно сохраняется и загружается благодаря механизму свойств, уже реализованному в проекте.

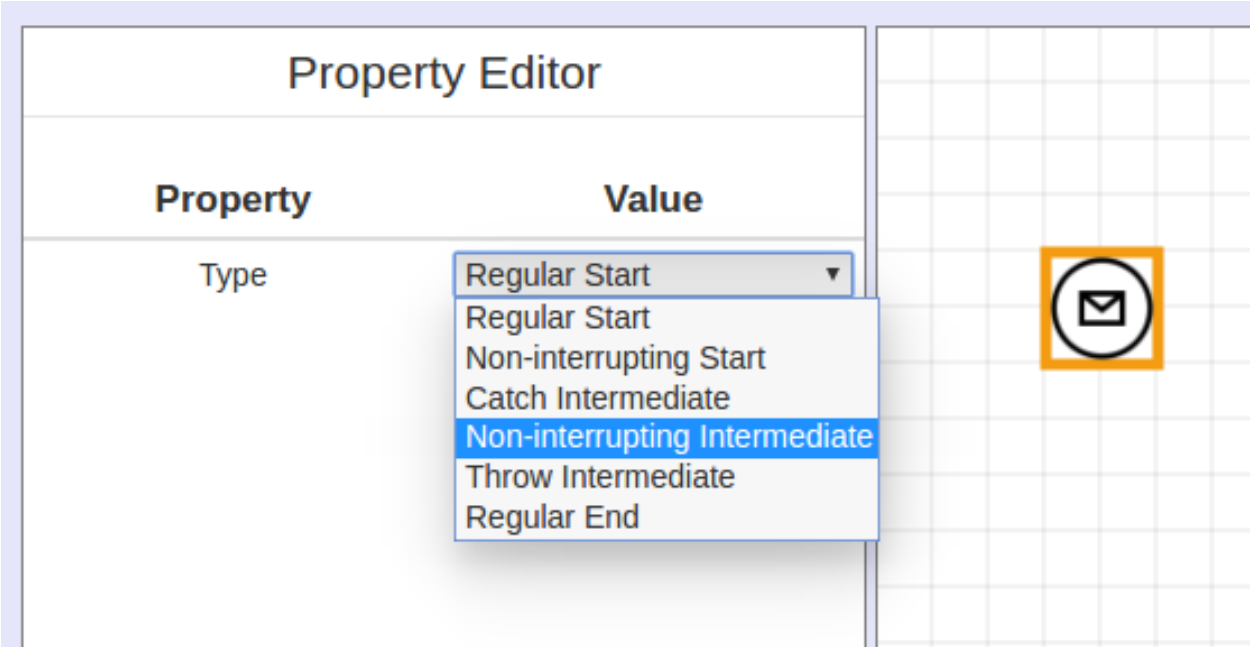

Рис. 7: Событие типа «сообщение» и выпадающее меню с его подтипами

2.1.2. Возможность выбора типа связи

Создание связей между элементами на диаграмме в библиотеке JointJS происходит с помощью объекта атрибутов в формате SVG [9]. Соответственно, чтобы создавать связи разных типов, необходимо создать шаблоны для этих объектов. Была реализована возможность написать шаблон в конфигурационном файле редактора как часть описания типа связи. Этот шаблон обрабатывается вместе с остальной информацией о связи в парсере типов при загрузке клиентской части проекта. Далее, список шаблонов передается в сцену редактора, которая хранит информацию обо всех объектах диаграммы, а также осуществляет контроль за добавлением и удалением этих объектов. При создании объекта связи конструктору передается соответствующий шаблон, который клонируется и используется в качестве объекта атрибутов. Например, так выглядит шаблон для описания потока сообщений (рис. 8):

```
''attrs'': {
    ''.connection'': { ''stroke'': ''black'' },
    ''.marker-source'': { ''fill'': ''white'',
        ''d'': ''M 10 0 L 0 5 L 10 10 L 20 5 z'' },
    ''.marker-target'': { ''fill'': ''black'',
        ''d'': ''M 10 0 L 0 5 L 10 10 z'' }
}
```

Для корректного сохранения и загрузки диаграмм к существующей таблице базы данных для хранения связей диаграмм был добавлен еще один атрибут — тип связи, и обновлены методы, которые позволяют отправлять и принимать данные об элементах диаграммы. Также в палитре элементов появилась новая вкладка, в которой можно увидеть список доступных связей (рис. 8). В этой панели можно выбрать тип текущего используей связи, а можно перетащить элемент с палитры на сцену. В последнем случае связь подсоединится к элементу, на который она был перетащена.

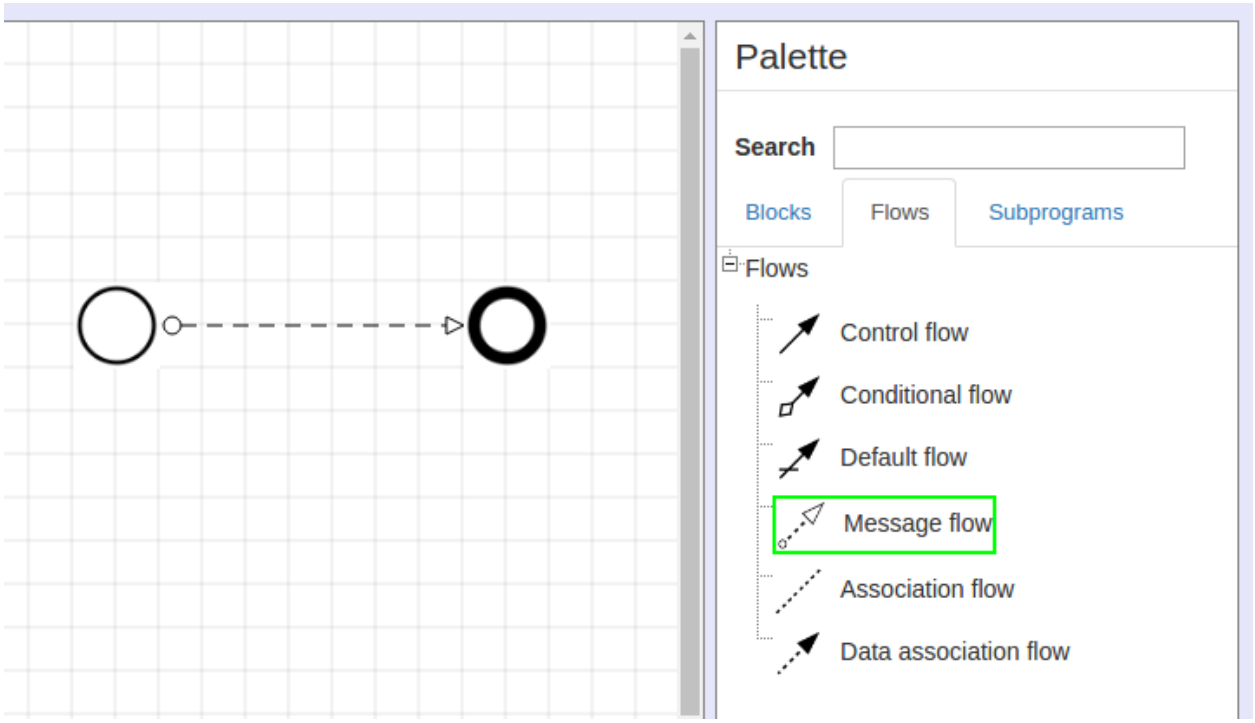

Рис. 8: События, соединенные потоком сообщений, и панель связей

2.1.3. Текстовые элементы

Благодаря возможностям библиотеки JointJS отображение текста внутри элемента не является очень сложной задачей. С помощью шаблона в SVG-формате, подобного шаблону для связей, можно создавать элементы с текстом внутри. В этом шаблоне возможно задать стиль и размер шрифта, выравнивание, сам отображаемый текст, а также его положение внутри элемента. Например, следующий шаблон задает текст черного цвета шрифта 14pt семейств «Arial, helvetica, sans-serif», расположенный ровно в середине элемента, с выравниванием посередине. Такой шаблон используется для текста элемента BPMN «задание»:

```
''innerText'': {
    ''fill'': ''#000000'',
    ''font-size'': 14,
    ''ref-x'': 0.5,
    ''ref-y'': 0.5,
    ''y-alignment'': ''middle'',
```

```
    ''text-anchor'': ''middle'',
    ''font-family'': ''Arial, helvetica, sans-serif''
}
```

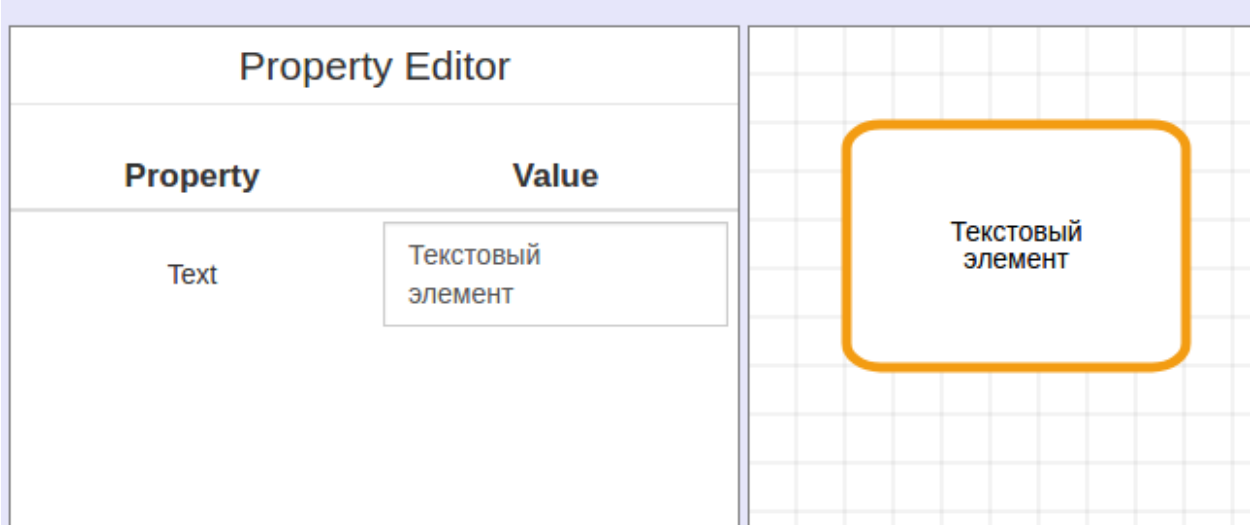

Рис. 9: Текстовый элемент и окно для ввода пользовательского текста

Таким образом, задача свелась к тому, чтобы дать пользователю возможность этот текст менять, а также корректно его сохранять и загружать. Для этого был использован уже имеющийся в проекте механизм свойств (properties). Был добавлен новый класс InnerTextView, который формирует HTML-элемент с тегом textarea для текста пользователя в редакторе свойств (рис. 9). Также в контроллер свойств (PropertyEditController) была добавлена обработка события изменения этого HTML-элемента, в ходе которой изменяется текст внутри элемента диаграммы. Если текст сохранять и загружать так же, как и другие свойства, то проблем с текстом не возникнет.

2.1.4. Контейнерные элементы

Класс DiagramScene, отвечающий за непосредственную работу с элементами диаграммы, является наследником класса joint.dia.Paper из библиотеки JointJS. Начиная с JointJS версии 1.0, есть возможность при создании сцены задать параметр «embeddingMode». Если он равен «true», то активируется режим «автоматического наследования». Это

значит, что при если переместить элемент диаграммы над некоторым другим и отпустить кнопку мыши, то перемещенный элемент автоматически станет ребенком того, на который он был перемещен. Это, в свою очередь, значит, что при перемещении или удалении родительского элемента аналогичное действия рекурсивно совершат все его дети. За корректность такого присоединения отвечает функция isValidEmbedding, которую также можно задать при создании объекта сцены. Если функция вернет false, то такого присоединения не произойдет. В рамках данной работы был создан новый класс для контейнерных элементов DiagramContainer, наследующий класс DefaultDiagramNode, содержащий информацию о своих детях. Таким образом, функция isValidEmbedding равна true тогда, когда элемент, к которому пытаются присоединить другой элемент, является объектом класса DiagramContainer. Чтобы элемент был контейнерным, в конфигурационном файле элементу необходимо задать опцию «container».

Такая функциональность, безусловно, упрощает задачу реализации контейнерных элементов. Однако, остается проблема сохранения и загрузки диаграмм, а также поддержки механизма undo-redo. Для сохранения и загрузки в соответствующей Thrift-структуре было добавлено дополнительное поле parent, содержащее ID родителя данного элемента, а также импорт и экспорт этого значения как на клиентской, так и на серверной стороне. Значение parent можно узнать и задать с помощью методов JointJS.

Несколько сложнее поддержать механизм undo-redo. Теперь при изменении состояния элемента рекурсивно запоминаются состояния всех его дочерних элементов, в том числе все ассоциации, которые были связаны хотя бы с одним из них. При удалении запоминаются сами элементы, при перемещении — их координаты. Из сформированных при этом отдельных команд получается одна – MultiCommand. Однако этого недостаточно, так как при перемещении элемента у него может измениться родитель. Для этого каждому элементу было добавлено поле parent, указывающее на его родителя. Если у связанного с элементом объекта Joint родитель поменяется автоматически, то

это поле продолжит хранить старое значение. Если после перемещения оказалось, что родитель объекта Joint не совпадает с элементом в поле parent, то в MultiCommand будет добавлена команда нового типа — EmbedCommand, которая отвечает за изменение родительской связи.

Также для удобства была реализована возможность присоединить элемент к контейнеру сразу при выставлении его на сцену. Элемент становится ребенком того контейнера, с каким он пересекается и чей z-index больше всех (то есть, кто визуально на диаграмме находится выше всех).

2.1.5. Конструктор элементов

Вся реализованная до этого функциональность относилась к ядру платформы, и доступна во всех редакторах, но пулы и дорожки являются слишком специфичными элементами, и их реализацию в ядро помещать не стоит. Однако создание всех элементов происходит в ядре, в классе SceneController. Появилась проблема: не изменяя значительно структуру кода, дать возможность в ядре создавать элементы, классов для которых в ядре нет.

Для этого был создан класс ElementConstructor. Он по набору данных создает и возвращает необходимый элемент, реализующий интерфейс DiagramNode. Ссылку на объект этого класса хранит DiagramController. Для того, чтобы создавать элементы BPMN (или другие специфичные элементы в других редакторах), реализация DiagramEditorController для редактора некоторого языка заменяет ссылку конструктора элементов на наследника этого класса, реализованного для этого редактора. Такое решение также централизует создание элементов, что уменьшает повторяемость кода.

2.2. Добавление дополнительной функциональности

Отображение подтипов элементов палитры. Стандарт BPMN имеет большое количество событий и их подтипов. Отображение их в палитре элементов сплошным списком, где разные подтипы разных эле-

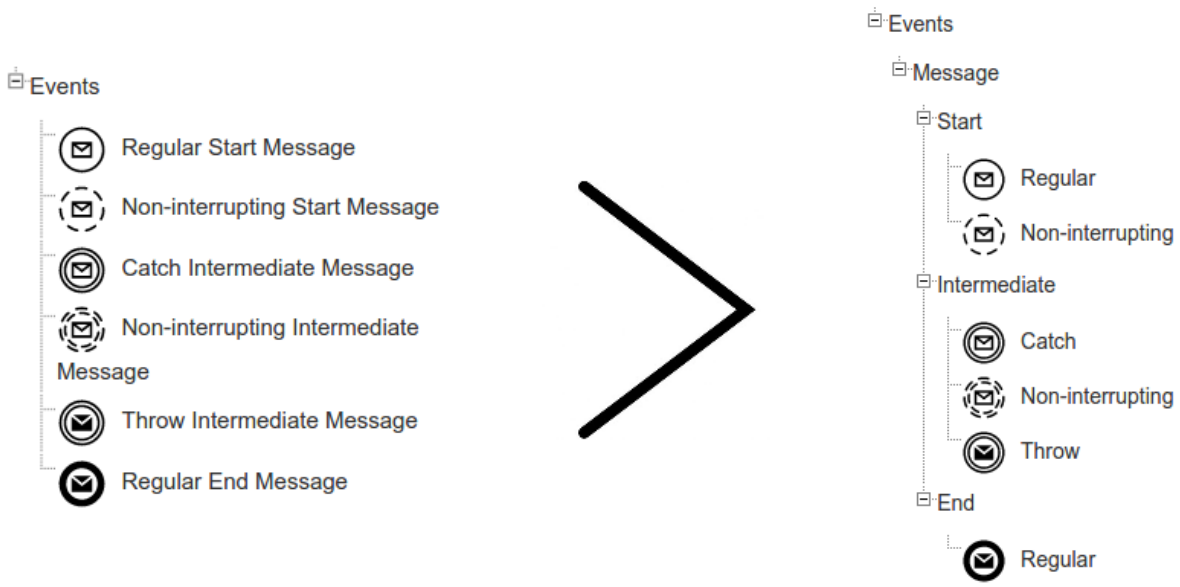

Рис. 10: Список событий без использования дерева элементов и с ним

ментов стоят рядом, будет тяжелым для восприятия пользователя. Поэтому было реализовано иерархическое отображение элементов в виде дерева: в корне стоит название самого события, оно делится на типы «начальное», «промежуточное» и «конечное», а те в свою очередь содержат конкретные подтипы (рис. 10). Для отображения иерархии используется дерево подтипов PaletteTree, описанное в пункте 2.1.1.

Поиск элементов палитры. Так как элементов BPMN много, все они могут не помещаться в окно палитры, и долгая прокрутка этого окна может быть неприятной. Поэтому был реализован поиск, оставляющий в палитре только элементы, подходящие под написанный шаблон (рис. 11).

Поиск разбивает написанный пользователем шаблон по символу пробела на несколько подшаблонов. Элемент считается подходящим под шаблон, если каждый подшаблон содержится как подстрока в пути элемента в дереве подтипов. Таким образом, можно искать не только по названию элемента, но и оставлять в поиске, к примеру, только стартовые события. Палитра мгновенно перерисовывается при любом изменении строки поиска, что очень удобно при составлении диаграмм.

Отображение границы элементов диаграмм. До начала дан-

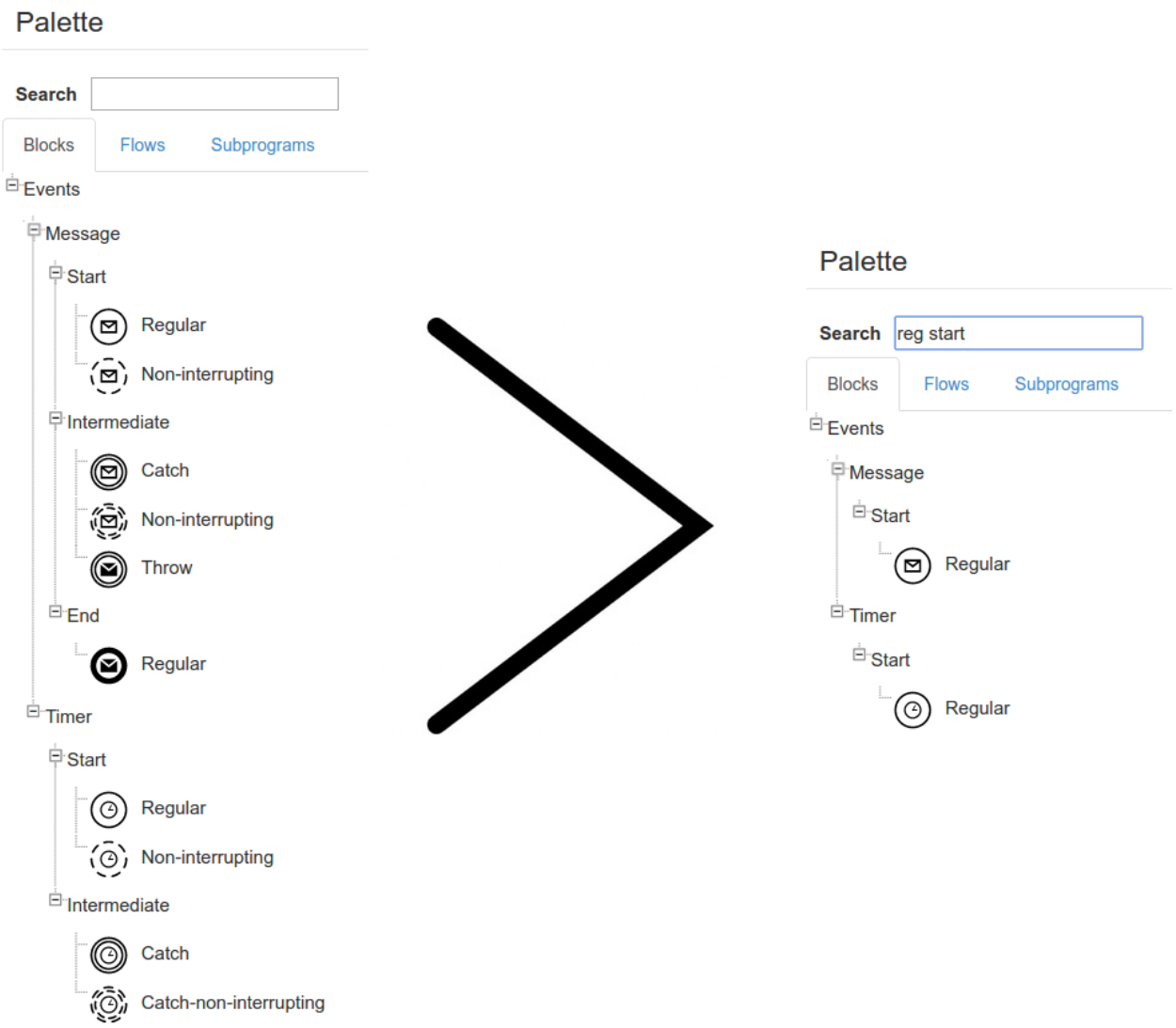

Рис. 11: Применение поиска элементов палитры

ной работы каждый элемент диаграммы визуально был статичной картинкой, и до появления контейнерных элементов это не было проблемой. Контейнерные элементы предполагают наличие границы у элемента, и возможность этот элемент расширять. При увеличении элемента пропорционально увеличивалась и картинка. Граница контейнерного элемента на картинке тоже увеличивалась соответственно, и при сильном растяжении элемента граница на этой картинке занимала огромное количество места и выглядела совершенно не к месту. Было принято решение для контейнерных элементов не использовать картинку, а дать возможность отображать границу с помощью CSS. Для этого в конфигурационном файле редактора появилась возможность задать такую

границу и скрыть картинку при помощи опции «isHidden». Так, например, выглядит описание границы элемента «подпроцесс».

```
''border'': {
    ''stroke'': ''black'',
    ''stroke-width'': ''5px'',
    ''rx'': ''5px'',
    ''ry'': ''5px''
}
```

Это решение не только решило вышеописанную проблему, но и дало возможность подсветить границу контейнерного элемента, что пригодилось для визуального выделения выбранного пользователем элемента диаграммы.

Использование z-индекса. Элементы диаграммы могут пересекаться и накладываться друг на друга, особенно при использовании контейнерных элементов. Ранее сохраненные элементы диаграммы загружались в произвольном порядке, и контейнерный элемент мог визуально оказаться выше своих детей. Теперь z-индекс элементов корректно сохраняется и загружается.

Перекрытие ассоциаций другими элементами. Ранее при выделении контейнерного элемента связи, содержащиеся в нем лишь частично, перекрывались этим контейнерном, что было абсолютно нелогично. Теперь при любом изменении z-индекса элемента обновляется z-индекс связанных с ним связей, равный максимальному из z-индексов его начального и конечного элемента.

Сокрытие элементов из палитры. Если есть желание не давать пользователю возможность создания каких-либо элементов, а генерировать их только программно, то с помощью опции invisible в конфигурационном файле можно скрыть элемент из палитры.

Запрет на растяжение элемента. Пользователю можно запретить растягивать элементы некоторых типов. Для этого необходимо указать опцию «static» в конфигурационном файле редактора.

Исправление ошибок. В процессе работы было исправлено несколько ошибок в проекте, вызванные его переходом на новые технологии. Восстановлена работа в браузере Firefox, растяжение элементов, рисование жестов и создание связей с помощью правой кнопки мыши после перехода на JointJS версии 1.1. Также исправлена работа с контейнерами Map из ECMAScript 6. Также добавлена возможность объединять контейнеры Map, что очень часто используется в проекте.

3. Редактор диаграмм BPMN

3.1. Прототип редактора

Первым необходимым этапом было создание пустого редактора, который впоследствии будет развиваться. По аналогии с редактором роботов, был создан класс BPMNDiagramEditorController, наследующий класс DiagramEditorController из ядра проекта (рис. 12).

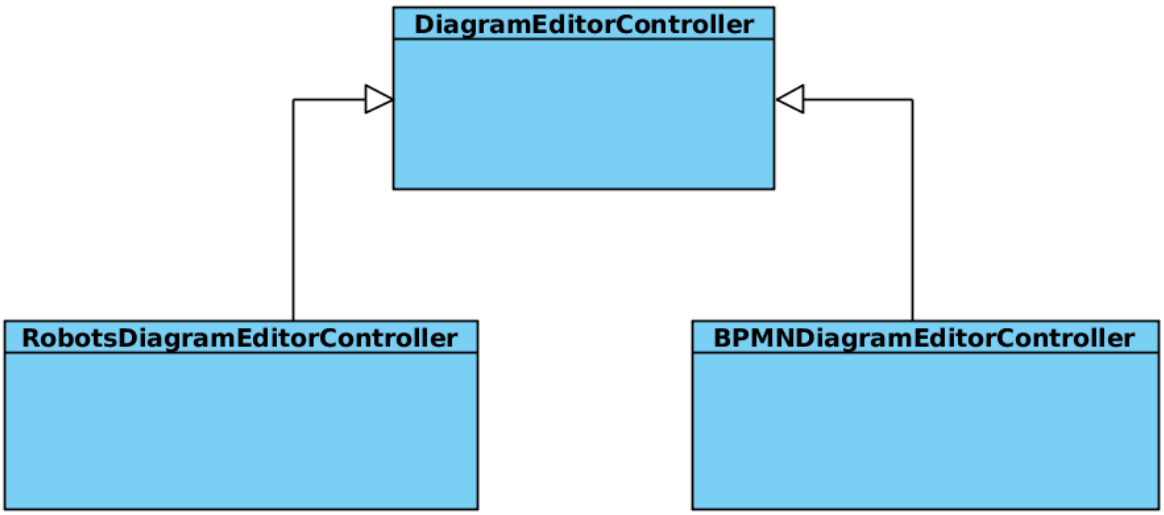

Рис. 12: Диаграмма классов для контроллеров редакторов

В этом контроллере создаются основные объекты и задаются основные процессы, необходимые для работы редактора. Кроме того, здесь инициализируется ранее реализованный поиск элементов палитры.

Далее необходимо было сделать небольшие исправления на серверной части: дать пользователю возможность загружать оба редактора по разным URL и доработать запрос о загрузке типов элементов редактора так, чтобы он мог посылать типы различных языков диаграмм. Ссылки на оба редактора стали доступны на главной панели проекта (dashboard).

3.2. Элементы BPMN

После реализованной в ядре функциональности поддержать события, логические операторы, потоки и задания BPMN не представляет труда. Для этого нужно лишь корректно занести данные о них в конфигурационный файл редактора, а также загрузить иконки для их отображения. В данный момент поддержаны все события и их подтипы, все логические операторы и потоки. Также поддержаны базовые действия: задание и развернутый подпроцесс. Реализация других действий более специфична и в поставленные задачи не входила.

3.3. Пулы и дорожки

3.3.1. Реализация

Для создания пулов был реализован класс BPMNElementConstructor, являющийся наследником класса ElementConstructor, описанным в пункте 2.1.5. При создании пула создается два элемента — заголовок пула и дорожка, связанная с ним. Заголовок скрыт из палитры элементов, и самостоятельно его создать нельзя. Для создания сразу двух элементов в SceneController была реализована возможность создавать элемент и его детей вместе, одной командой.

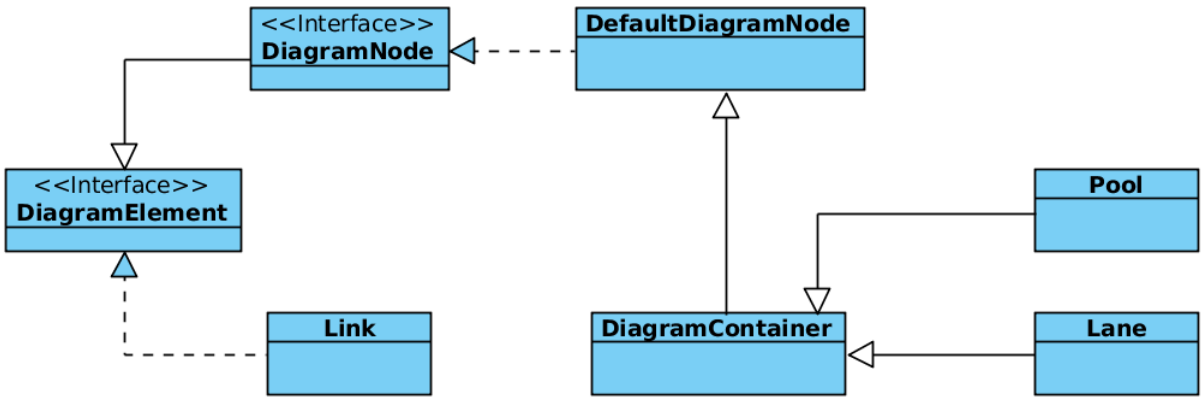

Рис. 13: Диаграмма классов для элементов BPMN

Классы Pool и Lane, реализующие функциональность пулов и дорожек соответственно, являются наследниками класса DiagramContainer

(рис. 13). Реализованная в них функция isValidEmbedding ограничивает типы элементов, которые могут быть их детьми. Дорожки при любом растяжении запускают алгоритм обновления пула. Он также запускается при добавлении и удалении детей пула. Этим гарантируется работа механизма undo-redo.

С помощью реализованных пулов стало возможным создавать диаграммы сложных бизнес-процессов (рис. 14).

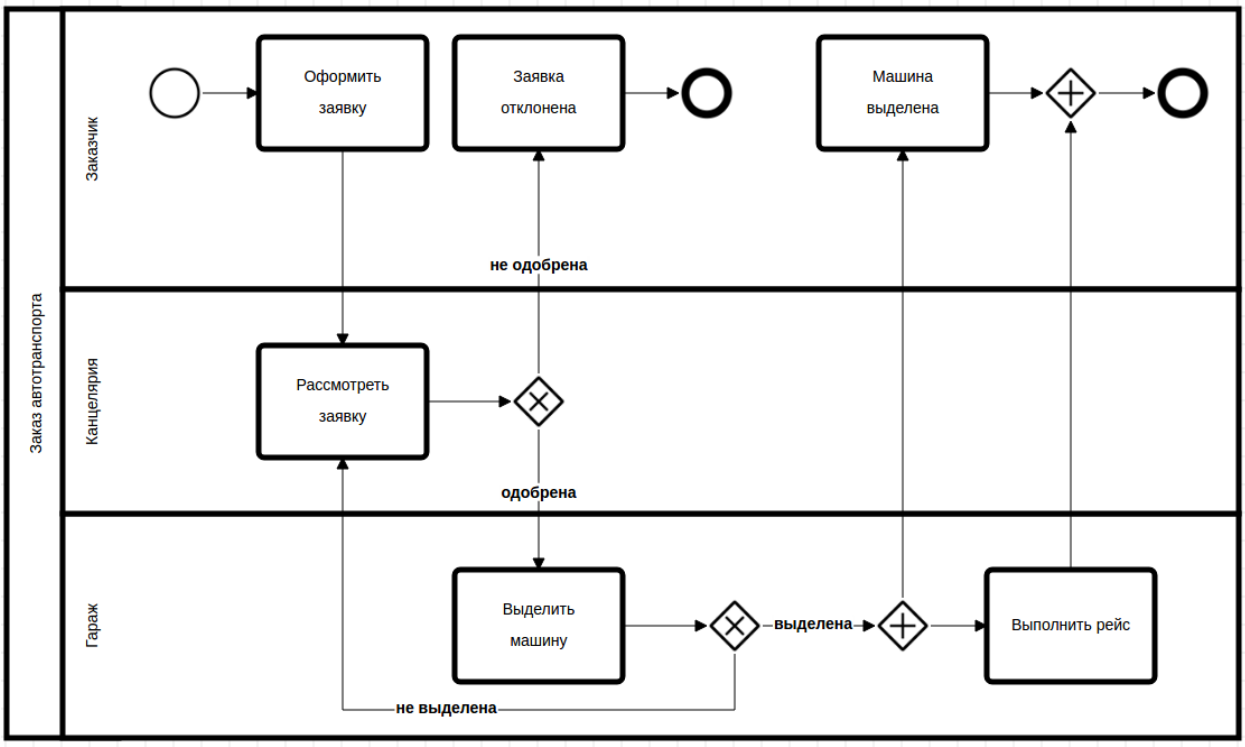

Рис. 14: Диаграмма бизнес-процесса заказа автотранспорта

3.3.2. Алгоритм обновления пула

При каждом изменении одного из своих детей пул должен перестраиваться практически полностью. Для этого был разработан алгоритм обновления размеров пула и всех его детей.

Во-первых, при изменении элемента необходимо найти сам пул, который будет изменяться. Для этого нужно просто переходить по ссылке parent до тех пор, пока она не окажется пустой.

Во-вторых, нужно обновить высоту всех дорожек рекурсивно. Для этого:

1. отсортируем список всех детей по y-координате их центра;
2. запустим алгоритм обновления высоты рекурсивно для каждого непосредственного ребенка;
3. если у элемента нет детей, то высота остается прежней; иначе его высота равна сумме высот всех детей;
4. переместим детей на их места в элементе согласно предварительно отсортированному списку.

Заметим, что важно сначала отсортировать детей, а потом обновлять их высоту; в противном случае у расширившейся дорожки сверху центр может переместиться ниже, чем у дорожки снизу, и дорожки поменяются местами.

В-третьих, необходимо обновить длину всех дорожек. Если обновление произошло по причине того, что одна их дорожек изменила длину, то обновлять необходимо по правому краю этой дорожки. Если по причине того, что дорожку добавили в пул, то обновление будет происходить по самому правому краю всех дорожек. Для этого нужно рекурсивно спуститься и получить список всех крайних дорожек. Все дорожки из этого списка выравниваются по правому краю, изменяя свой размер соответствующим образом.

3.3.3. Работа с пулами

Создаются пулы и дорожки как и все остальные элементы путем перетаскивания на сцену из палитры элементов. Перетаскиваются пулы за заголовок вне зависимости от положения на диаграмме. Дорожки перетаскиваются как обычные элементы. Если потащить дорожку, подсоединенную к пулу, то она вытащится из пула.

Для того, чтобы добавить дорожку в пул, можно либо перетащить ее до пересечения с заголовком пула, либо поместить дорожку из палитры на этот заголовок. Для создания разделенных дорожек нужно перетащить элемент пула и подсоединить его к заголовку другого пула.

Заключение

В ходе данной работы были получены следующие результаты.

- Определено подмножество BPMN для дальнейшей реализации.
- Расширена необходимыми инструментами платформа WMP.
  - Поддержано задание подтипов элементов и смена подтипа элемента на сцене.
  - Поддержано создание разных типов связей и добавлена панель связей.
  - Поддержаны контейнерные элементы.
- Реализован BPMN-редактор на базе платформы WMP.
- Исправлен ряд ошибок в платформе.
  - Восстановлена корректная работа приложения в браузере Mozilla Firefox.
  - Восстановлена возможность рисовать жесты и создавать связи с помощью правой кнопки мыши.
  - Восстановлена корректная работа механизма растяжения элементов.

Список литературы

[1] Apache Software Foundation. Apache Thrift. — 2016. — URL: https://thrift.apache.org/ (дата обращения: 20.05.2017).

[2] Bruce Silver. BPMN Method and Style: A levels-based methodology for BPM process modeling and improvement using BPMN 2.0. — 2009. — ISBN: 0982368100.

[3] CYBERTECH CO. LTD. TRIK Studio. — 2015. — URL: http://www.trikset.com/index_en.html (дата обращения: 20.05.2017).

[4] David Flanagan. JavaScript: The Definitive Guide. — O'Reilly Media, 2011. — ISBN: 978-0-596-80552-4.

[5] Group Object Management. Business Process Model and Notation (BPMN). — 2011. — URL: http://www.omg.org/spec/BPMN/2.0/PDF (дата обращения: 20.04.2017).

[6] Oracle. JavaServer Pages Technology. — URL: http://www.oracle.com/technetwork/java/javaee/jsp/index.html.

[7] QReal Research Group. Web Modeling Project (WMP). — 2017. — URL: https://github.com/qreal/wmp (дата обращения: 16.05.2017).

[8] Steve Fenton. Pro TypeScript: Application-Scale JavaScript Development. — Apress, 2014. — ISBN: 9781430267911.

[9] W3. Scalable Vector Graphics (SVG). — URL: https://www.w3.org/TR/SVG/ (дата обращения: 20.05.2017).

[10] Wikipedia. BPMN // Википедия, свободная энциклопедия. — 2017. — URL: https://ru.wikipedia.org/wiki/BPMN (дата обращения: 16.05.2017).

[11] Безгузиков Артемий Валерьевич. Микросервисная архитектура QReal-Web. — 2016. — URL: http://se.math.spbu.ru/SE/

YearlyProjects/spring-2016/344/344-Bezguzikov-report.pdf (дата обращения: 20.04.2017).

[12] Библиотека JointJS. — URL: https://www.jointjs.com/opensource (дата обращения: 20.04.2017).

[13] Кознов Д. В. Визуальное моделирование информационных e-сервисов в публичной сфере. — С.-Петерб. ун-та, 2014.

[14] Нотация BPMN. — URL: http://www.businessstudio.ru/wiki/docs/current/doku.php/ru/csdesign/bpmodeling/bpmn_notation (дата обращения: 20.04.2017).