Федеральное государственное унитарное предприятие
Российский федеральный ядерный центр -
Всероссийский научно-исследовательский институт экспериментальной физики

УТВЕРЖДЕН
07623615.00435-03 33 01 -ЛУ

КОМПЛЕКС ПРОГРАММ В ЗАЩИЩЕННОМ ИСПОЛНЕНИИ
«СИСТЕМА ПОЛНОГО ЖИЗНЕННОГО ЦИКЛА ИЗДЕЛИЙ
«ЦИФРОВОЕ ПРЕДПРИЯТИЕ»

Программный модуль

«Система моделирования жизненного цикла изделий»

Руководство программиста

07623615.00435-03 33 01

Листов 67

Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

2021

АННОТАЦИЯ

В данном документе приведено руководство программиста по работе с программным модулем «Система моделирования процессов жизненного цикла изделий» (далее – программный модуль BPMS).

В разделе «Назначение и условия применения программы» содержатся сведения о назначении и целях программного модуля BPMS, приведены сведения о минимальном составе программных и технических средств, а также требования к квалификации программиста.

В разделе «Характеристики программного модуля BPMS» содержатся сведения об исходном коде программного модуля, составе его дистрибутива, инструментах среды программирования и среды исполнения, а также о режимах его работы.

В разделе «Обращение к программе» описан процесс подготовки рабочего места программиста, а также сборки, конфигурировании и запуске программного модуля BPMS.

Раздел «Рекомендации по разработке» описывает основные принципы, используемые при работе программиста в программном модуле BPMS для изменения текущих функциональных возможностей компонентов программного модуля BPMS, создания новых функциональных возможностей, а также сборки этих компонентов из исходных кодов.

В разделе «Входные и выходные данные» приведены сведения о составе и форматах внешних данных, с которыми способен взаимодействовать программный модуль BPMS.

Настоящий документ составлен в соответствии с требованиями ГОСТ 19.504-79 Межгосударственный стандарт. Единая система программной документации. Руководство программиста. Требования к содержанию и оформлению.

СОДЕРЖАНИЕ

1. НАЗНАЧЕНИЕ И УСЛОВИЯ ПРИМЕНЕНИЯ ПРОГРАММНОГО МОДУЛЯ BPMS

1.1. Назначение программного модуля BPMS

Программный модуль «Система моделирования процессов жизненного цикла изделий» (далее – программный модуль BPMS) предназначен для моделирования бизнес-архитектуры, поддержания процессного управления, реинжиниринга и оптимизации процессов, анализа и автоматизации деятельности. Под процессным управлением понимается формирование целенаправленного поведения организации посредством выделения, описания и менеджмента системы взаимосвязанных и взаимодополняющих процессов деятельности и их ресурсного окружения.

Основные задачи, решаемые программным модулем BPMS – управление процессами жизненного цикла сложных инженерных изделий, оценка организации процессов и распределение ресурсов на этапах жизненного цикла изделий.

Программный модуль BPMS обеспечивает полноту и прозрачность моделируемых процессов, а также удобство в управлении изменениями в моделях процессов.

1.2. Условия применения программного модуля BPMS

1.2.1. Требования к аппаратному и программному обеспечению рабочих станций

Для функционирования программного модуля BPMS необходимо обеспечить наличие персонального компьютера, удовлетворяющего следующим условиям:

- двухъядерный процессор с архитектурой x64 и частотой не менее 1,8 ГГц;
- материнская плата на базе набора микросхем компании Intel или совместимых;
- объем оперативной памяти не менее 8 Гб;
- сетевой адаптер Ethernet PCI 10/100Base-T (или совместимая);
- HDD/SSD не менее 512 Mb;
- видеокарта с поддерживаемым разрешением не менее 1920x1080 пикселей;
- монитор ЖК с диагональю экрана не менее 19 дюймов;

- компьютерная клавиатура;

- компьютерная мышь.

К программному обеспечению рабочих станций программиста BPMS предъявляются следующие требования:

- операционная система (ОС) – Astra Linux Special Edition версии 1.6 («Смоленск») и выше, либо Microsoft Windows x64 версии 8.1 и выше;

- Java 8 (LTS) JRE;

- Apache Maven, версия 3.6.3 Tycho;

- интернет-браузер, позволяющий просматривать файлы формата html;

- пакет офисных программ, соответствующий выбранной ОС.

1.2.2. Требования к квалификации программиста

Основными задачами программиста являются:

- изменение текущих функциональных возможностей инструментов программного модуля BPMS;

- создание новых функциональных возможностей инструментов программного модуля BPMS;

- сборка усовершенствованных инструментов программного модуля BPMS из исходных кодов.

К программисту программного модуля BPMS предъявляются следующие требования:

- знание языков программирования Java, QVT, M2T, JavaScript;

- опыт работы с платформой Eclipse;

- знание технологий построения ПО: ECORE, EMF, Acceleo, Eclipse Sirius;

- навыки метамоделирования (MOF).

2. ХАРАКТЕРИСТИКИ ПРОГРАММНОГО МОДУЛЯ BPMS

Программный модуль BPMS поставляется на компакт-дисках в двух вариантах исполнения:

- исходный код программного модуля;

- дистрибутив, пригодный для установки и последующего запуска инструментов программного модуля BPMS.

Дополнительно поставляется стороннее ПО: Java8(LTS) JDK, OpenJDK 8u265b11, Java8(LTS) JRE, Apache Maven (версия 3.6.3, Tycho).

2.1. Исходный код программного модуля BPMS

Каталог с файлами исходного кода программного модуля BPMS содержится в архиве «bpms.zip». Состав архива «bpms.zip» приведен в таблице 1.

Таблица 1 – Файлы исходного кода программного модуля BPMS в составе архивного файла «bpms.zip»

Название	Описание
bundles	Основные модули, составляющие реализацию инструментов BPMS Modeler и BPMS Notator
examples	Тестовые программные модули
features	Вспомогательные проекты, используемые для группировки проектов из папки «bundles». Features объединяют схожие по назначению проекты в группы
products	Определения инструментов BPMS Modeler и BPMS Notator, содержат информацию о том какие модули из папки «bundles» используются в каждом из инструментов
releng	Вспомогательные проекты, используемые при разработке инструмента
.mvn	Пространство сборки

Каталог «bundles» содержит основные модули, составляющие реализацию инструментов BPMS Modeler и BPMS Notator, указанные в таблице 2.

Таблица 2 – Каталог «bundles»

Название	Описание
bpms.modeler.help.doc	Справочная документация инструмента моделирования
org.eclipse.gef.nl_ru	Сообщения на русском языке для плагина «org.eclipse.gef»
org.eclipse.gmf.runtime.diagram.ui.actions.nl_ru	Сообщения на русском языке для плагина «org.eclipse.gmf. runtime. diagram.ui. actions»
org.eclipse.gmf.runtime.diagram.ui.nl_ru	Сообщения на русском языке для плагина «org.eclipse.gmf. runtime. diagram.ui»

Продолжение таблицы 2

Название	Описание
org.eclipse.gmf.runtime.diagram.ui.properties.nl_ru	Сообщения на русском языке для плагина «org.eclipse.gmf.runtime.diagram.ui.properties»
org.eclipse.gmf.runtime.diagram.ui.providers.nl_ru	Сообщения на русском языке для плагина «org.eclipse.gmf.runtime.diagram.ui.providers»
org.eclipse.sirius.common.ui.ext.nl_ru	Сообщения на русском языке для плагина «org.eclipse.sirius.common.ui.ext»
org.eclipse.sirius.common.ui.nl_ru	Сообщения на русском языке для плагина «org. eclipse. sirius. common.ui»
org.eclipse.sirius.diagram.nl_ru	Сообщения на русском языке для плагина «org. eclipse. sirius. diagram»
org.eclipse.sirius.diagram.ui.ext.nl_ru	Сообщения на русском языке для плагина «org.eclipse.sirius.diagram.ui.ext»
org.eclipse.sirius.diagram.ui.nl_ru	Сообщения на русском языке для плагина «org. eclipse. sirius. diagram.ui»
org.eclipse.sirius.table.ui.nl_ru	Сообщения на русском языке для плагина «org. eclipse. sirius. table.ui»
org.eclipse.sirius.tree.ui.nl_ru	Сообщения на русском языке для плагина «org. eclipse. sirius. tree.ui»
org.eclipse.sirius.ui.editor.nl_ru	Сообщения на русском языке для плагина «org. eclipse. sirius.ui. editor»
org.eclipse.sirius.ui.ext.nl_ru	Сообщения на русском языке для плагина «org.eclipse.sirius.ui.ext»
org.eclipse.sirius.ui.nl_ru	Сообщения на русском языке для плагина «org. eclipse. sirius.ui»
org.eclipse.sirius.ui.properties.nl_ru	Сообщения на русском языке для плагина «org. eclipse. sirius.ui. properties»
org.eclipse.ui.views.nl_ru	Сообщения на русском языке для плагина «org. eclipse.ui. views»
ru.vniief.bpms.codegen	Преобразования моделей и генераторы документов на основе моделей
ru.vniief.bpms.codegen.ui	Пользовательский интерфейс для преобразований моделей и генераторы документов на основе моделей
ru.vniief.bpms.eef.core.ext.widgets.mask	Контроллер для поля ввода текста с использованием маски
ru.vniief.bpms.eef.ext.widgets.mask	Метамодель для описания полей для ввода текста с использованием маски (для EEF)
ru.vniief.bpms.eef.ide.ui.ext.widgets.mask	Поле ввода текста с использованием маски
ru.vniief.bpms.joda.money	Модуль для работы с денежным типом данных
ru.vniief.bpms.methodology	Метамодель методологий моделирования
ru.vniief.bpms.methodology.design	Графическая нотация для методологий моделирования
ru.vniief.bpms.methodology.edit	Базовая функциональность для редактирования методологий моделирования
ru.vniief.bpms.methodology.editor	Древовидный редактор методологий моделирования
ru.vniief.bpms.methodology.free	Свободная методология моделирования

Продолжение таблицы 2

Название	Описание
ru.vniief.bpms.methodology.rosatom	Методология моделирования «Росатома»
ru.vniief.bpms.model.core	Базовая метамодель
ru.vniief.bpms.model.core.design	Графическая нотация для базовых объектов моделей
ru.vniief.bpms.model.core.edit	Базовая функциональность для редактирования базовых объектов моделей
ru.vniief.bpms.model.core.editor	Древовидный редактор базовых объектов моделей
ru.vniief.bpms.model.epc	Метамодель EPC-моделей
ru.vniief.bpms.model.epc.design	Графическая нотация для EPC-моделей
ru.vniief.bpms.model.epc.edit	Базовая функциональность для редактирования EPC-моделей
ru.vniief.bpms.model.epc.editor	Древовидный редактор EPC-моделей
ru.vniief.bpms.model.function	Метамодель перечня функций
ru.vniief.bpms.model.function.design	Графическая нотация для перечня функций
ru.vniief.bpms.model.function.edit	Базовая функциональность для редактирования перечня функций
ru.vniief.bpms.model.function.editor	Древовидный редактор перечня функций
ru.vniief.bpms.model.goal	Метамодель дерева целей и показателей
ru.vniief.bpms.model.goal.design	Графическая нотация для дерева целей и показателей
ru.vniief.bpms.model.goal.edit	Базовая функциональность для редактирования дерева целей и показателей
ru.vniief.bpms.model.goal.editor	Древовидный редактор дерева целей и показателей
ru.vniief.bpms.model.org	Метамодель организационной структуры
ru.vniief.bpms.model.org.design	Графическая нотация для организационной структуры
ru.vniief.bpms.model.org.edit	Базовая функциональность для редактирования организационной структуры
ru.vniief.bpms.model.org.editor	Древовидный редактор организационной структуры
ru.vniief.bpms.model.product	Метамодель перечня продуктов и услуг
ru.vniief.bpms.model.product.design	Графическая нотация для перечня продуктов и услуг
ru.vniief.bpms.model.product.edit	Базовая функциональность для редактирования перечня продуктов и услуг
ru.vniief.bpms.model.product.editor	Древовидный редактор перечня продуктов и услуг
ru.vniief.bpms.model.res	Метамодель перечня ресурсов
ru.vniief.bpms.model.res.design	Графическая нотация для перечня ресурсов
ru.vniief.bpms.model.res.edit	Базовая функциональность для редактирования перечня ресурсов
ru.vniief.bpms.model.res.editor	Древовидный редактор перечня ресурсов
ru.vniief.bpms.model.system	Метамодель перечня информационных систем
ru.vniief.bpms.model.system.design	Графическая нотация для перечня информационных систем
ru.vniief.bpms.model.system.edit	Базовая функциональность для ре-

Окочание таблицы 2

Название	Описание
	дактирования перечня информационных систем
ru.vniief.bpms.model.system.editor	Древовидный редактор перечня информационных систем
rug.vniief.bpms.model.vad1	Метамодель VAD моделей 1-го уровня
ru.vniief.bpms.model.vad1.design	Графическая нотация для VAD моделей 1-го уровня
ru.vniief.bpms.model.vad1.edit	Базовая функциональность для редактирования VAD моделей 1-го уровня
ru.vniief.bpms.model.vad1.editor	Древовидный редактор VAD моделей 1-го уровня
rub.vniief.bpms.model.vad2	Метамодель VAD моделей 2-го уровня
ru.vniief.bpms.model.vad2.design	Графическая нотация для VAD моделей 2-го уровня
ru.vniief.bpms.model.vad2.edit	Базовая функциональность для редактирования VAD моделей 2-го уровня
ru.vniief.bpms.model.vad2.editor	Древовидный редактор VAD моделей 2-го уровня
ru.vniief.bpms.modeler.product	Стартовые плагины приложений
ru.vniief.bpms.notator.product	Стартовые плагины приложений
ru.vniief.bpms.sirius.editor.properties.ext.widgets.mask	Поле для указания маски ввода текста
ru.vniief.bpms.sirius.ext	Расширения для Eclipse Sirius
ru.vniief.bpms.sirius.properties.ext. widgets. mask	Метамодель для описания полей для ввода текста с использованием маски (для Eclipse Sirius)
ru.vniief.bpms.sirius.properties.ext. widgets. mask. edit	Базовая функциональность для редактирования описания полей для ввода текста с использованием маски (для Eclipse Sirius)
ru.vniief.bpms.sirius.ui.properties.ext. widgets. mask	Преобразование спецификаций полей для ввода текста с использованием маски между EEF и Eclipse Sirius
ru.vniief.bpms.ui.explorer	Функциональность для работы с навигатором по проектам и моделям
ru.vniief.bpms.ui.project	Функциональность для работы с проектами
ru.vniief.bpms.util	Вспомогательные функции

Каталог «examples» содержит тестовые программные модули, состав указан в таблице 3.

Таблица 3 – Каталог examples

Название	Описание
import	Каталог с тестовыми AML-моделями
ru.vniief.bpms.reports	Каталог с проектом для генерации отчетности
templates	Каталог с шаблонами отчетности
БД_НМО	Каталог с тестовым проектом

2.2. Содержание дистрибутива программного модуля BPMS

Дистрибутив программного модуля BPMS поставляется в двух вариантах исполнения – для операционных систем семейства Windows и Linux.

Комплект поставки включает два инструмента, которые представляют собой различные сборки исполняемого объектного кода для целевой ОС из исходного кода:

1) BPMS Notator (рабочее место метаэтика/методолога);

2) BPMS Modeler (рабочее место моделировщика).

Содержание дистрибутива приведено в таблице 4.

Таблица 4 – Содержание дистрибутива

Название архива	Наименование ПО	Описание
bpms.modeler-linux.gtk.x86_64.zip	BPMS Modeler	Инструмент моделирования для ОС Astra Linux Smolensk
bpms.notator-linux.gtk.x86_64.zip	BPMS Notator	Инструмент создания нотаций и фильтров для ОС Astra Linux Smolensk
bpms.modeler-win32.win32.x86_64.zip	BPMS Modeler	Инструмент моделирования для ОС Windows 10
bpms.notator-win32.win32.x86_64.zip	BPMS Notator	Инструмент создания нотаций и фильтров для ОС Windows 10

2.3. Инструменты среды программирования

В качестве инструментов программирования используются:

- Java8(LTS) JDK, OpenJDK 8u265b11– комплект разработчика Java;

- Apache Maven, версия 3.6.3, Tycho – система автоматической сборки.

Комплект поставки включает инструменты среды программирования, описанные в таблице 5.

Таблица 5 – Инструменты среды программирования

Название архива	Наименование ПО	Описание
zulu8.48.0.53-ca-jdk8.0.265-win_x64.msi	Java8(LTS) JDK, OpenJDK 8u265b11	Среда исполнения инструментов программирования для ОС Windows 10
zulu8.48.0.53-ca-jdk8.0.265-linux_x64.tar.gz	Java8(LTS) JDK, OpenJDK 8u265b11	Среда исполнения инструментов программирования для ОС Astra Linux Smolensk
apache-maven-3.6.3-bin.zip	Apache Maven, версия 3.6.3, Tycho	Инструмент сборки ПО

2.4. Инструменты среды исполнения

В качестве инструментов среды исполнения используются: Java8(LTS) JRE в соответствии с таблицей 6.

Таблица 6 – Инструменты среды исполнения

Название архива	Наименование ПО	Описание
zulu8.48.0.53-ca-jre8.0.265-win_x64.msi	Java8(LTS) JRE	Среда исполнения ПО для ОС Windows 10
zulu8.48.0.53-ca-jre8.0.265-linux_x64.tar.gz	Java8(LTS) JRE	ПО для ОС Astra Linux Смоленск

2.5. Режим работы программного модуля BPMS

Программный модуль является десктопным приложением и работает в автономном режиме на локальной рабочей станции. Автономный режим работы – это режим, в котором программный модуль BPMS работает как самостоятельное решение для одного пользователя, все данные программного модуля хранятся в локальном расположении на рабочей станции.

В сетевой версии программного модуля BPMS связь с серверной частью и другими пользователями осуществляется с помощью сервисов Технологической Платформы.

2.6. Самовосстанавливаемость программного модуля

При отключении питания и сбоях в работе операционной системы (ОС), внезапного выключения рабочей станции или перезагрузке, программный модуль BPMS восстанавливается до последней сохраненной версии.

3. ОБРАЩЕНИЕ К ПРОГРАММЕ

3.1. Общие сведения

Для работы с программным модулем BPMS необходимо выполнить следующие шаги:

- подготовить рабочее место программиста;
- выполнить сборку программного модуля BPMS из исходных кодов;
- запустить программный модуль BPMS.

3.2. Подготовка рабочего места программиста

Перед сборкой программного модуля BPMS необходимо определенным образом подготовить рабочее место программиста. На компьютер программиста необходимо установить инструменты среды программирования. Инструменты среды программирования – это программное обеспечение, предназначенное для доработки инструментов программного модуля BPMS и сборки их из исходных кодов. Процесс подготовки состоит из следующих шагов:

- установить на рабочей станции OpenJDK 8u265b11;
- установить на рабочей станции Apache Maven, версия 3.6.3, Tycho.

Подробное описание процесса установки специального программного окружения приведено в документе 07623615.00435-01 32 01 «Программный модуль “Система моделирования процессов жизненного цикла изделий”. Руководство системного программиста».

Для сетевой версии программного модуля BPMS необходим программный модуль «Технологическая платформа». Подробное описание процесса установки программного модуля «Технологическая платформа» приведено в документе 07623615.00436-02 32 01 «Программный модуль “Технологическая платформа”. Руководство системного программиста».

3.3. Сборка программного модуля BPMS из исходных кодов

3.3.1. Сборка программного модуля

В рамках программного модуля BPMS выделены два компонента, которые представляют собой различные сборки исполняемого объектного кода для целевой ОС из исходного кода:

- BPMS Modeler;

- BPMS Notator.

Процесс сборки является автоматизированным и производится с использованием Apache Maven, версия 3.6.3, Tycho. Управление процессом сборки осуществляется с помощью файлов конфигурации. Входом процесса является исходный код программного модуля BPMS, в архиве «bpms.zip». Выходом процесса являются дистрибутивы пригодные для установки и последующего запуска инструментов.

Процесс сборки состоит из следующих шагов:

1) вставить компакт-диск, содержащий исходные коды в привод компьютера;

2) скачать архив «bpms.zip», содержащий исходные коды;

3) распаковать архив с исходным кодом («bpms.zip»);

4) используя терминал переходим в корневую папку исходного кода `bpms, после чего запускаем процесс сборки командой *`mvn clean verify`*.

При успешном окончании процесса получается следующий результат:

[INFO] ----------

[INFO] BUILD SUCCESS

[INFO] ----------

[INFO] Total time: 09:22 min

[INFO] Finished at: 2020-07-02T15:22:12+03:00

[INFO] ----------

Результаты процесса находятся в папках:

- BPMS modeler\bumps\products\bpms.modeler.product.definition \target\products;

- BPMS Notator`\bombs\products\bpms. notator.product.definition\target\roducts.

Процесс сборки использует следующие конфигурационные файлы:

- главный файл является точкой входа в процесс сборки и определяет её состав;
- настроечный файл определяет параметры процесса сборки;
- файл платформы определяет способ сборки целевой платформы для поиска и разрешения зависимостей.

Расположение и содержимое файлов приведено ниже.

3.3.2. Конфигурация автоматической сборки

Главный файл конфигурации является точкой входа в процесс сборки и определяет её состав.

Расположение: – `bpms/pom.xml`

Содержание:

```
<!-- parent pom -->

<project>
   <modelVersion>4.0.0</modelVersion>
   <groupId>ru.vniief.bpms</groupId>
   <artifactId>root</artifactId>
   <version>1.0.0-SNAPSHOT</version>
   <packaging>pom</packaging>
   <parent>
      <groupId>ru.vniief.bpms</groupId>
      <artifactId>releng</artifactId>
      <version>1.0.0-SNAPSHOT</version>
      <relativePath>./releng/bpms.releng</relativePath>
   </parent>

   <modules>
      <module>./bundles/ru.vniief.bpms.codegen</module>
```

<module>./bundles/ru.vniief.bpms.codegen.ui</module>

<module>./bundles/ru.vniief.bpms.methodology</module>
<module>./bundles/ru.vniief.bpms.methodology.design</module>
<module>./bundles/ru.vniief.bpms.methodology.edit</module>
<module>./bundles/ru.vniief.bpms.methodology.editor</module>

<module>./bundles/ru.vniief.bpms.methodology.rosatom</module>
<module>./bundles/ru.vniief.bpms.methodology.free</module>

<module>./bundles/ru.vniief.bpms.model.core</module>
<module>./bundles/ru.vniief.bpms.model.core.design</module>
<module>./bundles/ru.vniief.bpms.model.core.edit</module>
<module>./bundles/ru.vniief.bpms.model.core.editor</module>

<module>./bundles/ru.vniief.bpms.model.epc</module>
<module>./bundles/ru.vniief.bpms.model.epc.design</module>
<module>./bundles/ru.vniief.bpms.model.epc.edit</module>
<module>./bundles/ru.vniief.bpms.model.epc.editor</module>

<module>./bundles/ru.vniief.bpms.model.function</module>
<module>./bundles/ru.vniief.bpms.model.function.edit</module>
<module>./bundles/ru.vniief.bpms.model.function.editor</module>

<module>./bundles/ru.vniief.bpms.model.goal</module>
<module>./bundles/ru.vniief.bpms.model.goal.edit</module>
<module>./bundles/ru.vniief.bpms.model.goal.editor</module>

<module>./bundles/ru.vniief.bpms.model.org</module>
<module>./bundles/ru.vniief.bpms.model.org.design</module>

```
<module>./bundles/ru.vniief.bpms.model.org.edit</module>
<module>./bundles/ru.vniief.bpms.model.org.editor</module>

<module>./bundles/ru.vniief.bpms.model.product</module>
<module>./bundles/ru.vniief.bpms.model.product.edit</module>
<module>./bundles/ru.vniief.bpms.model.product.editor</module>

<module>./bundles/ru.vniief.bpms.model.vad1</module>
<module>./bundles/ru.vniief.bpms.model.vad1.design</module>
<module>./bundles/ru.vniief.bpms.model.vad1.edit</module>
<module>./bundles/ru.vniief.bpms.model.vad1.editor</module>
<module>./bundles/ru.vniief.bpms.model.vad2</module>
<module>./bundles/ru.vniief.bpms.model.vad2.design</module>
<module>./bundles/ru.vniief.bpms.model.vad2.edit</module>
<module>./bundles/ru.vniief.bpms.model.vad2.editor</module>

<module>./bundles/ru.vniief.bpms.ui.explorer</module>
<module>./bundles/ru.vniief.bpms.ui.project</module>
<module>./bundles/ru.vniief.bpms.util</module>

<module>./bundles/ru.vniief.bpms.sirius.ext</module>

<module>./bundles/bpms.modeler.help.doc</module>


<!-- product(application) plugins -->
<module>./bundles/ru.vniief.bpms.modeler.product</module>
<module>./bundles/ru.vniief.bpms.notator.product</module>

<!-- target platform -->
```

```
        <module>./releng/bpms.targets</module>

    <!-- features -->
        <module>./features/bpms.epc.feature</module>
        <module>./features/bpms.model.feature</module>
        <module>./features/bpms.modeler.feature</module>
        <module>./features/bpms.modeler.doc.feature</module>
        <module>./features/bpms.notator.feature</module>
        <module>./features/bpms.org.feature</module>
        <module>./features/bpms.pgf.feature</module>
        <module>./features/bpms.vad.feature</module>

    <!-- NLS – platform -->
        <module>./features/bpms.nls.feature</module>
    <!-- product definitions -->
        <module>./products/bpms.modeler.product.definition</module>
        <module>./products/bpms.notator.product.definition</module>
      </modules>

    </project>
```

Настроечный файл определяет плагины Apache Maven, опции компилятора, опции упаковщика, расположение таргет-платформы и операционную систему для сборки инструментов BPMS Notator, BPMS Modeler.

Расположение: – `\bpms\releng\bpms.releng\pom.xml`

Содержание:

```
<project>
  <modelVersion>4.0.0</modelVersion>
```

```
<groupId>ru.vniief.bpms</groupId>
<artifactId>releng</artifactId>
<version>1.0.0-SNAPSHOT</version>
<packaging>pom</packaging>

<properties>
  <tycho.version>1.7.0</tycho.version>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <eclipse-repo.url>http://download.eclipse.org/releases/latest</eclipse-
repo.url>
  <java.version>8</java.version>
</properties>

<build>
  <plugins>
    <plugin>
      <groupId>org.eclipse.tycho</groupId>
      <artifactId>tycho-maven-plugin</artifactId>
      <version>${tycho.version}</version>
      <extensions>true</extensions>
    </plugin>
    <plugin>
      <groupId>org.eclipse.tycho</groupId>
      <artifactId>tycho-p2-director-plugin</artifactId>
      <version>${tycho.version}</version>
    </plugin>

    <plugin>
      <groupId>org.eclipse.tycho</groupId>
      <artifactId>tycho-compiler-plugin</artifactId>
```

```
<version>${tycho.version}</version>
<configuration>
<compilerArgument>-
warn:+discouraged,forbidden</compilerArgument>
<useProjectSettings>false</useProjectSettings>
<source>${java.version}</source>
<target>${java.version}</target>
</configuration>
</plugin>
<!--Enable the replacement of the SNAPSHOT version in the final product
configuration-->
<plugin>
<groupId>org.eclipse.tycho</groupId>
<artifactId>tycho-packaging-plugin</artifactId>
<version>${tycho.version}</version>
<executions>
<execution>
<phase>package</phase>
<id>package-feature</id>
<configuration>
<final-
Name>${project.artifactId}_${unqualifiedVersion}.${buildQualifier}</finalName>
</configuration>
</execution>
</executions>
</plugin>
<plugin>
<groupId>org.eclipse.tycho</groupId>
<artifactId>target-platform-configuration</artifactId>
<version>${tycho.version}</version>
```

```
            <configuration>
              <target>
                <artifact>
                  <groupId>ru.vniief.bpms</groupId>
                  <artifactId>target-2020-6-Sirius-6.3.2</artifactId>
                  <version>1.0.0-SNAPSHOT</version>
                </artifact>
              </target>
              <environments>
                <environment>
                  <os>linux</os>
                  <ws>gtk</ws>
                  <arch>x86_64</arch>
                </environment>
                <environment>
                  <os>win32</os>
                  <ws>win32</ws>
                  <arch>x86_64</arch>
                </environment>
              </environments>
            </configuration>
          </plugin>
        </plugins>
      </build>
    </project>
```

В случае отсутствия доступа к сети следующие пути нужно заменить на пути, ведущие к локальному зеркалу:

- http://download.eclipse.org/releases/latest;
- https://maven.apache.org/.

Файл платформы определяет способ сборки целевой платформы для поиска и разрешения зависимостей.

Расположение: – `\bpms\releng\bpms.targets/pom.xml`

Содержание:

```
<project>
   <modelVersion>4.0.0</modelVersion>
   <parent>
      <groupId>ru.vniief.bpms</groupId>
      <artifactId>root</artifactId>
      <version>1.0.0-SNAPSHOT</version>
      <relativePath>../../</relativePath>
   </parent>
   <groupId>ru.vniief.bpms</groupId>
   <artifactId>target-2020-6-Sirius-6.3.2</artifactId>
   <version>1.0.0-SNAPSHOT</version>
   <packaging>eclipse-target-definition</packaging>
</project>
```

3.4. Запуск программного модуля BPMS

Сведения по установке и первичному запуску BPMS Modeler и BPMS Notator указаны в документе 5201/0008-21-06 «Программный модуль «Система моделирования процессов жизненного цикла изделий». Руководство системного программиста».

3.4.1. Запуск BPMS Modeler в меню BPMS Notator

Для проверки разработанной функциональности в инструменте BPMS Notator до сборки BPMS Modeler есть возможность предварительно посмотреть результат, полученный после доработки и изменения нотации. Для этого необходимо открыть BPMS Modeler в меню BPMS Notator.

Чтобы открыть BPMS Modeler в меню BPMS Notator необходимо в навигаторе слева развернуть проект «bpms.modeler.product.definition» совершить двойной щелчок ЛКМ на файле «ru.vniief.bpms.modeler.product.product», как указано на рис. 1.

Выбор в навигаторе BPMS Notator файла «ru.vniief.bpms.modeler.product.product»

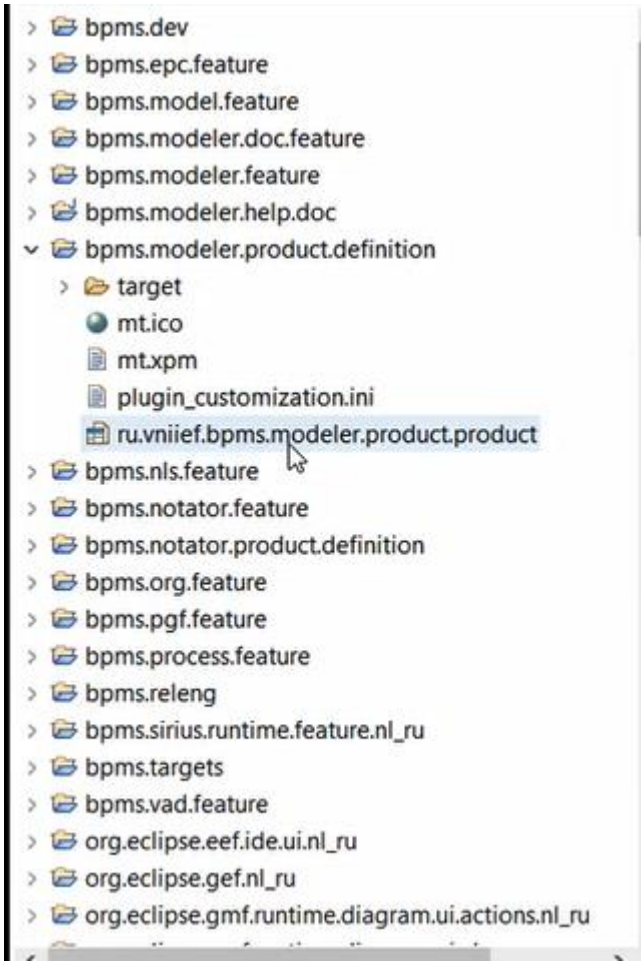

Рисунок 1

В появившемся окне, показанном на рис. 2, нажать по ссылке и запустить Eclipse-приложение.

Запуск BPMS Modeler

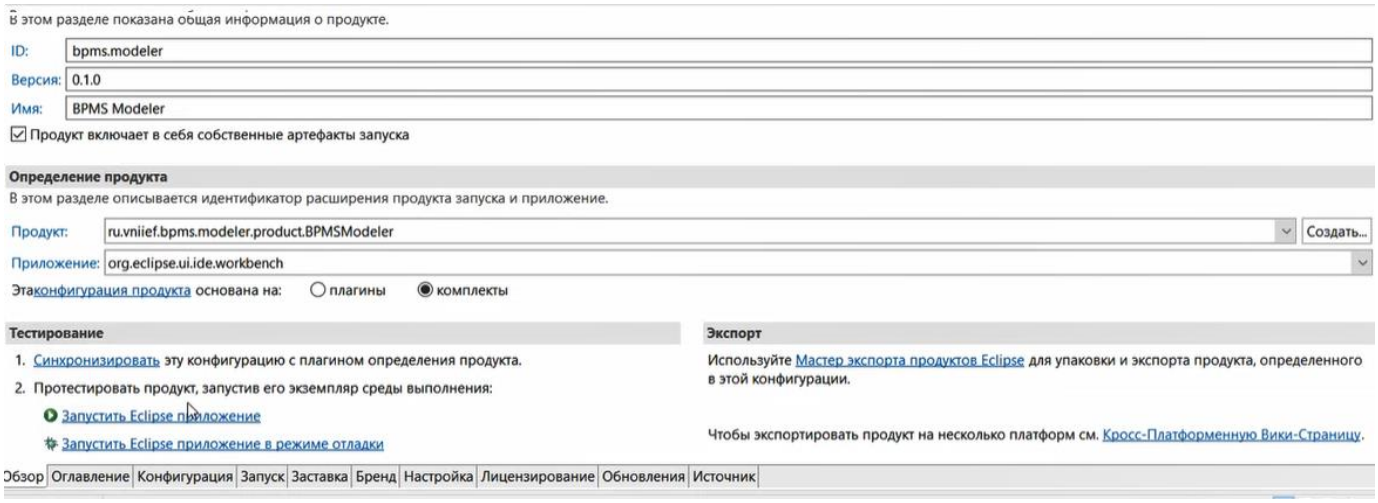

Рисунок 2

Далее в окне «Сохранить и запустить» нажать на кнопку «OK» в соответствии с рис. 3.

Окно «Сохранить и запустить»

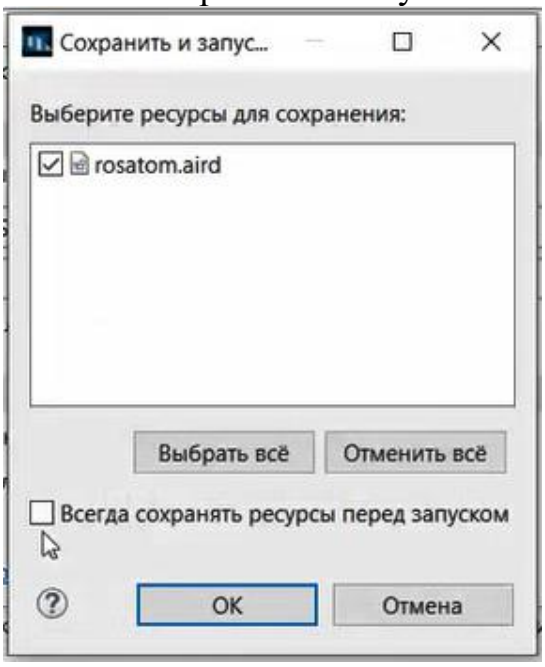

Рисунок 3

В результате отобразится стартовое окно BPMS Modeler, как показано на рис. 4, и выполнится запуск BPMS Modeler.

Стартовое окно BPMS Modeler

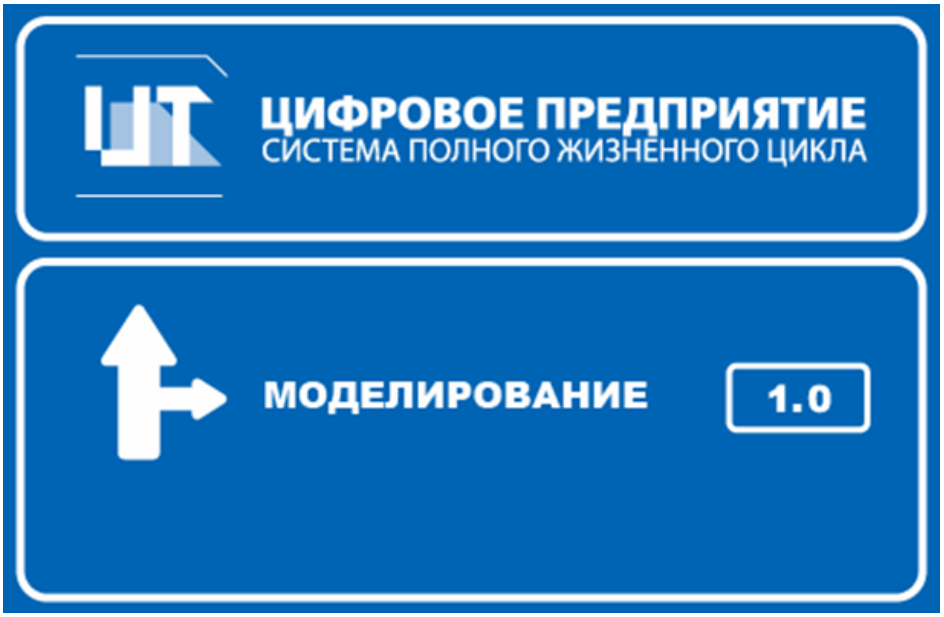

Рисунок 4

Для того, чтобы пользователь мог работать с измененными нотациями, необходительно выполнить сборку инструмента BPMS Modeler в соответствии с п. 3.2.

4. РЕКОМЕНДАЦИИ ПО РАЗРАБОТКЕ

4.1. Общее описание

Настоящие рекомендации по разработке описывают основные принципы, используемые при работе программиста в BPMS Notator для создания и изменения функциональных возможностей BPMS Modeler.

В целом подход к разработке описан в рамках проекта «Архитектура, управляемая моделями» (Model-Driven Architecture, MDA):

- https://www.omg.org/news/meetings/workshops/UML_2003_Manual/00-2_MDA_Guide_v1.0.1.pdf (1-я редакция – более подробная);
- https://www.omg.org/cgi-bin/doc?ormsc/14-06-01 (2-я редакция).

Суть подхода в том, что при разработке на верхнем уровне все описывается в виде моделей. Затем на основе моделей формируется реализация ПО, отчетов или иных артефактов.

Основные принципы подхода к разработке указаны ниже.

4.2. Ссылки на родительские объекты

Все объекты модели, кроме корневого, принадлежат некоторому родительскому объекту. Например, корневым объектом модели «Карта процессов» является объект «Карта процессов» (ProcessLandscape), а модель «Карта процессов» – это родительский объект для группы процессов в нотации, соответствующей предметной области «Карта процессов» в этой модели.

При создании ссылок рекомендуется делать неявные ссылки на родительские объекты, для этого предлагается ссылаться на родительский объект с помощью методов oclContainer () или eContainer ().

Примечание. Программист в инструменте метаэтика определяет модель, ее связи и вложенность. Карта процессов (ProcessLandscape) используется в инструменте моделирования при моделировании процессов предметной области «Карта процессов».

4.3. Иерархии объектов в моделях

В случае, когда создается иерархическая модель (например, организационная структура, дерево продуктов, дерево целей), в которой объекты нижестоящих моделей вкладываются в объекты вышестоящих моделей, можно использовать следующие представления:

- вышестоящие объекты определяются для нижестоящих в виде родительских объектов;
- все объекты всех моделей хранятся на одном уровне, иерархия задается в виде ссылок.

Также возможен гибридный вариант, при котором объекты одной диаграммы хранятся в плоском списке, а объекты вложенных диаграмм помещаются в родительский объект.

Применение родительских объектов имеет следующие преимущества:

- процедура копирования проще и быстрее, когда объект копируется со вложенной структурой;
- ускоряется процедура поиска и формирования дерева объектов (например: модель дерева целей или дерево процессов);
- интерфейс при просмотре модели в древовидном редакторе не нагружен лишними деталями, объекты меньше, и они сгруппированы.

Преимущества плоского списка: проще получить список всех объектов при поиске объекта или при построении диаграммы.

Родительский объект образует собственное пространство имен. Если в разных частях иерархии могут присутствовать одноименные, но при этом разные объекты, то, во избежание путаницы, следует хранить объекты в виде дерева. С другой стороны, если необходимо контролировать уникальность имен объектов во всем дереве, то следует хранить объекты в плоском списке.

При разработке метамоделей в программном модуле BPMS выбран вариант хранения в виде дерева по следующим причинам:

- плоский список сложно разбить на несколько подмоделей (это нужно для того, чтобы у разных фрагментов дерева могли быть разные владельцы, а также

для того, чтобы для каждого фрагмента можно было создавать свою диаграмму);

- гибридный вариант решает проблему с диаграммами, но в этом случае непонятно, по какому принципу часть объектов хранятся в плоской структуре, а часть вкладываются в родительские;

- в разных частях дерева могут быть одноименные объекты, если совпадение имен в каких-то случаях нежелательно, то проблема решается с помощью правил валидации, инструментов анализа и т. п.;

- справочники получаются большими и работать со списками из тысяч записей становится неудобно.

Исключением являются редакторы диаграмм, для них в метамодели создаются вычисляемые свойства, объединенные в плоский список объектов, которые должны присутствовать на диаграмме.

4.4. Вложенные пакеты

Описание ошибок при работе с вложенными пакетами и способы их устранения приведены по следующим ссылкам:

- https://eclipsesource.com/blogs/2013/03/19/emf-dos-and-donts-5/;
- https://www.eclipse.org/forums/index.php/t/1085561/.

4.5. Строковые свойства в модели

При работе с моделями пользователю необходимо задавать на панели «Свойства» значения атрибутов со строковым типом данных (например «название», «описание»), начиная со строчных букв (за исключением имен собственных, аббревиатур и т.п.).

Если требуется, чтобы на диаграмме или в отчете, сгенерированном в результате анализа, текст выводился с заглавной буквы, следует добавить соответствующее преобразование в спецификацию графической нотации или в генератор документа.

При формировании различных артефактов из модели (человеко-читаемых документов и т.п.) проще преобразовать первую строчную букву в прописную, если это требуется. Значительно сложнее преобразовать прописную букву в строчную.

4.5.1. Пример задания правила для отображения атрибутов со строковым типом данных.

Сначала необходимо открыть спецификацию редактора диаграмм Eclipse Sirius. Например, для открытия спецификации диаграмм для EPC-моделей необходимо развернуть в панели «Структура модели» проект «ru.vniief.bpms.model.epc.design», развернуть папку «description» и открыть файл «epc.odesign» с помощью двойного щелчка ЛКМ.

Затем в редакторе спецификаций Eclipse Sirius следует найти правила отображения фигур на диаграммах, для которых необходимо изменить правила отображения атрибутов со строковым типом данных. Например, для настройки атрибутов со строковым типом данных для событий (Event) в EPC-моделях необходимо найти параметры фигуры, в окне свойств открыть вкладку Label и в поле «Выражение заголовка» задать выражение «*aql:self.name.toUpperFirst()*», как показано на рис. 5.

Пример заполнения строки «Выражения заголовка»

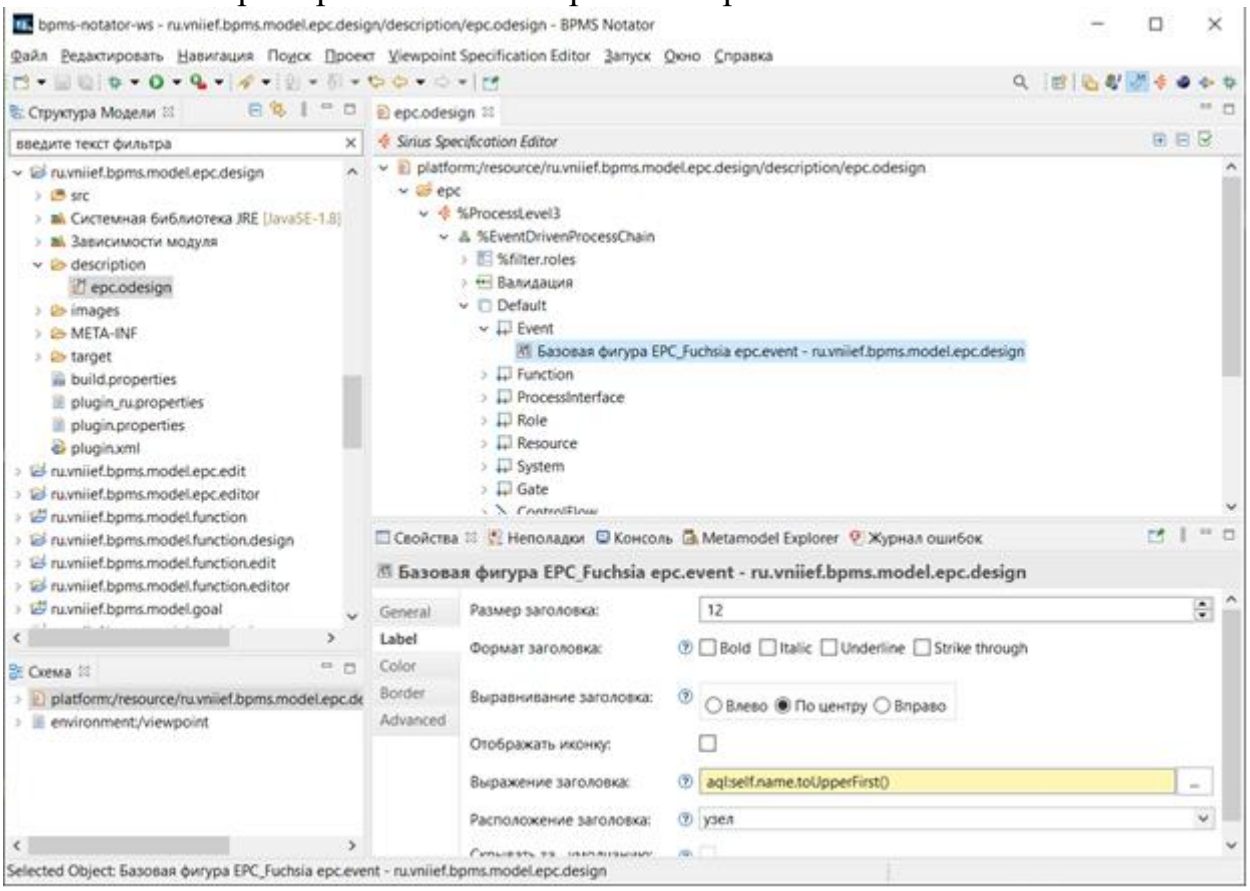

Рисунок 5

После этого надписи на фигурах будут отображаться с заглавной буквы.

4.6. Работа с OCL-правилами

4.6.1. Общее описание OCL-правил

OCL-правила – это правила, написанные на объектном языке ограничений (OCL). Язык OCL служит для описания правил проверки моделей, их экземпляров или данных, не предусмотренных нотацией. Если правила описываются на уровне метамодели (например, в метамодели языка EPC), то они позволяют производить проверку модели. Если правила описываются на уровне модели (например, в конкретной EPC-модели, либо в описании структуры электронного документа), то они позволяют проверять экземпляры этих моделей (правильно ли выполняется процесс, правильно заполнен электронный документ).

Для работы с OCL-правилами необходимо открывать ecore-модель в специализированном редакторе объектных ограничений OCL (OCLinEcore Editor), который является частью инструмента среды программирования.

Правила задаются в виде структуры данных фиксированной длины, которая содержит определенное число и последовательность элементов следующего вида:

```
invariant NameRequired: Tuple {
  message = 'Element should have a name',
  status = name <> null and name.size() > 0,
  severity = -1
}. status;
```

Элементы правила:

- *message* – сообщение об ошибке, может быть OCL-выражением;
- *status* – OCL-выражение, которое вычисляется при выполнении проверки;
- *severity* – вид проверки;
- *severity <0* – ошибка;
- *severity = 0* – информация;
- *severity> 0* – предупреждение.

4.6.2. Пример создания правила с использованием OCL

Правила на языке OCL определяются на уровне метамодели нотации. Для создания или изменения правил необходимо выполнить следующие шаги:

1) открыть проект, содержащий метамодель (например, для нотации EPC) в инструменте создания нотаций и фильтров (рис. 6);

2) открыть метамодель в редакторе OCLinEcore, например, файл «epc.ecore» (рис. 7);

3) написать необходимый инвариант, используя соглашения редактора OCLinEcore и синтаксис языка OCL (рис. 8);

4) сохранить изменения и выполнить генерацию кода по метамодели (рис. 9);

5) пересобрать инструмент создания нотаций BPMS Notator и инструмент моделирования BPMS Modeler, следуя процедуре, указанной в п. 3.2.

Проект, содержащий метамодель нотации EPC

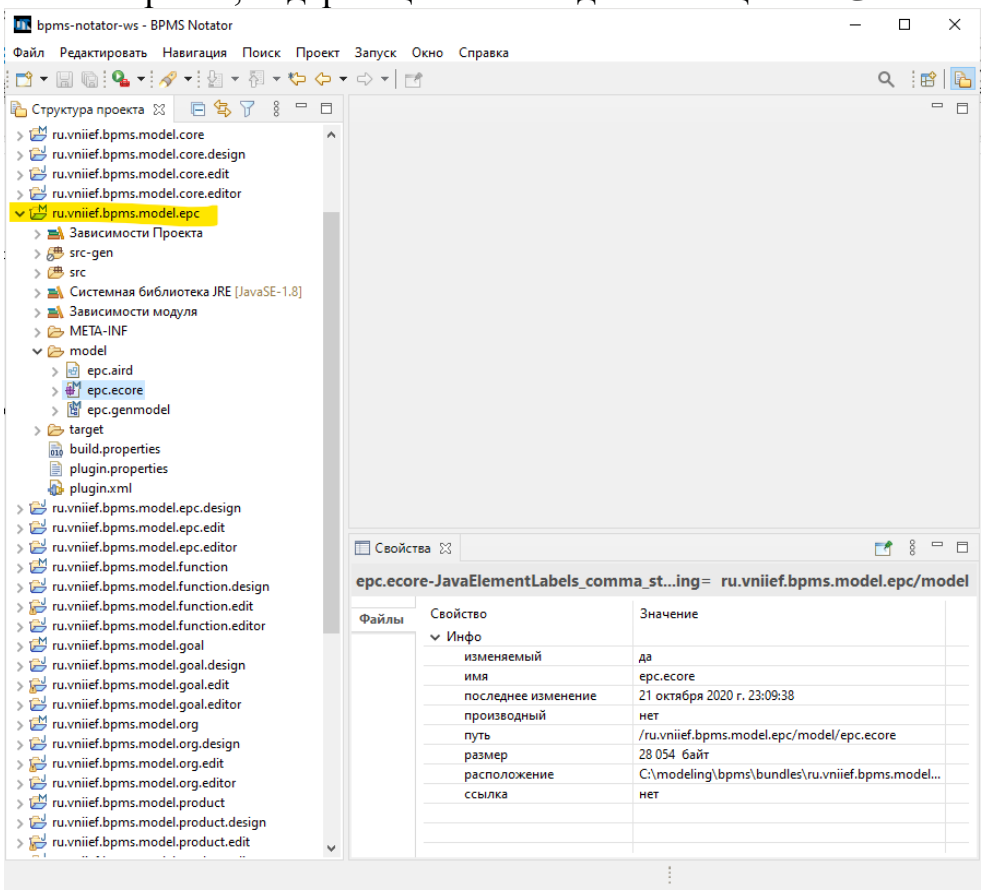

Рисунок 6

Метамодель нотации в редакторе OCLinEcore

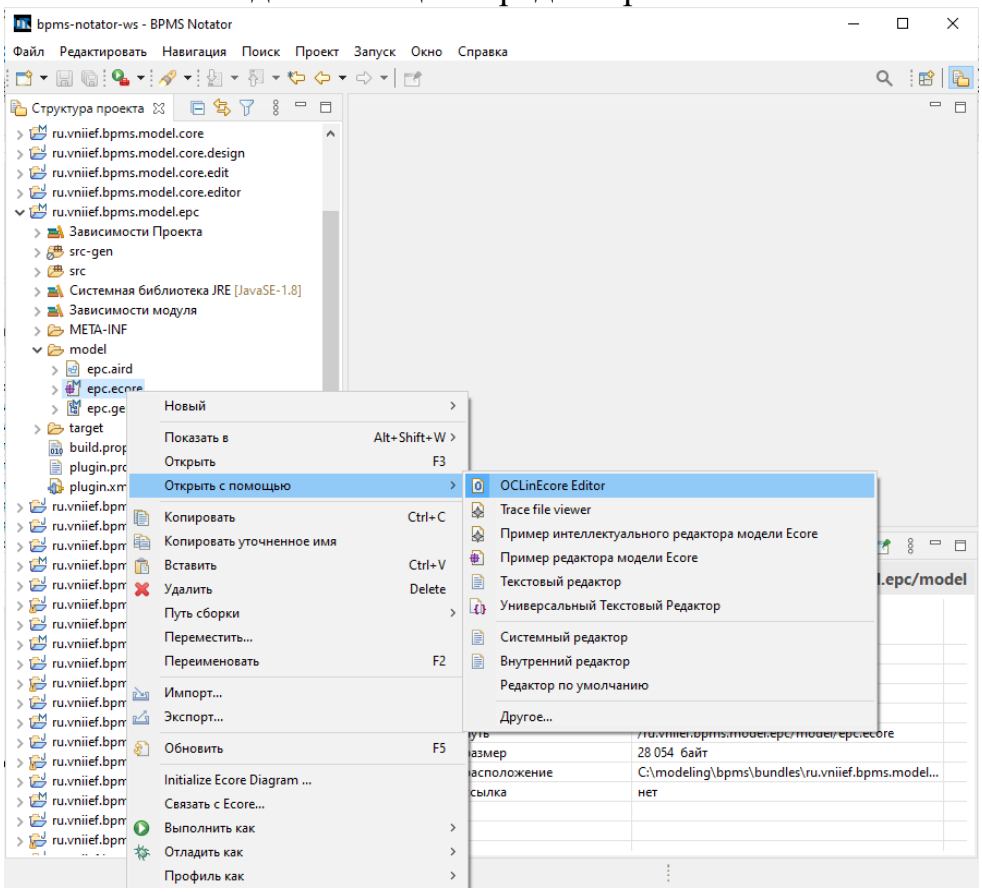

Рисунок 7

Инвариант с использованием соглашения редактора OCLinEcore и синтаксис языка OCL

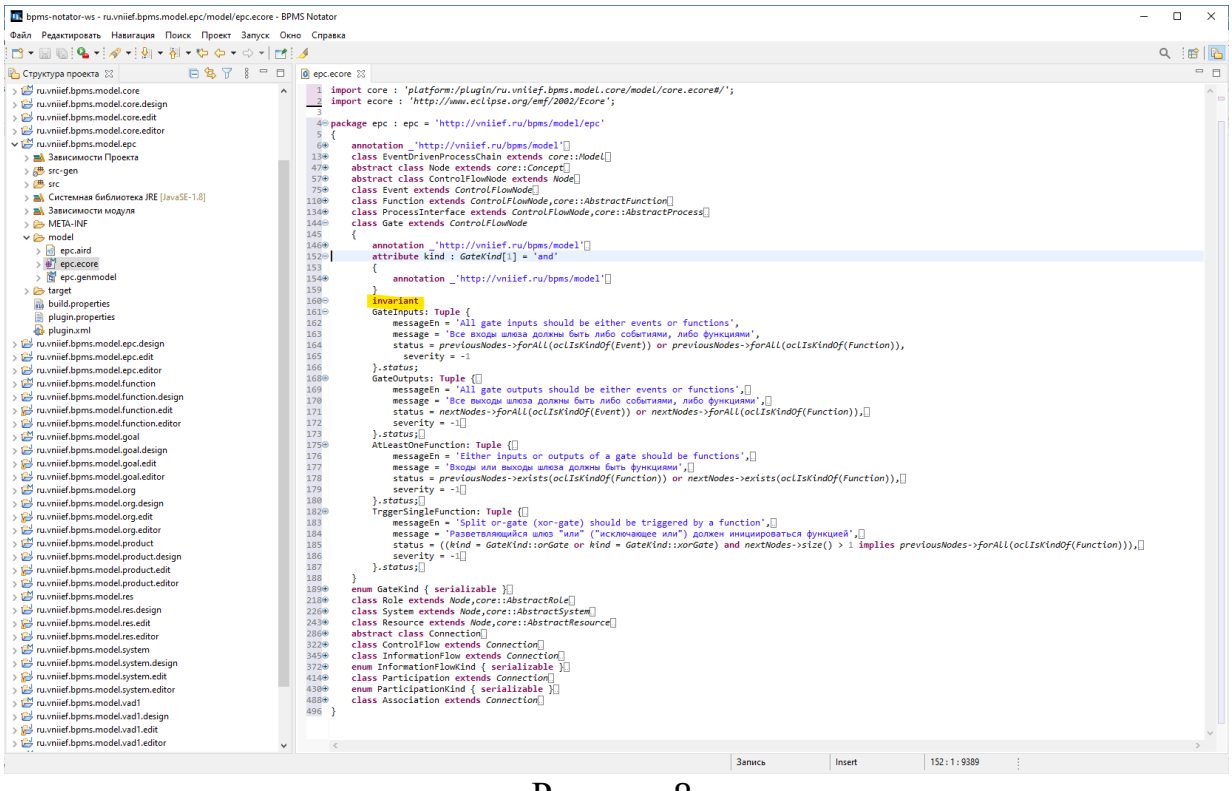

Рисунок 8

Сохранение изменений

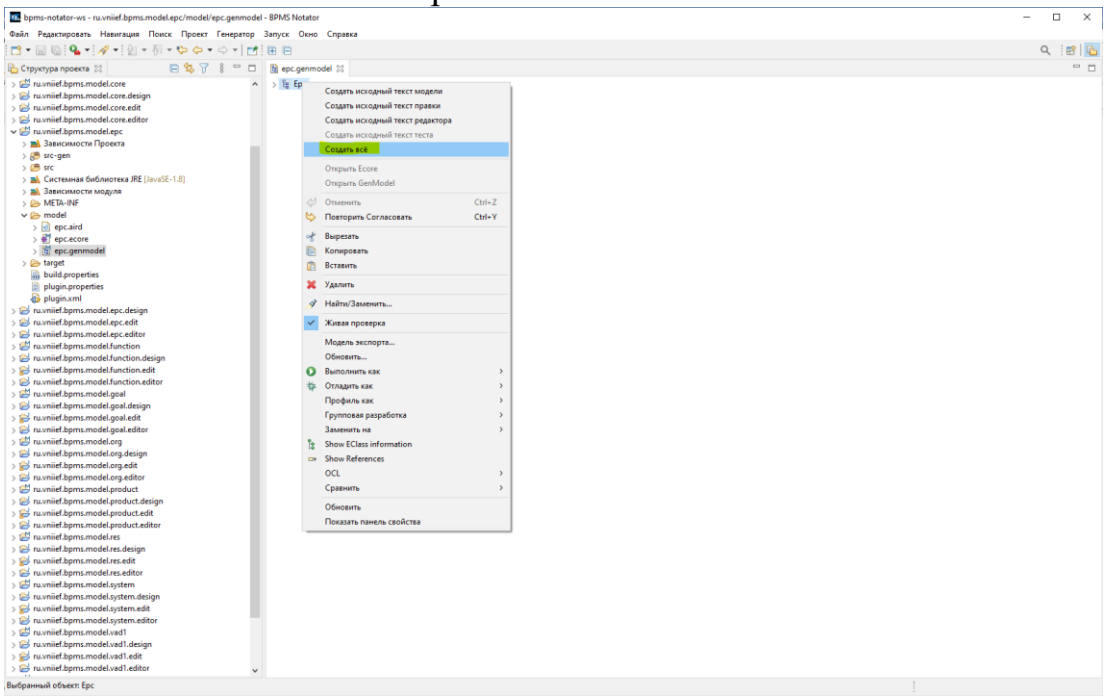

Рисунок 9

Информацию о написании инвариантов и использовании редактора можно почерпнуть в подразделе «OCLinEcore tutorial» раздела «OCL Documentation» справочной системы Eclipse. Определение языка OCL приведено в стандарте OMG, https://www.omg.org/spec/OCL/About-OCL/.

Пример написания инварианта для узла типа «шлюз» выглядит следующим образом:

```
class Gate extends ControlFlowNode
        {
            annotation _'http://vniief.ru/bpms/model'
            (
                name                                                    =
'\u043B\u043E\u0433\u0438\u0447\u0435\u0441\u043A\u0438\u0439
\u0448\u043B\u044E\u0437',
                description =
                '\u041B\u043E\u0433\u0438\u0447\u0435\u0441\u043A\u0438\u0439
\u0448\u043B\u044E\u0437
\u043F\u0440\u0435\u0434\u0441\u0442\u0430\u0432\u043B\u044F\u0435\u0442
```

\u043B\u043E\u0433\u0438\u0447\u0435\u0441\u043A\u0438\u0435 \u0432\u0437\u0430\u0438\u043C\u043E\u043E\u0442\u043D\u043E\u0448\u0435\u043D\u0438\u044F \u044D\u043B\u0435\u043C\u0435\u043D\u0442\u043E\u0432 \u043F\u043E\u0442\u043E\u043A\u0430 \u0443\u043F\u0440\u0430\u0432\u043B\u0435\u043D\u0438\u044F. \u041E\u043D \u043F\u043E\u0437\u0432\u043E\u043B\u044F\u0435\u0442 \u0441\u043E\u0435\u0434\u0438\u043D\u044F\u0442\u044C \u0438\u043B\u0438 \u0440\u0430\u0437\u0432\u0435\u0442\u0432\u043B\u044F\u0442\u044C \u043F\u043E\u0442\u043E\u043A \u0443\u043F\u0440\u0430\u0432\u043B\u0435\u043D\u0438\u044F \u0441\u043E \u0441\u043B\u0435\u0434\u0443\u044E\u0449\u0438\u043C\u0438 \u0438\u0437 \u043D\u0438\u0445. \u041E\u043F\u0440\u0435\u0434\u0435\u043B\u0435\u043D\u044B \u0442\u0440\u0438 \u0432\u0438\u0434\u0430 \u0448\u043B\u044E\u0437\u043E\u0432: XOR, AND \u0438 OR.'

```
    );
    attribute kind : GateKind[1] = 'and'
    {
        annotation _'http://vniief.ru/bpms/model'
        (
            name = '\u0432\u0438\u0434',
            description = '\u0432\u0438\u0434 \u0448\u043B\u044E\u0437\u0430'
        );
    }
    invariant
    GateInputs: Tuple {
        messageEn = 'All gate inputs should be either events or functions',
        message = 'Все входы шлюза должны быть либо событиями, либо функциями',
```

status = previousNodes->forAll(oclIsKindOf(Event)) or previousNodes->forAll(oclIsKindOf(Function)),

severity = -1

}.status;

invariant

GateOutputs: Tuple {

messageEn = 'All gate outputs should be either events or functions',

message = 'Все выходы шлюза должны быть либо событиями, либо функциями',

status = nextNodes->forAll(oclIsKindOf(Event)) or nextNodes->forAll(oclIsKindOf(Function)),

severity = -1

}.status;

invariant

AtLeastOneFunction: Tuple {

messageEn = 'Either inputs or outputs of a gate should be functions',

message = 'Входы или выходы шлюза должны быть функциями',

status = previousNodes->exists(oclIsKindOf(Function)) or nextNodes->exists(oclIsKindOf(Function)),

severity = -1

}.status;

invariant

TrggerSingleFunction: Tuple {

messageEn = 'Split or-gate (xor-gate) should be triggered by a function',

message = 'Разветвляющийся шлюз "или" ("исключающее или") должен инициироваться функцией',

status = ((kind = GateKind::orGate or kind = GateKind::xorGate) and nextNodes->size() > 1 implies previousNodes->forAll(oclIsKindOf(Function))),

severity = -1

}.status;

}

Здесь и далее, редактор OCL поддерживает знаки киррилицы в виде набора символов в формате Unicode (UTF), например /u404E.

4.7. Написание правил на обязательность заполнения атрибутов

Если атрибут обязателен к заполнению, то на уровне метамодели задается множественности вида 1…1.

В данном случае проверка задается таким образом:

```
name.size() > 0
```

Если же возможен ввод значения равного пустой строке, необходимо задать правило вида:

```
name <> null and name.size() > 0
```

Если атрибут обязательный, рекомендуется делать проверку на неравенство *null.* – для унификации, и чтобы не допустить ошибок в правилах при изменении обязательности.

Если у атрибута задана множественность 1...1, то при генерации Java-кода проверка `*name <> null*` будет игнорироваться. При этом, если значение атрибута будет всё-таки не задано (равно *null*), то при выполнении проверки возникает исключение. Возможны следующие варианты решения:

- делать строковые атрибуты всегда опциональными и проверять, что они заполнены с помощью OCL-правил;

- изменять генератор OCL, чтобы он не игнорировал условия вида `*name <> null*` для обязательных атрибутов.

4.8. Преобразования моделей в текстовое представление

4.8.1. Пример преобразования модель-текст

Для преобразования моделей в текстовое представление могут использоваться различные технологии. Одна из них – это язык MOF Model to Text Transformation Language (MOF M2T). Язык описан в одноименном международном стандарте, разработанном консорциумом Object Management Group. Язык реализован в рамках проекта Eclipse Acceleo. В качестве выходного формата могут использоваться HTML, CSV, код на некотором языке программирования (например, Java или JavaScript) и др.

В качестве примера шаблона на языке MOF M2T рассмотрим формирование регламента процесса на основе модели в нотации EPC. Пример модели представлен на рис. 10.

Пример модели в нотации EPC

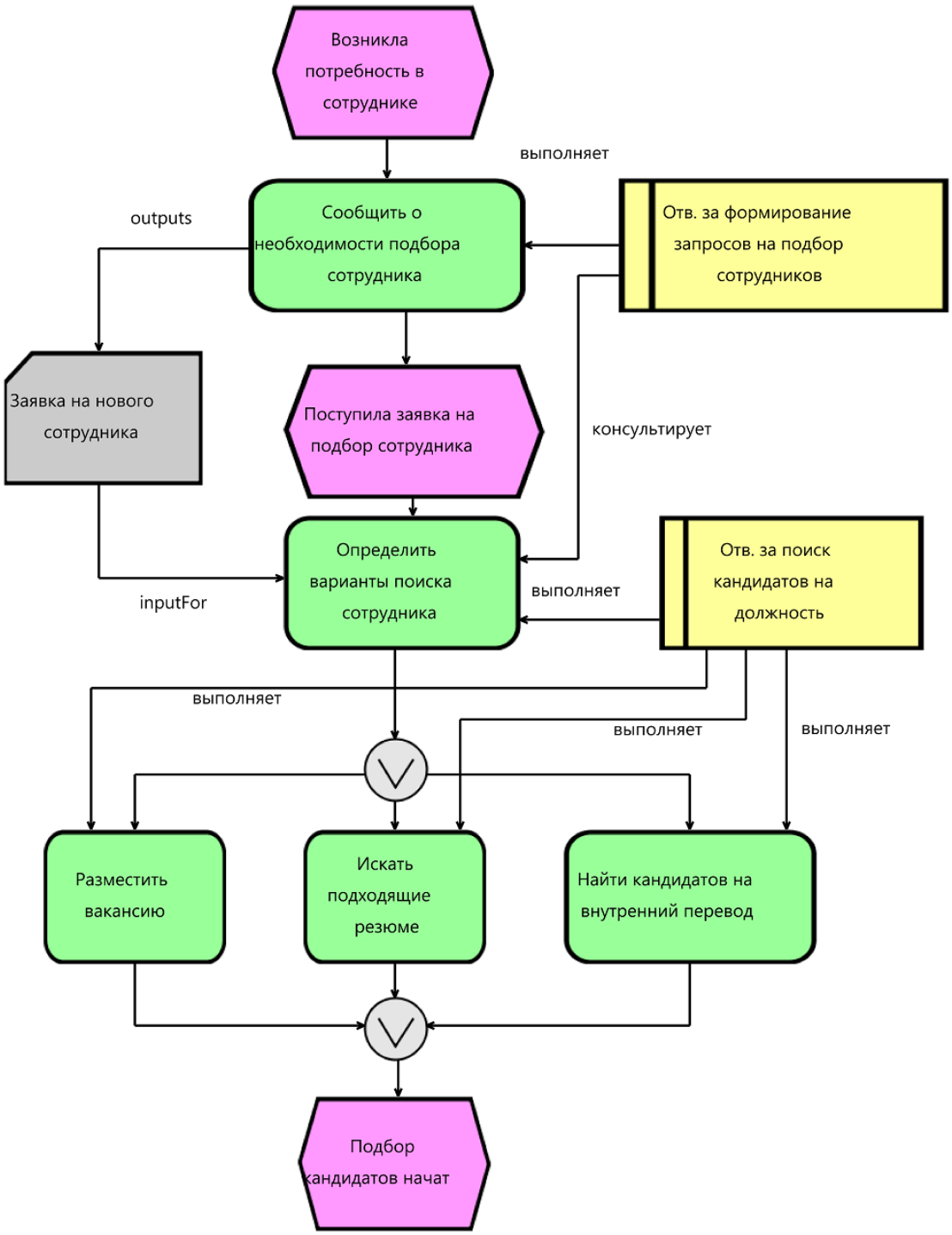

Рисунок 10

Упрощенный пример регламента, формируемого на основе данной модели, приведен в таблице 7.

Таблица 7 – Пример регламента процесса

Шаг процесса	Дополнительные участники	Входы	Выходы
Событие «возникла потребность в сотруднике»			
Ответственный за формирование запросов на подбор сотрудников выполняет функцию «сообщить о необходимости подбора сотрудника»			Заявка на нового сотрудника
Событие «поступила заявка на подбор сотрудника»			
Ответственный за поиск кандидатов на должность выполняет функцию «определить варианты поиска сотрудника»	Консультирует: ответственный за формирование запросов на подбор сотрудников	Заявка на нового сотрудника	
Переход на один или несколько шагов 6, 7 и (или) 8			
Ответственный за поиск кандидатов на должность выполняет функцию «разместить вакансию». Переход на шаг 9			
Ответственный за поиск кандидатов на должность выполняет функцию «искать подходящие резюме». Переход на шаг 9			
Ответственный за поиск кандидатов на должность выполняет функцию «найти кандидатов на внутренний перевод»			
Ожидается выполнение одного или более шагов 6, 7 и/или 8			
Событие «подбор кандидатов начат»			

Рассмотрим шаблон на языке MOF M2T, с помощью которого модель может быть преобразована в приведенную выше таблицу. Данное руководство не является исчерпывающим описанием языка MOF M2T, далее будут описаны только базовые принципы. Для получения более детальной информации следует обратиться к стандарту OMG MOF M2T и документации Eclipse Acceleo.

Преобразование начинается с указания используемой в нем кодировки (UTF-8), указания названия преобразования и ссылки на метамодель.

[comment encoding = UTF-8 /]

[module generateEpcReglament('http://vniief.ru/bpms/model/epc')/]

Метамодель описывает структуру допустимых входных моделей для преобразования (типы объектов и связей, атрибуты). Если входная модель не соответствует

метамодели (например, вместо EPC-модели передана BPMN-модель), то при запуске преобразования будет выдана ошибка.

Метамодели описываются отдельно от преобразования на языке MOF (Ecore), а в преобразовании указывается лишь идентификатор метамодели (обычно в виде URI).

Преобразования строятся из блоков. Начало блока обозначается открывающим тегом, а окончание блока – закрывающим тегом. Например, блок *template* может иметь следующую структуру:

[template ...]

...

[/template]

Между открывающим и закрывающим тегами указывается содержимое блока. Если у блока нет содержимого, то может использоваться сокращенная запись вида:

[comment ... /]

Основными строительными блоками преобразования модель-текст являются шаблоны. Далее приведен основной шаблон преобразования. Комментарии к шаблонам преобразования даны в таблице 8.

Таблица 8 – Комментарии к шаблонам преобразования.

Шаблон преобразования	**Комментарии**
[template public generateEpcReglament(model : Event-DrivenProcessChain)]	Определяет: - область видимости – может принимать значения «public», «private»; - название шаблона (generateEpcReglament); -параметры, принимаемые шаблоном (модель типа EventDrivenProcessChain)
[comment @main/]	Комментарий, указывающий на то, что данный шаблон является основным в преобразовании. При запуске преобразования ко входной модели будет применен именно этот шаблон.
[file (model.name.toUpperFirst() + '.html', false, 'utf-8')]	Блок «file», указывает на то, что последующее содержимое должно выводиться в файл.
<!DOCTYPE HTML> *<html>* *<head>*	Статичная HTML-разметка

Окончание таблицы 8

Шаблон преобразования	Комментарии
<title>[model.name/]</title>	OCL-выражение, которое получает название модели
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" /> *<style type="text/css">* *table { border-collapse: collapse; }* *caption { text-align: right; margin-bottom: 0.5em; }* *tr { vertical-align: top; }* *td, th { border: 1px solid black; padding: 0.1em 0.5em; }* *td p { margin: 0.5em 0em 0em 0em; }* *td p:first-of-type { margin-top: 0em; }* *.process-flow td:nth-of-type(1) { text-align: right; }* *.process-flow td:nth-of-type(3),* *.process-flow td:nth-of-type(4),* *.process-flow td:nth-of-type(5) { width: 15%; }* *.process-flow .performer { font-weight: bold; }* *.process-flow .performer.unknown { font-style: italic; }* *</style>* *</head>* *<body>* *<table class="process-flow">* *<caption>Таблица 1. Порядок выполнения процесса «[model.name/]»</caption>* *<thead>* *<tr>* *<th>№</th>* *<th>Шаг процесса</th>* *<th>Дополнительные участники</th>* *<th>Входы</th>* *<th>Выходы</th>* *</tr>* *</thead>* *<tbody>*	Статичная HTML-разметка
[for (node : ControlFlowNode | model.nodes->selectByKind(ControlFlowNode))] *[genStep(node)/]* *[/for]*	Блок выполняет последовательный перебор всех узлов в EPC-модели (события, функции, логические операторы), к каждому из которых применяет шаблон «genStep»

Перечень элементов для перебора так же задается на языке OCL. В остальных видах блоков выражения для доступа к объектам, составляющим модель, и к их атрибутам описываются так же на языке OCL. Это универсальный язык, используемый как для валидации моделей, так и для навигации по ним.

Язык MOF M2T поддерживает полиморфизм. Это значит, что для разных типов объектов может быть задана разная структура шаблона. Шаблон «genStep» яв-

ляется полиморфным. Сначала задается структура шаблона по умолчанию для произвольных узлов EPC-модели:

[template public genStep(node : ControlFlowNode)]

[/template]

Затем задаётся шаблон для событий в EPC-модели:

[template public genStep(node : Event)]

<tr>

<td>[node.ordinal()/]</td>

<td>Событие «[node.name/]»[genNextStep(node)/]</td>

<td></td>

<td></td>

<td></td>

</tr>

[/template]

Для каждого события в выходном файле формируется строка таблицы, в первом столбце которой выводится порядковый номер узла в EPC-модели, а во втором – название события.

Шаблон для функций уже сложнее:

[template public genStep(node : Function)]

<tr>

<td>[node.ordinal()/]</td>

<td>[genPerformer(node, true)/] выполняет функцию «[node.name/][genNextStep(node)/]</td>

<td>

[for (p : Participation | node.model().connections->selectByKind(Participation)->

select(kind <> ParticipationKind::responsible and target = node))]

[if (p.kind = ParticipationKind::accountable)]

* Контроль: [p.source.name/] *

[elseif (p.kind = ParticipationKind::supportive)]

```
  <p>Поддержка: [p.source.name/]</p>
  [elseif (p.kind = ParticipationKind::consulted)]
  <p>Консультирует: [p.source.name/]</p>
  [elseif (p.kind = ParticipationKind::informed)]
  <p>Информирование: [p.source.name/]</p>
  [/if]
 [/for]
 </td>
 <td>[node.inputInformation().name->sep(', ')/]</td>
 <td>[node.outputInformation().name->sep(', ')/]</td>
</tr>
[/template]
```

Во втором столбце выводится не только название функции, но и название роли, ответственной за выполнение данной функции. В третьем столбце выводятся дополнительные роли, связанные с функцией. В четвертом и пятом столбцах выводятся входные и выходные данные для функции соответственно. Также в данном шаблоне используется новый тип блоков *«[if ...]...[elseif ...]...[else]...[/if]»,* позволяющий формировать тот или иной текст в зависимости от выполнения указанных условий.

Шаблоны для логических операторов «и», «или» и «исключающее или» аналогичны шаблонам, описанным ранее:

```
[template public genStep(node : Gate) ? (kind = GateKind::andGate)]
<tr>
 <td>[node.ordinal()/]</td>
 <td>
 [if (node.previousNodes->size() > 1)]
  Ожидается выполнение шагов [node.previousNodes.ordinal()->sepLast(', ' и
')/][if (node.nextNodes->size() > 1)].[/if]
 [/if]
 [if (node.nextNodes->size() = 1)]
```

[genNextStep(node.nextNodes->at(1))/]

[else]

Переход на шаги [node.nextNodes.ordinal()->sepLast(', ' и ')/]

[/if]

</td>

<td></td>

<td></td>

<td></td>

</tr>

[/template]

[template public genStep(node : Gate) ? (kind = GateKind::xorGate)]

<tr>

<td>[node.ordinal()/]</td>

<td>

[if (node.previousNodes->size() > 1)]

Ожидается выполнение одного из шагов [node.previousNodes.ordinal()->sepLast(', ' или ')/][if (node.nextNodes->size() > 1)].[/if]

[/if]

[if (node.nextNodes->size() = 1)]

[genNextStep(node.nextNodes->at(1))/]

[else]

Переход на один из шагов [node.nextNodes.ordinal()->sepLast(', ' или ')/]

[/if]

</td>

<td></td>

<td></td>

<td></td>

</tr>

[/template]

[template public genStep(node : Gate) ? (kind = GateKind::orGate)]

<tr>

<td>[node.ordinal()/]</td>

<td>

[if (node.previousNodes->size() > 1)]

Ожидается выполнение одного или более шагов [node.previousNodes.ordinal()->sepLast(', ', ' и (или) ')/][if (node.nextNodes->size() > 1)].[/if]

[/if]

[if (node.nextNodes->size() = 1)]

[genNextStep(node.nextNodes->at(1))/]

[else]

Переход на один или несколько шагов [node.nextNodes.ordinal()->sepLast(', ', ' и (или) ')/]

[/if]

</td>

<td></td>

<td></td>

<td></td>

</tr>

[/template]

Основное отличие данных шаблонов – это наличие предусловия в первой строке. Решение о применении одного из этих шаблонов к объекту исходной модели принимается не только на основании его типа (в данном случае – это *Gate*), но и на основании дополнительных условий (в данном случае – это тип логического оператора).

Как правило, некоторые части шаблонов выносятся в отдельные вспомогательные шаблоны. Это позволяет упростить основные шаблоны и позволяет сократить размер преобразования в целом за счет повторного использования вынесенных

фрагментов. Далее приведены два вспомогательных шаблона. Первый шаблон формирует перечень следующих шагов в процессе:

```
[template public genNextStep(node : ControlFlowNode)]
[if node.nextNodes->size() = 1]
 [let next : Integer = node.nextNodes->any(true).ordinal()]
  [if (node.ordinal() + 1 <> next)]
. Переход на шаг [next/]
  [/if]
 [/let]
[elseif node.nextNodes->size() > 1]
. Переход на шаги [node.nextNodes.ordinal()->sepLast(', ' и ')/]
[/if]
[/template]
```

Второй шаблон формирует название роли, ответственной за выполнение функции:

```
[template public genPerformer(function : Function, upperFirst : Boolean)]
 [let performer : Role = function.performer()]
<span class="performer">[performer.name.toUpperFirst(upperFirst)/]</span>
 [else]
<span class="performer unknown">['участник'.toUpperFirst(upperFirst)/]</span>
 [/let]
[/template]
```

Кроме шаблонов важными строительными блоками преобразования модель-текст являются запросы. Так же как и шаблоны, запросы имеют название, область видимости, входные параметры. Как и шаблоны, запросы могут быть полиморфными, т.е. при применении к разным типам объектов выполнять разные операции. Основное отличие запросов от шаблонов заключается в том, что вместо текста они

возвращают либо объекты модели, либо некоторые значения (например, значения атрибутов объектов). Запросы пишутся на языке OCL.

В данном преобразовании используются следующие вспомогательные запросы для работы со строками:

```
[query public toUpperFirst(str : String, upperFirst : Boolean) : String =
  if upperFirst then str.toUpperFirst() else str endif /]

[query public sepLast(seq : Sequence(String), sep : String,
                      lastSep : String) : String =
  seq->subSequence(1, seq->size() – 1)->
    iterate(str; res : String = '' |
      if res = '' then str else res + sep + str endif) +
  lastSep + seq->last() /]

[query public sepLast(seq : Sequence(Integer), sep : String,
                      lastSep : String) : String =
  seq.toString()->sepLast(sep, lastSep) /]
```

Запрос «*toUpperFirst*» преобразует в строке первую букву в заглавную при условии, что параметр «*upperFirst*» имеет истинное значение.

Запрос «*sepLast*» преобразует последовательность строк в одну строку. При этом между строками вставляется разделитель «*sep*» (например, запятая или точка с запятой), а между двумя последними элементами вставляется разделитель «*lastSep*» (например, «и» или «или»). Данный запрос является полиморфным. Если вместо последовательности строк ему передается последовательность целых чисел, то числа сначала преобразуются в строки, а затем они объединяются в одну строку в соответствии с описанной выше логикой.

Более сложные запросы работают уже не со строками, а с объектами, составляющими модель:

```
[query public model(node : ControlFlowNode) : EventDrivenProcessChain =
  node.eContainer(EventDrivenProcessChain) /]
```

```
[query public ordinal(node : ControlFlowNode) : Integer =
  node.model().nodes->selectByKind(ControlFlowNode)->indexOf(node) /]

[query public performer(function : Function) : Role =
  let p : Participation = function.model().connections->
    selectByKind(Participation)->
    any(kind = ParticipationKind::responsible and target = function) in
  if p <> null then p.source.oclAsType(Role) else null endif /]

[query public inputInformation(function : Function) : Sequence(Resource) =
  function.model().connections->selectByKind(InformationFlow)->
    select(target = function and (kind = InformationFlowKind::inputFor or
                                  kind = InformationFlowKind::usedBy)).
    source->selectByKind(Resource) /]

[query public outputInformation(function : Function) : Sequence(Resource) =
  function.model().connections->selectByKind(InformationFlow)->
    select(source = function and (kind = InformationFlowKind::outputs or
                                  kind = InformationFlowKind::creates)).
    target->selectByKind(Resource) /]
```

Запрос «*model*» возвращает модель, которой принадлежит событие, функция или другой узел EPC-модели.

Запрос «*ordinal*» возвращает порядковый номер узла в EPC-модели.

Запрос «*performer*» возвращает исполнителя заданной функции.

Запросы «*inputInformation*» и «*outputInformation*» возвращают соответственно входные и выходные сведения для функции в рамках процесса.

4.9. Реализация вспомогательных функций с состоянием на Java

Вспомогательные функции с состоянием на языке Java должны быть реализованы в виде синглетона – множества с единственным элементом. Например, множество *{0}*. Из Acceleo их следует вызывать с помощью команды *template*, а не *query*, так как результаты последних кэшируются и должны зависеть только от входных параметров, а не состояния. Важно, чтобы *template* использовались во всей цепочке вызовов.

4.9.1. Пример реализация вспомогательных функций с состоянием на Java

В языке MOF M2T (Acceleo) запросы и шаблоны, как правило, не имеют состояния. Однако, если требуется, например, реализовать нумерацию таблиц или рисунков в генерируемом документе, то шаблон с состоянием может быть реализован следующим образом.

На языке Java реализуется класс следующего вида:

```
package ru.vniief.bpms.codegen.service;

import java.util.HashMap;
import java.util.Map;

public class Counter {

    private static volatile Counter instance;

    private final Map<Object, Map<String, Integer>> contexts = new HashMap<Object, Map<String, Integer>>();

    private Counter() {
    }
```

```
public static Counter getInstance() {
    if (instance == null) {
        synchronized (Counter.class) {
            if (instance == null) {
                instance = new Counter();
            }
        }
    }
    return instance;
}
private Map<String, Integer> getCounters(Object context) {
    if (contexts.containsKey(context)) {
        return contexts.get(context);
    }
    final Map<String, Integer> counters = new HashMap<String, Integer>();
    contexts.put(context, counters);
    return counters;
}

public int incCounter(Object context, String str) {
    final Map<String, Integer> counters = getCounters(context);
    if (counters.containsKey(str)) {
        int counter = counters.get(str) + 1;
        counters.put(str, counter);
        return counter;
    }
    counters.put(str, 1);
    return 1;
}
```

```
    public int getCounter(Object context, String str) {
        final Map<String, Integer> counters = getCounters(context);
        if (counters.containsKey(str)) {
            return counters.get(str);
        }
        return 0;
    }
}
```

Комментарий к шаблонам вспомогательных функций приведен в таблице 9.

Таблица 9- Пример регламента процесса

Шаблон вспомогательных функций	Комментарии
`private static volatile Counter instance;`	Класс является синглетоном (т.е. в приложении может использоваться только один экземпляр этого класса). В данном поле хранится ссылка на этот единственный экземпляр класса
`private final Map<Object, Map<String, Integer>> contexts = new HashMap<Object, Map<String, Integer>>();`	Счетчики таблиц, рисунков или других объектов, которыми управляет данный класс
`private Counter() { }`	Прямое создание экземпляров данного класса запрещено
`public static Counter getInstance() { if (instance == null) { synchronized (Counter.class) { if (instance == null) { instance = new Counter(); } } } return instance; }`	Метод возвращает единственный экземпляр класса. Если он ещё не создан, то предварительно создает его
`private Map<String, Integer> getCounters(Object context) { if (contexts.containsKey(context)) { return contexts.get(context); } final Map<String, Integer> counters = new HashMap<String, Integer>(); contexts.put(context, counters); return counters; }`	Метод возвращает перечень счетчиков, используемых в определенном контексте. В качестве контекста может выступать любой объект, например, EPC-модель или группа процессов в VAD-модели
`public int incCounter(Object context, String str) { final Map<String, Integer> counters = getCounters(context); if (counters.containsKey(str)) {`	Увеличивает значение счетчика

Окончание таблицы 9

Шаблон вспомогательных функций	Комментарии
int counter = counters.get(str) + 1; *counters.put(str, counter);* *return counter;* *}* *counters.put(str, 1);* *return 1;* *}*	
public int getCounter(Object context, String str) { *final Map<String, Integer> counters = getCounters(context);* *if (counters.containsKey(str)) {* *return counters.get(str);* *}* *return 0;* *}*	Возвращает значение счетчика

Для удобства вызова реализованных на языке Java методов в преобразовании модель-текст могут использоваться шаблоны следующего вида:

[template public incCounter(ctx : OclAny, str : String)][invoke(
'ru.vniief.bpms.codegen.service.CommonService',
'incCounter(org.eclipse.emf.ecore.EObject, java.lang.String)',
Sequence{ctx, str})/][/template]

[template public getCounter(ctx : OclAny, str : String)][invoke(
'ru.vniief.bpms.codegen.service.CommonService',
'getCounter(org.eclipse.emf.ecore.EObject, java.lang.String)',
Sequence{ctx, str})/][/template]

Первый шаблон увеличивает значение счетчика. Второй шаблон возвращает значение счетчика.

Используются именно шаблоны (*template*), а не запросы (*query*), т.е. Acceleo кеширует результаты работы запросов. При использовании запросов данные методы будут возвращать фиксированное значение.

4.10. Поддержка национальных языков

4.10.1. Общие сведения

Используемые в программном модуле плагины совместимы с системой поддержки национальных языков платформы Eclipse. Есть три случая встречающихся при разработке:

1) плагин входит в состав платформы или ее подпроекта и имеет существующий перевод в виде фрагмента, находящегося в репозитории программных артефактов. Необходимо убедится, что фрагмент с переводом присутствует в таргет-платформе для сборки и указан в файлах конфигурации процесса сборки;

2) плагин создается в ходе разработки системы; при создании применяется следующая процедура:

- строки, подлежащие переводу, экстернализируются с использованием стандартной процедуры, обеспечиваемой IDE Eclipse (меню «Externalize strings…» в контекстном меню редактора кода). При этом генерируется необходимый код и файл ресурсов со значениями строк по умолчанию (plugin.properties). Строки, не подлежащие переводу, отмечаются инлайн-комментарием *//$NON-NLS-n$?* ;
- файл «plugin.properties» копируется в файл «plugin_ru.properties»;
- осуществляется перевод строк значений ключей в файле «plugin_ru.properties» на русский язык. Строки на национальных языках в файлах «*.properties» должны записываться в виде управляющих последовательностей Unicode.

3) плагин входит в состав проекта, не являющегося частью платформы, и не имеет перевода в репозитории. При создании плагина применяется следующая процедура:

- для перевода плагина надо создать новый проект для фрагмента к переводимому плагину, соблюдая следующие правила:

  а) расположение проектов фрагментов *bmps/bundles*, расположение фич для перевода *bpms/features*;

  б) тип проекта – *фрагмент*;

  в) имя папки проекта и проекта – *<имя-хост-бандла>.nl_ru*;

  г) пункт синглетное – *да*;

  д) пункт Java проект – *нет*;

  е) плагин ID – *<имя-хост-бандла>.nl_ru*;

  ж) имя плагина – *<имя-хост-бандла> NLS Russian Support*;

  и) вендор – имя поставщика фрагмента;

  к) хост плагин – выбрать из списка переводимый плагин;

  л) минимальная версия хоста – устанавливается автоматически;

  м) имя файла свойств – *plugin_ru.properties*, первоначальное содержимое берем из хост плагина;

  н) свойства билда – включать *plugin_ru.properties* в бинарный билд;

  п) в файле *plugin_ru.properties* кириллические символы должны быть представлены в виде управляющих последовательностей Unicode;

  р) существуют строки, которые запрещены к переводу, при переводе на них строки нужно обратить особое внимание чтобы переводом не сломать инструмент;

  с) фрагменты объединяются в опции. Опции экспортируются и могут устанавливаться в продукт или включаются в билд.

4.10.2. Пример содержимого файлов поддержки национальных языков

Исходный файл на английском языке – plugin.properties

Function = Function

FunctionHarmonization = Function Harmonization

Harmonization = Harmonization

Indicator = Indicator

IndicatorHarmonization = Indicator Harmonization

Product = Product

ProductHarmonization = Product Harmonization

RelatedObjects = Related Objects

Role = Role

RoleHarmonization = Role Harmonization

pluginName = ru.vniief.bpms.model.core.design

providerName = HWMNBN

viewpointName = MyViewpoint

Соответствующий файл на русском языке –

Function = \u0424\u0443\u043D\u043A\u0446\u0438\u044F

FunctionHarmonization = \u0413\u0430\u0440\u043C\u043E\u043D\u0438\u0437\u0430\u0446\u0438\u044F \u0444\u0443\u043D\u043A\u0446\u0438\u0439

Harmonization = \u0413\u0430\u0440\u043C\u043E\u043D\u0438\u0437\u0430\u0446\u0438\u044F

Indicator = \u041A\u043B\u044E\u0447\u0435\u0432\u043E\u0439 \u043F\u043E\u043A\u0430\u0437\u0430\u0442\u0435\u043B\u044C IndicatorHarmonization = \u0413\u0430\u0440\u043C\u043E\u043D\u0438\u0437\u0430\u0446\u0438\u044F \u043A\u043B\u044E\u0447\u0435\u0432\u044B\u0445 \u043F\u043E\u043A\u0430\u0437\u0430\u0442\u0435\u043B\u0435\u0439

Product = \u041F\u0440\u043E\u0434\u0443\u043A\u0442 \u0438\u043B\u0438 \u0443\u0441\u043B\u0443\u0433\u0430

ProductHarmonization = \u0413\u0430\u0440\u043C\u043E\u043D\u0438\u0437\u0430\u0446\u0438\u044F

\u043F\u0440\u043E\u0434\u0443\u043A\u0442\u043E\u0432 \u0438 \u0443\u0441\u043B\u0443\u0433

RelatedObjects = \u0421\u0432\u044F\u0437\u0430\u043D\u043D\u044B\u0435 \u043E\u0431\u044A\u0435\u043A\u0442\u044B

Role = \u0420\u043E\u043B\u044C

RoleHarmonization = \u0413\u0430\u0440\u043C\u043E\u043D\u0438\u0437\u0430\u0446\u0438\u044F \u0440\u043E\u043B\u0435\u0439

pluginName = ru.vniief.bpms.model.core.design

providerName = HWMNBN

viewpointName = MyViewpoint

5. ВХОДНЫЕ И ВЫХОДНЫЕ ДАННЫЕ

5.1. Входные данные

Ко входным данным относятся:

- действия пользователей программного модуля BPMS, создающие посредством человеко-машинного интерфейса артефакты поддерживаемых программным модулем типов;
- импортируемые из файловой системы данные, содержащие артефакты поддерживаемых программным модулем BPMS типов, записанные в поддерживаемых программным модулем BPMS форматах;
- данные из репозитория, представляющие собой версии артефактов поддерживаемых программным модулем BPMS типов;
- данные для анализа из систем исполнения моделей, записанные в поддерживаемых программным модулем BPMS форматах;
- модели и нотации моделирования, получаемые с технологической платформы.

5.2. Выходные данные

К выходным данным относятся:

- человеко-читаемые представления артефактов программного модуля BPMS, выводимые через человеко-машинный интерфейс программного модуля BPMS;
- экспортируемые в файловую систему данные, содержащие артефакты программного модуля BPMS в поддерживаемых программным модулем BPMS форматах;
- передаваемые в репозиторий данные, представляющие собой новые версии артефактов программного модуля BPMS;
- отчеты, генерируемые программным модулем BPMS в виде офисных и текстовых документов;
- модели и нотации моделирования, передаваемые на технологическую платформу.

6. СООБЩЕНИЯ ОБ ОШИБКАХ ПРОГРАММИСТУ ПРОГРАММНОГО МОДУЛЯ BPMS

Сообщения об ошибках отображаются на панели «Неполадки» в рабочей среде BPMS Modeler и BPMS Notator. Пример рабочей среды BPMS Modeler при возникновении ошибки приведен на рис. 11.

Активная панель «Неполадки» в рабочей среде BPMS Modeler

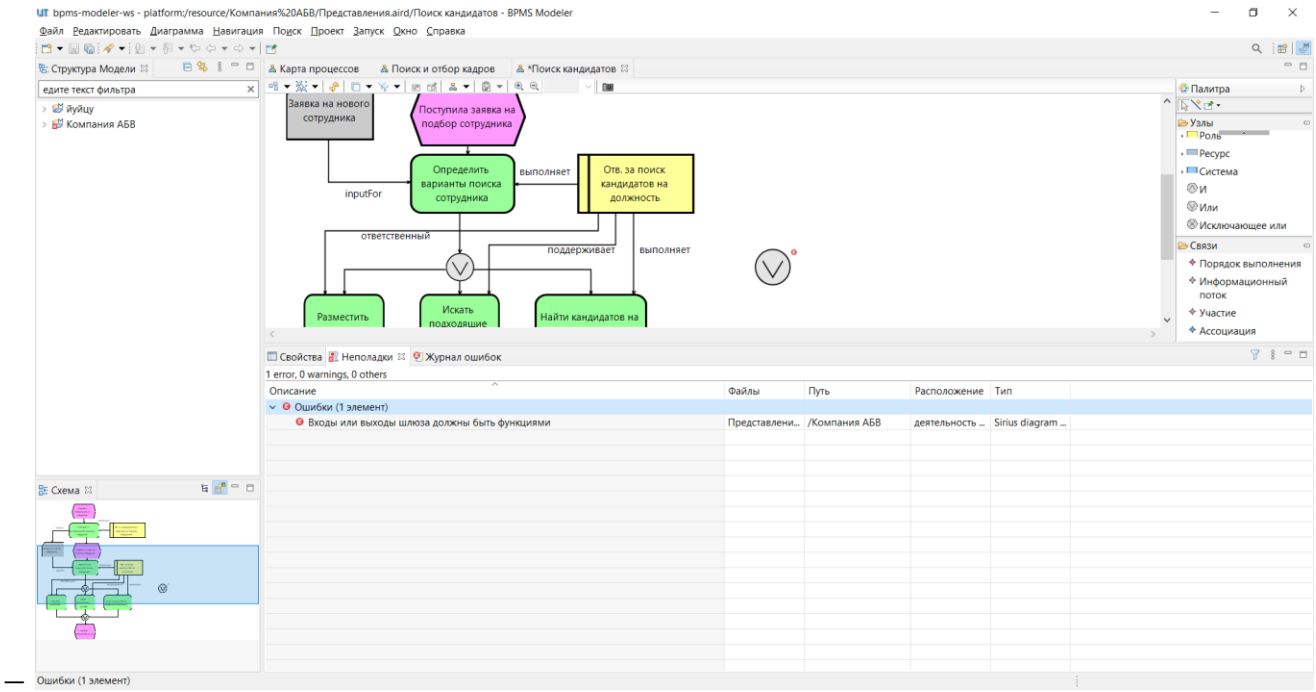

Рисунок 11

При работе с инструментом BPMS Modeler при валидации модели у программиста могут возникать следующие сообщения об ошибках:

- все входы шлюза должны быть либо событиями, либо функциями;
- все выходы шлюза должны быть либо событиями, либо функциями;
- входы или выходы шлюза должны быть функциями;
- разветвляющийся шлюз «или» («исключающее или») должен инициироваться функцией;
- роль должна быть владельцем хотя бы одного подпроцесса;
- информационный поток должен связывать функцию и ресурс;
- исходным узлом потока управления должно быть событие, функция, шлюз или процесс;

- целевым узлом потока управления должно быть событие, функция, шлюз или процесс;
- значения должны быть уникальными;
- роль должна быть владельцем хотя бы одного процесса;
- роль, являющаяся владельцем единственного процесса, должна принадлежать той же группе процессов, что и сам процесс;
- ключевой показатель должен быть связан хотя бы с одним подпроцессом;
- группа процессов должна содержать другие группы процессов или процессы;
- продукт/услуга должен быть входом или выходом подпроцесса;
- для элемента должно быть задано имя.

Общее описание правил валидации, содержащих сообщения об ошибках, приведено в таблице 10.

Таблица 10 – Правила валидации, содержащие сообщения об ошибках

Метамодель	**Вид объектов**	**Вид объектов (мнемоника)**	**Обозначение правила**	**Описание правила (рус.)**	**Описание правила (англ.)**	**Спецификация правила (OCL)**	**Вид правила**
событийная цепочка процесса	логический шлюз	Gate	GateInputs	все входы шлюза должны быть либо событиями, либо функциями	all gate inputs should be either events or functions	previousNodes->forAll(oclIsKindOf(Event)) or previousNodes->forAll(oclIsKindOf(Function))	требование
событийная цепочка процесса	логический шлюз	Gate	GateOutputs	все выходы шлюза должны быть либо событиями, либо функциями	all gate outputs should be either events or functions	nextNodes->forAll(oclIsKindOf(Event)) or nextNodes->forAll(oclIsKindOf(Function))	требование
событийная цепочка процесса	логический шлюз	Gate	AtLeastOneFunction	входы или выходы шлюза должны быть функциями	either inputs or outputs of a gate should be functions	previousNodes->exists(oclIsKindOf(Function)) or nextNodes->exists(oclIsKindOf(Function))	требование
событийная цепочка процесса	логический шлюз	Gate	TrggerSingleFunction	разветвляющийся шлюз "или" ("исключающее или") должен инициироваться функцией	split or-gate (xor-gate) should be triggered by a function	((kind = GateKind::orGate or kind = GateKind::xorGate) and nextNodes->size() > 1 implies previousNodes->forAll(oclIsKindOf(Function)))	требование
цепочка добавленного	роль	Role	OwnSubprocess	роль должна быть владельцем хотя	role should own at least one subprocess	Subprocess.allInstances	требование

Продолжение таблицы 10

Метамодель	Вид объектов	Вид объектов (мнемоника)	Обозначение правила	Описание правила (рус.)	Описание правила (англ.)	Спецификация правила (OCL)	Вид правила
качества				бы одного подпроцесса		()->exists(owner = self)	
событийная цепочка процесса	связь	Connection	SourceNotEqualTarget	связь должна иметь разные исходный и целевой узлы	connection should have different source and target nodes	source <> target	требование
событийная цепочка процесса	информационный поток	InformationFlow	FunctionInformationFlow	информационный поток должен связывать функцию и ресурс	information flow should connect a function and a resource	source.oclIsKindOf(Function) and target.oclIsKindOf(Resource) or source.oclIsKindOf(Resource) and target.oclIsKindOf(Function)	требование
событийная цепочка процесса	поток управления	ControlFlow	ControlFlowSource	исходным узлом потока управления должно быть событие, функция, шлюз или процесс	control flow source should be event, function, gate он process	source.oclIsKindOf(ControlFlowNode)	требование
событийная цепочка процесса	поток управления	ControlFlow	ControlFlowTarget	целевым узлом потока управления должно быть событие, функция, шлюз или процесс	control flow target should be event, function, gate or process	target.oclIsKindOf(ControlFlowNode)	требование
событийная цепочка процесса	событийная цепочка процесса	EventDrivenProcessChain	UniqueStringFeatures	значения должны быть уникальными	feature values must be unique	(true)	требование
карта процессов верхнего	роль	Role	OwnSubprocess	роль должна быть владельцем хотя	role should own at least one process	Process.allInstances)	требование

Окончание таблицы 10

Метамодель	**Вид объектов**	**Вид объектов (мнемоника)**	**Обозначение правила**	**Описание правила (рус.)**	**Описание правила (англ.)**	**Спецификация правила (OCL)**	**Вид правила**
уровня				бы одного процесса		()->exists(owner = self	
карта процессов верхнего уровня	роль	Role	SameProcessGroup	роль, являющаяся владельцем единственного процесса, должна принадлежать той же группе процессов, что и сам процесс	role owning a single process should belong to the same process group as this process	let processes = Process.allInstances()->select(owner = self) in processes->size() = 1 implies oclContainer() = processes->any(true).oclContainer()	требование
цепочка добавленного качества	ключевой показатель	Indicator	OwnSubprocess	ключевой показатель должен быть связан хотя бы с одним подпроцессом	indicator should be assigned to at least one subprocess	Subprocess.allInstances().indicators->includes(self)	требование
карта процессов верхнего уровня	группа процессов	ProcessGroup	NotEmpty	группа процессов должна содержать другие группы процессов или процессы	process group should contain subgroups or processes	groups->notEmpty() or processes->notEmpty()	рекомендация
цепочка добавленного качества	продукт/услуга	Product	OwnSubprocess	продукт/услуга должен быть входом или выходом подпроцесса	product should be either input or output of a subprocess	Subprocess.allInstances().inputs->includes(self) or Subprocess.allInstances().outputs->includes(self)	требование

ПЕРЕЧЕНЬ ТЕРМИНОВ

Apache Maven Tycho	–	инструмент автоматизации сборки проектов на основе описания их структуры в файлах на языке POM (англ.), являющемся подмножеством XML
Astra Linux Special Edition	–	сертифицированная операционная система со встроенными средствами защиты информации
AQL (Acceleo Query Language)	–	декларативный язык для описания запросов к объектным моделям
Eclipse Sirius	–	платформа создания и среда выполнения для разработки и использования редакторов представлений моделей
Java	–	строго типизированный объектно-ориентированный язык программирования
Java (LTS) JRE	–	обозначение базовой версии среды выполнения программного модуля BPMS
Java (LTS) JDK	–	обозначение базовой версии среды разработки и выполнения программного модуля BPMS
OCLinEcore Editor	–	редактор OCL-правил
OCL-правило	–	ограничение, выраженное на языке OCL
Windows	–	семейство коммерческих операционных систем корпорации Microsoft
Атрибут	–	некоторая характеристика элемента, имеющая имя и значение (например, название, тип или идентификатор)
Кэширование	–	процесс сохранения данных в выделенной области памяти, обеспечивающий оперативный доступ к ним при последующих запросах
Методология	–	совокупность принципов, методов и правил описания архитектуры предприятия, определенных при помощи предметных областей, типов моделей, нотаций, ограничений, правил валидации и представлений (например,

		методология «ARIS»)
Модель	–	представление архитектуры предприятия в виде совокупности объектов и связей, разработанное в соответствии с определенной нотацией
Моделирование	–	процесс описания архитектуры предприятия по правилам выбранной методологии
Нотация	–	совокупность типов объектов, типов связей, ограничений и спецификаций представлений, предназначенная для описания сущностей и отношений реального мира (например, UML, VAD, BPMN2.0, EPC, ERD, DFD, IDEF0)
Объект	–	информационная сущность с заданными свойствами описываемой предметной области моделирования
Плагин	–	(англ. plug-in, от plug in «подключать») – программный модуль, подключаемый к основной программе и предназначенный для расширения и/или использования её возможностей
Правило валидации	–	способ проверки ограничения, который может быть немедленным или отложенным (инициируется по запросу или другим действием)
Программист	–	опытный пользователь ПК, выполняющий мероприятия по разработке инструментов программного модуля BPMS, а также доработке этих инструментов и сборке программного модуля из исходных кодов
Проект	–	создаваемая, загружаемая и редактируемая в программном модуле BPMS совокупность ресурсов
Ресурс	–	объект данных, архитектурная единица программного модуля BPMS

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ

BPMS	–	система моделирования процессов жизненного цикла изделий;
BPMS Modeler	–	инструмент моделирования деятельности предприятия, собранный из исходных кодов программного модуля BPMS
BPMS Notator	–	инструмент для создания нотаций и фильтров, собранный из исходных кодов программного модуля BPMS
MOF (OMG MOF)	–	Object Management Group Meta-Object Facility, стандарт для формального описания языков моделирования (таких как UML, BPMN, EPC и др.)
OMG	–	Object Management Group, группа управления объектами – рабочая группа (консорциум), занимающаяся разработкой и продвижением объектно-ориентированных технологий и стандартов
ЛКМ	–	левая кнопка мыши
ОС	–	операционная система
ПО	–	программное обеспечение

Лист регистрации изменений									
Номера листов (страниц)					Всего листов (страниц) в докум.	№ документа	Входящий № сопроводительного докум. и дата	Подп.	Дата
Изм.	измененных	замененных	новых	аннулированных					