Национальный исследовательский университет – Высшая школа экономики
Факультет бизнес-информатики, отделение программной инженерии

УТВЕРЖДЕН
Заведующий кафедрой «Управление разработкой программного обеспечения»
_____ / Авдошин С.М./
«____»_____ 2012 г.

# КОМПОНЕНТНАЯ МОДЕЛЬ С ДЕКЛАРАТИВНЫМ ОПИСАНИЕМ СОСТАВНЫХ ТИПОВ: ПАРСЕРЫ

Руководство программиста

ЛИСТ УТВЕРЖДЕНИЯ

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

Руководитель работы
_____ / Гринкруг Е.М./
«____»_____ 2012 г.

Исполнитель: студент группы 271ПИ
_____ / Дубов М.С. /
«____»_____ 2012 г.

2012

Национальный исследовательский университет – Высшая школа экономики
Факультет бизнес-информатики, отделение программной инженерии

УТВЕРЖДЕН

# КОМПОНЕНТНАЯ МОДЕЛЬ С ДЕКЛАРАТИВНЫМ ОПИСАНИЕМ СОСТАВНЫХ ТИПОВ: ПАРСЕРЫ

Руководство программиста

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

Листов 10

2012

## Содержание

# 1. Назначение и условия применения программы

## *1.1. Функциональное назначение*

Программный комплекс предназначен для построения компонентных моделей на основе их описания на одном из поддерживаемых декларативных языков (VRML/X3D), а также для генерации декларативного описания уже существующих моделей.

## *1.2. Эксплуатационное назначение*

Библиотека предназначена для использования сторонними разработчиками при разработке ими других приложений, например:

- редакторов компонентных моделей;
- анализаторов ошибок в коде декларативного описания (например, в специальных текстовых редакторах)
- программ визуализации компонентных моделей (например, в виде 3D-сцен);
- программ-конвертеров между представлениями одной и той же модели на разных декларативных языках.

## *1.3. Требования к программной совместимости*

Для работы библиотеки необходима реализация виртуальной машины Java версии не ниже 6 (например, Java Runtime Environment). При использовании библиотека в ходе разработки других программных продуктов необходимо также наличие средств Java Development Kit.

## *1.4. Требования к составу и параметрам технических средств*

Библиотека распространяется на оптических дисках и через интернет. Для чтения оптических дисков компьютер должен быть оснащен устройством чтения дисков CD; операционная система должна поддерживать файловую систему iso9660.

# 2. Характеристики программы

## *2.1. Временные характеристики*

В случае анализа как VRML, так и X3D парсер работает за время $O(n)$, где $n$ – число символов во входном файле; построение графа сцены требует одного линейного прохода по файлу.

Генерация кода осуществляется за время $O(mn)$, где $m$ – число корневых узлов, а $n$ – глубина направленного ациклического графа сцены.

## *2.2. Характеристики памяти*

Максимальное количество используемой парсерами в ходе синтаксического анализа памяти линейно зависит от максимального уровня вложенности встречающихся в исходном тексте узлов. Построенный в результате граф сцены требует для хранения $O(mn)$ ячеек памяти ($m$ – число корневых узлов, а $n$ – максимальный уровень вложенности узлов).

### *2.3. Режим работы*

Парсер VRML предусматривает анализ ошибок во входном файле. Так, лексические ошибки распознаются при неверном написании названий полей и узлов; использование рефлексии при этом позволяет получать список идентификаторов, возможных на месте неверного, и предлагать программисту на основе этого списка наиболее вероятное исправление. Синтаксические ошибки имеют место, например, при отсутствии в нужных местах открывающих/закрывающих фигурных скобок. Встречая такие ошибки, парсер способен восстанавливаться, что позволяет ему продолжать чтение исходного файла и, таким образом, сообщать о как можно большем числе ошибок за один проход.

При наличии ошибок парсеры вместо ссылки на граф сцены возвращают *null*. Список ошибок при этом может быть получен методом *getParsingErrors()* класса Parser.

## 3. Обращение к программе

### *3.1. Библиотека парсеров*

#### *3.1.1. Абстрактный класс Parser*

| Имя | Модификаторы | Тип | Аргументы | Описание |
|---|---|---|---|---|
| **Методы** | | | | |
| **parse** | public | ArrayList <Node> | InputStream Reader | Принимает на вход строковый поток и возвращает граф сцены, либо null в случае наличия ошибок в исходном тексте. |
| **setUp Tokenizer** | protected | void | - | Настраивает лексический анализатор. |
| **init** | protected abstract | void | - | Инициализирует парсер, читает первую лексему из потока. |
| **parseScene** | protected abstract | void | - | Формирует граф сцены на основе синтаксического анализа входного файла. |
| **parseChild Node** | public abstract | Node | - | Читает из входного потока следующий узел и возвращает экземпляр соответствующего класса. |
| **parseValue Type** | protected | Object | Class<?> | Читает из входного потока значение заданного типа. |
| **tokenizer** | public | Stream Tokenizer | - | Возвращает объект, представляющий лексический анализатор. |
| **lookahead** | public | boolean | String | Сравнивает текущую лексему в потоке с аргументом. |
| **lookahead** | public | String | - | Возвращает текущую лексему в потоке. |
| **nextToken** | public abstract | boolean | - | Считывает следующую лексему из потока. |

| | | | | |
|---|---|---|---|---|
| **match** | public | boolean | String | Сопоставляет текущую лексему с аргументом и генерирует ошибку в случае несоответствия. |
| **tryMatch** | public | boolean | String | Сопоставляет текущую лексему с аргументом; запоминает возможную ошибку в случае несоответствия. |
| **registerError** | public | boolean | Error | Регистрирует ошибку парсинга. |
| **getParsing Errors** | public | ArrayList <Error> | - | После неудачной попытки парсинга возвращает сформированный список ошибок. |
| **classFor NodeName** | protected | Class<?> | String | Осуществляет поиск класса-узла по имени в одном из зарегистрированных пакетов с классами-узлами. |
| **create Instance** | protected | Node | String | Возвращает объект класса-узла по его имени. |
| **registerNode Package** | public | void | String | Регистрирует пакет с классами-узлами. |
| **Поля** | | | | |
| **tokenizer** | protected | Stream Tokenizer | | Лексический анализатор. |
| **sceneGraph** | protected | ArrayList <Node> | | Граф сцены, представляемый в виде списка корневых узлов. |
| **parsing Errors** | protected | ArrayList <Error> | | Список ошибок парсинга. |
| **possibleError** | protected | Error | | Возможная ошибка, зарегистрированная в tryXxx методе. |
| **nodePackages** | protected | ArrayList <String> | | Список зарегистрированных пакетов с классами-узлами. |

### *3.1.2. Класс VRMLParser*

| Имя | Модифика-торы | Тип | Аргументы | Описание |
|---|---|---|---|---|
| **Методы** | | | | |
| **setUp Tokenizer** | protected | void | - | Настраивает лексический анализатор. |
| **init** | protected | void | - | Инициализирует парсер, читает первую лексему из потока. |
| **parseScene** | protected | void | - | Формирует граф сцены на |

| | | | | |
|---|---|---|---|---|
| | | | | основе синтаксического анализа входного файла. |
| **parseXxx […]** | public | boolean | - | Один из методов, соответствующих продукциям грамматики VRML. |
| **nextToken** | public | boolean | - | Считывает следующую лексему из потока. |
| **lookahead IsId** | private | boolean | - | Определяет, является ли текущая лексема идентификатором. |
| **lookahead IsFieldName** | private | boolean | - | Определяет, является ли текущая лексема названием одного из полей текущего узла. |
| **tryMatch FieldId** | private | boolean | - | Осуществляет попытку сопоставления текущей лексемы с названием поля текущего узла. |
| **tryMatch TypeId** | private | boolean | - | Осуществляет попытку сопоставления текущей лексемы с именем узла. |
| **panicMode Recovery** | private | boolean | - | Восстановление после ошибок парсинга в «режиме паники». |
| **instantiate Node** | private | boolean | - | Инстанциирует узел по его типу. |
| **instantiate NodeById** | private | boolean | - | Инстанциирует узел по его ID с помощью хэш-таблицы узлов. |
| **addRootNode** | private | boolean | - | Добавляет текущий узел на первый уровень графа |
| **matchField ValueAnd SetField** | private | boolean | - | Считывает значение поля и записывает его в объект-узел. |
| **initFields** | private | void | - | Инициализирует private-поля класса. |
| **Поля** | | | | |
| **defNodes Table** | private | HashMap <String, Node> | | Хэш-таблица DEF-узлов (узлов с ID). |
| **lookahead** | protected | String | | Текущая лексема. |
| **currentId** | private | String | | Текущий ID. |
| **currentType** | private | String | | Тип текущего узла. |
| **currentNodes** | private | Stack<Node> | | Стек обрабатываемых узлов. |
| **currentField** | private | Stack <String> | | Стек считываемых полей. |

### *3.1.3. Класс X3DParser*

| Имя | Модификаторы | Тип | Аргументы | Описание |
|---|---|---|---|---|
| **Методы** | | | | |
| **setUp Tokenizer** | protected | void | - | Настраивает лексический анализатор. |
| **init** | protected | void | - | Инициализирует парсер, читает первую лексему из потока. |
| **parseScene** | protected | void | - | Формирует граф сцены на основе синтаксического анализа входного файла. |
| **parseXML** | private | void | - | Осуществляет чтение XML и вызов обработчиков SAX-событий. |
| **openingTag** | private | void | String | Обработчик SAX-события «Открывающий тег» |
| **closingTag** | private | void | String | Обработчик SAX-события «Закрывающий тег» |
| **attribute** | private | void | String | Обработчик SAX-события «Атрибут» |
| **textNode** | private | void | String | Обработчик SAX-события «Текстовый узел» |
| **nextToken** | public | boolean | - | Считывает следующую лексему из потока. |
| **match AttributeId** | private | boolean | - | Считывает текущую лексему, которая должна быть идентификатором. |
| **matchField ValueAnd SetField** | private | boolean | String | Считывает значение поля и записывает его в объект-узел. |
| **initFields** | private | void | - | Инициализирует private-поля класса. |
| **Поля** | | | | |
| **defNodes Table** | private | HashMap <String, Node> | | Хэш-таблица DEF-узлов (узлов с ID). |
| **lookahead** | protected | String | | Текущая лексема. |
| **readingTag** | private | boolean | | Определяет, происходит ли в данный момент считывание тега. |
| **current Attribute** | private | String | | Имя текущего атрибута. |
| **currentNodes** | private | Stack<Node> | | Стек обрабатываемых узлов. |
| **currentTags** | private | Stack <String> | | Стек считываемых тегов. |
| **fieldValue Name Attributes** | private | Stack <String> | | Вспомогательный стек для чтения значений типа MFNode. |

| | | | | |
|---|---|---|---|---|
| **fieldValue MFNodes** | private | Stack <String> | | Вспомогательный стек для чтения значений типа MFNode. |

## *3.2. Библиотека средств генерации кода*

### *3.2.1. Абстрактный класс CodeGenerator*

| Имя | Модификаторы | Тип | Аргументы | Описание |
|---|---|---|---|---|
| **Методы** | | | | |
| **generate** | public abstract | void | ArrayList <Node>, PrintStream | Генерирует декларативное описание графа сцены. |
| **VRMLtoX3D** | public static | boolean | InputStream Reader, PrintStream | Конвертирует код на VRML в код на X3D. |
| **X3DtoVRML** | public static | boolean | - | Конвертирует код на X3D в код на VRML. |

### *3.2.2. Класс VRMLCodeGenerator*

| Имя | Модификаторы | Тип | Аргументы | Описание |
|---|---|---|---|---|
| **Методы** | | | | |
| **generate** | public | void | ArrayList <Node>, PrintStream | Генерирует декларативное описание графа сцены. |
| **process** | private | void | Node | Обрабатывает один узел и рекурсивно все его дочерние узлы. |
| **Поля** | | | | |
| **nodes** | private | Stack<Node> | | Стек обрабатываемых узлов. |
| **output** | private | PrintStream | | Выходной поток. |
| **defNodes** | private | HashSet <String> | | Хэш-таблица встреченных именованных узлов. |

### *3.2.3. Класс X3DCodeGenerator*

| Имя | Модификаторы | Тип | Аргументы | Описание |
|---|---|---|---|---|
| **Методы** | | | | |
| **generate** | public | void | ArrayList <Node>, PrintStream | Генерирует декларативное описание графа сцены. |
| **process** | private | void | Node | Обрабатывает один узел и рекурсивно все его дочерние узлы. |
| **Поля** | | | | |

| | | | | |
|---|---|---|---|---|
| **nodes** | private | Stack<Node> | | Стек обрабатываемых узлов. |
| **output** | private | PrintStream | | Выходной поток. |
| **defNodes** | private | HashSet <String> | | Хэш-таблица встреченных именованных узлов. |

### *3.3. Стандартные узлы VRML/X3D*

#### *3.3.1. Абстрактный базовый класс Node*

| Имя | Модифика-торы | Тип | Аргументы | Описание |
|---|---|---|---|---|
| **Методы** | | | | |
| **setId** | public | void | String | Задает ID узла. |
| **getId** | public | String | - | Возвращает ID узла. |
| **Node** | public | - | - | Публичный конструктор без параметров. |
| **containerField** | public abstract | string | | Возвращает значение свойства containerField узла (необходимо для X3D-парсинга). |
| **Поля** | | | | |
| **id** | private | String | | ID узла. |
| **serialVersionUID** | private static final | long | | Для сериализации узла. |

#### *3.3.2. Стандартные и сторонние классы-узлы*

Библиотека содержит набор стандартных VRML-узлов, реализованных в виде JavaBeans-компонент. Все эти узлы соответствуют набору требований:

- Реализуют public-конструктор без параметров;
- Обеспечивают доступ к полю xxx на чтение через метод getXxx();
- Обеспечивают доступ к полю xxx на запись через метод setXxx(T value).

Стандартная библиотека может быть расширена пользовательскими узлами, которые также должны быть построены в соответствии со стандартом JavaBeans.

## 4. Входные и выходные данные

### *4.1. Организация входной информации*

Входными данными для парсеров VRML и X3D являются текстовые файлы, содержащие декларативное описание сцены на этих языках в соответствии со стандартом ISO, не содержащие выражения ROUTE и PROTO. Число узлов, которые могут быть описаны в этих файлах, также ограничено стандартом, однако может быть расширено программистом с помощью реализации соответствующих Java-классов и регистрации их перед использованием парсера.

### *4.2. Организация выходной информации*

Выходные данные парсера – направленный ациклический граф сцены, представленный в виде массива корневых узлов. Каждый же узел представляет собой экземпляр специального класса, соответствующего определенному типу узла и являющегося наследником определенного в библиотеке класса Node. Каждый из таких классов должен быть реализован в соответствии со стандартом JavaBeans, что позволяет выполнять их интроспекцию с помощью механизма рефлексии. Так, ссылки на дочерние узлы содержатся в getter'ах узлов и легко могут быть получены в ходе обхода графа.

Выходные данные парсеров являются входными данными для средства кодогенерации; входные для парсеров, соответственно, выходными для генераторов кода.

## 5. Сообщения

При наличии ошибок парсеры вместо ссылки на граф сцены возвращают *null*; список возникших ошибок (типа *ArrayList<Error>*)может быть получен с помощью метода парсера *getParsingErrors()*.

Каждая ошибка содержит в себе свое описание, которое включает краткую ее диагностику, возможные пути исправления и номер строки кода, где она была обнаружена. Эти информация может быть получена для каждого объекта методом *getMessage()* класса Error.

Каждая ошибка может быть одного из следующих типов:

- *LexicalError* – лексическая ошибка;
- *SyntaxError* – синтаксическая ошибка;
- *TypeMismatchError* – ошибка несоответствия типов;
- *Warning* – предупреждение;
- *ParsingError* – ошибка работы парсера.