УДК 004.4'2

Артамонова А.А.

бакалавр по направлению «Информационные системы и технологии»,

Национальный исследовательский ядерный университет «МИФИ»,

Российская Федерация, Москва

РАЗРАБОТКА ИНСТРУМЕНТА ВИЗУАЛЬНОГО ПРОЕКТИРОВАНИЯ, РЕДАКТИРОВАНИЯ И ГЕНЕРАЦИИ КОДА JAVA-ПРИЛОЖЕНИЙ

Аннотация: проектирование информационных систем становится все более и более трудоемким и ответственным этапом. Для упрощения данной задачи, обеспечения высокого качества и минимизации ошибок используются CASE-инструменты, в частности, инструменты для построения UML-диаграмм. В статье приведено описание разработки инструмента графического проектирования, редактирования, и генерации кода Java-приложений, в том числе модульных приложений с использованием Jigsaw, минимизирующего недостатки существующих UML-средств проектирования.

Ключевые слова: кодогенерация, CASE-инструменты, Java, InteliJ, разработка плагина

Artamonova A.A.

bachelor in specialty «Information systems and technologies»,

National Research Nuclear University MEPhI),

Moscow, Russian Federation

DEVELOPMENT OF A TOOL FOR VISUAL DESIGN, EDITION AND CODE GENERATION FOR JAVA APPLICATIONS

Abstract: the design of information systems is becoming more and more labor-consuming and responsible stage. To simplify this task, ensure high quality and minimize errors, CASE tools are used, in particular, tools for constructing UML diagrams. The article describes the development of a tool for graphical design, editing, and code generation for Java applications, including modular applications using Jigsaw, which minimizes the shortcomings of existing UML design tools.
Key words: code generation, CASE-tools, Java, InteliJ, UML

Введение

В современном мире компьютерные технологии занимают все больше и больше сфер, информационные системы усложняются, и их проектирование становится трудоемким и ответственным этапом. Для упрощения процесса проектирования информационных систем, обеспечения высокого качества и минимизации ошибок используются CASE-инструменты. Типичными CASE-инструментами, используемыми проектирования программных продуктов, являются инструменты для построения UML-диаграмм. Однако существующие UML-инструменты обладают множеством недостатков, связанных с абстрагированностью UML от конкретных языков программирования. Целью работы является разработка инструмента графического проектирования, редактирования, и генерации кода Java-приложений, в том числе модульных приложений с использованием Jigsaw [1], представляющий из себя набор связанных пакетов и классов и используемые ими данные и ресурсы [2]. Инструмент должен минимизировать недостатки существующих UML-средств проектирования.

Требования

Целью работы является разработка инструмента графического проектирования Java-приложений, в т.ч. модульных. В результате изучения современных проблем визуального проектирования Java-приложений, к разрабатываемому инструменту был выдвинут ряд требований, представленных ниже.

Представление структуры разрабатываемого приложения

Инструмент должен иметь возможность визуально отображать следующие компоненты приложения: модуль, пакет, класс, интерфейс, энумерация, члены классов. Дочерние компоненты по отношению к остальным должны располагаться в соответствии с распространенным стандартом, т.е находиться внутри контейнера родительского компонента, при этом родительские отношения определяются как соответствующие в файловой системе. Также инструмент должен иметь возможность визуального представления членов класса, компонент, соответствующий члену класса, при этом должен располагаться внутри контейнера этого класса. Визуальное представление члена класса должно включать в себя информацию о его типе, модификаторах и имени.

Построение диаграммы и кодогенерация

Под диаграммой понимается графическое представление структуры приложения или его компонента, удовлетворяющее требованиям, описанным в предыдущем пункте.

Инструмент должен обладать механизмами построения диаграмм на основе стороннего корректного исходного кода. А также позволять немедленную синхронизацию диаграммы с соответствующими файлами в файловой системе. Также должен позволять генерировать или

модифицировать файлы исходного кода на основе диаграммы.

Контекст разработки

Инструмент должен позволять вести разработку как на уровне всего приложения, так и в контексте определенного компонента приложения, в частности, модуля, пакета, класса. Т.е. отображаемая структурная диаграмма должна быть представлена компонентом, соответствующим текущему контексту разработки и дочерними элементами этого компонента.

Навигация

Инструмент должен обеспечивать возможность изменения контекста разработки с помощью пользовательского интерфейса. В частности, поддерживать перечисленные ниже навигационные операции:

- открытие контекста. Переход к произвольному контексту, если соответствующий компонент отображается на текущей диаграмме;
- вверх. Переход к контексту, соответствующий компонент которого в настоящий момент является родительским по отношению к компоненту текущего контекста;
- назад. Переход к контексту, который был открыт до текущего, если соответствующий компонент существует;
- вперед. Переход к контексту, с которого пользователь вернулся, используя операцию «назад», если соответствующий компонент существует.

Добавление, рефакторинг и удаление элементов

Инструмент должен позволять добавлять новые компоненты и члены классов на диаграмму, если полученное расположение компонентов является корректной структурой приложения, а также производить их удаление с помощью пользовательского интерфейса. Кроме того, он должен

предоставлять возможность производить рефакторинг идентификаторов этих элементов, обновляя изменения для всех ссылок на него в исходных кодах приложения и изменять его тип и модификаторы.

Помощь при добавлении элементов

Инструмент должен устанавливать наиболее подходящие модификаторы для добавляемых элементов в зависимости от контекста разработки, если их можно явно определить на основе текущего контекста и характера добавляемого элемента и его родителя.

Описание реализации решения

Интеграция с платформой реализована через механизм расширений. Класс PluginEngine реализует интерфейс фабрики ToolWindow. Этот класс инициализирует пользовательский интерфейс, и впоследствии все основные управляющие функции передаются ему. Также этот класс реализует механизм отслеживания событий виртуальной файловой системы IntelliJ, обрабатывая добавление файлов в снимок, их удаление или изменение контента. Все внесенные изменения немедленно отображаются на диаграмме.

Пользовательский интерфейс

Окно плагина разделено на три области:

- рабочая область, на которой отображается диаграмма в соответствии с текущим контекстом проектирования;
- панель инструментов. На ней располагаются элементы пользовательского интерфейса, отвечающие за добавление элементов диаграммы, кнопки навигации и т.д.;
- область модульного окружения. На ней отображаются классы из

других модулей, обладающие модификаторами доступа protected или public, входящие в пакеты, требуемые для открытого модуля;

- для загрузки изображений реализован класс GUIManager. При инициализации он рекурсивно обходит соответствующий каталог ресурсов, определяет файлы изображений и загружает их в память, впоследствии предоставляя доступ к ним по имени файла, которое содержало изображение без расширения.

Структура дерева проекта

Для представления структуры проекта реализован класс дерева. Узлы дерева являются подклассами класса Node, который в свою очередь является подклассом графического компонента библиотеки Swing JPanel. Каждый узел соответствует одному из следующих компонентов приложения: корень проекта, модуль, пакет, класс, энумерация, интерфейс.

Представление перечисленных выше компонентов наследниками одного класса позволяет унифицировать API, разделяемый компонентами, в частности, менеджмент шапки элемента, менеджмент контента, графический интерфейс для удаления и рефакторинга элемента, графическое выделение элемента и механизмы навигации. Дерево перестраивается всякий раз, когда файл снимка был изменен или изменения были внесены с помощью пользовательского интерфейса плагина. К актуальному состоянию дерева можно получить доступ через класс PluginEngine. Для отображения текущего контекста в рабочую область добавляется узел, соответствующий этому контексту, и таким образом отображаются только подкаталоги каталога текущего контекста проектирования и содержащиеся в них файлы.

Построение дерева проекта

Построение дерева состоит из двух этапов, на первом этапе создается корень дерева являющийся корнем проекта, затем с помощью API платформы формируется список модулей проекта, для каждого модуля создается узел дерева. Второй этап - рекурсивное построение дочерних ветвей модулей. Рекурсивный метод построения представлен на рисунке 2.

Получение дочерних элементов в циклах реализовано через API соответствующих PSI-структур. Метод createClass на основе PSI-класса, создает узел одного из следующих типов: Enum, Interface или SimpeClass. После построения дерева вызывается соответствующее событие.

Размещение элементов на диаграмме

Для задачи обратного обхода дерева компонентов, описанной в соответствующей теоретической главе, в библиотеке AWT используется интерфейс менеджеров компоновки [3]. Поэтому для размещения элементов диаграммы, соответствующих компонентам, имеющим общего непосредственного родителя, т.е. элементов, располагающихся в одном контейнере, разработан класс, реализующий соответствующий интерфейс библиотеки AWT. Упаковка реализована на основе метода максимальных прямоугольников. Структура полученных при разбиении прямоугольников хранится в виде бинарного дерева. При добавлении нового графического элемента в контейнер для корня дерева вызывается рекурсивный метод insert(), размещающий прямоугольник, соответствующий предпочитаемым размерам компонента(component.getPrefferedSize()) с учетом границ, затем координаты добавляемого компонента устанавливаются в соответствии узлом, в который он был размещен.

Навигация

Для перемещения между контекстами проектирования реализован класс ViewManager, который содержит стек узлов дерева, представляющих ранее открытые контексты и индекс текущего контекста в стеке. API класса представлен методом для получения узла дерева, соответствующего текущему контексту и методами переходов между контекстами:

- back() – возвращается назад, декрементируя индекс контекста до ближайшего существующего файла - как правило, это предыдущий открытый контекст, если он не был удален;
- open(<узел, соответствующий открываемому контексту>) – инкрементирует индекс текущего контекста, помещает корневой файл контекста по индексу, удаляет элементы стека, индексами, большими индекса текущего;
- up() – если текущий контекст не является корнем проекта, открывает новый контекст с помощью метода open() на основе родительского узла по отношению к узлу текущего контекста;
- forw() – если узел в стеке, соответствующий инкрементированному индексу текущего контекста, существует, то инкрементирует индекс.

Каждый из перечисленных методов навигации также вызывает обновление диаграммы в соответствии с новым рабочим контекстом. Методы back(), forw(), up() вызываются при нажатии на соответствующую кнопку на панели инструментов. Каждый узел дерева прослушивает события мыши и при нажатии на него левой клавишей вызывает метод open(), передавая себя в качестве аргумента.

Фильтрация элементов

Элементы, обладающие модификаторами доступа, фильтруются. Фильтрация происходит после построения дерева на этапе добавления

соответствующих элементов на диаграмму. Механизм фильтрации реализован в классе Filter, и представлен двумя методами – isClassVisible() и isMemberVisible(), для классов и членов классов соответственно. Методы возвращают булево значение, на основе которого элемент добавляется на родительский контейнер, соответствующий родительскому элементу или нет.

Члены классов

Узлы, являющиеся классами и интерфейсами, представлены различными классами, наследниками Clazz. При перепостроении дерева или при изменении текущего контекста проектирования вызывается метод этого класса, в котором на основе PSI-файла, соответствующего данному классу или интерфейсу, создаются элементы, представляющие члены классов разрабатываемого приложения. Эти элементы представлены классами Field и Method, унаследованных от абстрактного Memeber. Оба класса сами отвечают за свое графическое представление, т.к. для методов и полей оно отличается, а механизмы удаления и редактирования членов реализованы в базовом классе. для редактирования и создания членов класса разработан класс фабрики. Этот класс также имеет графическое представление в виде текстового поля, которое размещается в конце списка членов класса на месте кнопки добавления членов, если создается новый член, или замещает графический элемент, соответствующий редактируемому члену класса. При редактировании элемента изначальным значением текстового поля является текущее описание члена класса на языке программирования, при создании элемента добавляется лишь рекомендованный модификатор доступа. После потери фокуса или при событии ввода, введенный текст проверяется на валидность с точки зрения корректности описания метода или поля, затем определяется тип добавляемого элемента (поле или метод), после чего на основе текста создается новый PSI-элемент, который замещает

редактируемый или добавляется в конец класса в случае добавления члена. После редактирования или добавления элемента дерево перестраивается.

Интеллектуальный выбор модификаторов доступа

Для определения наиболее подходящего модификатора доступа для добавляемого члена или класса реализован класс ModificatorAdvisor, предоставляющий два статических метода для членов и классов, которые на основе расположения добавляемого элемента по отношению к текущему контексту возвращают рекомендуемый модификатор. Отношение непосредственной вложенности класса определяется через определение эквивалентности родительского пакета, полученного с помощью API-дерева, и узла, соответствующего текущему контексту проектирования. В случае добавления класса рекомендуемым модификатором является “public”, кроме случаев, когда класс добавляется в пакет, соответствующий текущему контексту проектирования.

Модули

Сложные системы насчитывают тысячи классов, и декомпозиции системы на классы недостаточно [2][3]. Таким образом, возникает задача декомпозиции системы на более крупные модули, выделения их API из API содержащихся классов. В данном решении каждый объект модуля загружает список экспортируемых пакетов и модулей, от которых он зависит из файла. Эта информация используется для следующих задач: отображение окружения модуля на соответствующей панели; фильтрация отображаемых пакетов, в случае, если текущий рабочий уровень является корнем проекта; добавление иконки экспорта к графическим элементам, соответствующим экспортируемым пакетам на других уровнях. При изменении экспортируемых пакетов или зависимости модулей с помощью

пользовательского интерфейса плагина, эти изменения немедленно вносятся в соответствующий дескриптор модуля с помощью механизмов платформы. Для определения предполагаемых экспортируемых модулем пакетов производится рекурсивный обход всех пакетов модуля и для каждого осуществляется поиск ссылок на него в проекте. Этими ссылками являются, например операторы импортирования класса, находящегося в этом пакете. Все полученные элементы поочередно проверяются на принадлежность к текущему модулю. Если какой-либо элемент ссылающийся, на данный пакет принадлежит другому модулю, пакет добавляется в список экспортируемых и дальнейшая проверка ссылающихся на него элементов прерывается.

В процессе определения зависимостей модуля от других перебираются все классы, которые содержит данный модуль. Для каждого класса, с помощью механизмов платформы определяется список PSI-элементов, представляющих из себя операторы импортирования, на основе этих списков составляется общий набор неповторяющихся операторов для всего модуля, затем для каждого выполняется поиск элемента, на который ссылается соответствующий оператор. Затем с помощью механизмов платформы выполняется поиск модуля проекта, к которому он относится. Если полученный модуль отличается от текущего, он добавляется в список зависимостей текущего модуля.

Заключение

В ходе работы были сформированы требования к инструменту, позволяющему решить некоторые существующие проблемы средств визуального проектирования.

Был разработан инструмент, удовлетворяющий всем выдвинутым требованиям, кроме того, часть реализованного функционала инструмента выходит за рамки поставленных требований. В настоящий момент с помощью плагина уже было спроектировано два тестовых проекта, а в

финальных стадиях разработки инструмент использовался для модификации собственного архитектурного решения. Интеграция инструмента в популярную среду разработки позволила использовать инструмент одновременно с использованием IDE, при этом плагин не расходует большого количества дополнительных ресурсов, т.к использует данные сформированные платформой. Использование оптимизированных механизмов платформы позволило сделать быстродействующий инструмент. В ходе изучения средств визуального проектирования не было найдено инструментов с аналогичным функционалом. Есть все основания полагать, что разработанный инструмент будет востребован и конкурентоспособен.

В настоящий момент диаграммы, разрабатываемые с помощью плагина, не соответствуют стандартам UML, однако при создании документации к приложению может понадобиться экспорт текущей диаграммы в UML, что планируется реализовать в ходе последующей работы над проектом. В настоящее время в разработке активно применяются шаблоны проектирования [15], поэтому планируется реализовать в инструменте возможность добавления готовых конструкций, что поспособствует уменьшению скорости разработки и повысит качество кода.

Использованные источники:

1. Project Jigsaw. [Электронный ресурс]- Режим доступа: http://openjdk.java.net/projects/jigsaw/
2. A. Jecan. Java 9 Modularity. Project Jigsaw and Scalable Java Applications. 2017.
3. J. Zukowski. Java AWT Reference, March, 1997.
4. L. Hatton. Re-examining the Defect-Density versus Component Size Distribution IEEE Software, 1997.
5. E.S. Raymond. The Art of UNIX Programming. Addison Wesley, 2003.
6. Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес. Приемы объектно-ориентированного проектирования. Питер, 2015.