Санкт-Петербургский государственный университет

Математико-механический факультет

Кафедра системного программирования

Реализация автоматизированного преобразования кода в расширении Eclipse для языка haXe

Дипломная работа студентки 461 группы

Савенко Марии Олеговны

Научный руководитель
Старший преподаватель ..... /В.С.Полозов/
/подпись /

Рецензент
Старший преподаватель ..... /Ю.В.Литвинов/
/подпись /

«Допустить к защите»
Заведующий кафедрой ..... /А.Н.Терехов/
/подпись /

Санкт-Петербург

2012

SAINT PETERSBURG STATE UNIVERSITY

Mathematics & Mechanics Faculty

Software Engineering Chair

Automation code transformations implementation in haXe language plug-in for Eclipse IDE

by

Maria Savenko

Bachelor's graduation paper

Supervisor
Senior Lecturer ..... /V.S.Polozov/
/ signature /

Reviewer
Senior Lecturer ..... /Y.V.Litvinov/
/ signature /

«Approved by»
Head of Chair ..... /A.N.Terekhov/
/signature /

Saint Petersburg

2012

Оглавление

Введение

Неотъемлемой частью процесса разработки программного обеспечения является использование интегрированных средств разработки (IDE, Integrated Development Environment). Иногда подобную среду для некоторого языка предоставляет та же компания или сообщество, которые разработали этот язык. Когда подобного готового решения не предоставляют, то появление среды разработки полностью зависит от популярности языка. И, бывает, чем популярнее язык, тем больше разнообразных инструментов для работы с ним.

HaXe [4] достаточно молодой, строго типизированный, мультиплатформенный язык программирования с открытым исходным кодом. На данный момент он продолжает развиваться, и последний релиз состоялся в апреле 2012 – версия 2.09. Компилятор haXe поддерживает многие популярные на сегодняшний день платформы: JavaScript, Flash, NekoVM, PHP, C++, C#. Синтаксис языка очень похож на синтаксис языков JavaScript или ActionScript.

Сейчас уже существует некоторое количество популярных сред разработки, поддерживающих haXe. Это Flash Develop [5], «EclihX» [6] - расширение для среды Eclipse [7], расширение [8] для среды FDT [9] и расширение [10] для среды IntelliJ IDEA [11]. Однако расширение для IDEA доступно только для платной версии этой среды, и почти все упомянутые среды поддерживают компиляцию только под 2-3 платформы. Если смотреть с точки зрения функциональности, то все названные среды предлагают пользователю такие удобства как: создание и настройка проекта, подсветка синтаксиса, автодополнение имен и подсветка ошибок, как правило, найденных компилятором haXe. Но все из перечисленного, кроме подсветки ошибок на основе сообщений от компилятора, может предложить и современный текстовый редактор, ориентированный на программирование, тогда как в большинстве современных IDE для других языков доступны статический анализ кода и связанная с ним функциональность.

Все это повлекло создание нового расширения для среды Eclipse, добавляющего поддержку haXe и лежащего в основе данной работы. Прототип расширения был спроектирован в рамках дипломной работы Кондратьева А.Е. в 2010 году. В нем были почти или полностью реализованы все возможности, доступные в других средах разработки для haXe.

Основной целью данной дипломной работы является исследование и разработка алгоритмов статического анализа и автоматизированных преобразований кода. Однако существующая версия данного расширения не стабильна, а доступ к уже существующей функциональности непривычен для рядового разработчика. Поэтому кратко поставленные задачи можно охарактеризовать так:

- добавить поддержку библиотек haXe;
- добавить элементы автоматизированного преобразования кода;
- улучшить пользовательский интерфейс.

1. Постановка задачи

Задачей данной работы является поддержка автоматизированных преобразований кода в IDE для haXe. В рамках работы были выделены две подзадачи.

- Поиск использований какой-либо переменной или функции в проекте или группе проектов.
- Добаление возможность переименования пользовательских типов, функций и переменных.

Для реализации указанных задач требуется выполнение ряда модификаций существующего проекта, которые были выделены в качестве подзадач:

- добавить поддержку работы с проектом и группой проектов;
- добавить поддержку библиотек haXe;
- доработать внутреннее представление типов;
- упростить доступ пользователя к различной функциональности плагина.

Также в проекте практически отсутствует тестирование, поэтому добавление тестов тоже выделено в отдельную подзадачу.

2. Обзор предметной области

Особенности языка haXe

Прежде всего, haXe является проектом с открытым исходным кодом. Согласно выступлению Николя Канасса на 24C3, язык специально разрабатывался таким образом, чтобы иметь возможность абсорбировать технологии от различных производителей. Автор намеренно отказался от всех существующих языков и создал свой, где проводится четкая грань между общим кодом и кодом, специфичным для конкретной технологии. Такой подход позволяет максимально полно поддерживать функциональность целевых платформ.

Open Source стратегия в таком контексте является гарантией того, что язык будет развиваться и обеспечивает максимально плодотворную почву для разработки компиляторов или интерпретаторов на новые целевые платформы. Открытость также снимает запрет на использование сторонних технологий в коммерческих целях: haXe не стремится конкурировать с Flash, Javascript или Silverlight – это, прежде всего, инструмент для разработчика, помогающий комфортно использовать распространенные web-платформы и не тратить дополнительных усилий для переноса независимого кода между ними.

Обычно, каждая новая платформа идет со своим собственным языком, поэтому при выборе наилучшей платформы для какой-либо задачи разработчик часто отдает предпочтение тому языку, который лучше всего знает. В этом случае haXe является универсальным языком – код на haXe можно компилировать для разных платформ, независимо от языка этих платформ. С помощью haXe и связанных с ним технологий (Neko, SWHX, SysTools, SPOD и т.д.) можно создавать приложения, способные работать под Windows, Mac OS или Linux. Также с помощью haXe можно писать как клиентские, так и серверные программы для web, и настольные приложения для любых платформ. Поддерживаемые на сегодняшний момент платформы для компиляции: JavaScript, Flash, NekoVM, PHP, C++, C#. В скором времени разработчики обещают добавить поддержку Java.

Язык является строго типизированным, а это значит, что большинство ошибок, связанных с типами, будут выявлены еще во время компиляции. Для констант тип выводится автоматически. Константами в haXe являются значения таких типов: Int, Float, String, Bool, EReg и специальное Unknown<0>. Можно и не указывать тип при объявлении функций и локальных переменных – при первом обращении к ним их тип будет уточнен. При создании же все объекты с необъявленным типом будут иметь тип Unknown<0>, и, таким образом, оставаться строго типизированными. Неясным поведением может обладать, разве что, тип Dynamic.

Что касается синтаксиса, то он очень похож на Java, PHP, JavaScript или ActionScript. Все выражения в haXe равноправны – их можно без проблем вкладывать одно в другое рекурсивно. Есть поддержка объявления анонимных объектов и функций.

На haXe уже написано несколько успешных, в том числе и коммерческих проектов, среди которых, например, on-line игры MyMiniCity [12] и Alpha Bounce [13], а так же проект Comapping [14]. Одним из основных препятствий для широкого распространения языка в коммерческих проектах является отсутствие удобной многофункциональной интегрированной среды разработки для него.

Сравнительный анализ IDE для haXe

На данный момент наиболее распространённым продуктом для программирования на haXe является IDE FlashDevelop и дополнение к IDE FDT. Многие используют и просто различные виды текстовых редакторов, такие как TextMate [15], JEdit [16], VIM [17] и другие, с установленными дополнениями, отвечающими за подсветку кода и его автодополнение.

Мной были протестированы наиболее популярные в haXe-сообществе IDE и расширения для IDE, поддерживающие данный язык.

FlashDevelop

Является бесплатной средой разработки, лицензированной по MIT. Работает на платформе Windows. Основным языком, для которого ведется поддержка, является ActionScript (AS) 2 и 3, однако так же поддерживает MXML, haXe, и подсвечивает синтаксис для Python, HTML, XML, PHP, CSS. Является расширяемой с помощью отдельно устанавливаемых дополнений (plug-in) средой.

Для удобства работы с haXe доступны такие функции:

- широкий выбор платформ для компиляции (AS2, AS3, Neko, C++, Air AS3 projector, js, php, nme) в настройках проекта или возможность не выбирать платформу;
- создание проекта;
- распознавание и использование конфигурационных (build) файлов в дереве проекта;
- подсветка кода;
- использование шаблонов кода как вручную так и автоматически;
- автодополнение кода;
- переход от использования к определению;
- визуализация структуры кода;
- отображение ошибок, полученных от компилятора;
- навигация по списку ошибок;
- переименование переменных.

Здесь стоит особенно выделить возможность переименовывать переменные. Данная функция – первый шаг к добавлению инструментов рефакторинга к средам разработки для haXe. Пока функция доступна только для локальных в рамках класса или функции переменных.

Из других явных преимуществ надо отметить удобное автодополнение кода, которое не только отображает краткое и полное название типа или объекта и пакет, в котором он находится, но и сортирует список в зависимости от контекста. Например, после ключевого слова ‘new' первыми будут представлены типы с подходящими названиями и только потом имена экземпляров объектов.

Из недостатков стоит отметить, что настройки создаются на весь проект целиком, тогда как сам проект может являться составлением многих отдельно компилируемых частей. В этом случае можно скомпилировать вручную при помощи build файлов, но удобства автоматической перекомпиляции и отображения ошибок будут недоступны. Так же были замечены проблемы при прорисовке отметок ошибок в коде – при исправлении ошибки отметка исчезала, но подчеркивание оставалось в тексте до следующей сборки.

Проект EclihX

Этот проект был начат студентом кафедры системного программирования математико-механического факультета СПбГУ Николаем Красько в рамках его дипломной работы.

На данный момент доступна стандартная функциональность: подсветка синтаксиса, автодополнение кода и компиляция проекта. Подсветка ошибок ведется на основе информации от компилятора. Еще в проект встроена возможность отладки кода для платформы Flash. Проект реализован в виде подключаемого модуля для Eclipse и продолжает развиваться.

Расширение для среды IntelliJ IDEA

Начат как индивидуальная разработка, вскоре получив статус «закрытой». Совсем недавно свежие версии данного подключаемого модуля стали доступны для скачивания. Однако модуль доступен только для платной IDEA Ultimate (испытания проводились на 30ти дневной испытательной версии).

Список возможностей данного расширения впечатляет:

- подсветка синтаксиса;
- переход от использования к определению;
- автодополнение кода;
- компиляция на Neko VM, Flash;
- отладчик для Flash;
- форматирование кода;
- документирование кода;
- поддержка NMML и HXML;
- функции рефакторинга:
    - переименование;
    - перемещение ресурсов.

В список заявленных возможностей также входил вывод типов, но при проверке явные ошибки в использовании типов не отмечались редактором.

Из достоинств надо отметить легкость установки самого модуля и подключение библиотек, возможность посмотреть дерево использований и, несомненно, функции рефакторинга.

Из недостатков:

- Отсутствие сворачивания частей кода
- Недостаточно хорошее автодополнение кода
- Отсутствует проверка на наличие ошибок в коде до компиляции

Однако проект активно развивается и возможно, что в скором будущем будут доступны новые возможности.

Расширение для FDT

Среда FDT построена на базе Eclipse и, прежде всего, нацелена на поддержку платформ Flash, Flex и Air. Сама среда доступна для скачивания в двух вариантах: бесплатная версия с ограниченной функциональностью и версия, для которой необходимо приобрести лицензию. Расширение,

которое добавляет поддержку haXe в среду, может быть скачано и установлено на любую из этих версий.

Расширение поддерживает компиляцию только для следующих целевых платформ: Flash, JS и PHP. Также среда предлагает уже стандартную подсветку синтаксиса и автодополнение функций и переменных. Еще модуль добавляет много шаблонов для создания проектов и файлов.

Отдельно стоит отметить подсветку ошибок в режиме реального времени. В среде включена автоматическая перекомпиляция проекта при изменении кода, поэтому расширение, получая предупреждение или сообщение об ошибке от компилятора, заносит его в список и помечает соответствующее место в коде. Разработчики отметили, что не могут предложить пользователям ничего сверх того, что предлагает компилятор haXe, поэтому ничего, связанного с анализом кода, в расширении для FDT не доступно.

Процесс разработки расширений для IDE Eclipse

На сегодняшний момент все более популярными становятся платформы, изначально рассчитанные на поддержку нескольких языков и возможную поддержку еще большего количества в будущем. Обычно подобные платформы предоставляют пользователям возможности для создания собственных расширений среды. Одной из самых известных бесплатных платформ такого типа является Eclipse.

Архитектура Eclipse

В основе архитектуры Eclipse Platform лежит принцип использования подключаемых модулей (plug-in). Платформа позволяет подключать их вручную или автоматически из указанной заранее папки, и реализует механизмы их выполнения в среде.

За соединения с платформой отвечает файл манифеста (manifest), генерирующийся для каждого подключаемого модуля. Фактически это XML-документ, содержащий информацию о сборке и всех компонентах, используемых данной сборкой и, соответственно, обязанных присутствовать при исполнении данного модуля. При работе с дескриптором модуля среда Eclipse (точнее, загрузчик компонент) определяет наличие всех необходимых сборок. Но среда Eclipse, в случае необходимости, позволяет разработчику создать и использовать свой собственный загрузчик компонентов.

Однако файл манифеста не содержит ничего, имеющего отношения к функциональности, настройкам модуля, его взаимодействию с другими компонентами и многому другому. Эта дополнительная информация находится в другом файле, названном plugin.xml. Его основное назначение – описать использование «точек расширения» (extension points) Eclipse. Модуль может, как объявлять точки расширения для последующего использования, так и расширять любое количество из ранее объявленных. Например, подключаемый модуль, реализующий функциональность стандартного текстового редактора, предоставляет точку расширения и связанный с ней API для настройки и доработки существующей базовой функциональности разработчиком.

Запущенный экземпляр платформы Eclipse выполняется под управлением одной виртуальной машины Java (JVM). Что касается компонентов, то для загрузки каждого модуля используется свой загрузчик классов - classLoader. Такое архитектурное решение приводит к использованию

некоторых «внешних» библиотек, которые должны иметь доступ к ресурсам модуля (например, библиотека для ведения отчетов), поэтому необходимо обеспечить специальный механизм загрузки классов. Этот механизм называется «buddy loading», и управление им осуществляется с помощью задания специальных тегов манифеста модуля – Eclipse-BuddyPolicy и Eclipse-RegisterBuddy (реализовано начиная с Eclipse 3.3).

При запуске среды разработки на базе Eclipse Platform компонент Platform Runtime определяет набор доступных подключаемых модулей, читает их файлы манифестов и строит реестр модулей. При этом полная загрузка plug-in'а происходит только тогда, когда понадобится непосредственно исполнить принадлежащий ему код. Данный механизм обеспечивает возможность поддерживать большую базу установленных модулей.

Платформа предоставляет точки расширения с интерфейсами и базовой реализацией практически для всех абстракций и функциональности, которая может понадобиться при разработке современной IDE. Рассмотрим наиболее значимые.

- **Рабочее пространство** (workspace) – это компонент, определяющий основные объекты, с которыми могут работать пользователи и приложения Eclipse и предоставляющий средства по структуризации и управлению ими. В техническом плане, основная задача, которую решает эта компонента – унифицированный доступ к локальным и удаленным объектам. Базовые понятия, определяемые Workspace –это проект (project), папка (folder) и файл (file). В терминологии Eclipse все эти объекты называются ресурсами (resources).
- **Рабочая область** (workbench) определяет базовый пользовательский интерфейс (UI) Eclipse и предоставляет другим компонентам средства для создания своих собственных интерфейсов. Основными объектами интерфейса, с которыми работает Workbench, являются редакторы (editors), виды (views) и перспективы (perspectives).

Для создания пользовательского интерфейса в Eclipse используется компонента SWT (Standard Widget Toolkit). На ней построены все графические средства Eclipse. Поскольку SWT использует средства конкретной операционной системы, с помощью нее легко создавать мультиплатформенные приложения, выглядящие одинаково на всех системах.

Архитектура подключаемого модуля

Класс модуля (или класс-активатор) – это класс, экземпляр которого создается средой при загрузке этого расширения. Реализуя интерфейс org.osgi.framework.BundleActivator, он должен иметь методы start и stop, вызывающиеся соответственно сразу после загрузки и перед выгрузкой расширения. Платформа уже содержит несколько шаблонов данного класса, в зависимости от того, будет ли иметь будущее расширение графический интерфейс или нет. Разработчик может дописывать свой собственный класс-активатор на основе этих шаблонов.

Подходы к реализации сред разработки

Рассмотрим, что сейчас является «стандартным набором» компонент в полноценной IDE:

- текстовый редактор с подсветкой синтаксиса;
- встроенный или внешний компилятор и/или интерпретатор;
- отладчик;
- средства проведения рефакторинга:
    - переименование;

    - извлечение методов и классов;
    - перемещение и др.
- средства автоматизации сборки;
- средство связывания с системами управления версиями;
- графический менеджер для языков, поддерживающих графические интерфейсы;

Подсветка синтаксиса – это самая простая и наиболее распространенная компонента. Специализированные текстовые редакторы, которые не являются средой разработки как таковой, но предоставляют пользователям все различные инструменты для работы с кодом как с текстом, обычно имеют у себя набор подсветок синтаксиса для различных языков.

Компилятор и интерпретатор могут быть и встроенными, но, как правило, даже для языков с закрытым исходным кодом компилятор является отдельным приложением и просто вызывается из среды разработки.

Проект IMP (The IDE Meta-Tooling Platform)

IMP – проект, целью которого было получить инструмент для создания IDE под Eclipse. Основной целью проекта является упрощение разработки IDE для новых языков, а именно:

- генерация и управление синтаксическим анализатором, AST (Abstract Syntax Tree), семантическим анализом;
- подсветка синтаксиса;
- визуальное представление кода (outline view);
- навигация между файлами по гипер-ссылкам;
- сборщики проектов;
- пометка ошибок в редакторе;
- всплывающие подсказки;
- исполнение и отладка программы.

С технической точки зрения, IMP представляет из себя расширение для Eclipse, на основе которого разработчик пишет собственное расширение. В IMP входят: собственный текстовый редактор, инструмент для визуального представления кода, контекстные меню и страницы настроек параметров и др. Включить поддержку вышеописанных возможностей в собственном плагине можно с помощью точек расширения в IMP.

В основе работы почти всей функциональности IMP лежит AST-дерево, полученное с помощью анализа кода сгенерированным синтаксическим анализатором LPG [19] (сокращение от LALR Parser Generator). Однако пользователь может присоединить и собственный синтаксический анализатор.

Можно поделить проект IMP логически на три части:

- часть, недоступная для изменения;
- часть, обязательная для реализации пользователем – обычно это классы, определяющие поведение IDE на основе информации, полученной при анализе дерева;
- точки расширения, созданные разработчиками IMP, требующие реализации только в том случае, если нужно изменить поведение, связанное с точкой расширения объекта.

К сожалению, разработка проекта остановлена – последнее обновление было в декабре 2009 года. Поддержка рефакторинга не была закончена, хотя соответствующие интерфейсы были добавлены в проект. Помимо этого, во встроенном текстовом редакторе можно изменять только контекстное меню. Все это делает проект IMP хорошим стартом для введения поддержки собственного языка в среде Eclipse, но нежелательным при длительной, серьезной разработке.

Описание грамматики языка haXe на языке ANTLR

Проект IMP изначально был настроен для работы с LPG грамматикой, однако разработчиками проекта была добавлена возможность подключить собственный синтаксический анализатор (parser) для языка.

ANTLR – это набор инструментов по созданию на основе существующей грамматики лексических (lexer) и синтаксических анализаторов для нее на разные языки, в том числе и на язык Java. Поэтому инициатором данного проекта было принято решение использовать ANTLR в качестве языка для воспроизведения грамматики haXe, а затем сгенерировать анализаторы для нее с помощью инструментов ANTLR. При желании добавить поддержку какой-либо новой синтаксической конструкции в IDE, сначала ее описание вносится в грамматику, а затем лексический и синтаксический анализаторы перегенерируются.

До начала данного проекта еще не было попыток воспроизвести грамматику haXe именно на языке ANTLR, поэтому она была разработана с "нуля". Источниками информации о поведении компилятора haXe служила документация с официального сайта, существующая грамматика haXe на языке Gold [18] (являющаяся LR грамматикой) и анализ результатов компиляции собственных примеров на haXe. При описании грамматики можно явно указывать, какого типа класс должен использоваться в конкретном правиле – главное, чтобы все классы были наследниками класса CommonTree.

3. Архитектура проекта

Весь проект компилируется в единый plug-in для Eclipse, но логически его можно разделить на несколько частей:

- лексический и синтаксический анализаторы грамматики;
- классы, представляющие различные типы объектов в синтаксическом дереве;
- реализации частей проекта IMP;
- части, отвечающие за обработку синтаксического дерева – связывание использований с определениями, проверка ошибок типов;
- классы, отвечающие за элементы интерфейса.

В описании каждого из разделов будут приведены названия пакетов, в которых находятся классы, отвечающие за функциональность текущего раздела.

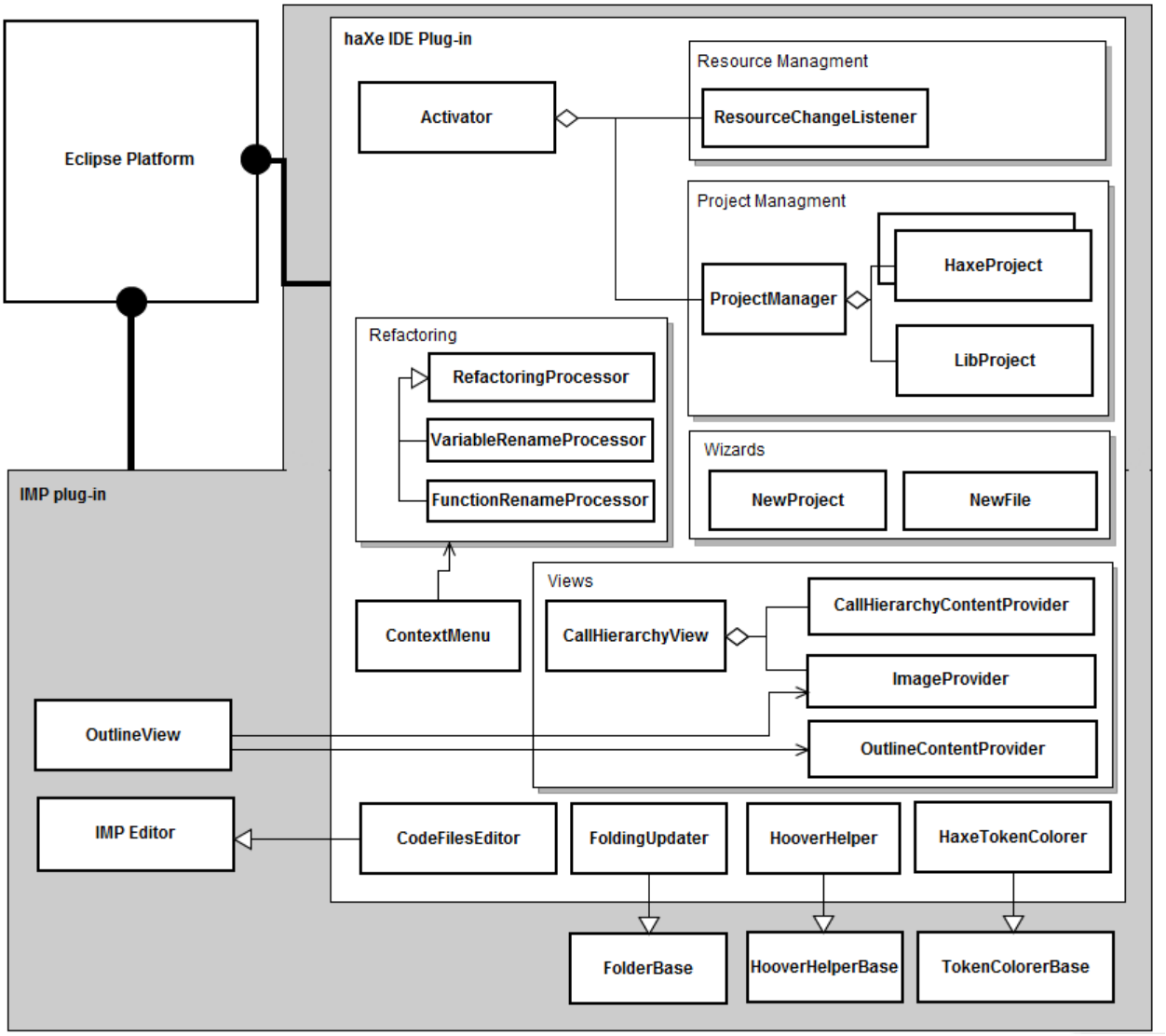

Рис 1. Архитектура проекта и связь с проектом IMP

Лексический и синтаксический анализаторы грамматики

Как уже было сказано в обзоре, проект IMP использует синтаксические деревья, полученные при разборе кода программы на целевом языке, в качестве главного источника информации о

программе. Для создания дерева используется синтаксический анализатор, сгенерированный по грамматике на языке ANTLR.

imp.parser.antlr

Содержит файл грамматики - Haxe.g, и полученные генерацией файлы лексического и синтаксического анализаторов – HaxeLexer и HaxeParser соответственно.

Классы-узлы синтаксического дерева

Для удобства последующего изучения синтаксического дерева, узлы дерева должны быть различных типов. Это позволит делать некоторые заключения о поведении программы только по типу узла, не изучая информацию, хранящуюся в нем. ANTLR позволяет строить AST из узлов различных типов – достаточно лишь указать в нужном правиле грамматике желаемый тип узла. Однако любой используемый в грамматике класс узла должен быть наследником класса CommonTree.

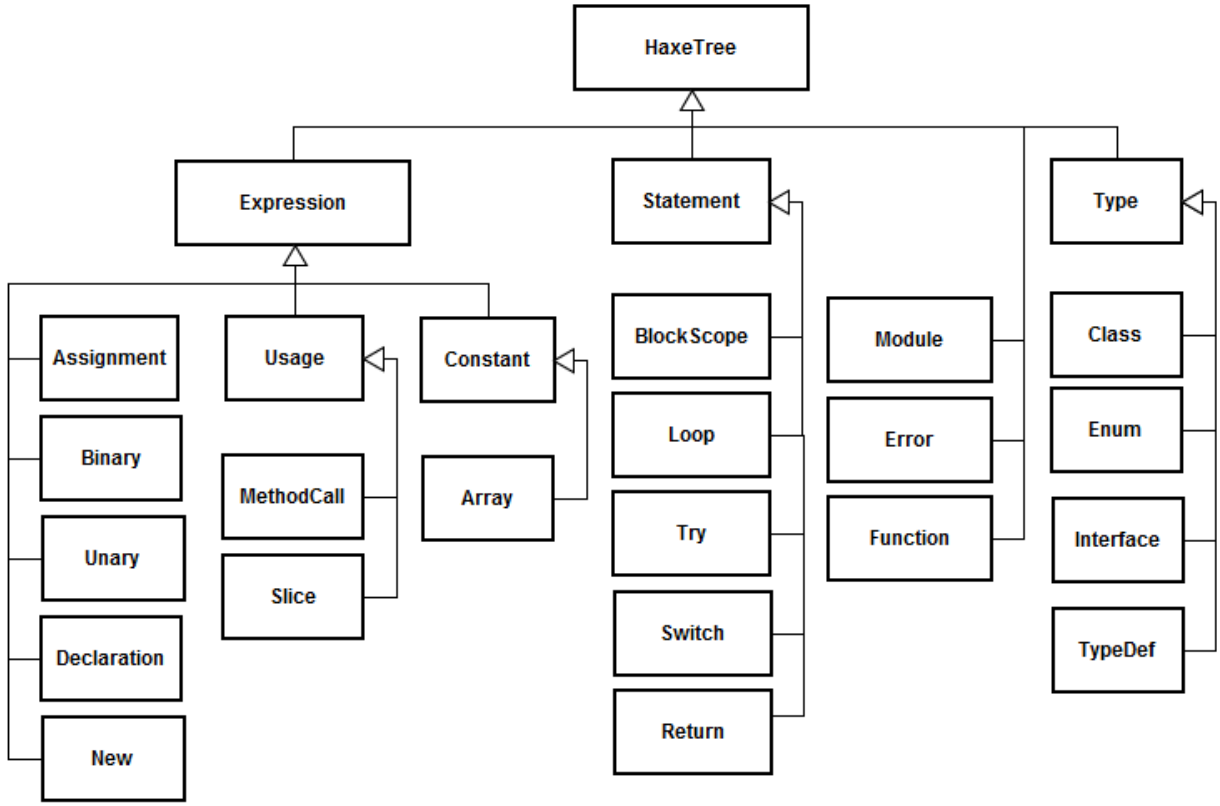

Рис 2. Схема наследования основных узлов AST

tree

В этом пакете находится класс HaxeTree, являющийся потомком CommonTree и служащий классом-родителем для всех узлов дерева. HaxeTree содержит адаптации методов CommonTree, такие как: вычисление начала и конца записи, относящейся к узлу в тексте программы, текстовое представление узла и т.д., для объектов, применяемых в проекте.

Здесь же находится HaxeTreeAdaptor, использующийся при создании узла нужного типа во время анализа программы, а также интерфейсы, используемые для придания некоторым узлам дерева общих свойств.

tree; tree.expression; tree.statement

Здесь находятся классы узлов дерева, характеризующие определенные синтаксические конструкции языка, например: Assignment для присваиваний, Declaration для определений и т.д.

tree.type

Содержит классы, которые могут использоваться в качестве типов при создании объектов в целевом языке. В соответствии с системой типов в haXe это: классы (Class), перечисления (Enum), интерфейсы (Interface) и определения типов (TypeDef).

Реализации точек расширения проекта IMP

Хотя в расширение на основе проекта IMP уже заложено много функциональности, поведение среды в таких областях как подсветка синтаксиса, визуализация структуры кода и т.д. будет зависеть от целевого языка. Такие части были оставлены для реализации разработчикам будущего расширения. Также в проекте IMP были созданы точки расширения для некоторых внутренних компонент, таких как текстовый редактор, меню настроек и панель инструментов, позволяющие вносить незначительные изменения в поведение или внешний вид этих компонент.

imp.builder

Содержит класс HaxeBuilder, ответственный за подключение компилятора haXe и за автоматическую сборку проекта. Является расширением класса BuilderBase из проекта IMP. Здесь же находится класс HaxeNature, являющийся одной из настроек проекта, по которой определяется, что это проект именно на языке haXe.

imp.parser

В пакете находятся классы для запуска анализатора на каком-либо тексте (HaxeParseController) и обработки сообщений, полученных во время анализа (HaxeMessageHandler). В IMP автоматически настроено запускать анализ для содержимого открытого в текстовом редакторе файла, и при любом редактировании этого файла в редакторе проводить анализ заново.

imp.preferences

Содержит класс, описывающий содержимое страницы настроек проекта, а также класс, где указаны все используемые в программе пути (например, пути к изображениям).

imp.utils

Содержит классы, необходимые для:

- задания правил подсветки синтаксиса;
- определения позиций для свертывания кода;
- генерации текста всплывающего сообщения для текущего объекта;
- определения положения в тексте для переходов по ссылке;
- генерации списка для функции автодополнения;
- генерации изображений и обозначений для окна с визуальным представлением кода.

Обработка синтаксического дерева

Следующим шагом для анализа кода программы на целевом языке является выявление связей между различными частями этой программы. Например, нужно установить связь между описанием типа и его использованиями, а также между описаниями членов класса или локальными переменными и их использованиями.

tree.exceptions

Содержит классы для специфических ошибок, возникающих при работе с деревом.

tree.utils

Содержит классы для работы с синтаксическим деревом.

AbstractTreeVisitor – реализует паттерн «посетитель», является базовым классом почти для всех классов, работающих с деревом.

Linker – отвечает за связывание использований с определениями. Данный класс был реализован в ходе курсовой работы и доработан в рамках данной работы.

ErrorProvider – отвечает за выявление ошибок в коде, через изучение дерева. В основном это ошибки, связанные с использованием типов.

ReferenceListBuilder – собирает информацию об использованиях какого-либо объекта в программе на целевом языке.

И другие классы, отвечающие за печать дерева, хранение локальных переменных, отображение ошибок в тексте программы и т.д.

Организация рабочего пространства

Если рассматривать Eclipse с точки зрения Java-программиста, то уже стал привычным определенный набор возможностей и способов представления информации. При добавлении поддержки нового языка поддержка точно таких же элементов создает некоторую целостность среды. Если рассматривать только интерфейсную часть, то, прежде всего, это поддержка различных «видов».

В прототипе текущего расширения рабочее пространство поддерживалось только внутренними механизмами проекта IMP. Из частей рабочего пространства, которые можно было изменять с помощью точек расширения IMP, были только специализированный текстовый редактор, контекстные меню, окно с визуальным представлением кода и характеристика проекта (nature). Сами же проекты, их создание и управление ими не было реализовано в явном виде и расширение использовало общие механизмы работы с проектами Eclipse по умолчанию. Из-за этого предоставляемые возможности некоторых готовых инструментов были ограничены, поэтому требовалась их адаптация к текущему расширению.

workspace

Содержит классы, упрощающие работу с рабочим пространством: работу с проектами (ProjectManager, HaxeProjectCreator, ProjectVisitor), слежение за изменениями ресурсов (ResourceChangeListener), работу с изображениями (SharedImages) и т.д.

workspace.commands

Здесь находятся классы действий для элементов контекстного меню редактора и видов.

workspace.contextMenu

Содержит дополнительные инструменты для управления меню.

workspace.editor

Помимо видов при работе над проектом, потребовалось расширить функциональность текстового редактора. Так как изменять сам редактор IMP нельзя, а только его контекстное меню, то на его основе был написан новый редактор.

workspace.elements

Пакет для классов, представляющих различные элементы рабочего пространства, связанные с реальными объектами файловой системы. Здесь находятся классы, представляющие: файлы с кодом на haXe (CodeFile), конфигурационные файлы (BuildFile), пользовательские проекты (HaxeProject), проект библиотек (HaxeLibProject) и связанные с ними интерфейсы.

workspace.refactoring

Содержит реализации обработчиков рефакторинга для различных групп или типов узлов синтаксического дерева.

workspace.views

Содержит все элементы, связанные с видами данного проекта. В данном проекте понадобилось отображать информацию обо всех использованиях какого-либо объекта. Аналогом для языка Java в Eclipse является вид с иерархией вызовов, в котором при поиске функции/переменной отображается список функций в которых они вызывались/использовались, а для поля класса еще указывается место его определения. У нас за само окно вида с деревом вызовов отвечает класс CallHierarchyView, за отображение текста элемента вида отвечает CallHierarchyLabelProvider, а за поведение элементов в дереве отвечает CallHierarchyContentProvider.

workspace.wizards

Содержит классы, отвечающие за визуализацию и поведение помощников для создания проектов и файлов.

4. Особенности реализации

Грамматика haXe

В грамматику были внесены некоторые изменения – добавлены правила для:

- вызова метода;
- вырезки из массива;
- обращения к членам классов.

В самом проекте были добавлены классы, соответствующие названным конструкциям, а так же ряд классов для уже существующих конструкций, не требовавшихся ранее в прототипе ввиду его узкой направленности и сейчас облегчающих навигацию по дереву и хранение информации. Список добавленных классов-узлов:

- Module – корень любого синтаксического дерева, соответствующего файлу;
- Классы, не исполняющие роль классификаторов и не используемые напрямую синтаксическим анализатором:
    - TypeTag – обозначает любое объявление типа в объявлении переменных, функций, параметрических классов и т.д.;
    - NodeWithModifier – обозначает узел, который может содержать информацию об области своей видимости или типе поведения (возможные модификаторы – static, dynamic, override, public, private);
    - NodeWithScopeAndModifier – узел, который может содержать информацию об области своей видимости и при этом должен содержать блок;
    - Statement – представляет ту часть синтаксических конструкций, которая может использоваться самостоятельно в блоке;
    - Expression – представляет ту часть синтаксических конструкций, которые могут использоваться только в составе других конструкций.
- BlockScope – представляет множество выражений, разделенных точкой с запятой, заключенное в фигурные скобки;
- Array – узел, представляющий массив-константу;
- MethodCall – узел, представляющий вызов функции;
- NewNode – обозначает узел, содержащий вызов конструктора;
- Slice – представляет взятие выборки из массива-переменной;
- Unary – представляет все унарные выражения haXe;
- Return – узел, представляющий оператор возврата из функции.

Поддержка проектов

В прототипе расширения была доступна обработка только текущего просматриваемого файла проекта. В рамках курсовой работы был реализован вывод примитивных типов, но из-за отсутствия возможности организации связи между файлами система примитивных типов была реализована прямо внутри расширения, а вызов функций пользовательских типов был невозможен.

Для полноценной работы такой подход неприемлем. Поэтому были поставлены две связанные между собой задачи: введение поддержки проектов и реорганизация системы типов с информацией из реальных файлов как для пользовательских, так и для библиотечных типов.

Структура проекта haXe

Перед тем как перейти к следующим пунктам, стоит определить некоторые понятия, относящиеся к проектам на haXe.

В проекте должен присутствовать как минимум один файл с кодом проекта, называемый главным файлом, и конфигурационный файл. Файлы с кодом имеют расширение «hx», а конфигурационные файлы имеют расширение «hxml». Главный файл проекта является тем файлом, с которого начнется компиляция, имя и путь до этого файла указываются в соответствующем ему конфигурационном файле. Главный файл может ссылаться на другие файлы с кодом. В проекте может присутствовать много главных файлов, делящих проект на части, а одному главному файлу могут соответствовать много файлов конфигурационных.

Несмотря на то, что проект не должен иметь какой-либо четкой структуры, есть несколько правил, которых нужно придерживаться. В файлах с кодом может быть указан пакет, к которому этот файл относится. Если главный файл не ссылается на любые другие файлы, это не обязательно, но в любом другом случае это необходимо для поиска подключаемых файлов. Указанный в главном файле пакет является по существу путем до корня проекта и в пакетах, указанных в подключаемых файлах, должен отражаться путь от этого корня до файла. При этом вовсе не обязательно, чтобы пакеты всех файлов одной папки совпадали, если часть этих файлов не используется в компилируемой части проекта, связанной с другой частью файлов.

Создание проекта haXe

Для создания проектов в Eclipse традиционно используются помощники (wizard). Для проектов данного расширения был сделан подобный, ссылка на него доступна в меню помощников по созданию проектов и файлов. В качестве задаваемых параметров на странице создания проекта доступны: имя конфигурационного файла, платформа для компиляции, пути к папкам для исходников и результатов компиляции, а также имя и путь до главного файла проекта. Полученная информация обрабатывается, и для файла конфигурации генерируется содержимое в виде, естественном для такого типа файлов для haXe.

Общий механизм работы с проектами

При запуске данный модуль обрабатывает список всех проектов рабочего пространства, и составляет список проектов, являющихся haXe-проектами (для которых включена haXe nature), для быстрого доступа к ним из любой части плагина и удобной работы с их содержимым. Каждый проект в списке является экземпляром класса HaxeProject, хранящим ссылку на оригинальный ресурс проекта, а также обладающий дополнительными возможностями по поиску, обработке, хранению и работе с файлами этого проекта.

После создания каждый из проектов составляет список собственных файлов кода и список конфигурационных файлов. Виртуальным представителем файлов кода являются CodeFile, а конфигурационных файлов – BuildFile. Содержимое файлов кода обрабатывает синтаксический анализатор, и получившееся дерево сохраняется в соответствующий экземпляр CodeFile.

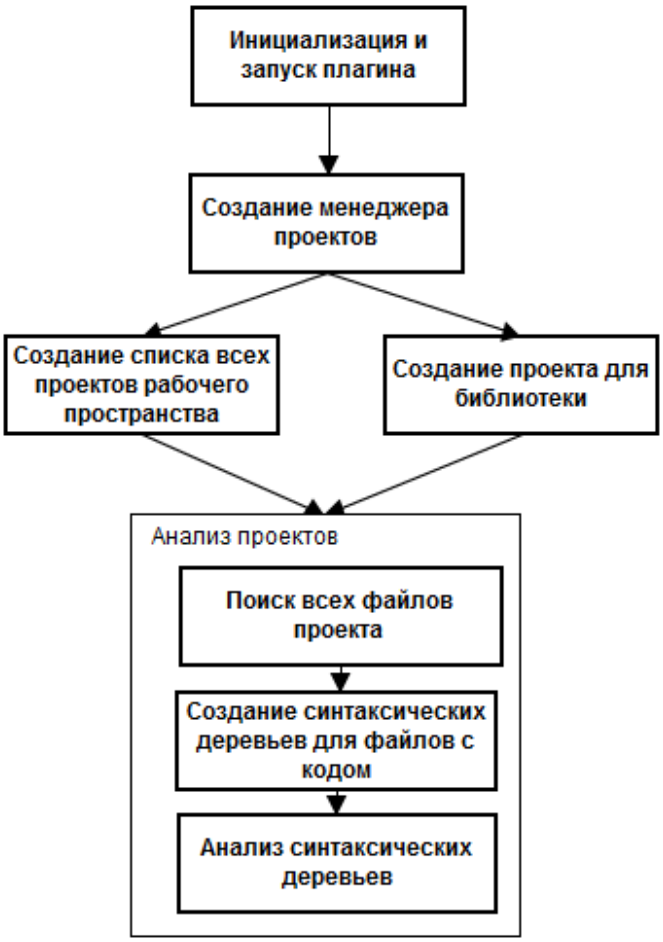

Рис 3. Обработка проектов после запуска расширения

После завершения обработки всех файлов конкретного проекта для него можно провести анализ связей. На данном этапе все использования типов, функций и переменных связываются с их определениями с помощью ссылок на соответствующие узлы синтаксического дерева.

Поиск определений ведется в порядке, установленном разработчиками haXe в документации к языку:

- параметризующие типы текущего класса;
- примитивные типы (находятся в корневой директории библиотеки);
- типы, объявленные в текущем файле;
- типы, объявленные в импортируемых файлах (если в данном пакете объявление не найдено);
- если определение все еще не обнаружено, загружается соответствующий файл и тип ищется внутри него;

На этом этапе проводится и вывод типов. После завершения анализа все корректные выражения будут иметь определенный тип.

Использование библиотек haXe

Библиотеки haXe представляют собой набор файлов с кодом на haXe, в каждом из которых объявлен один или несколько типов. В корне библиотеки лежат стандартные типы, которые

можно использовать в программе на haXe под любую платформу или послужившие основой для аналогичного типа под эту платформу, остальные типы рассортированы по пакетам с названиями платформ, при компилировании под которые можно будет использовать эти типы. Таким образом, с файлами библиотек можно работать так же, как и с файлами haXe-проекта пользователя.

Механизм работы с библиотекой

Внутри данного расширения библиотека представлена в виде проекта, за исключением того, что проект библиотеки создается не на основе реального ресурса рабочего пространства, а является виртуальной сущностью и хранит ссылки на файлы библиотек. Как и для обычного проекта, для проекта библиотеки создается список файлов, входящих в нее. Файлы обрабатываются синтаксическим анализатором, и по их содержимому генерируются синтаксические деревья. На последнем этапе проводится анализ связей для всех файлов внутри библиотеки.

Используемая версия библиотеки

Используемый в проекте синтаксический анализатор пока что не распознает препроцессорные команды в тексе программы. Однако такие команды активно используются в файлах библиотек, например, для открытия или закрытия доступа в зависимости от версии платформы (например: haXe можно скомпилировать под конкретную версию flash платформы, начиная с 6-ой). Поэтому было решено сосредоточиться на платформе flash 9-ой и 10-ой версии и для них создать версию библиотек без препроцессорных команд, которую можно было бы использовать в данном проекте.

Особенности работы с проектом библиотеки

Как правило, специальных инструментов для работы с библиотекой в таком виде не требуется – поиск типа проводится по названию сразу после поиска типа в параметрах класса. Однако для некоторых бинарных выражений для выведения типа требуется либо сравнение типа компоненты выражения с определенным типом из библиотеки (например: почти все арифметические операции доступны только для чисел), либо результатом выражения может служить тип, не связанный с типом компонент, в него входящих (например: логические выражения). Все типы, требуемые в таких случаях, являются примитивными и их число меньше, чем число доступных примитивных типов в библиотеке. Поэтому для всех них были созданы специальные методы поиска.

Сбор информации об использованиях

Для реализации основной задачи данной дипломной работы – поддержки автоматизированных преобразований кода, а именно рефакторинга, требовалось иметь информацию обо всех использованиях какой-либо переменной, типа или функции. После того как файлы в проекте обработаны и связаны со своими определениями, собрание данной информации не составляет сложную задачу.

Этапы работы

1. Проводится поиск узла, соответствующего месту, выделенному пользователем в тексте программы. Например, текущий механизм запускается каждый раз, когда пользователь меняет положение курсора в тексте программы. Если выделенному месту не соответствует никакой конкретный узел, то ищется узел, наиболее подходящий по параметрам – отступ

начала текста программы, который представляет этот узел, должен быть меньше или равен абсолютной позиции в файле (offset), а offset конца должен быть больше или равен чем соответствующий отступ выделения.

2. Узел проверяется на возможность проведения для него поиска использований, и, если он не пригоден, обрабатывается. Если поиск невозможен даже после обработки узла, либо сама обработка невозможна, то алгоритм прекращает свою работу. Существует два вида обработки узлов для применения алгоритма.
    a. Обработка для вывода результатов в CallHierarchyView. В этом случае достаточно найти первый родительский узел, соответствующий использованию или объявлению переменной или функции.
    b. Обработка для применения рефакторинга (в данной работе был реализован рефакторинг переименования, поэтому рассматривается именно он). В этом случае выделенная область должна охватывать полностью и точно, либо являться частью объекта, для которого можно провести преобразование рефакторинга (например для переменной выделена часть ее имени, но не захвачено никакой лишней области). В случае бинарных выражений и выделения нулевой длины, выбирается подузел, наиболее близкий к выделению. В случае любого другого некорректного выделения алгоритм завершит свою работу.
3. На этом этапе у нас есть целевой узел, для которого возможен поиск определений. Теперь достаточно взять ссылку на его узел-объявление и найти все узлы, у которых эта ссылка на узел-объявление ведет на тот же самый узел. Этим поиском занимается ReferenceListBuilder.

Rename-преобразование

Поддержка реализации пользовательских автоматизированных преобразований в Eclipse

В Eclipse встроены средства, облегчающие добавление возможностей рефакторинга в проект. За хранение и применение всех изменений, связанных с рефакторингом, отвечает класс Change. Подклассы Change реализуют шаблоны основных типов изменений, которые производятся при рефакторинге: текстовые изменения, удаление или перемещение ресурсов, композиция изменений.

За создание набора изменений для конкретного случая применения рефакторинга отвечает пользовательский подкласс класса RefactorProcessor. Он также проверяет условия корректности применения преобразований в каждом конкретном случае.

Архитектура рефакторинг-модуля данного проекта

Основным классом, отвечающим за переименование, является RenameProcessor. Наследники RenameProcessor реализуют создание наборов изменений и проверку условий для разных типов узлов синтаксического дерева. Например, VariableRenameProcessor отвечает за переименование полей пользовательских типов и локальных переменных, а FunctionRenameProcessor отвечает за переименование функций или интерфейсов.

RenameProcessor предоставляет механизмы для проверки начальных и конечных условий переименования, а также он ответственен за создание набора изменений, являющихся сутью операции рефакторинга.

Проверка начальных условий корректности применения преобразований

Проверка начальных условий должна быть настолько короткой, насколько возможно, и проводиться еще до получения информации о новом имени цели преобразования. Как правило, результаты такой проверки используют еще до вызова пользователем, для запрета возможности такого вызова – например, для сокрытия соответствующего пункта меню.

Для возможности использовать функцию переименования, файл, содержащий определение переименовываемого объекта, должен:

- не принадлежать библиотеке haXe;
- быть разрешенным для редактирования текущим пользователем.

Проверка окончательных условий корректности применения преобразований

Проверка окончательных условий проводится после создания набора преобразований, но до их применения. Успешное ее завершение приведет к применению преобразований к коду или ресурсам проекта.

В данном проекте проверяется только корректность имени. Классическим условием является запрет на использование пустой строки или старого имени в качестве нового. Правильное имя должно начинаться с буквы и может содержать только латинские буквы, цифры и знак подчеркивания. Еще новое имя не должно совпадать ни с одним из ключевых слов языка, за исключением ключевого слова «new» при переименовании функций, т.к. конструктор класса имеет такое имя.

Особые случаи, возникающие при использовании переименования

Все члены класса должны иметь разные имена, независимо от вида – поле, функция, конструктор. Поэтому, например, если после переименования поле и функция имеют одинаковое имя, то компилятор haXe выдаст ошибку «Duplicate class field exception». В случае локальных переменных, совпадение их имени с именем типа, функции или другой переменной допустимо.

Ситуации, которые могут возникнуть после переименования с использованием ранее использовавшегося имени.

- **Две локальные переменные имеют одинаковое имя**. Ошибка будет быстро обнаружена при выводе типов, если первая переменная активно использовалась, а тип второй переменной является подтипом типа первой переменной или совсем с ним не связан. В обратном случае поведение программы будет непредсказуемым, а поиск ошибки будет полностью зависеть от пользователя.
- **Имя локальной переменной совпадает с именем функции, в которой эта локальная переменная находится**. Единственный побочный эффект – невозможность использования рекурсии. Уже существующие рекурсивные вызовы будут отмечены расширением как ошибочные.
- **Новое имя совпадает с именем текущего пользовательского типа.** Это отразится только на местах в программе, где вызовы статических членов класса были сделаны через имя типа. Такие места будут отмечены расширением как ошибочные в случае, если тип новой переменной отличен от текущего класса. К статическим полям класса по-прежнему можно будет обращаться с помощью оператора «this».

- **Новое имя совпадает с именем существующего пользовательского типа, отличного от текущего**. В данную категорию входят также имена стандартных типов из библиотек haXe. В этом случае доступ к статическим полям типа с тем же именем будет невозможен в методе, где была объявлена переменная, в случае с локальной переменной, или во всем классе, в случае с полем класса. Подобные места будут отмечены расширением как ошибочные. Создавать новые переменные упомянутого типа, однако, будет по-прежнему возможно.

Учет возникновения особых случаев

Из всех рассмотренных ситуаций только первая может привести к трудно обнаруживаемой ошибке. Поэтому, хотя переименование запрещено не будет, пользователь будет уведомлен о возможном совпадении имен. Проверка на совпадение имен проводится во время проверки окончательных условий и для успешного завершения должна вернуть утвердительный ответ от пользователя.

5. Заключение

Главной задачей данной работы была поддержка автоматизированных преобразований кода в IDE для haXe. Она успешна выполнена.

- Реализован сбор информации обо всех использованиях какой-либо переменной или функции в проекте или группе проектов на основе контекста.
- Добавлена возможность переименования пользовательских типов или объектов.

В ходе работы были решены дополнительные задачи.

- Библиотеки haXe обработаны для улучшенной совместимости с текущей версией грамматики.
- В среду добавлена поддержка проекта и группы проектов.
- Упрощен доступ пользователя к различной функциональности плагина:
    - wizard’ы проекта и файлов;
    - страница настройки параметров расширения;
    - содержимое build файла генерируется по настройкам, заданным при его создании
- Доработано внутреннее представление типов, улучшен вывод типов.
- Добавлено тестирование для парсера грамматики и тестирование реализованной функциональности вывода типов.

6. Список литературы

1. Бенджамин Пирс «Типы в языках программирования». Изд-о «Лямбда пресс», «Добросвет», 2012. 680 стр.
2. Альфред В. Ахо, Моника С. Лам, Рави Сети, Джеффри Д. Ульман «Компиляторы: принципы, технологии и инструментарий», 2е изд. Изд-о «Вильямс», 2008. 1184 стр.
3. Мартин Фаулер, Кент Бек, Джон Брант, Дон Робертс, Уильям Апдайк «Рефакторинг. Улучшение существующего кода» . Изд-о «Символ-Плюс», 2008. 432 стр.
4. Официальный сайт языка haXe, URL: http://haxe.org (дата обращения: 3.06.2012)
5. Официальный сайт IDE Flash Develop, URL: http://www.flashdevelop.org/ (дата обращения: 3.06.2012)
6. Официальный сайт расширения EclihX для IDE Eclipse, URL: http://www.eclihx.org/ (дата обращения: 3.06.2012)
7. Официальный сайт IDE Eclipse, URL: http://www.eclipse.org/ (дата обращения: 3.06.2012)
8. Раздел официального сайта FDT, посвященный расширению, включающего поддержку haXe, URL: http://fdt.powerflasher.com/docs/An_Overview_of_haXe_%26_FDT (дата обращения: 3.06.2012)
9. Официальный сайт IDE FDT, URL: http://fdt.powerflasher.com/ (дата обращения: 3.06.2012)
10. Раздел официального сайта среды IDE Intellij IDEA, где можно скачать расширение для этой среды, включающее поддержку haXe, URL: http://plugins.intellij.net/plugin/?idea&id=6873 (дата обращения: 3.06.2012)
11. Официальный сайт IDE Intellij IDEA, URL: http://www.jetbrains.com/idea/ (дата обращения: 3.06.2012)
12. Официальный сайт on-line игры MyMiniCity; создан при использовании haXe, URL: http://myminicity.com/ (дата обращения: 3.06.2012)
13. Официальный сайт on-line игры Alpha Bounce; создан при использовании haXe, URL: http://www.alphabounce.com/ (дата обращения: 3.06.2012)
14. Официальный сайт проекта Comapping; создан при использовании haXe, URL: http://www.comapping.com/ (дата обращения: 3.06.2012)
15. Официальный сайт текстового редактора TextMate, URL: http://macromates.com/ (дата обращения: 3.06.2012)
16. Официальный сайт текстового редактора JEdit, URL: http://www.jedit.org/ (дата обращения: 3.06.2012)
17. Официальный сайт текстового редактора Vim, URL: http://www.vim.org/ (дата обращения: 3.06.2012)
18. Грамматика написанная Ником Сабавски для языка haXe под парсеро-генератор GOLD, URL: http://www.semitwist.com/download/haxepred/Haxe_gold_0.2.zip (дата обращения: 3.06.2012)
19. Официальный сайт проекта LPG, URL: http://sourceforge.net/projects/lpg/ (дата обращения: 3.06.2012)
20. Официальный сайт на русском, посвященный Java, URL: http://www.java.com/ru/ (дата обращения: 3.06.2012)
21. Официальный сайт Plug-in Development Environment (PDE), URL: http://www.eclipse.org/pde/ (дата обращения: 3.06.2012)
22. Красько Н.Л. Разработка отладчика для программ на языке haXe и целевой платформы Adobe Flash 9. Дипломная работа, СПбГУ, 2008. 30 стр.
23. Кондратьев А.Е. Eclipse плагин для программирования на haXe. Дипломная работа, СПбГУ, 2010. 26 стр.
24. Jeffrey L. Overbey, Ralph E. Johnson «Regrowing a Language. Refactoring Tools Allow Programming Languages to Evolve» // OOPSLA '09, pages 493-502, New York, ACM, 2009
25. Dig and R. Johnson. « How do APIs evolve? A story of refactoring» // J. Software Maintenance and Evolution: Research and Practice, 18(2):83–107, 2006