Санкт-Петербургский Государственный Университет
Математико-механический факультет

Кафедра системного программирования

Eclipse-плагин для программирования на haXe

Выпускная работа бакалавра
студента 461 группы

Кондратьева Анатолия Евгеньевича

Научный руководитель	..... /подпись/	ст. преп. В.С. Полозов
Рецензент	..... /подпись/	ст. преп. Н.Н. Вояковская
“Допустить к защите” заведующий кафедрой	..... /подпись/	д.ф.-м.н., проф. А.Н. Терехов

Санкт-Петербург
2010

SAINT PETERSBURG STATE UNIVERSITY
Mathematics & Mechanics Faculty

Software Engineering Chair

HaXe programming language support in Eclipse IDE

by

Anatoly Kondratyev

Bachelor's graduation paper

Supervisor	..... /signature/	Senior Lect. V.S. Polozov
Reviewer	..... /signature/	Senior Lect. N.N. Voyakovskaya
«Approved by» Head of Chair	..... /signature/	Professor A. N. Terekhov

Saint Petersburg
2010

Содержание

1 Введение

У понятия "язык программирования" нет чёткого определения, каждый специалист понимает его немного, но по своему. Они стали появляться в середине 50-ых годов XX века, некоторые из них дошли до нашего времени. В последующие годы появлялось все больше и больше новых языков программирования, и они продолжают появляться до сих пор. В основном, причиной создания новых языков является нацеленность на решение задач в конкретной области.

Говоря о языках программирования, следует отметить тенденцию к появлению платформ, на которых исполняются или отдельные языки, или целые семейства. Наиболее распространённые примеры: Java (с JVM – Java Virtual Machine), .NET (с CLR – Common Language Runtime), Adobe Flash (Adobe Flash Player).

Неотъемлемой частью процесса разработки является использование интегрированных средств разработки (IDE, Integrated developer environment) - это система программных средств, используемая программистами для разработки программного обеспечения [7]. В начале своего развития практически любой язык программирования имеет этап, когда программы на нем пишутся в примитивном текстовом редакторе, а код передаётся компилятору или интерпретатору. Разумеется, это не способствует удобной и быстрой разработки. Ниже перечислены компоненты, которые сейчас уже де-факто считаются стандартными для полноценной работы IDE:

- Текстовый редактор
- Система подсветка синтаксиса
- Компилятор и/или интерпретатор
- Средства автоматизации сборки
- Средства проведения рефакторинга:
  - переименование,
  - извлечение методов и классов,
  - перемещение и др.
- Визуальное представление кода (outline view)
- Отладчик
- Средство связывания с системами управления версиями
- Для языков, предусматривающих построение пользовательских интерфейсов:
  - графический менеджер
- Для объектно-ориентированной разработки ПО так же включаются:
  - браузер классов,

– диаграмму иерархии классов.

Наличие все этих компонент значительно упрощает процесс разработки и отладки кода. Сейчас существуют среды разработки предназначенные и для нескольких языков (Eclipse [12], Microsoft Visual Studio [14] и прочие), они изначально создаются с расчётом на расширение с помощью плагинов, для поддержки новых языков.

Одним из языков, который ещё только развивается, и не имеет хорошего средства разработки, является язык haXe.

haXe — объектно-ориентированный, строго типизированный язык программирования высокого уровня, ориентированный на разработку Web-приложений [24]. Разработка его началась в 2005 году, французской компанией Motion-Twin. На данный момент язык продолжает развиваться, последний релиз, 2.05, был выпущен 9 января 2010.

При создании этого языка разработчики преследовали 2 идеи, первой было создание языка, не привязанного к конкретного платформе (как Java привязана к JVM (Java Virtual Machine)). Имеется в виду, что при компиляции можно указывать, под какую из возможных платформ код должен быть скомпилирован. На данный момент возможна компиляция для JavaScript, Flash, NekoVM [23], PHP; ведутся работы над возможностью компиляции для JVM. Второй идеей была направленность на Web-разработку. Так, одним из желанием создателей было создание языка, на котором одинаково удобно можно было бы создать и front-end (часть программной системы, которая непосредственно взаимодействует с пользователем), и back-end (часть системы, которая инкапсулирует компоненты, обрабатывающие выходную информацию от front-end).

Несмотря на недавнее появление, этот язык имеет быстрорастущее сообщество поклонников, а в интернете появились сайты написанные полностью на haXe, или со вставками на нем (haXe Website [24], MyMiniCity [26], Alpha Bounce [27]); так же серия компаний выполняет коммерческие проекты с использованием haXe. Не в последнюю очередь такая популярность объясняется бесплатностью и открытостью кода.

Один из вопросов, который встаёт перед программистом, решившим воспользоваться haXe, состоит в выборе среды программирования. Создатели языка не предоставляют какой-либо среды программирования от себя, но существует некоторое количество других разработок. К сожалению, все они имеют те или иные недостатки, которые будут рассмотрены позже. Таким образом, задача усовершенствования или создания новой среды разработки для haXe, устраняющей недостатки существующих IDE, является актуальной проблемой, решение которой и было главной задачей бакалаврской работы.

2 Обзор

В данном обзоре будут рассмотрены существующие IDE (IDE от Integrated Development Environment — интегрированная среда разработки) для haXe, а также рассмотрен вопрос создания новой среды разработки.

При решении задачи написания среды разработки для какого-либо языка, перед разработчиком встаёт множество вопросов, среди них:

- Какой язык использовать для реализации основных компонент?
- Как реализовывать синтаксический и лексический анализатор?
- Как должны взаимодействовать между собой различные модули среды?

В случае реализации IDE для одного языка часто пишется не новая среда программирования, а плагин к одной из существующих. В данном обзоре будут рассмотрены возможности по написанию плагинов для NetBeans [13], IntelliJ IDEA [15] и Eclipse [12].

2.1 Сравнительный анализ IDE для haXe

На данный момент наиболее распространённым продуктом для программирования на haXe является FlashDevelop, а также и eclihx [31] - дополнение к Eclipse. Многие же используют просто различные виды текстовых редакторов, таких как TextMate [20], JEdit [21], GEdit [37], UEx [38], VIM [22] и другие, с установленными дополнениями отвечающие за подсветку кода и его автодополнение.

FlashDevelop [16] является бесплатной средой разработки, лицензированной по MIT [43]. Изначально разработка была ориентирована на платформу Windows (версия 3.0.6 работает с WindowsXP, Windows Vista, Windows 7).

При работе в FlashDevelop с haXe программисту доступны такие функции, как:

- Подсветка кода
- Использование шаблонов кода
- Автодополнение кода
- Навигация по коду
  - Переход от использования к определению по F4
- Визуализация структуры кода

К сожалению, никакие средства рефакторинга и отладки кода не предоставляются, что снижает эффективность разработки.

Проект Eclihx [31]. Этот проект был начат в рамках дипломной работы студента кафедры системного программирования математико-механического факультета СПбГУ Николаем Красько. Целью проекта было введение системы отладки при компиляции для Flash платформы. В данный момент доступна так же стандартная функциональность подсветки, автодополнения и компиляции проекта. Проект реализован в виде плагина для Eclipse. Этот проект продолжает развиваться.

В заключение можно сказать, что вопрос реализации новой среды программирования или усовершенствовании существующей с целью введения улучшенной навигации по коду, способов отображения кода (таких как дерево полей и методов класса), иерархии классов, всплывающих подсказок по элементам кода, а так же средств рефакторинга и отладки, является актуальным и своевременным.

2.2 Выбор базовой IDE

Выбирая между улучшением существующих проектов (плагина к FlashDevelop), и написание нового для Eclipse, NetBeans или IntelliJ IDEA, было решено рассматривать только последние три варианта, поскольку FlashDevelop – редактор с подсветкой построенный на основе свободной компоненты – редактора Scintilla. Проект активно развивается, однако поддержка новых языков во FlashDevelop осуществляется на уровне проекта, прозрачного вызова компилятора, и удобного редактора. Разработчику языка не предоставляются средства для создания и хранения промежуточного представления программы, и осуществления преобразований программы (рефакторинга). Если же писать свой редактор для Eclipse, NetBeans или IntelliJ IDEA, то он разрабатывается как внешний модуль, и практически никогда не требует изменения самой IDE, поскольку вся необходимая функциональность предоставляется базовой средой разработки.

2.2.1 Описание среды NetBeans

Создатели NetBeans IDE [13] отдельно выделяют два инструмента для поддержки новых языков: проект Шлиман [39] (Project Schliemann) и GSF [40] (Generic Scripting Framework).

Проект Шлиман (Project Schliemann) [39]. Основная идея, ради которой задумывался этот проект, это возможность за короткое время, практически не прикладывая усилий, добавить базовую функциональность лексического и синтаксического анализа, но без поддержки семантического анализа. Таким образом, используя проект Шлиман, можно указать как отображать язык, но не как с ним работать. Имеется в виду, что добавление поддержки рефакторинга, компиляции, запуска программы, используя проект Шлиман, чрезвычайно затруднительно, и создатели проекта советуют не пытаться пробовать реализовать это.

GSF - Generic Scripting Framework [40] является вторым механизмом для введения поддержки новых языков в NetBeans IDE. В отличии от Шлимана, где для реализации поддержки достаточно описать несколько конфигурационных файлов, в GSF от программиста требуется уже организовать самому поддержку лексера и парсера, и в последствии работать через GSF уже с имеющимися деревьями выводов. Главная идея такого подхода состоит в том, что считается, что парсер и лексер у пользователя уже имеются, и он может сконцентрироваться на их улучшении, в то время как GSF возьмёт на себя связывание получаемых результатов с тем, что отображается в IDE. То есть, GSF будет вызывать парсер в нужные моменты (в момент начального индексирования, в момент первого открытия, спустя несколько секунд после активности пользователя и т.д.). Программисту же остаётся только следить за деревом парсера и абстрактным синтаксическим деревом, и указывать, какие их узлы как связывать с классами IDE.

В данном случае слово скриптовой (scripting) в названии не является характеристикой языка, под который можно писать плагины с помощью GSF, наглядным подтверждением этому является то, что в NetBeans поддержка Ruby (который не является скриптовым), реализована как раз с помощью GSF.

Одна из основных проблем, которые встают перед разработчиком, решившим воспользоваться GSF для своей работы это то, что этот инструмент находиться ещё в стадии разработки и не имеет публичного фиксированного API, и создатели оставляют за собой право менять его в следующих версиях.

2.2.2 Описание среды IntelliJ IDEA

Создатели IntelliJ IDEA [15] пока не предоставляют инструментов наподобие GSF или Шлимана, как в NetBeans, зато у них есть открытый и фиксированный API для описания взаимодействия среда программирования - язык. При этом, отдельно нужно реализовывать систему лексического анализа, которая будет удовлетворять API IDEA, отдельно парсер, отдельно придётся регистрировать в IDE расширение нового языка, для адекватного его отражения, и т.д.

Лексер предлагается реализовывать с помощью JFlex-а [19], и существует плагин JFlex Support для более удобной работы через IDEA. На втором шаге нужно реализовывать систему семантического анализа, при этом в данный момент нельзя взять имеющуюся грамматику языка и воспользоваться парсеро-генератором, необходимо вручную реализовать рекурсивно-нисходящий парсер, с огладкой на механизм взаимодействия дерева абстрактного синтаксического дерева и дерева парсера с принципами взаимодействия IDE с языком.

2.2.3 Описание среды Eclipse

Платформа Eclipse [12] изначально задумывалась не как единый продукт, а как множество модулей, взаимодействующих друг с другом. Поэтому сразу же вопрос написания разнообразных плагинов был широко освещён. Говоря о модульности Eclipse необходимо упомянуть суб-проект Eclipse PDE (Plug-in Development Envi-

ronment) [28] – проект для написания, тестирования, отладки, сборки и размещения Eclipse-плагинов.

В отличии от IntelliJ IDEA, где синтаксический и лексический анализ нужно было проводить вручную, при работе с Eclipse можно воспользоваться лексером и парсером сгенерированными каким-либо сторонним инструментом, например ANTLR [17], по грамматике языка.

Xtext [29] – фрэймворк для разработки программ на произвольном предметно-ориентированном языке [6]. Основная его идея такова: описывается грамматика языка так, чтобы она соответствовала расширенной форме Бэкуса-Наура (РБНФ) [10] – это формальная система определения синтаксиса, в которой одни синтаксические категории последовательно определяются через другие. Встроенный в Xtext генератор создаёт парсер, мета-AST (AST – Abstract syntax tree, Абстрактное синтаксическое дерево) [5, 8] модель, и используя эту модель создаётся текстовый редактор с возможностями подсветки кода, визуальное представление структуры кода (outline view), навигацию по коду через ссылки. Одна из существенных проблем при использовании Xtext-а – необходимость реализовывать грамматику, используя родные средства Xtext (которые являются расширенными средствами ANTLR [17]), что не всегда приемлемо.

IMP [32] – проект, целью которого было получить инструмент для создания IDE под Eclipse. Основным постулатом проекта является упрощение разработки IDE для новых языков, а именно:

- Генерация и управление парсером, AST-деревом, семантическим анализом
- Подсветка синтаксиса, визуальное представление (outline view), навигация между файлами по гипер-ссылкам, сборщики проектов, вывод ошибок
- Поддержка рефакторинга:
  - переименование переменных,
  - перенос элементов кода (Move),
  - извлечение методов (Extract Method),
  - и другие
- Статический анализ кода:
  - всплывающие подсказки
- Исполнение и отладка программы

В основе проектов на IMP лежит AST-дерево, полученное с помощью генератора парсеров LPG [18] (сокращение от LALR Parser Generator). Многие возможности оказываются завязанными на специфичные возможности AST-дерева, которое конструируется по парсеру от LPG. Тем не менее, создатели заявляют, что в случае,

если у пользователя есть уже готовый парсер, то для реализации базовых возможностей IMP-а достаточно имплементировать ряд интерфейсов. Подробнее об этом будет сказано позже.

В заключение можно сказать, что выбирая между проектом Xtext и IMP, в данной ситуации предпочтительнее IMP, поскольку он позволяет использовать произвольный парсер, а, главное, имеет более удачный список возможностей, больше соответствующий полноценной IDE.

2.2.4 Выводы

Таким образом, при выборе среды для написания плагина для haXe, выбор можно проводить между IntelliJ IDEA и Eclipse. Возможности NetBeans можно не учитывать, поскольку с помощью проекта Шлимана нельзя получить необходимую функциональность, а GSF находится ещё в стадии разработки и не имеет зафиксированной документации. К тому же, в свете поглощения компании Sun Microsystems компанией Oracle Corporation, неизвестно будет ли и дальше поддерживаться и развиваться NetBeans.

Делая выбор между IntelliJ IDEA и Eclipse можно отметить, что будучи в большинстве своём схожими продуктами, Eclipse в сфере написания дополнений можно считать более предпочтительным, поскольку необязательно полностью создавать свой парсер и лексер, а можно воспользоваться сгенерированными; при работе с Eclipse можно пользоваться PDE и одним из указанных выше плагинов (например, IMP), что значительно упростит разработку.

2.3 Принципы написания плагинов для Eclipse

Несмотря на то, что сейчас существует большое число разнообразных IDE, которые сильно отличаются друг от друга, практически все они имеют схожу архитектуру. В основном, всякую IDE можно разделить на серию модулей, в зависимости от функциональности. Два наиболее общих модуля, или, как ещё называют, уровня, это:

- ядро,
- графический интерфейс пользователя (ГИП).

Ядро – это база всякой IDE, которая состоит из лексического и/или синтаксического анализатора, который распознает элементы языка. Графический интерфейс – это то, что строится на основе ядра, и включает такие элементы как подсветка синтаксиса, визуальное представление кода и прочее. Все компоненты IDE могут быть отнесены к уровню ядра или ГИП.

Теперь подробнее рассмотрим архитектуру самого Eclipse. Как было сказано ранее, она не монолитна, а состоит из множества модулей. В принципе, можно сказать, что сам Eclipse является всего лишь небольшим ядром, содержащим загрузчик

плагинов, а это ядро окружено сотнями обычных плагинов, которые и составляют структуру IDE. Каждый из плагинов может как зависеть напрямую от ядра, так и от других дополнений. В любом случае, все они исполняются в окружении ядра, как-то влияя на всю IDE в целом. Компонуя разные плагины, можно получить среду разработки ориентированную на разные задачи[3].

Обсудим структуру самого плагина. Он состоит из следующих частей:

- Java-классы – обычные классы, располагающиеся в стандартной иерархии внутри JAR файла[35].
- Ресурсы: иконки и прочие вспомогательные данные – так же хранятся внутри JAR-файла. Их положение может указывается в plugin.xml, или непосредственно в кое самих классов.
- META-INF/MANIFEST.MF – файл, описывающий аспекты выполнения плагина, такие как идентификатор, версию, зависимости плагина.
- plugin.xml – файл в формате XML, описывающий расширения и точки расширения.

JAR-файл, содержащий плагин, должен иметь определённое имя и находится внутри определённой директории Eclipse, где IDE сможет найти и загрузить его. Имя файла должно содержать идентификатор плагина, потом подчёркивание, и версию: $com.qualityeclipse.favorites\_1.0.0.jar$.

Отдельно стоит осветить понятия расширений и точек расширений, упомянутые выше. Плагин может объявлять точки расширения так, чтобы другой плагин мог изменять (обычно увеличивать) функциональность оригинального плагина в соответствии с нуждами разработчика. Такой механизм позволяет делать дополнения независимыми, поскольку начальный плагин в этом случае может ничего не знать о дополнении, которое его расширит. Для каждой точки расширения должна быть указана её категория и список атрибутов, необходимых, чтобы этой точкой воспользоваться.

Подробнее вопросы связанные с написанием плагинов будут рассмотрены позже.

2.4 Обзор проекта IMP

Стоит вернутся к проекту IMP [32], заявленному ранее. Как было сказано, этот проект направленный на быстрое создание средств разработки для произвольных языков. С самого начала стоит отметить, что этот проект по неизвестным причинам не является популярным в сообществе Eclipse, несмотря на его возможности. Так же ранее было упомянут проект LPG, который используется IMP для генерации лексера, парсера и построения AST. Возможно, это является одной из причин, по которой проект не очень популярен – разработчики признают, что они работают и тестируют IMP используя парсер, полученный с помощью LPG. Пытаясь решить эту проблему, создатели выделили серию интерфейсов для работы с синтаксическим анализатором и абстрактным синтаксическим деревом, и постарались

привести весь код в соответствии с этими интерфейсами. Таким образом, утверждалось, что любой парсер, который можно будет подключить к классам, имплементирующим данные интерфейсы, будет корректно работать. К сожалению, к моменту написания работы такого опыта у разработчиков и пользователей было очень мало, к примеру, с парсером сгенерированным ANTLR ещё никто не пробовал работать, поэтому они приглашали желающих экспериментировать и сообщать о своих успехах.

В ходе данной работы было принято решение помочь проекту IMP и попробовать использовать их проект с парсером и лексером, сгенерированным ANTLR. Это позволило бы создать мост между стандартными классами и библиотеками времени исполнения ANTLR и компонентами IMP, а так же выявить места в проекте, не соответствующие объявленным интерфейсам. Этот мост впоследствии могли бы использовать как сами создатели IMP при реализации поддержки ANTLR, так и обычные пользователи.

Говоря о преимуществах IMP, нужно пояснить его структуру. Этот проект является некоторой перемычкой между стандартными точками расширений компонент Eclipse и тем, что желает реализовать пользователь. IMP позволяет упростить работу, изменяя plugin.xml, MANIFEST.MF: он добавляет необходимые точки расширения и каркасы классов, необходимые для реализации логики. Иначе говоря, теперь программисту не нужно знать сколько и каких точек расширения нужно добавить, и какие создать классы для того чтобы ввести определённую функциональность. Используя мастера от IMP-а, можно указать, какая функциональность нужна, и IMP сам добавит нужные точки расширений. Для человека, который никогда ранее не работал с написанием сложных плагинов для Eclipse, эта функциональность очень полезна, поскольку позволяет не тратить время на разбор зависимостей между плагинами.

2.5 Обзор проекта ANTLR

Как было сказано в предыдущем параграфе, было принято решение использовать парсер, сгенерированный ANTLR [17]. ANTLR (ANother Tool for Language Recognition – "ещё одно средство распознавания языков") – это генератор парсеров, позволяющий автоматически создавать программу-парсер (как и лексический анализатор) на одном из целевых языков программирования по описанию LL(*)-грамматики [11] на языке, близком к РБНФ [10]. Так же позволяет конструировать компиляторы, интерпритаторы и трансляторы с различных формальных языков. В рамках данной работы особенно интересно, что ANTLR так же предоставляет удобные средства восстановления после ошибок и систем сообщений о них. Стоит отметить, что ANTLR – проект с открытым свободным кодом, версия 3.0 (на момент написания данной работы являющаяся последней) распространяется по лицензии BSD [41].

ANTLR выгодно отличается от многих других парсеро-генераторов наличием собственной среды разработки ANTLRWorks, позволяющей удобно создавать и отлаживать грамматики: это многооконный редактор, поддерживающий подсветку

синтаксиса, автодополнение, визуальное отображение грамматик, строящееся в реальном времени по мере ввода, отладчик, инструменты для рефакторинга, средства приведения правил к LL виду и т.д.

3 Постановка задачи

При написании этой работы был составлен следующий список задач:

- Описание грамматики haXe для ANTLR.
- Описание AST-дерева для последующей работы и обновление грамматики для генерации такого дерева.
- Реализация базовых алгоритмов на AST-дереве.
- Интеграция полученного парсера с IMP.
- Использование IMP для создания среды разработки для haXe.

В рамках последнего пункта отдельно выделялись такие подзадачи, как:

- Реализация подсветки синтаксиса.
- Реализация навигации по файлу (использование гипер-ссылок от место использования переменной к месту определения).
- Автодополнение кода.
- Диагностика ошибок.

4 Реализация

Ниже будут описаны основные архитектурные решения и проблемы, связанные с данной работой.

4.1 Описание грамматики

4.1.1 Общее описание

На первом этапе написание плагина требовалось создать грамматику, чтобы по ней сгенерировать лексический и синтаксический анализатор. Как было сказано ранее, был выбран парсеро-генератор ANTLR [17]. Наиболее простым решением было бы найти уже существующую грамматику haXe под ANTLR, но таковая не была найдена на момент написания диплома. Была обнаружена грамматика для парсера-генератора GOLD (акроним от Grammar Oriented Language Developer) [34]. GOLD – это парсеро-генератор, использующий LALR алгоритм. Автор грамматики, Ник Сабалавски, признавал, что его грамматика несовершенна, поскольку он не имел достаточного опыта в создании грамматик, и приводил список выявленных им ограничений [9]. Поэтому первой проблемой, которая выявилась при написании грамматики, это перевод грамматики для ANTLR, а так же избавление от бОльшей части ограничений.

Как было сказано при описании ANTLR, для разработки грамматик предоставляется IDE ANTLRWorks. Вместе с тем, существует так же дополнение для Eclipse [30], для разработки под ANTLR. В результате сравнения возможностей плагина и ANTLRWorks было принято решение воспользоваться последним, поскольку плагин находится в состоянии застоя: Создатели гарантируют совместимость только с версиями Eclipse 3.02 и 3.1 (на текущий момент официальная версия Eclipse – 3.5). Кроме того, ANTLWorks предоставляет более богатый инструментарий для разработки.

Следующей проблемой было наличие в языке haXe команд для препроцессора. После обсуждения с научным руководителем, было принято решение не пытаться осуществить полную поддержку команд препроцессора на уровне одного парсера, поскольку это чрезвычайно трудоёмко и затруднительно. Поэтому было решено игнорировать команды препроцессора и не встраивать их в AST-дерево.

К сожалению, не все ограничения на грамматику удалось обойти, незначительная их часть сохранились. Описание доступно в рамках проекта.

4.1.2 Структура AST-дерева

При решении задачи построения AST-дерева использовались классические подходы. Для каждой стандартной структуры языка haXe был построен соответствующий узел:

- определение класса,
- определение функции,

- определение переменной,
- присвоение переменной значения,
- использование переменной,
- операторы циклов Do-While и While-Do,
- оператор for () {...},
- условный оператор if () then {...} else {...},
- оператор switch,
- использование блока.

Каждый узел является наследником класса haxe.imp.parser.antlr.tree.ExtendedCommonTree, который, в свою очередь, является расширением класса org.antlr.runtime.tree.CommonTree.

У каждого узла можно по умолчанию получить номер строки и столбца – это позиция, с которой началось определение данного узла. Создатели ANTLR считают, что эти данные более выгодны, нежели отступ от начала файла. Этот вопрос будет освещён подробнее чуть позже, в разделе интеграции с IMP-ом.

Отдельно стоит упомянуть класс haxe.imp.parser.antlr.utils.HaxeType, который используется для определения типа переменной. Его использование обосновано необходимостью статического вывода типа переменной, и содержит следующие данные:

- иерархию наследования данного класса,
- список интерфейсов, которые реализует данный класс,
- полное имя класса (с пакетом),
- ссылку на узел, в котором хранится определение этого класса.

Эти данные используются уже сейчас при простом выводе типов, и станут необходимыми впоследствии, когда вывод типов будет усовершенствован.

4.2 Операции над деревом

Для реализации функциональности IDE над деревом необходимо уметь проводить серию операций, таких как: поиск наиболее низкого узла, соответствующего данной позиции, нахождение верхней левой и нижней правой координаты, соответствующей данному поддереву, нахождение места определения переменной по её использованию, и другие.

Для поиска узла, соответствующего данной координате, использовался поиск сверху-вниз. В силу того, что не все узлы были упорядочены (имеется в виду, что

для строчки $i = 3$; корнем будет "=", а детьми: i и 3), прямым спуском провести такой поиск нельзя. В некоторых случаях можно спуститься в узел смежный с требующимся, что вызывало определённые сложности.

Для поиска узла, в котором была определена переменная, используется поиск снизу вверх в пределах одного файла. В будущем этот подход планируется изменить (и эти изменения уже частично проведены). Идея этих изменений – хранить список переменных, со ссылкой на места их определения в каждом блоке программы (в определении класса, функции или просто блока). Это необходимо сделать для реализации гипер-ссылок не только в пределах одного файла, но и между файлами.

Третьим алгоритмом можно назвать как раз подсчёт областей видимости для каждой из переменных. Это проход по дереву сверху-вниз, с сохранением в ключевых точках всех переменных, доступных в данном блоке программы. Одна из проблем, с которой можно столкнуться при реализации подобного алгоритма – подсчёт типов переменных. Язык haXe не запрещает не указывать тип переменной:

```
var foo:Int = 5; //сразу указывается целочисленный тип
var bar = 5 + "- good mark";  //тип переменной bar будет String
```

Таким образом, задача вывода типов может быть выделена в отдельную объёмную подзадачу. В рамках же данной работы было решено только анализировать и выводить элементарные типы: Int, Float и String. Преобразования с этими данными оцениваются корректно, и невозможные приведения расцениваются как ошибки приведения типов, о чем сообщается в IDE.

Данная функциональность тестировалась с помощью фрэймворка JUnit. Написанные тесты доступны в рамках проекта.

4.3 Интеграция с IMP

Как было объявлено ранее, одной из задач проекта была демонстрация возможности работы проекта IMP с парсером и AST-деревом, полученным от ANTLR. Так же было сказано, что по словам разработчиков проекта, для реализации базовой функциональности достаточно реализовать следующие интерфейсы:

- org.eclipse.imp.parser.IParseController;
- org.eclipse.imp.parser.ISourcePositionLocator;
- org.eclipse.imp.parser.IMessageHandler;

Перечислены только три интерфейса, как наиболее близкие и важные при работе с парсером. В ходе разработки было выявлена ещё серия второстепенных интерфейсов, которые требовалось реализовать для внесения конкретной функциональности в IDE.

Рассмотрим подробнее каждый из этих интерфейсов. IParseController отвечает за инициализацию парсера и синтаксический разбор текста. Очевидно, что из-за

этого ему нужно хранить путь к файлу, анализ которого он осуществляет в данный момент, последнее полученное AST-дерево, итератор по токенам лексического анализатора. Так же, он хранит ссылку на IMessageHandler и ряд вспомогательных объектов. IMessageHandler – интерфейс, отвечающий за передачу информации, получаемой во время парсинга (или просто операций над деревом) к fronte-end-у. В простом случае это может быть сообщение об ошибке парсинга, или, в более сложном, проблемы при выводе типов. Из сказанного вытекает проблема: узлы AST-дерева должны каким-то образом передавать (или сохранять) информацию для IMessageHandler. Последний крупный интерфейс, необходимый для имплементации – ISourcePositionLocator. Этот интерфейс предназначается для работы с AST-деревом. В его методы входит:

- нахождение узла по смещению от начала файла,
- нахождение узла на началу смещения и концу смещения (это может давать не самый нижний узел),
- нахождение смещения узла,
- нахождение конца смещения узла,
- нахождение текстовой длины узла,
- нахождение файла, соответствующего данному узлу.

Если первые 5 методов являются обычными операциями над деревом, то последний пункт является нестандартным для AST-дерева ANTLR и так же нуждался в реализации. Тем не менее, первые 5 методов тоже представляют некоторую проблему. Как было сказано ранее, ANTLR приветствует работу не с отступами, а с позицией столбца и строки. К счастью, функциональность отступов не была полностью вырезана из ANTLR, но скрыта в классе org.antlr.runtime.CommonToken.

Стоит вернутся к архитектуре IMP и взаимодействию с ним. Как было сказано, одна из целей IMP – избавить программиста от необходимости думать о том, какие точки расширения нужны пользователю для той или иной функциональности. То есть, используя IMP достаточно указать, какая функциональность необходима, и IMP создаёт расширение на нужную функциональность. Уже внутри этого расширения идёт подключение к стандартным точкам расширения Eclipse-компонент. Это чрезвычайно удобно, поскольку, несмотря на то, что Eclipse – платформа, ориентированная на расширения, литературы, описывающей аспекты создания нетривиальных IDE пугающе мало.

На изображении 1 демонстрируется схема зависимости создаваемого в рамках работы плагина от IMP. Разумеется, для корректной работы плагина необходимо установить на Eclipse и сам IMP-плагин.

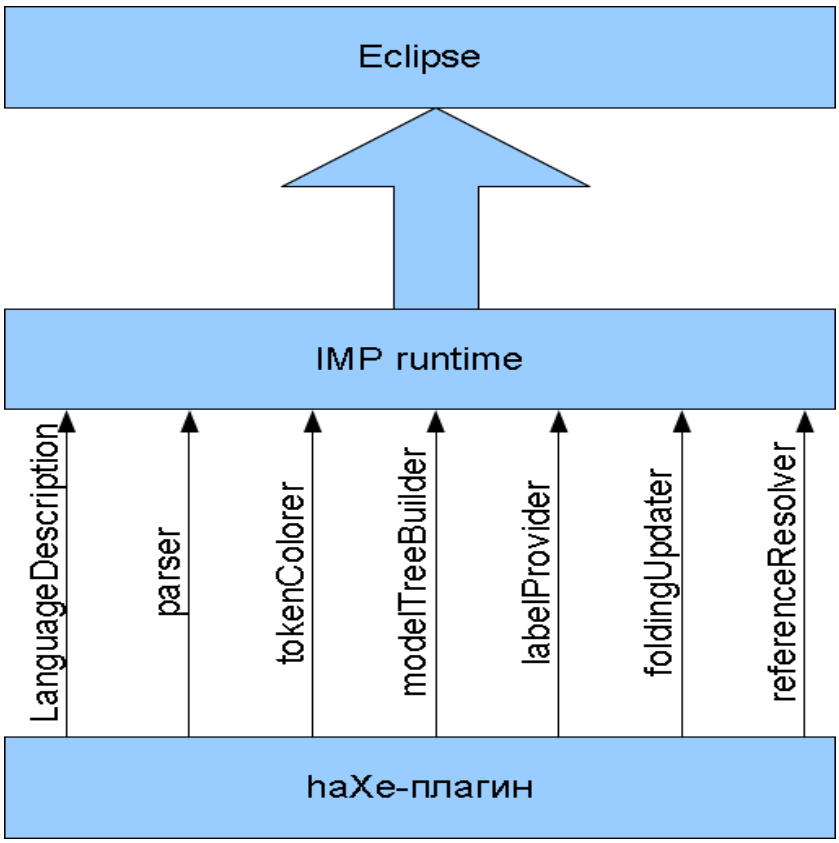

Рис. 1: Структура взаимодействия haXe-плагина с Eclipse посредством IMP

4.4 Реализация возможностей IDE

В рамках работы была реализована следующая функциональность:

- Подсветка синтаксиса
- Визуальное отображение структуры кода (outline view)
- Свёртка элементов кода
- Всплывающие подсказки, показывающие тип переменных
- Выявление ошибок
  - Повторного определения переменной
  - Использования неопределённой переменной
  - Некорректного приведения элементарных типов
- Гипер-ссылки к определению переменной от места использования
- Автодополнение кода

Рассмотрим подробнее каждую из этих сущностей. Каждая из них подключается через определённую точку расширения, но в основе любого дополнения от IMP лежит точка расширения с названием "IMP Programming Language Description"(Plug-in ID: org.eclipse.imp.runtime, Point ID: languageDescription). Это ключевая точка, через которую нужно описать язык, для которого создаётся IDE. В рамках описания указываются следующие атрибуты:

- Физическое имя языка (в данном случае haXe).
- Описание языка (не является необходимым атрибутом), например, "Открытый язык для Web-разработки".
- Расширение, соответствующее файлам данного языка, если их несколько, то указываются через запятую. В случае работы с haXe, это hx.
- Каноническое имя языка, из которого произошёл данный. Этот атрибут так же не является необходимым. Например, PHP вырос из HTML.
- Иконка, а точнее, путь к иконке, которая будет отображаться с файлами, соответствующим данному языку. Не является необходимым.
- URL сайта, который содержит информацию о языке. В случае с haXe можно указать официальный сайт языка: http://haxe.org/ .
- Класс-валидатор. Полное имя класса, который может установить, является ли код в данном файле действительно кодом на указанном языке. Для реализации класса-валидатора можно расширить класс org.eclipse.imp.code. LanguageValidator. Не является необходимым атрибутом.

На базе этого расширения строятся все остальные.

Следующее по значимости расширение – org.eclipse.imp.metatooling.parser, расширение, отвечающее за подключение и работу парсера. Для него необходимо указать всего два атрибута: целевой язык (haXe), и полное имя класса, который реализует интерфейс IParseController, о котором было сказано ранее.

Подключение этого расширения так же изменяет класс-активатор, отвечающий за инициализацию плагина. Плагины в Eclipse используются по схеме "ленивой загрузки", то есть они инициилизируются не при загрузки Eclipse, а только когда в них возникает необходимость. Activator-класс является своеобразной "точкой входа", через которую выполняется загрузка плагина, чтение необходимых ресурсов и прочее.

Все дальнейшие расширения являются не столь значимы, но некоторые из них зависят друг от друга. Большинство расширений имеет только два атрибута – целевой язык, и полное имя класса, реализующее конкретный интерфейс для данной функциональности. Они не будут рассматриваться так же подробно, как первые два, поскольку код содержит всю необходимую информацию в комментариях. Ниже будет просто перечислено, какие расширения отвечают за ту или иную функциональность:

- Подсветка синтаксиса
  - org.eclipse.imp.metatooling.tokenColorer
- Визуализация структуры кода (outline view)
  - org.eclipse.imp.runtime.modelTreeBuilder
- Всплывающие подсказки по элементам кода
  - org.eclipse.imp.metatooling.HoverHelper
- Сворачивание элементов кода
  - org.eclipse.imp.runtime.foldingUpdater
- Использование гипер-ссылок (переход от использования к определению)
  - org.eclipse.imp.runtime.referenceResolvers
- Автодополнение кода
  - org.eclipse.imp.metatooling.contentProposer

Тем не менее, стоит указать на некоторые характерные особенности, выявленные в ходе использования данных расширений. Так, для подсветки синтаксиса можно использовать и родные средства Eclipse, но в этом случае анализ того, какой тип подсветки произвести, будет проводится не на основе потока токенов, а на основе вручную заданных текстовых правил. В большинстве случаев это приемлемый подход, но не имеет смысла фактически повторять правила лексического анализатора, когда он уже реализован. IMP-же, при подсветки синтаксиса, оперирует именно потоком токенов. Всё, что нужно сделать программисту - соотнести определённый тип токенов с нужным цветом. Более того, предусмотрена ситуация, когда цвет элементов кода может зависеть не только от типа токенов, но и от построенного AST-дерева (к примеру, некоторые программисты предпочитают, чтобы константы отличались от переменных). В этом случае можно находить узел, соответствующий данному токену в AST-дереве и уже на основе данных, полученных из этого узла, проводить подсветку.

Так же стоит упомянуть следующую проблему: при реализации свёртки кода, необходимо знать координаты открывающей и закрывающей фигурных скобочек. При описании грамматики и реализации парсера этот аспект не был учтён. Дополнительные проблему привносит тот факт, что, поскольку ANTLR генерирует LL-парсер, то в момент создания соответствующего узла, координата закрывающей скобочки недоступна. После изучения грамматики для языка Java, одобренного создателями ANTLR, было выяснено, что в AST дерево действительно передаётся координата только открывающей фигурной скобочки. Было принято решение вручную изменить парсер, сгенерированный ANTLR таким образом, чтобы сохранять

и координату закрывающей скобочки тоже. Эта операция была с успехом произведена, подробные инструкции по поводу того, как следует изменить парсер, сгенерированный ANTLR, чтобы он корректно работал с плагином, находятся в корне проекта.

В заключение можно сказать, что при реализации той или иной функциональности, требуется обход дерева. Создатели IMP предлагают использовать для этого стандартный паттерн Посетитель (Visitor). Разумеется, они предоставляли Посетителей для дерева, сгенерированного LPG, в ходе данной работы были созданы несколько посетителей, ориентированных на обход AST-дерева от ANTLR.

Так же стоит отметить возможность совместного использования проекта EclihX, упомянутого ранее, с проектом, созданным в ходе данной работы. Поскольку оба они являются плагинами для Eclipse, то пользователь может поставить оба, получив таким образом функциональность по работе с кодом от данного проекта, а функциональность по работе с haXe-проектом (имеется в виду сборка, создание новых файлов и прочее), от EclihX. К сожалению, при таком использовании часть функциональность EclihX по работе с кодом и отладке будет утеряна.

5 Заключение

Данная работа закладывает фундамент для создания интегрированной среды разработки для языка haXe на базе платформы Eclipse. Текущая версия среды предоставляет базовую функциональность, необходимую разработчику при создании приложений на языке haXe:

- Подсветка кода проекта
- Визуальное представление кода
- Свёртывание элементов кода
- Всплывающие подсказки по элементам кода
- Частичная диагностика ошибок, включающая:
  - Повторное определение переменной;
  - Использование неопределённой переменной;
  - Ошибки приведения примитивных типов;
- Переход от использования переменной к определению по ctrl-щелчок
- Использование автодополнение кода

Помимо этого, в рамках проекта была получена грамматика языка haXe для ANTLR. Так же была подтверждено утверждение об использовании парсера, сгенерированного ANTLR для проекта IMP. В рамках последнего утверждения была получена серия замечаний, которые стоит учитывать при построении грамматики для ANTLR.

В будущем планируется расширение функциональности, в том числе интеграция с проектом EclihX, для внесения возможности отладки, а так же введение средств для рефакторинга кода. Ещё один интересный аспект, который не был рассмотрен, это описание грамматики для анализа команд макропроцессора и его интеграция с проектом.

Список литературы

[1] Альфред В. Ахо, Моника С. Лам, Рави Сети, Джеффри Д. Ульман. Компиляторы: принципы, технологии и инструментарий, 2 издание. Изд-во: «Вильямс», 2008. 1184 с.

[2] Роберт У. Себеста. Основные концепции языков программирования (Concepts of Programming Languages). Изд-во Вильямс, 2001. 672 с.

[3] Eric Clayberg, Dan Rubel. Eclipse Plug-ins, 3-е издание. Изд-во:Addison-Wesley 852 с.

[4] Красько Н.Л. Разработка отладчика для программ на языке haXe и целевой платформы Adobe Flash 9. Дипломная работа, СПбГУ, 2008. 30 с.

[5] Cambridge, Nicola Howarth. Abstract Syntax Tree Design, 23 августа, 1995. http://www.ansa.co.uk/ANSATech/95/Primary/155101.pdf

[6] JetBrains, Сергей Дмитриев. Language Oriented Programming: The Next Programming Paradigm, ноябрь, 2004. http://www.jetbrains.com/mps/docs/Language_Oriented_Programming.pdf/

[7] Oracle Corporation, Dana Nourie. Getting Started with an Integrated Development Environment (IDE), Март 24, 2005. http://java.sun.com/developer/technicalArticles/tools/intro.html.

[8] University of Alabama, Joel Jones. Abstract Syntax Tree Implementation Idioms. http://www.hillside.net/plop/plop2003/Papers/Jones-ImplementingASTs.pdf

[9] Грамматика написанная Ником Сабавски для языка haXe под парсерогенератор GOLD, http://www.semitwist.com/download/haxepred/Haxe_gold_0.2.zip

[10] Международный стандарт по Расширенной форме Бэкуса-Наура, http://www.cl.cam.ac.uk/ mgk25/iso-14977.pdf

[11] Описание LL грамматик, http://www.cs.uaf.edu/ cs331/notes/LL.pdf

[12] Официальный сайт IDE Eclipse, http://www.eclipse.org/

[13] Официальный сайт IDE NetBeans, http://netbeans.org/

[14] Официальный сайт IDE Microsoft Visual Studio, http://www.microsoft.com/visualstudio/

[15] Официальный сайт IDE Intellij IDEA,
http://www.jetbrains.com/idea/

[16] Официальный сайт IDE FlashDevelop,
http://www.flashdevelop.org/

[17] Официальный сайт проекта ANTLR и среды ANTLRWorks,
http://www.antlr.org/

[18] Официальный сайт проекта LPG,
http://sourceforge.net/projects/lpg/

[19] Официальный сайт проекта JFlex,
http://jflex.de/

[20] Официальный сайт текстового редактора TextMate,
http://macromates.com/

[21] Официальный сайт текстового редактора JEdit,
http://www.jedit.org/

[22] Официальный сайт текстового редактора Vim,
http://www.vim.org/

[23] Официальный сайт языка Neko и виртуальной машины NekoVM,
http://nekovm.org/

[24] Официальный сайт языка haXe,
http://haxe.org/

[25] Официальный сайт языка Flex,
http://flex.org/

[26] Официальный сайт on-line игры MyMiniCity; создан при использовании haXe,
http://myminicity.com/

[27] Официальный сайт on-line игры Alpha Bounce; создан при использовании haXe,
http://www.alphabounce.com/

[28] Официальный сайт Plug-in Development Environment (PDE),
http://www.eclipse.org/pde/

[29] Официальная страница плагина Xtext,
http://www.eclipse.org/Xtext/

[30] Официальная страница плагина для ANTLR под Eclipse,
http://antlreclipse.sourceforge.net/

[31] Официальная страница проекта eclihx,
http://code.google.com/p/eclihx/

[32] Официальная страница проекта IMP,
http://www.eclipse.org/imp/

[33] Официальная страница проекта ANTLRWorks,
http://www.antlr.org/works/index.html

[34] Официальная страница проекта GOLD,
http://www.devincook.com/goldparser/

[35] Спецификация JAR файлов,
http://java.sun.com/javase/6/docs/technotes/guides/jar/jar.html

[36] Список языков программирования,
http://hopl.murdoch.edu.au/

[37] Страница проекта gedit,
http://projects.gnome.org/gedit/

[38] Страница проекта UltraEdit, текстового редактора UEx,
http://www.ultraedit.com/

[39] Страница описание Проекта Шлимана,
http://wiki.netbeans.org/Schliemann

[40] Страница описания проекта Generic Scripting Framework,
http://wiki.netbeans.org/Gsf

[41] Текст BSD лицензии,
http://www.opensource.org/licenses/bsd-license.php

[42] Текст GNU General Public License, v2,
http://www.gnu.org/licenses/gpl-2.0.html

[43] Текст MIT лицензии,
http://www.opensource.org/licenses/mit-license.php