САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Математико-механический факультет

Верещагина Елизавета Алексеевна

Сборка и публикация бинарных пакетов

Курсовая работа

Научный руководитель:
ст. преп. Григорьев С. В.

Санкт-Петербург
2015

Оглавление

Введение

Самым распространенным способом организации программного обеспечения является создание модульной структуры. Большая задача, поставленная перед разработчиками, разбивается на подзадачи. Для каждой из них выбирается алгоритм решения, который реализуется в отдельной компоненте – модуле. Преимуществами такой организации работы являются:

- Локальное исправление ошибок, т.к. каждый модуль реализован как самостоятельная сущность.
- Упрощение командной разработки.
- Возможность разработки на разных языках программирования.
- Переиспользование компонент в других проектах для решения новых задач.
- С точки зрения пользователя, которому нужен конкретный набор функций, удобно выбрать инструмент, реализующий нужную функциональность, и работать только с ним, а не с большим проектом.

Одним из недостатков модульной структуры проекта является сложный процесс сборки. Чем больше в проекте существует подсистем, тем сложнее выполнять ручную сборку. Удобным решением является автоматизированная сборка. Например, FAKE [1] - FSharp Make - система автоматизации сборки. Fake - предметно-ориентированный язык для написания сборочных скриптов. С их помощью производится запуск сборки, выполнение тестов, автоматическая генерация файлов, содержащих информацию о сборке (assemblyinfo), и т.д.

Метод разработки, когда при каждом подтверждении изменения кода в системе управления версиями на удаленном сервере производится автоматизированная сборка проекта, называется непрерывной сборкой (continuous integration - CI). Такой подход позволяет обнаружить ошибки интеграции и исправить их, что существенно снижает затраты на разработку и отладку проекта. Также благодаря этому производится постоянное поддержание правильной версии и продуктов сборки.

В данной работе рассматривается модульный инструмент для исследования грамматик и платформа разработки - YaccConstructor [5]. Его сборка автоматизирована, осуществляется на сервере TeamCity [3].

Многие из представленных в YaccConstructor модулей независимы между собой, т.е. могут быть использованы как готовый инструмент разработки. Например, фронтенды могут быть использованы пользователем для разбора конкретной грамматики, а генераторы - для получения парсеров, основанных на разных алгоритмах для решения пользовательской задачи.

Одним из признанных способов распространения программного обеспечения является публикация пакетов. NuGet [2] - менеджер пакетов на платформе Microsoft, который обеспечивает создание и использования (установки, удаления, обновления) бинарных пакетов для конкретного решения. Бинарные пакеты содержат исполняемые файлы, которые были скомпилированы для работы на определенной архитектуре процессора. NuGet предоставляет возможность публикации пакетов в общей галерее, а так же создания собственной. Myget.org - сервис, предоставляющий такую возможность. Также в нем доступны пакеты публичной галереи NuGet. Создание собственной галереи пакетов удобно для разработчиков, которые хотят использовать внутренние библиотеки более структурировано.

Сборка и публикация функциональных компонент YaccConstructor в качестве бинарных пакетов в системе MyGet является одним из способов упрощения работы пользователя с отдельным инструментом, требуемым для решения конкретных задач.

1. Постановка задачи

Целью данной работы является упрощение работы пользователя с конкретными функциональными компонентами YaccConstructor.

Для достижения поставленной цели поставлены следующие задачи:

- Изучить структуру проекта YaccConstructor.
- Определить структуру бинарных пакетов, учитывая предоставляемую функциональность.
- Провести предварительное тестирование пакетов.
- Реализовать сборку и публикацию пакетов согласно разработанной структуре.
- Провести апробацию полученной функциональности.

2. Обзор

Ниже представлены аналоги инструмента, выбранного для решения поставленных задач, по разным параметрам.

2.1. Аналоги по способу распространения

Система управления пакетами (package management system) [4] — программное обеспечение, позволяющее управлять процессом установки, удаления, настройки и обновления различных компонентов программного обеспечения. Такие системы используются в UNIX-подобных операционных системах. Программное обеспечение представляется в виде особых пакетов, содержащих помимо дистрибутива программного обеспечения набор определённых метаданных, которые сохраняются в системной базе данных пакетов. Примеры наиболее распространенных систем управления пакетами:

- RPM - менеджер пакетов. Создание пакета осуществляется с помощью команды rpmbuild при наличии spec-файла. Это обычный текстовой файл, который содержит в себе название пакета, версию, номер релиза, архитектуру, под которую собран пакет, инструкции по сборке и установке пакета.
- dpkg - основа системы управления пакетами в Debian. Не предоставляет данных о зависимостях между пакетами и загрузку из сетевого репозитория.

2.2. Аналоги по способу хранения

Существует несколько вариантов работы с пакетами в системах контроля версий (version control system - vcs). В первом содержимое пакета помещается в хранилище кода. Недостатком такого подхода является большой объем занимаемого места в репозитории. При втором подходе пакеты не помещаются в хранилище, хранится только информация о них. Загрузка пакетов происходит только на этапе сборки проекта. Такой подход реализован в NuGet - менеджере пакетов для платформы Microsoft, который распространяется как расширение Visual Studio - интегрированной среды разработки. Работа с ним может быть автоматизирована с помощью скриптов. Преимуществом NuGet является локальность - пользователь устанавливает нужные пакеты только в проект, над которым ведется разработка. При необходимости обновления версий пакетов производятся централизованно для всего решения, а не индивидуально для каждого проекта. Также, если необходимо, ведется автоматическое создание записей перенаправления версий в файлах конфигурации (.config).

3. Реализация

3.1. Выделение, сборка и публикация бинарных пакетов

В проекте YaccConstructor реализованы модули, реализующие различные задачи. В результате анализа предоставляемой ими функциональности была принято решение о структуре каждого бинарного пакета. На диаграмме пакетов (Рис. 1) видны зависимости между выделенными компонентами.

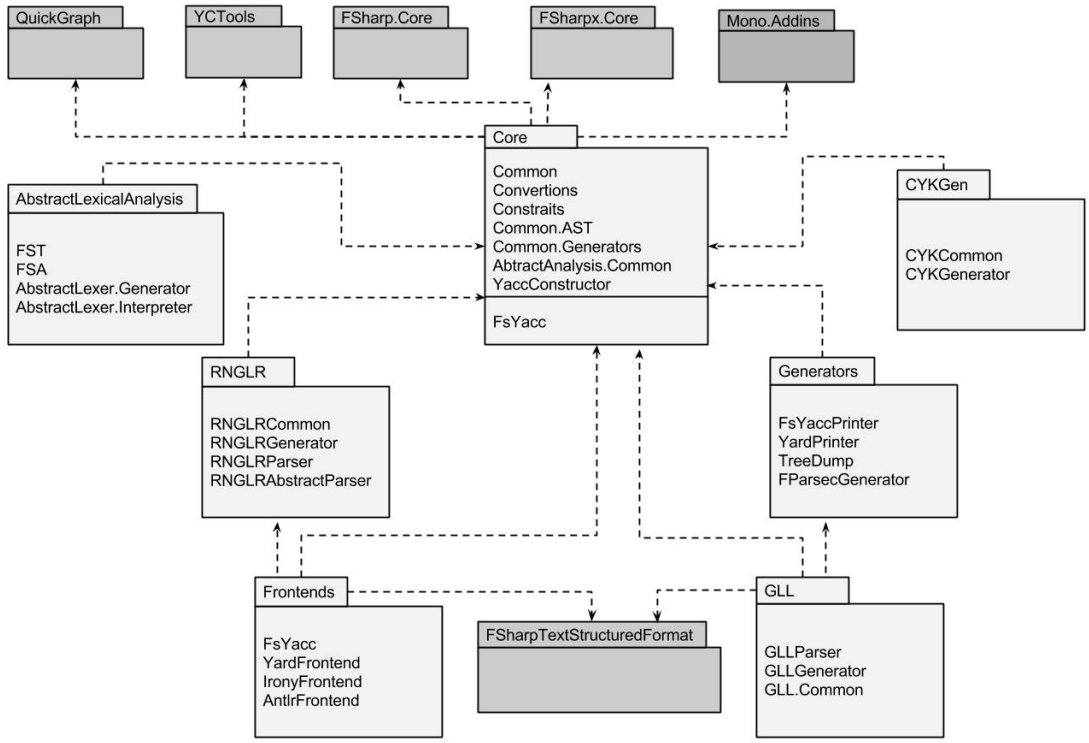

Рис. 1: Диаграмма пакетов. Темно серым показаны сторонние библиотеки, светло серым - пакеты YaccConstructor. Стрелками обозначены зависимости между пакетами.

Ниже представлено краткое описание каждого из них:

- Core
  Ядро YaccConstructor: IL - внутреннее представление грамматики, преобразования грамматики и проверка корректности, управление модулями. В пакете содержится YaccConstructor.exe, с помощью которого осуществляется запуск конкретного инструмента.

- Generators
  Генератор (бэкенд) - инструмент, генерерующий из внутреннего представления грамматики (IL) дерево разбора, код на языке FSharp и т.д. Пакет содержит следующие генераторы: FsYaccPrinter, YardPrinter, TreeDump, FParsecGenerator.

- Frontends
  Фронтенд - модуль, транслирующий представление грамматики в IL согласно языку спецификации грамматики. Пакет содержит FsYacc, YardFrontend, IronyFrontend и AntlrFrontend.

- RNGLR
  Генератор, который согласно описанной грамматике создает восходящий парсер, основанный на алгоритме RNGLR. Работает с контекстно-свободными грамматиками.

- GLL
  Генератор, который создает нисходящий парсер для работы с неоднозначными грамматиками, а так же для разбора контекстно-свободных грамматик, включая леворекурсивные.

- AbstractLexicalAnalysis
  Генератор лексических анализаторов, основанных на конечном преобразователе (finite state transducer - FST), для разбора встроенных языков.

- CYK
  Генератор, который создает восходящий парсер, основанный на алгоритме CYK. Работает с контекстно-свободными грамматиками.

Для каждого пакета на языке XML написан файл спецификации (NuSpec). В нем указан путь к исполняемым файлам и библиотекам, которые должны быть собраны в пакет, а также зависимые библиотеки или пакеты. Зависимые библиотеки - библиотеки, необходимые для работы данного пакета. При установке пакета зависимые библиотеки автоматически устанавливаются в тот же проект решения, что и данный пакет. Также в nuspec присутствует краткое описание пакета, указан автор, версия, тэги для поиска, и т.д.

Для использования какого-либо модуля требуется запуск YaccConstructor.exe. Чтобы обеспечить видимость модулей после установки пакетов в решение, были внесены изменения в файл YaccConstructor.addins, который собирается в пакет Core. В этом файле указан путь к директории, в которой будет осуществляться поиск установленных модулей.

При наличии файлов спецификации с помощью команд nuget pack и nuget push, исполняемых из командной строки, производится сборка и публикация пакетов. Перед публикацией пакетов на myget.org было произведено предварительное тестирование. Выполнена локальная сборка всех пакетов и проверено наличие в пакетах всех необходимых файлов. Также проверено, что все модули доступны для работы.

3.2. Настройка автоматизированной сборки

В YaccConstructor была реализована сборка и публикация пакетов в системе NuGet. При каждом запуске сборки проекта на сервере TeamCity в файле с хранимой версией производится инкремент разряда версии сборки. Эта версия с приписанным к ней суффиксом -On-branch–local проставляется во все файлы спецификации.
Внесены изменения в скрипты сборки, для того, чтобы при сборке пакетов не происходило переименования имен исполняемых файлов в папке Bin/Release.
Сборка и публикация пакета YC.Tools, реализованые в проекте, были внесены в общую сборку пакетов. Файл спецификации YC.Tools помещен в единую директорию с файлами спецификации выделенных пакетов.

3.3. Тестирование

Тестирование полученного решения было проведено на примере среды разработки - IDE (Integrated development environment) для параллельного вычислителя, построенного на виртуальной архитектуре. Из исходного кода на ассемблере требуется получить дерево разбора, с которым будет работать интерпретатор.
Для проведения тестирования были выполнены следующие шаги:

- Бинарные пакеты, содержащие требуемые инструменты, установлены в проект. Для данного решения выбраны YardFrontend и RNGLR генератор. Установлены пакеты YC.Core, RNGLR и Frontends.
- По описанной грамматике с помощью YardFrontend получили внутреннее представление грамматики.
- На основе полученного внутреннего представления грамматики с помощью генератора RNGLR сгенерирован парсер.
- Полученный парсер использовали для генерации дерева разбора кода, интерпретируемого в IDE.

Заключение

В ходе работы были достигнуты следующие результаты:

- Изучена структура проекта YaccConstructor.
- Изучена функциональность каждого модуля, определена структура опубликованных бинарных пакетов.
- Произведено предварительное тестирование бинарных пакетов.
- Реализована сборка и публикация пакетов согласно разработанной структуре.
- Проведено тестирование полученной функциональности.

Код опубликован в репозитории проекта YaccConstructor:
https://github.com/YaccConstructor.
Имя пользователя, под которым работал автор: VereshchaginaE

Список литературы

[1] FAKE. Документация по языку. — URL: http://fsharp.github.io/FAKE/.

[2] NuGet. Официальный сайт. — URL: https://www.nuget.org/.

[3] TeamCity. Официальный сайт. — URL: https://www.jetbrains.com/teamcity/.

[4] Wikipedia. Package management system // Википедия, свободная энциклопедия. — URL: https://en.wikipedia.org/wiki/Package_manager.

[5] YaccConstructor. Сайт проекта. — URL: https://code.google.com/p/recursive-ascent/wiki/YaccConstructor.