РЕАЛИЗАЦИЯ СИСТЕМЫ ПРОЕКТИРОВАНИЯ И ТЕСТИРОВАНИЯ КАРКАСОВ ПРОГРАММНЫХ СИСТЕМ

Шатилина Ю. Е. - студент, Крючкова Е. Н. - к.ф-м.н., профессор,Алтайский государственный технический университет (г. Барнаул)

Современные методы и практики быстрой разработки программного обеспечения нацелены на минимизацию рисков, путём сведения процесса к серии коротких циклов, каждый их которых представляет программный проект в миниатюре и включает все задачи, необходимые для выдачи минимального прироста по функциональности. При таком подходе особенно важно представлять всю систему целиком на ранних этапах разработки. Для решения подобных задач применяются прототипы программных систем, которые позволяют вести разработку проекта методом непрерывной интеграции [1].

Прототип архитектуры программной системы – это упрощённое представление желаемой архитектуры, без учёта деталей и аспектов реализации. Прототипы архитектуры создаются, чтобы смоделировать будущую систему в целом. Ни один из отдельных модулей в прототипе не должен быть особенно функциональным. Основная цель создания прототипа архитектуры в том, чтобы смоделировать разрабатываемую систему и понять, как она будет выглядеть в собранном виде, опуская детали [2].

Поскольку большинство прототипов создаётся с целью моделирования рассматриваемой системы в целом, то полученный прототип есть не что иное, как одноразовая программа, необходимая для того, чтобы получить ответы на определённый ряд вопросов. Главная цель составления такого прототипа – это получить понимание того, как система будет выглядеть в собранном виде, не вдаваясь в детали и особенности. С этой точки зрения может показаться удобным создание прототипов посредством языков очень высокого уровня, а точнее языков более высокого уровня по сравнению с языком, используемым при написании системы. К таким языкам, например, можно отнести Perl и Python. Язык сценариев высокого уровня позволяет опустить многие детали (например, указание типов данных), но при этом создавать функциональный фрагмент программы. Такие языки также позволят при необходимости соединить низкоуровневые фрагменты в новые сочетания.

Согласно приведенным выше рассуждениям, можно сделать вывод о том, что основополагающая проблема прототипирования архитектур программных проектов заключается в отсутствии на рынке целого класса специализированных систем, комплексно удовлетворяющих выдвинутым требованиям. Кроме того, современные тенденции развития методологий и практик в области быстрой интегральной разработки программного обеспечения, лишь подтверждают необходимость в появлении подобных инструментов прототипирования.

Основополагающая идея предлагаемого подхода заключается в использовании унифицированного языка в процессе описания прототипов, а также свойств конечных автоматов в процессе тестирования.

Описание прототипа, составленное с использованием унифицированного языка, является входными данными для системы. Затем на основе описания идёт построение его внутреннего представления. Внутренне представление прототипа должно позволять провести анализ исходного прототипа. С этой точки зрения в качестве внутреннего представления можно выбрать конечный автомат, представленный графом переходов. На основе полученного внутреннего представления можно провести его оптимизацию [3] (например, детерминизацию и минимизацию графа переходов) и его проверку с учётом исходных требований. Будем считать, что прототип является корректным, если в его внутреннем представлении возможен переход из каждого состояния конечного автомата в любое другое состояние. Иными словами, конечный автомат должен представлять собой полный граф (каждая пара различных вершин является смежной). После тестирования и оптимизации прототипа можно переходить к этапу кодогенерации, на котором внутреннее представление транслируется в описание на целевом языке.

Исходя из вышесказанного, можно выделить следующие основные стадии в работе системы:

* лексический анализ;
* парсинг;
* оптимизация внутреннего представления;
* анализ внутреннего представления;
* тестирование.

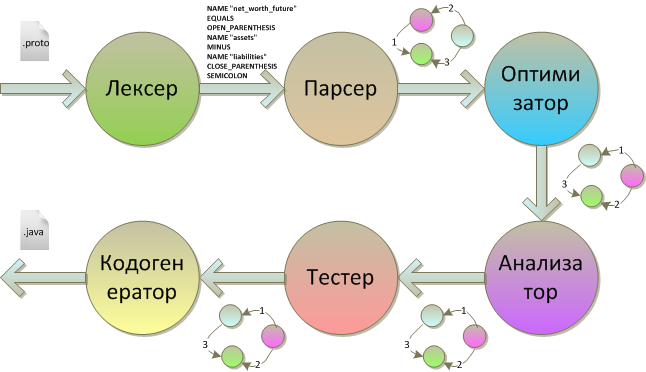


Рисунок 1 – Модель системы

В соответствии с выделенными стадиями работы системы можно разделить всю систему на несколько подсистем (см. Рисунок 1):

* Лексер – делает лексический анализ исходного описания прототипа;
* Парсер – строит внутреннее представление на основе потока токенов;
* Оптимизатор – выполняет ряд оптимизации над внутренним представлением, например, детерминизацию и минимизацию графа переходов для упрощения процесса тестирования;
* Анализатор – осуществляет поиск шаблонов проектирования, типичных ошибок в построении архитектуры, формирует отчет и рекомендации наоснове полученных данных;
* Тестер – производит тестирование внутреннего представления на предмет работоспособности архитектуры и соответствия предъявляемым требованиям. В результате работы тестер выдает однозначный ответ о пригодности прототипа;
* Кодогенератор – позволяет получить исходный код внутреннего представления на целевом языке.

В некотором смысле, можно считать модель рассмотренной системы моделью оптимизирующего компилятора, включающего дополнительную стадию тестирования и осуществляющего трансляцию описания прототипов с унифицированного исходного языка в целевой для рассматриваемой платформы.

Автором было проведено исследование современных решений в области прототипирования программного обеспечения, оценка их эффективности и применимости согласно выдвинутой модели требований, а также выводы о необходимости появления нового класса инструментов прототипирования, в виду неготовности существующих решений удовлетворять ранее выдвинутым требованиям. Кроме того, в работе детально представлена предлагаемая автором модель системы описания и тестирования прототипов архитектуры программного обеспечения и рассмотрена ее реализация с точки зрения современных технологий программирования.

Список литературы

1. Макконелл С. Совершенный код. Мастер-класс. Практическое руководство по разработке программного обеспечения / С. Макконелл. – Издательство «Русская редакция», 2011. – 896 с: ил.
2. Хант Э. Программист-прагматик. Путь от подмастерья к мастеру / Э. Хант, Д. Томас - СПб.: Питер, 2010. – 830 с.
3. Ахо А. Компиляторы: принципы, технологии и инструментарий, 2-е издание / А. Ахо, М. Лам, Р. Сети, Д. Уильман – ООО «И. Д. Вильямс», 2011. – 1184 с: ил.