На правах рукописи

Кручинин Алексей Николаевич

АВТОМАТИЧЕСКАЯ ГЕНЕРАЦИЯ ПРОГРАММНЫХ КОМПОНЕНТ ПО ВЫСОКОУРОВНЕВЫМ СПЕЦИФИКАЦИЯМ

Специальность 05.13.11 – математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей

А В Т О Р Е Ф Е Р А Т
диссертации на соискание ученой степени
кандидата технических наук

Ростов-на-Дону
2006 г.

Работа выполнена в Южно-Российском региональном центре информатизации Ростовского государственного университета.

Научный руководитель: кандидат технических наук, доцент,
старший научный сотрудник
Литвиненко Александр Николаевич

Официальные оппоненты: доктор физико-математических наук, профессор
Горбунов-Посадов Михаил Михайлович

доктор технических наук, профессор
Божич Владимир Иванович

Ведущая организация: Ростовский государственный
строительный университет

Защита состоится «30» ноября 2006 г. в 11 часов на заседании диссертационного совета К.212.208.04 по физико-математическим и техническим наукам в Ростовском Государственном Университете по адресу:
344090, г. Ростов-на-Дону, пр. Стачки 200/1, корпус 2, ЮГИНФО РГУ, к. 206.

С диссертацией можно ознакомиться в научной библиотеке РГУ по адресу: г. Ростов-на-Дону, ул. Пушкинская, 148.

Автореферат разослан «____» _____2006 г.

Ученый секретарь
диссертационного совета,
кандидат физико-математических наук Муратова Г. В.

ОБЩАЯ ХАРАКТЕРИСТИКА РАБОТЫ

Актуальность темы.

Развитие программного кода, связанное с процессом сопровождения программной системы, вследствие высокой конкуренции на рынке программного обеспечения, должно происходить оперативно, и при этом, не в ущерб работоспособности этой системы. По оценке многих специалистов поддержка программного продукта может занимать до 70% всего времени в рамках его жизненного цикла. В силу многих факторов, на этапе проектирования системы вопросы сопровождения часто не затрагиваются, что приводит к появлению сложного «наслоенного» кода, который со временем становится неочевидным и чрезвычайно трудным для понимания и модификаций.

В работе Дейкстры[1] был предложен метод «разделения ответственностей» (Separation of Concerns), согласно которому четко локализованный код гораздо предпочтительнее, чем функционально-нелокализованный, так как последний сложнее для обозрения и понимания. Дейкстра заключал, что локализованный код проще поддается отладке и повторному использованию, и, следовательно, такой код удобнее поддерживать. Дейкстра указал на необходимость локализации кода, но им не были сформулированы принципы декомпозиции. Структурный и объектно-ориентированный подходы предлагают способы достижения поставленной цели, однако, далеко не всегда успешные. Несмотря на глубокие различия, они имеют общую черту — использование функциональной декомпозиции на всех этапах жизненного цикла системы. При этом многие требования, которые должны найти отражение в программном коде, не могут быть четко реализованы в терминах функциональной декомпозиции по причине того, что они относятся к функционированию всей системы в целом, а не к ее отдельным компонентам. Обычно решение этой проблемы состоит в ручном «запутывании» кода с целью повышения эффективности системы, в ущерб четкой локализации кода[2]. Необходим отдельный уровень декомпозиции для четкого их обозначения и реализации.

Одна из главных проблем, которой в работе уделяется особое внимание — это проблема рассредоточенности кода, известная по работам А.Л. Фуксмана, как проблема «сосредоточенного описания рассредоточенных действий», или

---

[1] E.W. Dijkstra, A Discipine of Programming. Prentice Hall, Englewood Cliffs, 1976.
[2] Иськив А.И., Иванова И.В., Использование аспектной декомпозиции при создании информационных систем, МГТУ, 2000.

проблема «сквозной функциональности» в терминологии аспектно-ориентированного подхода.

Существует несколько подходов для достижения приемлемой локализации кода. Особенно значимые успехи были получены в рамках аспектно-ориентированного подхода, претендующего на решение проблемы рассредоточенности кода. Однако в действительности, аспектная ориентированность является расширением объектно-ориентированного подхода, что не позволяет перенести его идеи на императивные и декларативные языки, а также на произвольные текстовые файлы. Аспектная декомпозиция позволяет разбить программу на модули-аспекты, где каждый аспект — это функционал системы, влияющий на нее, как на целое, и пересекающий ее. Невыделенные явно аспекты тесно переплетаются с кодом программы.

Таким образом, исследование в области поиска методов локализации кода является актуальным. Новые методы должны позволить разделить пересекающуюся логику, что способствовало бы упрощению сопровождения программного обеспечения.

В данной работе предлагается решение проблемы реализации сквозной функциональности, позволяющее сократить стоимость внесения изменений в программный код. Работа является продолжением и развитием работ А.Л. Фуксмана, а также идей о рассредоточенном коде и о вертикальном слоении программы.

Областью исследования диссертационной работы является аспектная декомпозиция кода и его сопровождение, а также принципы и подходы, которые лежат в основе модульного программирования.

Предметом исследования диссертационной работы выступает такая структуризация программного кода, которая учитывает возможные и наиболее вероятные его модификации.

Цель и задачи работы.

Целью данной диссертационной работы является разработка методов структуризации программного кода, которые бы способствовали повышению качества сопровождения разрабатываемой системы.

Для достижения поставленной цели решаются следующие **задачи**:

- разработка методов применения аспектно-ориентированной парадигмы, не ограниченных исключительно объектно-ориентированными языками, путем разрешения следующих задач:
  - расширение области применимости точек связывания на не объектно-ориентированные языки;

    - расширение области применимости аспектов на не объектно-ориентированные языки;
- разработка подхода для расширения масштабов повторного использования аспектного кода;
- разработка способа автоматической генерации программного кода по высокоуровневым спецификациям;
- реализация метода автоматической генерации программного кода по высокоуровневым спецификациям на основе предлагаемой архитектуры.

Научная новизна.

1. Предложена общая классификация точек связывания, разработан новый вид точек связывания «прямые точки связывания» и определено его место в предложенном классе;
2. разработан статический подход «вплетения кода», основанный на прямых точках связывания;
3. использована параметризация модуля изменений методом морфологического анализа для расширения области его применимости;
4. разработан «метод цепочек спецификаций», где в качестве звена цепи используется спецификация, что позволило:
    - использовать вертикальные и горизонтальные преобразования над спецификациями;
    - выражать аспекты звеньями цепи;
5. разработан метод автоматической генерации программного кода по высокоуровневым спецификациям и метод его безболезненной инсталляции;
6. разработан и реализован программный комплекс «XpectEngine» для преобразования высокоуровневого аспектного кода к низкоуровневому коду XML, HTML, JavaScript, ActionScript, а также к тексту документаций.

Практическая ценность.

Результаты проведенных исследований были использованы при создании программного комплекса «XpectEngine» в ФГНУ НИИ «Спецвузавтоматика», который был зарегистрирован и используется в настоящее время в Министерстве экономики, торговли, внешних международных внешнеэкономических связей Ростовской области, а также для поддержки сетевого портала администрации города Батайска, о чем имеются справка и акт о внедрении.

Апробация результатов работы.

Результаты работы докладывались и обсуждались на следующих научных конференциях:

- международная конференция «Технологии Microsoft в теории и практике программирования» (г. Москва., МГУ им. Ломоносова, 2004);
- международная конференция «Ломоносов-2004» (г. Москва., МГУ им. Ломоносова, 2004);
- XI Всероссийская научно-методическая конференция «Телематика-2004» (г. Санкт-Петербург, ИТМО, 2004);
- V всероссийская научно-техническая конференция «Теоретические и прикладные вопросы современных информационных технологий» (г. Улан-Удэ, ВСГТУ, 2004);
- X международная открытая научная конференция «Современные проблемы информатизации в технике и технологиях» — 3 доклада (г. Воронеж, ВГТУ, 2005);
- XI международная открытая научная конференция «Современные проблемы информатизации в моделировании и программировании» — 3 доклада (г. Воронеж, ВГТУ, 2006).

Публикации. По теме диссертационной работы опубликовано 14 печатных работ: 1 статья в издании, рекомендуемом ВАК; 1 статья в электронном журнале; свидетельство об официальной регистрации программы для ЭВМ; 11 статей в сборниках трудов и сборниках тезисов конференций.

Структура и объем диссертации. Диссертационная работа состоит из введения, четырех глав, заключения, списка литературы из 70 наименований. Основной текст диссертационной работы изложен на 130 машинописных страницах, включая 12 рисунков и 3 таблицы.

Диссертационная работа выполнена под руководством кандидата технических наук, доцента кафедры Информатики и вычислительного эксперимента Ростовского государственного университета А.Н. Литвиненко, которому автор выражает глубокую признательность.

ОСНОВНОЕ СОДЕРЖАНИЕ РАБОТЫ

Во введении обосновывается актуальность темы диссертации, формулируется цель исследований, показывается научная новизна и практическая значимость полученных результатов.

Первая глава диссертации состоит из четырех частей. В ней дается обзор существующих технологий программирования в части методов, позволяющих повысить качество сопровождения кода.

Цель главы в рассмотрении существующих подходов к сопровождению кода: выяснение недостатков и установление общих принципов в различных подходах. Также рассматриваются виды спецификаций и их применение при разработке программного обеспечения.

В первой части главы рассматривается принцип разделения ответственностей и компонентное программирование, как основа построения подхода для эффективного сопровождения кода.

Принцип разделения ответственностей (Separation of Concerns) позволяет организовать сокрытие сложности и расширить границы повторного использования благодаря абстракциям, таким как модуль, функция или метод. Данный принцип позволяет также осуществлять поддержку кода и раздельное тестирование, он лежит в основе большинства существующих в настоящее время подходов к проектированию программных систем.

Главный субъект *компонентного программирования* — «программная компонента» — абстракция, вбирающая в себя естественное представление об объекте, который должен быть добавлен или изъят из системы без нарушения работоспособности приложения. Компоненты строятся из принципов взаимозаменяемости и многократной возможности их повторного использования. Примером здесь может служить технология COM корпорации Microsoft. Если компоненты являются результатом автоматизированной генерации, то говорят о *порождающем программировании*, идея которого раскрывается принципом «лучше сгенерировать необходимые компоненты, чем искать их»[1].

Во второй части главы рассматриваются работы в области *расширяемого программирования* (extensible programming). Согласно Никлаусу Вирту «расширяемое программирование подразумевает, что можно добавлять модуль без необходимости вносить какие-либо изменения в существующие модули»[2].

---

[1] Чернецки К., Айзенекер У. Порождающее программирование: методы, инструменты, применение. Для профессионалов. – СПб.: Питер, 2005. – 731 с.

[2] Вирт Н. Язык программирования Оберон (пересмотренное сообщение) // Системная информатика. — Новосибирск: Наука, 1991. — Вып.1. — С. 40–63.

Технология вертикальных слоев А.Л. Фуксмана основана на расширяющих функциях с целью сосредоточения описаний рассредоточенных действий (фрагментов кода, находящихся в разных частях программы). Расширяющая функция — это логически упорядоченная совокупность фрагментов кода. Основная идея подхода — разделение программы на горизонтальные слои (фундаментальные, редко изменяемые) и вертикальные (подверженные пересмотру и модификациям). Цель подхода — облегчение модернизации системы по мере ее эксплуатации.

Расширяемые программы М.М. Горбунова-Посадова основаны на каркасном подходе, а также поддержке однородных наборов и пространствах однородности для единообразной обработки фрагментов кода. Цель подхода — создание средств безболезненного (т.е. не нарушающего работоспособности) развития программы. Основной механизм подхода — пополнение множества сменных модулей, размещаемых в гнездах каркаса.

В третьей части первой главы рассматривается аспектно-ориентированный подход (АОП) — технология, позволяющая собирать программы из взаимно пересекающихся блоков (crosscutting concerns). В рамках АОП программа понимается как набор модулей, каждый из которых отвечает за определенный аспект — мотив или критерий, вбирающий в себя соответствующую функциональность.

Аспектно-ориентированный подход является инструментальным средством, поддерживающим более чем одно функциональное измерение или отдельную функциональную сторону разрабатываемой системы.

АОП стало ответом на вопрос о сопровождении кода, в котором тесно переплетены отдельные аспекты или грани системы. В рамках подхода говорят о разрешении проблемы «сквозной функциональности» — четкое разделение системной и предметной области задачи.

Коротко рассматриваются 3 методики аспектной декомпозиции: субъектно-ориентированное программирование, фильтры композиции, адаптивное или деметерево программирование. Приводятся объекты АОП: точка присоединения (JoinPoint) — однозначно-определенная точка вставки в программе; срез (PointCut) — набор точек присоединения, задаваемых правилом; инструкция (Advice) — тройка срез, конструкция времени исполнения и фрагмент кода, предназначенный для вставки; представление (Introduction) — правило изменения структуры класса; аспект (Aspect) — совокупность срезов и инструкций.

Аспектно-ориентированному проектированию и программированию в работе уделяется особое внимание, поскольку данная парадигма вобрала в себя идеи основных подходов в области поддержки и сопровождения кода. Существу-

ет множество реализаций АОП[1], сводящихся к построению генератора вплетения аспектов в добавляемые компоненты, однако все эти реализации имеют недостатки, главным из которых является жесткая привязанность к конкретным объектно-ориентированным языкам, их семантическим и синтаксическим особенностям.

Вторым недостатком АОП является тот факт, что однажды разработанный аспект, как новая абстракция, применяемая в программе, очень часто слабо переносим или зачастую вообще не переносим в другие проекты. Таким образом, компонентная составляющая парадигмы соблюдена в настоящее время недостаточно.

В четвертой части главы рассказывается о роли спецификаций в построение гибкого сопровождаемого приложения. Рассматриваются отдельные виды спецификаций (UML, TDN, GDN) и способы их выработки. Отдельно рассматриваются функциональные спецификации.

Развитие инструментов построения программного обеспечения все больше происходит по двум направлениям: эволюционирование структур данных (*структуризация*), с целью как можно точнее и ближе к реальному миру описать предметную область при помощи выбранных инструментов, и *абстрагирование* — выработка таких моделей управления системой, которые бы позволили с минимальными усилиями достигнуть необходимого результата.

Абстракции позволяют выделять существенные черты системы, игнорируя ее подробности. Таким образом, абстракция есть частный случай разделения задач, в рамках которого проблема анализа важных аспектов отделяется от проблемы рассмотрения менее значимых деталей[2].

Способ определения аспектов в аспектно-ориентированных системах — это лишь первый этап структуризации. Вторым важным этапом в построении эффективно-сопровождаемой системы является описание способов управления и интеграции аспектов. Подобные описания представляют собой функциональные спецификации, основными чертами которых являются качество, строгость и полнота.

Выводы первой главы заключаются в необходимости разрешения указанных недостатков аспектно-ориентированного подхода с целью построения метода структуризации, который бы позволил организовывать сквозную функциональность как в объектно-ориентированных, так и в не объектно-ориентированных системах. Указывается также необходимость выработки эф-

[1] наиболее известная реализация АОП — AspectJ компании Xerox.
[2] Гецци К., Джазайери М., Мандриоли Д. Основы инженерии программного обеспечения. 2-е изд. / Пер. с англ. – СПб.: БХВ-Петербург, 2005. – 832 с.

фективного способа построения спецификаций с целью формализации, увеличения уровня абстракции и усиления строгости описания функционирования системы.

Вторая глава диссертации состоит из трех частей и посвящена описанию нового метода реализации аспектно-ориентированного подхода. Приводится последовательное построение метода. Подробно рассматриваются объекты метода, а также приводится их классификация и сравнение с уже существующими объектами в других подходах.

Цель главы — разработка общего метода применения аспектно-ориентированной парадигмы в объектно- и не объектно-ориентированных языках.

В первой части главы раскрывается понятие «рассредоточенный код» и устанавливаются фундаментальные причины его появления.

Рассредоточенный код — результат структурной декомпозиции программы, при котором фрагменты кода, отвечающие за отдельный функционал, не образуют непрерывный текст.

В работе впервые предлагается разделение рассредоточенного кода на три основных подвида: запутанный (code-tangling), переплетенный (code-weaving) и спутанный код (obfuscating-code).

Запутанный код (code-tangling) — рассредоточенный код, появляющийся в результате выбора парадигмы программирования (объективный фактор).

Спутанный код (obfuscating-code) — рассредоточенный код, появляющийся, как следствие многих факторов, включая сложную оптимизацию кода, выбранную модульность, а также непрофессиональное проектирование и неоправданную декомпозицию (субъективный фактор).

Переплетенный код (code-weaving) — рассредоточенный код, получаемый в результате работы генератора вплетения.

Организация работы с рассредоточенностью (структуризация и упорядочивание кода) — важнейшая цель на пути построения нового метода аспектной декомпозиции.

Во второй части главы разрабатывается метод аспектной декомпозиции; раскрывается понятие «точка присоединения» и проводится их полная классификация по способу определения. Раскрывается сущность фактора транзакционности и его роль в генерации программного кода.

Общей чертой большинства подходов в области сопровождения кода является выработка инструментов консолидации рассредоточенных фрагментов логики в единую систему. Такой процесс называют вплетением функционала (code-

weaving), он базируется на так называемых точках присоединения (join point) — специальном инструменте связи кода модулей и кода программы.

В работе вводится разделение методов указания точек связывания на «статические» и «динамические» в зависимости от момента генерации переплетенного кода: статические применяются во время препроцессорной обработки (т.е. до этапа компиляции), тогда как динамические — во время выполнения программы («на-лету»).

К статическим методам были отнесены следующие:

1. метод координат — точка связывания задается номером и позицией в строке;
2. предлагаемый в данной работе «метод прямой инсталляции», в рамках которого точка связывания указывается в коде явно в виде комментария специального формата или неявно в зависимости от языка и программы;
3. метод сигнатур, предложенный Г. Кишалесом для определения среза.

К динамическим были отнесены следующие:

1. событийный подход[1], основанный на определении событий, возбудителях событий, а также их обработчиках, которые предварительно регистрируются в специальной «таблице регистрации»;
2. метод сигнатур для определения среза в процессе выполнения программы.

Предложенный в диссертационной работе «метод прямой инсталляции» основан на определении и оформлении в коде программы специальных меток — точек вставки, задаваемых комментариями, определяемыми синтаксисом соответствующего языка. Предложено дополнять данные комментарии специфическими конструкциями для их различения от общепринятых комментариев. Например, в языке C такого рода комментарий может иметь вид (1), в HTML — вид (2).

```
/*!{name}*/        (1),
<!--!{name}-->     (2).
```

Идентификатор `name` позволяет генератору, производящему инсталляцию кода в точки вставок на этапе препроцессирования, обнаруживать определенные метки с целью инсталляции связанных с ними фрагментов. Кроме того, в зависимости от контекста языка и программы, некоторые программные конструкции могут быть определены как точки вставок неявно. Например, начало определения блока `<head>` (первая строка после `<head>`), и его завершение (строка пе-

[1] Салтыкова Н.Н. Разработка метода проектирования модифицируемых СУБД-приложений: Дисс. на соискание ученой степени канд. тех. наук. - Ростов-на-Дону, 2002 г. – 174 с.

ред </head>) — это неявные прямые точки вставок с соответствующими предопределенными идентификаторами (head_start и head_end).

Основное отличие метода прямой инсталляции от метода сигнатур заключается в том, что в первом случае точки вставки могут не зависеть от контекста языковых конструкций и могут быть определены в не объектно-ориентированном коде. Метод сигнатур позволяет работать исключительно с методами класса, поскольку в настоящее время жестко привязан к объектно-ориентированному подходу.

Метод построения аспектного кода основан на следующих этапах:

- выделение ядра и определение точек связывания;
- выделение рассредоточенного кода и его оформление в виде модулей изменений, а также оформление прямых точек связывания, выявленных на первом этапе;
- параметризация фрагментов модуля изменений.

Первым этапом является выделение в существующем коде так называемого ядра или каркаса — той части кода, которая не подвергается изменениям и носит постоянный характер.

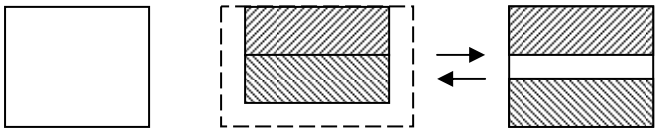
Второй этап заключается в выявлении рассредоточенного кода. По смыслу полученный код должен определять отдельный функционал системы, ее отдельный аспект.

Модуль изменений — это модуль, содержащий фрагменты кода, и точки его привязки к первичному коду программы. Понятия «модуль изменений» и «аспект» являются схожими, однако имеется одно важное отличие, заключающееся в том, что модуль изменений может содержать фрагменты кода, вообще говоря, несвязанные общей логикой. Под аспектом же понимают логически-связанную совокупность фрагментов кода, относящихся к некоторому функционалу программы, и характеристику времени и места их привязки. Таким образом, в архитектуре излагаемого метода аспект выражает абстракцию более высокого уровня, чем модуль изменений.

Установление ядра и выявление модуля изменений, как правило, проходят параллельно. Совокупность фрагментов, полученных на данных этапах, образуют необходимый текст программы. При этом очевидно, что код программы может быть разделен на модифицирующиеся и постоянные части неединственным образом. Основное отличие первых двух этапов состоит не в способе разделения, т.е. не в отнесении текущего фрагмента кода, например, к ядру или к модулю изменений, а в установлении причины, по которой необходимо осуществить имен-

но такое разделение. Таким образом, важны мотивы, которые устанавливают свойства кода для его разделения.

Применение программы-генератора для осуществления вплетения кода к паре «ядро кода, модуль изменений» влечет инсталляцию фрагментов в ядро в точках связывания (рис. 1).



Исходный код Модуль изменений Переплетенный код

рис. 1.

Существуют типичные задачи, которые могут встречаться во множестве программ. У таких задач входная информация имеет сложно-структурированный характер и не может быть выражена отдельными параметрами. Почти всегда реализация подобных задач приводит к возникновению запутанного кода, например, задача добавления многопользовательской работы к системе способна породить лавину изменений в существующем коде. Решение той же задачи в похожей программе может мало отличаться от реализации первой задачи, однако в то же время иметь некоторые существенные отличия, характеризующие уникальность конкретной задачи.

Модуль изменений разрешает вопрос упорядочивания и объединения рассредоточенного кода с целью дальнейшего его вплетения генератором, однако его статичность не способна разрешить задачу интеграции пусть близкого, но отличающегося кода.

Третьим этапом является параметризация фрагментов модуля изменений, что позволяет перейти от конкретной задачи к целому классу подобных задач.

В качестве одного из методов для параметризации предлагается методов морфологического анализа[1], суть которого заключается в том, что сначала определяется пространство поиска, которое обязательно должно включать в себя искомое решение, а затем строится его сужение путем варьирования параметров. Применение морфологического анализа позволяет выявить существенные параметры для определенных классов задач, что в свою очередь помогает установить возможные места вариаций в каждом отдельном фрагменте кода.

Установленные параметры ложатся в основу прототипа для разработки модели данных. В свою очередь прототип представляет собой отражение структуры данных текущего фрагмента кода, что способствует продвижению внутренних

[1] Одрин В.М. Метод морфологического анализа технических систем. М.: ВНИИПИ, 1989.

стандартов системы, которые устанавливаются регулярными правилами. *Регулярные правила* — это соглашения, которые вводит разработчик, часто во время проектирования, и которых он придерживается во время всего жизненного цикла системы.

Таким образом, разработчик получает возможность гибкой работы с частью возможностей некоторой системы, управляя существенными действенными параметрами, скрывая от себя редко используемые. Заполнение прототипа данными или фиксация кода, осуществляемая генератором, приводит к решению конкретной задачи.

На данном этапе построения метода, выработанные классы параметров отдельных фрагментов кода не связаны, т.е. генератор, оперирующий модулем изменений, устанавливает значения параметров в рамках отдельных фрагментов, не согласуясь с параметрами других фрагментов. Таким образом, выделенные и объединенные общей логикой, параметризованные и собранные в модуле изменений фрагменты кода на данном этапе не образуют аспект, поскольку не существует возможности централизованного управления распределенным во фрагментах кода функционалом. Данная задача разрешается в главе 3 путем построения формальных XML спецификаций.

В третьей части главы раскрывается понятие фактора транзакционности и устанавливаются его свойства.

Переход программы из одного рабочего состояния в последующее представляет собой т.н. цепочку развития (рисунок 2). Во время таких переходов модификации, вносимые в программу, должны отвечать принципу, который также позволял бы переводить систему из текущего состояния в предыдущее, получая деградированную систему. Таким образом, любое изменение в цепочке состояний системы носит характер *транзакционности* и обладает всеми свойствами, описанными в модели ACID (Atomicity, Consistence, Isolation, Durability).

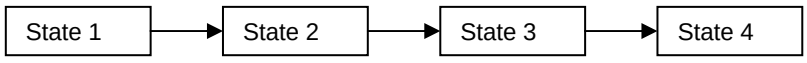


рис. 2.

Рассмотрение фактора транзакционности важно, поскольку при построении генератора вплетения необходимо учитывать, что возможны именно одновременная инсталляция или одновременное извлечение всех фрагментов кода модуля изменений, в противном случае, очевидно, работоспособность программы может быть нарушена.

Выводы второй главы. Разработан новый статический «метод прямой инсталляции», регулирующий правила определения точек связывания. Рассмотре-

ны первые три этапа метода автоматической генерации программного кода по высокоуровневым спецификациям. Раскрывается значимость применения метода морфологического анализа для выявления значимых родовых параметров, которые используются в построении регулярных правил.

В заключении рассмотрена важность поддержки фактора транзакционности при построении генератора вплетения.

В третьей главе «Применение XML спецификаций для моделирования сложно-структурированных программных объектов» излагается процесс проектирования функциональных XML спецификаций для упрощения сопровождения системы и увеличения масштабов ее повторного использования. Разрабатывается метод цепочек спецификаций.

Цель главы в построении дополнительных этапов к методу аспектной декомпозиции, которые включают работу с высокоуровневыми спецификациями. Применение спецификаций позволяет упростить сопровождение системы и увеличить масштабы ее повторного использования. Значения выявленных параметров отдельных фрагментов кода, относящихся к модулю изменений, задаются, согласно предлагаемому методу, централизованно с помощью спецификации аспектов.

В третьей главе также рассматривается процесс проектирования функциональных XML спецификаций и разрабатывается метод цепочек спецификаций.

В первой части главы формализуются объекты сопровождаемых систем; устанавливается их разнородный характер. Раскрывается принцип построения спецификации аспекта.

Необходимо стремиться оформлять отдельные фрагменты кода взаимозаменяемыми, предполагая их дальнейшее повторное использование. Аналогичной цели следует придерживаться во время построения модуля изменений. Модуль изменений — это сложно-структурированный объект системы, вбирающий в себя многие свойства, такие как логику, которая пересекает систему, регулярные зафиксированные разработчиком правила, выраженные параметрами встроенных фрагментов, а также точки присоединения.

Отдельные прототипы фрагментов кода служат для построения общего прототипа модуля изменений. Данный прототип и является спецификацией модуля изменений. Формальная спецификация модуля изменений называется спецификацией аспекта, если выполняются два условия: во-первых, если спецификация содержит информацию о точках присоединения к исходному коду, и во-вторых, если спецификации достаточно для восстановления всех фрагментов кода модуля изменений.

Спецификация аспекта едина для всех составляющих модуля изменений, при этом одни и те же параметры могут использоваться в различных фрагментах кода. Основная цель спецификации аспекта — сосредоточенное описания сложно-структурированных объектов, которые при добавлении в исходный код программы, могут быть выражены рассредоточенным кодом.

В качестве одного из методов построения прототипа спецификации предлагается использовать морфологический анализ для выделения и обобщения параметров модуля изменений. Некоторые параметры могут быть зависимыми, подобные зависимости могут относиться к регулярным конструкциям разработчика и фиксируются в прототипе.

В качестве инструментов построения прототипа модуля изменений было предложено использовать технологии XML и XML Schemas, а также DTD. Выбор обосновывается удобством и функциональностью проверки спецификаций на соответствие схемам, а также тем, что XML позволяет моделировать сколь угодно сложные структуры благодаря возможности декларативно, с высокой долей наглядности описывать вложенность и взаимодействие.

Во второй части разрабатывается метод цепочек спецификаций.

Вначале формализуется цепочечный подход и указываются его недостатки. В классическом цепочечном подходе реализуется стандартный принцип IPO (Input - Process - Output) — вход–обработка–выход; в задачи каждого звена цепи (модуля) входит выполнение ряда регламентированных операций для реализации каждой отдельной функции в полном составе, достаточных для завершения начатой обработки. Модули взаимодействуют между собой посредством передачи данных, при этом данные являются входными для одного модуля и выходными для другого.

В работе предложена модификация метода, основанная на применении в качестве звеньев цепи спецификаций (рисунок 3), а также промежуточных генераторов преобразований для переходов между звеньями, что позволяет:

- внедрять новые рассредоточенные функции с помощью звеньев цепи;
- применяя регулярные правила для описания сложно-структурированных объектов программ и переходя от менее абстрактных моделей объектов к более общим (вертикальные преобразования), изменять уровень абстракции входных данных;
- варьировать значения свойств отдельной части системы, при этом изменение параметров может отвечать управлению частью логики в рамках единого функционала. Преобразование спецификаций по цепи позволяет так-

же трансформировать структуру спецификации, что отвечает горизонтальным преобразованиям.

Поскольку в качестве звеньев цепи предложено использовать XML–спецификации, для реализации промежуточных генераторов естественно применять технологию XSLT.

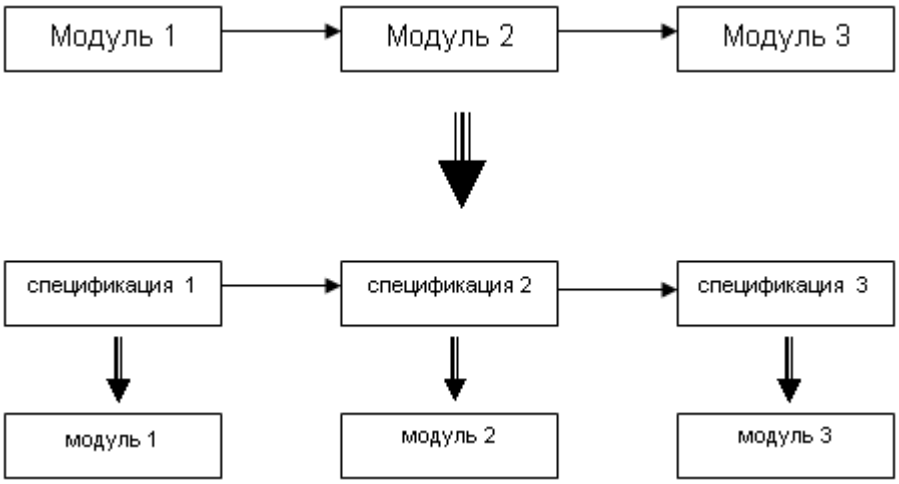


рис. 3.

Также инструкциями для генератора могут являться и вспомогательные объекты данных, используемые для пополнения исходных XML спецификаций.

Метод цепочек спецификаций позволил решить следующие задачи:

- увеличить масштабы повторного использования разрабатываемого программного кода;
- минимизировать затраты на разработку, благодаря возможности использования вертикальных и горизонтальных преобразований.

В третьей части главы раскрывается метод аспектной декомпозиции, а также его сопряжение с методом цепочек спецификаций.

В работе предлагается следующий метод генерации кода. На вход генератора вплетения подаются исходный код программы, размеченный точками присоединения, спецификации аспектов и промежуточные генераторы для поддержки цепочек спецификаций. Результатом работы системы является сгенерированный и вплетенный в систему код. Схема метода генерации кода (рис. 4) представлена следующими объектами:

1. цепочки, организуемые промежуточными генераторами, для осуществления трансформаций спецификаций;

2. исходный код с определенными точками связывания — местами будущего расширения программы;

3. «генератор вплетения» фрагментов кода, которые определяются на основе спецификации аспекта.

Цель генератора – порождение рассредоточенного переплетенного конечного кода.

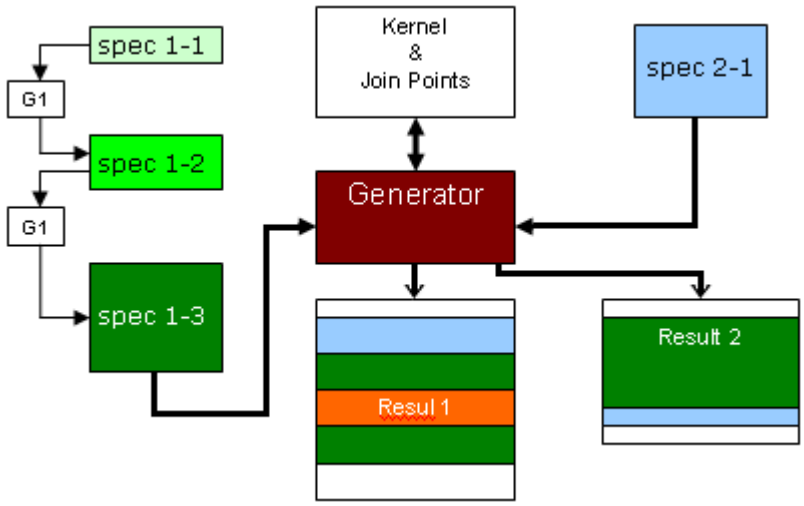


рис. 4.

Выводы третьей главы. Построен метод автоматической генерации программного кода, который связан с предложенным в главе методом цепочек спецификаций. Приведена схема метода.

В четвертой главе приведены оценки эффективности и сложности применения разработанного метода. Рассматривается реализация метода автоматической генерации кода на примере реализованного программного комплекса «XpectEngine». Указаны используемые программные средства и обосновывается их выбор. Определяется роль языков сценариев в реализации метода автоматической генерации кода по высокоуровневым спецификациям.

Цель главы — реализация метода автоматической генерации программного кода по высокоуровневым спецификациям на основе предлагаемой архитектуры и применение разработанных методов в реальных проектах.

В первой части главы рассматривается практическое применение метода автоматической генерации компонент на примере программного комплекса «XpectEngine». Приводятся используемые программные средства и обосновывается их выбор. Определяется роль языков сценариев в реализации метода автоматической генерации кода.

Разработанный программный комплекс «XpectEngine» предназначен для решения задач построения, расширения, сужения WWW-приложений, а также всевозможных текстов исходных кодов программ, путем преобразования высокоуровневого аспектного кода к низкоуровневому коду XML, HTML, JavaScript, ActionScript, а также к тексту документаций.

Компоненты расширения задаются на основе набора XML–спецификаций. Настраиваемые с помощью параметров компоненты безболезненно для окруже-

ния подключаются к исходному коду на основе построенного расширения аспектно-ориентированного подхода. В частности, применение инструмента позволило перейти от низкоуровневого разнородного кода (HTML, JavaScript, ActionScript) к модели WWW-приложения, что дало высокоуровневое его управление. Программный комплекс оказался особенно полезным там, где требования по качеству разработки приложений были высоки, а количество времени сильно ограничено.

Программный комплекс «XpectEngine» был разработан на JScript 5.x (Microsoft Windows Scripting Host — WSH) с применением Microsoft XML Core Services (MSXML) 4.0.

Выбор JScript 5.x Microsoft Windows Scripting Host (WSH) обосновывается тем, что язык сценариев, в силу интерпретируемости, позволяет «склеивать» отдельные существующие модули без необходимости многократной компиляции кода. Также используется возможность генерации кода «на лету» — модификация программы во время ее исполнения или непосредственно перед запуском.

Во второй части главы приводятся оценки эффективности и сложности реализаций различных задач при использовании методов и подходов, разработанных в данной диссертационной работе.

Сложность программы зависит от многих факторов: размера программы, способа структуризации, внутренних и внешних связей между модулями. Для измерения сложностей используют так называемые «метрики», численные значения сложности, устанавливаемые по различным критериям. Принято считать, что существуют 3 рода метрик для оценки сложности программ:

- по размеру программы;
- по сложности потока управления программ;
- по сложности потока данных программ.

Основываясь на данных численных экспериментов разнородных метрик, был сделан вывод о применимости выбранного подхода и его эффективности.

Сравнение оценок по метрике Холстеда (по размеру программы) для оценки усилия программиста при разработке программы на различных задачах показало от 1,3-х до 8-и кратного уменьшения показателя. Так, на «задаче о добавлении авторизации и журнализации в код методов» для объектно-ориентированной реализации коэффициент усилий (Halstead Effort) HEF = 6258. Аспектная реализация[1] при помощи AspectJ — HEF = 1001. Применение методов аспектной де-

[1] В. Павлов, Аспектно-ориентированное программирование // http://www.optim.ru/cs/2003/4/AOP2/AOP.asp

композиции кода, предложенных в данной работе, позволило снизить коэффициент до HEF = 857.

Цикломатическая сложность Маккейба (Cyclomatic Complexity, CC)[1] для оценки сложности потока управления уменьшалась на 9–10%. На «задаче банковских операциях» для объектно-ориентированной реализации коэффициент CC = 7. На реализации предложенным методом CC = 5. Аспектная реализация при помощи AspectJ — CC = 5. Аналогичные результаты получены для метрики Джилба — метрики оценки относительной сложности программ на основе количества операторов IF–THEN–ELSE в коде.

Ввиду предложенного синтаксиса описания прямых точек связывания, оценки метрики уровня комментированности или насыщенности внутреннего описания F=N/S, где N — количество комментариев, S — количество операторов исходного текста, всегда дают положительный прирост.

Таким образом, показано, что применение разработанного метода автоматической генерации кода по высокоуровневым спецификациям на различных задачах позволяет уменьшить сложность кода.

Выводы четвертой главы — предложена реализация метода на примере программного комплекса «XpectEngine». Его внедрение показало значительное сокращение времени внесения изменений в уже существующий код. Приведены оценки сложности кода, построенного с помощью предложенной реализации, в исследованных задачах.

В заключении сформулированы результаты исследований.

ОСНОВНЫЕ РЕЗУЛЬТАТЫ, ВЫНОСИМЫЕ НА ЗАЩИТУ

1. Предложена новая концепция построения программного кода с учетом возможных и наиболее вероятных его модификаций;
2. разработан статический автоматический подход генерации и вплетения программного кода, основанный на применении «прямых точек связываний» и высокоуровневых спецификаций;
3. разработан метод цепочек спецификаций для реализации вертикальных и горизонтальных преобразований спецификаций аспектов;
4. разработан программный комплекс «XpectEngine» для преобразования высокоуровневого аспектного кода к низкоуровневому результирующему коду.

[1] Гецци К., Джазайери М., Мандриоли Д. Основы инженерии программного обеспечения. 2-е изд. / Пер. с англ. – СПб.: БХВ-Петербург, 2005. – 832 с.

СПИСОК РАБОТ, ОПУБЛИКОВАННЫХ ПО ТЕМЕ ДИССЕРТАЦИИ

1. Брагилевский В.Н., Кручинин А.Н. Об одном методе представления графовых структур в XML // Современные проблемы информатизации в моделировании и программировании. XI междунар. науч. конф. Сб. трудов. – Воронеж, Научная книга, 2006. Выпуск №11. – С. 218-219.

2. Кручинин А.Н., Брагилевский В.Н. Классификация методов вплетения функционала // Современные проблемы информатизации в моделировании и программировании. XI междунар. науч. конф. Сб. трудов. – Воронеж, Научная книга, 2006. Выпуск №11. – С. 290-292.

3. Кручинин А.Н., Литвиненко А.Н., Колоколов И.А., Фактор транзакционности в жизненном цикле программы // Современные проблемы информатизации в моделировании и программировании. XI междунар. науч. конф. Сб. трудов. – Воронеж, Научная книга, 2006. Выпуск №11. – С. 235-236.

4. Кручинин А.Н, Литвиненко А.Н., Модифицированный цепочечный подход // Известия ВУЗОВ. Северо-кавказский регион. Естественные науки. – 2005. – № 4, С. 9-14.

5. Малышевский А.П., Литвиненко А.Н., Кручинин А.Н. XML модели объектов приложения при создании и сопровождении расширяемых СУБД приложений // Современные проблемы информатизации в технике и технологиях. X междунар. науч. конф. Сб. трудов. – Воронеж, Научная книга, 2005, Выпуск №10. – С. 229.

6. Кручинин А.Н., Литвиненко А.Н., Малышевский А.П. Языки сценариев, как склеивающие языки (glue languages) // Современные проблемы информатизации в технике и технологиях. X междунар. науч. конф. Сб. трудов. – Воронеж, Научная книга, 2005. Выпуск №10. – С. 253-254.

7. Кручинин А.Н., Литвиненко А.Н., Малышевский А.П. Метод сопровождения программного кода // Современные проблемы информатизации в технике и технологиях. X Междунар. науч. конф. Сб. трудов. – Воронеж, Научная книга, 2005, Выпуск №10. – С. 252-253.

8. Кручинин А.Н., Литвиненко А.Н. Решение задачи сопровождения программного кода // Теоретические и прикладные вопросы современных информационных технологий. Материалы V всеросс. научн.-техн. конф. – Улан-Удэ: ВСГТУ, 2004. Ч.2. – С. 239-241.

9. Кручинин А.Н., Литвиненко А.Н. Метод модификации программного кода на основе компонентного подхода // Сетевой электронный научный журнал. Системотехника. – 2004. – №2, http://systech.miem.edu.ru/2004/n2/Litvinenko.htm

10. Кручинин А.Н., Литвиненко А.Н. Применение цепочек спецификаций в аспектно-ориентированном программировании // Телематика-2004. Сб. трудов XI всеросс. научн.-метод. конф. – СПб.: ГУИТМО, 2004. – С. 151-152.

11. Кручинин А.Н. Метод модификации программного кода с применением цепочек спецификаций // Ломоносов-2004. Материалы междунар. науч. конф. по фундаментальным наукам. – М.: МГУ им. Ломоносова, UNESCO, 2004. – секц. Вычислительная математика и кибернетика. – С. 16.

12. Кручинин А.Н. Метод модификации программного кода на основе компонентного подхода с применением языка сценариев WSH, как склеивающего инструмента. (Пример построения системы для модернизации сайта с использованием XML спецификаций и XSLT трансформаций) // Технологии Microsoft в теории и практике программирования. Сб. тезисов междунар. науч. конф. – М.: МГУ им. Ломоносова, 2004. – С. 22-23.

13. Кручинин А.Н. Модульное проектирование однородных пространств на примере разборщика гипертекстовых структур // Рубикон. Сборник научных работ молодых ученых, 2002. – № 20, С. 96-100.

14. Кручинин А.Н., Колоколов И.А., Чудинов Н.В., Алиев А.Т. Комплекс преобразования высокоуровневого аспектного кода в низкоуровневое текстовое представление в системах электронного документооборота XpectEngine. Версия 1.1. // Свидетельство об официальной регистрации программы для ЭВМ №2006611415/Роспатент – М., 26.04.2006.