В интеграле (2.1) произведем замену переменной, полагая $\tau = |z|^{\frac{1}{b}} t$ и, затем действуя оператором D на левую и правую части равенства, получим

$$(a+b+1)\varphi(z) + b(z\varphi_z' - 3\bar{z}\varphi_{\bar{z}}') = 2\left(\frac{R_2 c^b}{|z|}\right)^{\frac{a+b+1}{b}} f\left(R_2 \frac{z}{|z|}\right) + c^{a+b+1} f^-(c^b z), \tag{2.12}$$

$$|z| > R_1.$$

Сравним (2.12) с ранее полученным

$$(a+b+1)\varphi(z) + b(z\varphi_z' + \bar{z}\varphi_{\bar{z}}') = c^{a+b+1} f^-(c^b z),$$

$$\varphi_{\bar{z}} = -\frac{1}{2b}\frac{z}{|z|^2}\left(\frac{R_2}{|z|}\right)^{\frac{a+b+1}{b}} f\left(R_2 \frac{z}{|z|}\right), \ \forall z : |z| \geq R_1.$$

Здесь точка $R_2 \cdot \frac{z}{|z|} \in \gamma$, при $\forall z : |z| \geq R_1.$, поэтому если $f(\eta) \in C_\alpha(\gamma)$, то $\varphi_{\bar{z}} \equiv \frac{\partial}{\partial \bar{z}} I_{a,b} F_{a,b} f \in C_\alpha(E)$.-

$u_i = c^b z_i$, $i = 1,2$.

Пусть $\lambda(z) = I_{a,b}\varphi(z)$, тогда согласно Мусхелишвили [1, стр.71]

$$|\lambda(u_1) - \lambda(u_2)| \leq H |u_1 - u_2|^{\lambda} = H c^{b\alpha} |z_1 - z_2|^{\alpha} \equiv H_1 |z_1 - z_2|^{\alpha} . \tag{2.13}$$

Из (2.13) следует, что функция

$$\lambda(z) = I_{a,b}\varphi(z) \in C_\alpha(E),$$

$0 < \alpha < 1$, где $I_{a,b}\varphi(z) = (a+b+1)F_{a,b}f + bD_1 F_{a,b} f$.

Так как $F_{a,b} f(z) \in C_\alpha(E)$, то и оператор $D_1 F_{a,b} f(z) = z(F_{a,b}f)_z' + \bar{z}(F_{a,b}f)_{\bar{z}}'$ тоже принадлежит $C_\alpha(E)$.

СПИСОК ЛИТЕРАТУРЫ

1. Мусхелишвили Н. И. Сингулярные интегральные уравнения. М., 1968.
2. Гахов Ф. Д. Краевые задачи. М.: Государственное издательство физико-математической литературы, 1963.
3. Векуа И. Н. Обобщенные аналитические функции. М.: Государственное издательство физико-математической литературы, 1959.

АНАЛИЗ СУЩЕСТВУЮЩИХ СРЕДСТВ АДАПТАЦИИ ПРОГРАММ

В. В. ДРОЖДИН, А. М. ВОЛОДИН
Пензенский государственный педагогический университет им. В. Г. Белинского
кафедра прикладной математики и информатики

Характерной особенностью большинства компьютерных систем являются изменяющиеся условия функционирования, поэтому для обеспечения нормального функционирования системы должны приспосабливаться к происходящим изменениям. Например, финансовые и энергосистемы должны сохранять работоспособность при любых сбоях оборудования и защищать себя при любых попытках взлома защиты. Адаптация систем – один из основных путей решения этих проблем.

Адаптивная система – это система, способная приспосабливаться к изменениям внешней среды и внутренней организации путем настройки своих параметров или изменения структуры [3]. Поэтому можно выделить два основных подхода к адаптации программного обеспечения. Первый – параметрическая адаптация, предполагающая изменение и фиксацию параметров, определяющих эффективное поведение системы в определенных условиях. Самый известный пример реализации такого подхода – протокол TCP, чье поведение меняется путем модификации значений, влияющих на управление пакетами передачи и их ретрансляцию в случае перегрузки сети.

Параметрическая адаптация достаточно проста в реализации, т.к. не требует изменения структуры и законов функционирования системы. Существует много способов, позволяющих приложению приспосабливаться к изменяющемся условиям путем настройки своих параметров, следуя при этом одной из имеющихся стратегий. Например, параметрическая адаптация может достигаться путем применения шаблонов проектирования. Так, паттерн «Стратегия» [1] позволяет определить семейство родственных алгоритмов (решающих одну задачу), которые можно повторно использовать в разных контекстах. Например, одни алгоритмы могут требовать больше памяти, другие – больше времени. Специфицировав эти алгоритмы, в процессе исполнения программы можно выбирать подходящую стратегию в зависимости от требований к быстродействию и памяти.

Однако параметрическая адаптация действует в ограниченном диапазоне и не позволяет добавлять новые алгоритмы и компоненты к приложению после завершения его проектирования. Поэтому такие приложения часто оказываются инертными и неспособными к эффективному решению сложных, быстро меняющихся задач. Они требуют много времени на разработку и настройку в соответствии с новыми требованиями пользователей.

Таким образом, используя параметрическую адаптацию, можно менять параметры системы или заставлять приложение следовать какой-либо из имеющихся стратегий, но добавлять новые стратегии или существенно изменять их нельзя. Поэтому на практике параметрическая адаптация оказывается мало эффективной.

Второй способ адаптации – структурная, или композиционная адаптация [2], которая предполагает приспособление системы путем изменения ее структуры. Программы со структурной адаптацией позволяют включать или заменять какие-либо алгоритмические или структурные компоненты программы другими, позволяющими ей оставаться адекватной изменяющимся условиям рабочей среды. Поэтому приложение способно использовать новые алгоритмы для решения задач, которые были неизвестны при его разработке.

Учитывая большие ограничения параметрической адаптации, далее будем рассматривать только методы структурной адаптации.

Структура программной системы определяется задачами, которые она решает, и конструктивными особенностями используемой технологии. Поскольку эти два фактора не могут жестко определить архитектуру, то появляется возможность разных решений. Правильный выбор архитектуры во многом определяет такие важные характеристики системы, как быстродействие, надежность, структурную гибкость (способность системы изменять свою структуру) и др.

Гибкость структуры системы обеспечивает решение трех ее основных задач:

- удовлетворение информационных потребностей различных категорий пользователей;

- приспособление программы к изменениям, происходящим в ходе ее эксплуатации;

- возможность сохранения работоспособности системы при любых сбоях оборудования.

Первая задача часто может быть решена формированием требуемой структуры из готовых программных модулей (компонентов) путем соединения их между собой на этапе разработки системы. Этот процесс можно усовершенствовать путем использования стандартизованных интерфейсов. Интерфейс в объектно-ориентированных языках программирования представляет собой множество требований (сигнатуру функций), предъявляемых к соответствующему классу, реализующему этот интерфейс.

Остальные задачи могут быть решены путем применения различных методов адаптации.

Разработчики программного обеспечения могут использовать различные средства структурной адаптации, которые можно классифицировать на основе того, когда и где осуществляется композиция программ.

Изменение структуры может осуществляться статически и динамически. Методы статической композиции реализуются во время разработки, компиляции или загрузки программы на исполнение, а методы динамической композиции – во время выполнения программы. В зависимости от времени реализации композиции адаптивные программные приложения делятся на фиксированные, специализированные, конфигурируемые, настраиваемые и изменяемые [2]. Первые три типа приложений используют методы статической композиции, а последние два – динамической композиции.

В фиксированных приложениях любое адаптивное поведение жестко прописывается в программе, и его невозможно модифицировать без изменения кода.

Специализируемые приложения реализуют возможности адаптации во время компиляции или сборки за счет конфигурации приложения для конкретной среды. Например, следуя аспектно-ориентированному программированию можно выделить методы решения общих задач (обеспечение отказоустойчивости и защита, журнализация и др.), называемые аспектами. Код, реализующий решение этих задач, разрабатывается независимо от остальных частей программы. Затем с помощью специального компилятора, например AspectJ, можно интегрировать аспекты с бизнес-логикой приложения. Чтобы такое специализированное приложение соответствовало новой среде, его достаточно перекомпилировать.

Аспектно-ориентированное программирование играет важную роль в динамической рекомпозиции, поскольку позволяет изолировать задачи общего характера от остальной части программы. Однако при традиционном аспектно-ориентированном подходе структура скомпилированной программы остается сложной. Для воплощения динамической реорганизации необходимы средства, обеспечивающие разделение задач во время их исполнения.

Конфигурируемые приложения могут использовать методы адаптации во время загрузки компонентов. Например, виртуальная машина Java (JVM) загружает классы только тогда, когда Java-приложение их впервые использует.

Наиболее гибкие подходы к структурной адаптации реализуются в процессе выполнения программы. При динамической композиции программные модули могут изменяться или заменяться во время исполнения. В зависимости от изменения бизнес-логики выделяют два типа приложений.

- Настраиваемое программное обеспечение запрещает модификацию кода программы, отвечающего за бизнес-логику. Оно поддерживает точную настройку реализации общих задач в ответ на изменение условий внешней среды;

- Изменяемое программное обеспечение позволяет вносить изменения в любые функции программы. Поэтому в результате динамической рекомпозиции исполняемая программа может трансформироваться в функционально другую программу.

Структурная адаптация может осуществляться на основе промежуточного программного обеспечения или в самом коде приложения.

Программное обеспечение промежуточного слоя представляет собой набор сервисов, отделяющих приложения от операционной среды и сетевых протоколов. Оно скрывает распределение ресурсов и гетерогенный характер платформы от бизнес-логики приложения. Поэтому именно на этом уровне целесообразно реализовать возможности адаптации, необходимые для выполнения общих задач.

Д. Шмидт выделяет четыре категории программных средств промежуточного слоя [2]:

1. Промежуточные средства базовой инфраструктуры (host-infrastructure middleware) размещаются над операционной системой и предоставляют высокоуровневый интерфейс API, скрывающий разнородность аппаратных средств, операционных систем и, до определенной степени, сетевых протоколов.

2. Промежуточные средства распределения (distribution middleware) предоставляют высокоуровневые программные абстракции (например, удаленные объекты), позволяя разработчикам писать распределенные приложения во многом так же, как и автономные программы. К этому уровню, в частности, относятся CORBA, DCOM и Java RMI.

3. Общие сервисы (common middleware) включают в себя обеспечение отказоустойчивости, безопасности и долговременности, а также транзакций с участием удаленных объектов.

4. Специализированные сервисы (domain-specific middleware) создаются с учетом требований конкретных классов приложений.

Структурная адаптация может быть реализована на одном из уровней промежуточного программного обеспечения. Примером адаптации на уровне средств базовой инфраструктуры является использование виртуальной машины, способной перехватывать и перенаправлять взаимодействия программных компонент. Например, JVM и среда Common Language Runtime (CLR) реализуют динамическую реорганизацию за счет служб отображения, предоставляемых, соответственно, интерпретатором Java и платформой .Net. Этот подход является очень гибким в отношении динамической реорганизации, поскольку позволяет использовать новый программный код во время исполнения программы. Однако использование виртуальной машины может ограничить переносимость приложений.

Наибольшими возможностями адаптации обладает программное обеспечение промежуточного слоя, созданное на основе парадигмы объектно-ориентированного программирования и опирающееся на популярные платформы типа CORBA, Java RMI и DCOM/.Net. Многие из подходов к реализации адаптивного инструментария промежуточного слоя предусматривают перехват и изменение сообщений.

Недостаток средств, опирающихся на программное обеспечение промежуточного слоя, заключается в том, что они применимы только к программам, которые пишутся для конкретной платформы. Эти средства также существенно снижают быстродействие системы. Более высокая адаптация может быть достигнута при использовании методов структурной адаптации непосредственно в самом приложении.

Можно выделить два основных способа реализации этих методов. Первый – создание кода приложения или его части с помощью языка, включающего языковые конструкции обращения к средствам динамической рекомпозиции. Некоторые языки (например, Python) явно предоставляют такую поддержку, а другие – расширяются соответствующим образом. Например, Open Java, R-Java, Handi-Wrap, PCL и Adaptive Java расширяют Java, поскольку включают в себя дополнительные ключевые слова и конструкции, повышающие выразительность адаптивного кода. Однако при создании программы разработчик должен явно включать эти конструкции в текст программы.

Второй способ – интеграция возможностей адаптации в функциональный код. AspectJ и Composition Filters позволяют интегрировать соответствующие возможности в существующие приложения во время компиляции. Инструменты, подобные TRAP/J, используют двухэтапный подход к поддержке динамической рекомпозиции. Сначала модуль интеграции аспектов добавляет обобщенные метки перехвата (в данном случае они реализуются как аспекты) в код приложения во время компиляции, а во время исполнения приложения сборщик динамически включает в него адаптивные компоненты.

Разработано немало механизмов структурной адаптации, и можно предположить, что они будут использоваться все шире по мере того, как программисты будут больше узнавать о методиках адаптивного программного обеспечения, а общество – готовиться к появлению самообучаемых и самоуправляемых компьютерных систем. Структурная адаптация – мощный механизм, но при отсутствии приемлемых инструментальных средств для автоматической генерации и проверки кода ее использование может негативно повлиять на целостность и безопасность системы. Для обеспечения нормального функционирования адаптируемые системы должны отвечать следующим требованиям.

1. Гарантия корректности. Модифицированная программа должна быть не хуже предыдущей. Проек-

тирование рекомпозируемых программ требует такой парадигмы программирования, которая поддерживает автоматическую проверку как функциональных, так и нефункциональных свойств системы [2].

Чтобы гарантировать корректность адаптируемой системы, разработчики должны сначала сертифицировать все компоненты, то есть подтвердить их корректность относительно спецификаций. Они могут пройти сертификацию либо за счет выбора компонентов, правильность которых уже подтверждена с помощью традиционных методов (тестирование, инспектирование, верификация на базе моделей и т.д.), либо с помощью автоматической генерации кода по спецификациям. Кроме функциональных, сертификация может включать и нефункциональные требования (например, требования к безопасности и производительности).

Кроме того, необходимы методики, гарантирующие, что модифицированная система по-прежнему работает приемлемым образом или безопасна. Для решения этой задачи принято использовать анализ зависимостей [2]. Также разработчики могут применять высокоуровневые контракты и инварианты для мониторинга корректности системы до, во время и после адаптации.

2. Безопасность. В то время как гарантия корректности связана преимущественно с целостностью системы, задача обеспечения безопасности – защитить систему от вторжений, не предоставив при этом нарушителям возможности использовать механизмы адаптации в своих целях. Адаптивная программная система должна не только проверять корректность исходных текстов компонентов, но и защищать от злоумышленников свое ядро. Существует множество хорошо изученных механизмов защиты систем, например, стойкое шифрование для гарантии конфиденциальности и аутентификации взаимодействия объектов.

3. Интероперабельность. Распределенные системы, способные адаптироваться к среде, должны настраивать отдельные компоненты и координировать такую настройку на различных системных уровнях и платформах. Программные компоненты, как правило, поставляются разными производителями, поэтому разработчику придется интегрировать механизмы адаптации для удовлетворения требований распределенного приложения. Проблема усложняется многообразием подходов к реализации адаптивного ПО на различных системных уровнях. А решения, относящиеся к одному и тому же уровню, зачастую несовместимы. Разработчикам необходимы инструментальные средства и методы интеграции адаптивных компонентов на разных уровнях одной системы, между несколькими системами и разными «платформами» адаптации.

4. Принятие решений. Адаптивные системы реагируют на меняющийся физический мир. Они должны действовать автономно, модифицируя структуру ПО в соответствии с изменением среды, предотвращая сбои и перерывы в работе. Чтобы определить, как, когда и где выполнять адаптацию системы, модули принятия решений используют информацию, получаемую от программных и аппаратных сенсоров. Некоторые системы могут даже требовать, чтобы эти модули «изучали» поведение пользователей и адаптировались к нему.

Одни исследователи создают программные инструменты принятия решений, опираясь на подходы на базе правил или теорию управления. Другие проектируют модули принятия решений, действия которых подобны «биологическим» процессам, таким, как нервная деятельность человека или поведение колоний насекомых. Эти подходы достаточно эффективны в определенных предметных областях, но динамика среды и сложность программ ограничивают их повсеместное распространение. Необходимы всесторонние исследования в области принятия решений для целей адаптации ПО. Высоко эффективные адаптируемые системы должны получать самую разную информацию от сенсоров, продолжать непрерывно обучаться с учетом собственного опыта и осваивать новые подходы и методы адаптации по мере их появления.

СПИСОК ЛИТЕРАТУРЫ

1. Гамма Э., Хелм Р., Джонсон Р. Приемы объектно-ориентированного программирования. Паттерны проектирования. СПб.: Питер, 2006. – 366 с.
2. Ф. Маккинли, С. М. Саджади, Э. Кастен, Б. Ченг. Композитная адаптация программ. // СУБД. Открытые системы № 9, 2004
3. Ордынцев В.М. Системы автоматизации экспериментальных научных исследований. М.: Машиностроение, 1984. – 328 с.

УДК 621.315.592

УЧЕТ СПИНОВЫХ СОСТОЯНИЙ В СПЕКТРАХ ПРИМЕСНОГО МАГНИТООПТИЧЕСКОГО ПОГЛОЩЕНИЯ ПОЛУПРОВОДНИКОВЫХ 1D-СТРУКТУР

Е. Н. КАЛИНИН
Пензенский государственный педагогический университет им. В. Г. Белинского
кафедра общей физики

В рамках водородоподобной модели решена задача о влиянии спиновых состояний локализованного электрона на спектр примесного магнитооптического поглощения полупроводниковых структур из квантовых проволок на основе InSb содержащих водородоподобные примесные центры. Получены аналитические выражения для волновых функций и энергетического спектра локализованного электрона, а также рассчитан коэффициент поглощения света полупроводниковой структуры с квантовыми проволоками.