МЕТОДЫ И ПРОГРАММНЫЕ СРЕДСТВА МАКРОМОДУЛЬНОЙ РАЗРАБОТКИ ПРОГРАММ

А.А. Сиднев

Нижегородский государственный университет им. Н.И. Лобачевского

Введение

Один из нерешённых вопросов в модульном подходе разработки программ – отсутствие стандартов на интерфейсы модулей. Разработчик библиотеки сам определяет удобные ему структуры хранения данных и интерфейсы функций, которые их обрабатывают. В результате, каждая библиотека получается уникальной. Во многих предметных областях существует множество библиотек с реализацией различных алгоритмов. Наличие большого количества библиотек приводит к необходимости выбора из них одной или нескольких наиболее подходящих. Поддержка нескольких библиотек усложняет структуру разрабатываемого проекта, а значит, трудозатраты на его разработку увеличиваются. Задача выбора библиотеки часто является многокритериальной, в связи с чем приходится идти на компромисс, выбирая некоторое «среднее» решение, теряя в эффективности реализации, удобстве использования или количестве поддерживаемых программно-аппаратных платформ. Кроме того, в процессе разработки могут возникнуть задачи, которые нельзя решить, используя текущую библиотеку. При переходе к использованию новой библиотеки разработчику придётся столкнуться с необходимостью выполнить модификацию разработанных структур данных и функций под те, которые используются в библиотеке. Такой переход может быть очень трудоёмким. Итак, основной недостаток классической модульной разработки программного обеспечения – это сильная зависимость от конкретной реализации библиотеки. Этот недостаток порождает три актуальные проблемы, которые тесно связаны и часто возникают одновременно:

- выбор наиболее подходящей библиотеки под текущие задачи проекта;
- поддержка нескольких библиотек;
- миграция на новую библиотеку.

Одно из лучших решений проблемы миграции – это разработка стандартного интерфейса для библиотек, решающих задачи одного класса. В результате все библиотеки реализуют одинаковый интерфейс, и задача перехода с одной библиотеки на другую решается просто (ярким примером является стандарт MPI [1]). У такого решения есть важный недостаток, который ограничивает его применимость, – разработка стандарта требует существенных усилий большой группы людей и занимает длительное время.

Сложность миграции приложения на новую библиотеку в первую очередь определяется качеством его проектирования и тем, была ли заложена при этом возможность миграции. Наиболее удачные решения, используемые при проектировании приложений, оформились в виде шаблонов проектирования [2]. С помощью предварительного анализа разрабатываемого ПО можно исключить потенциальные проблемы совместимости. В работах [3, 4] рассматриваются инструмент и метод анализа бинарной совместимости разделяемых библиотек под Linux, а также набор правил, следование которым обеспечивает совместимость. В работе [5] приводится анализ мобильности приложений между различными версиями ОС Linux. Поддержка нескольких библиотек реализуется за счёт использования средств автоматизации сборки [6]. Широкое развитие получили средства автоматизации программного реинжиниринга. Набор инструментов DMS [7] позволяет выполнять автоматический анализ, трансформацию программ с одного языка программирования на другой и их синтез. Предметно-ориентированный язык RASCAL [9] позволяется выполнять анализ исходных кодов и автоматическую трансформацию. Универсальные языки TXL [10], Stratego [11] позволяют преобразовывать исходные коды программ за счёт манипуляции нотациями языков программирования (описывается грамматика текущего языка и правила модификации синтаксиса к требуемому). Когда приложение работает с конкретной библиотекой, можно использовать шаблонный метод трансформации программ [12], который заключается в формализации структуры и поведения исходной и целевой библиотек с помощью специального языка. На основе этих описаний выполняется трансформация программы под новую библиотеку.

Применение предлагаемых подходов требует специальных навыков, а по части рассмотренных направлений ведутся дальнейшие исследования. В данной работе рассматривается решение, объединяющее в себе часть рассмотренных подходов, что позволяет устранить их недостатки и снизить остроту перечисленных выше проблем. Данная работа является развитием [8].

Макромодульная разработка программ

Если существует стандарт на интерфейсы, то можно разрабатывать программы независимо от конкретных реализаций библиотек. Обеспечить реализации существующих библиотек, решающих задачи одного класса, единым интерфейсом (даже при наличии стандарта) – тяжелая задача. Для решения этой

проблемы автором предлагается разрабатывать программы-переходники (адаптеры) для каждой библиотеки, которые обеспечат «стыковку» стандартных интерфейсов и используемых в библиотеке интерфейсов. Таким образом, модули программных библиотек должны либо разрабатываться при строгом соблюдении стандартных интерфейсов, либо библиотеки должны содержать адаптеры.

Для подготовки вызова модуля программной библиотеки в соответствии со стандартным интерфейсом необходимо выполнить описание модели вычислений. Чтобы избежать зависимости описания модели вычислений от конкретного языка программирования, оно выполняется на специальном макроязыке.

Для выбора наиболее эффективной реализации для текущей программно-аппаратной платформы среда выполнения вычислительной системы должна содержать планировщик, определяющий лучшие реализации модулей. Выбор может осуществляться статически во время сборки программы или динамически во время её выполнения. Последнее позволяет ориентироваться при разработке на более широкий круг программно-аппаратных платформ, т.к. для каждой из них выбирается лучшая из доступных реализаций.

Макромодульный подход ориентирован на практическую разработку программного обеспечения, поэтому он содержит как методологию построения программного обеспечения, так и программную среду, которая включает следующие элементы (Рис. 1):

1. Макроязык, который позволяет описывать модели вычислений подпрограмм в приложении пользователя.
2. Архитектуру среды сборки приложений, которая определяет схему и алгоритм взаимодействия компонент сборки.
3. Планировщик, выполняющий выбор библиотеки.
4. Архитектуру среды времени исполнения, которая определяет схему взаимодействия компонент времени исполнения.
5. Архитектуру адаптеров и требования к их реализации.

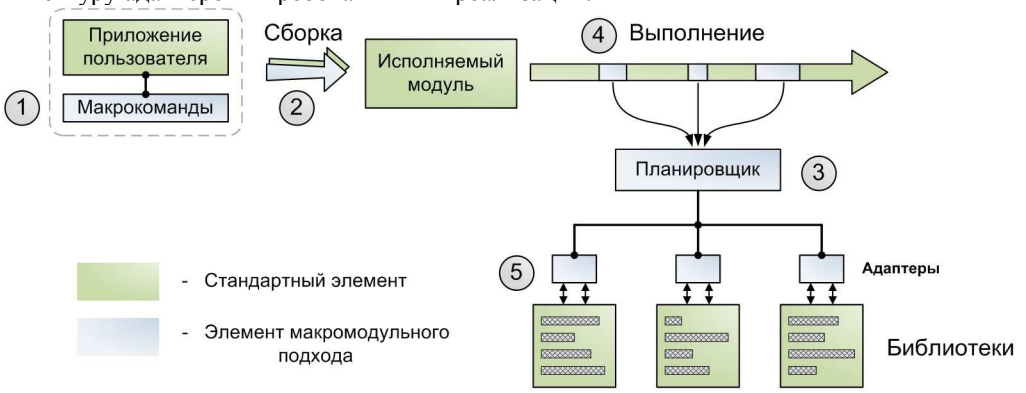

Рис. 1. Общая схема макромодульного подхода. Цифрами указано соответствие элементов макромодульного подхода и этапов разработки и выполнения макропрограммы

Результаты

Разработана среда поддержки, содержащая все необходимые элементы для сборки и исполнения макромодульных программ. Реализованы адаптеры для библиотек MKL, OpenBLAS и FFTW. Для апробации рассмотренного подхода написаны программы с использованием функций BLAS и FFTW3. Полученные программы могут использовать любую из библиотек без модификаций исходного кода. Реализован планировщик, выполняющий выбор наиболее эффективной библиотеки среди множества доступных в системе. Результаты экспериментов на 84 вычислительных системах, которые были построены на процессорах разных поколений (от Pentium 4 до современных на базе ядра Haswell), разных производителей (Intel, AMD), разного назначения (мобильные, настольные, серверные) и, как следствие, существенно отличающейся производительности, демонстрируют высокую эффективность работы планировщика. Ошибка оценки времени выполнения функций из библиотек не превосходит 15% при первых запусках программ. При последующих запусках точность оценки значительно повышается.

Заключение

На данный момент во многих предметных областях существует множество библиотек с реализацией различных алгоритмов. Отсутствие стандартов на интерфейсы этих библиотек порождает ряд проблем, связанных с поддержкой нескольких библиотек, миграцией на новые библиотеки и выбором наиболее подходящей под задачи проекта. Предложенный макромодульный подход позволяет снизить остроту перечисленных проблем и сократить трудозатраты на разработку ПО.

ЛИТЕРАТУРА:

1. Message Passing Interface Forum. – URL: [http://www.mpi-forum.org/].

2. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Д. Приемы объектно-ориентированного проектирования. Паттерны проектирования = Design Patterns: Elements of Reusable Object-Oriented Software. // СПб: «Питер». – 2007.
3. Shved P., Silakov D. Binary Compatibility of Shared Libraries Implemented in C++ on GNU/Linux Systems. // SYRCoSE. – 2009. URL: [http://syrcose.ispras.ru/2009/files/02_paper.pdf].
4. Ponomarenko A., Rubanov V., Khoroshilov A. A system for backward binary compatibility analysis of shared libraries in Linux // Proc. of Software Engineering Conference in Russia (CEE-SECR). – 2009. – P. 25–31.
5. Rubanov V. Automatic Analysis of Applications for Portability Across Linux Distributions // Proc. of the Third International Workshop on Foundations and Techniques for Open Source Software Certification. – 2009. – Vol. 20. – P. 1–9.
6. Махоткин А. GNU Autotools — инфраструктура для сборки программ: Automake. – URL: [http://squadette.ru/autotools-ru/article1.html].
7. DMS Software Reengineering Toolkit. – URL: [http://www.semdesigns.com/products/DMS/DMSToolkit.html]
8. Гергель В., Сиднев А. Методы и программные средства макромодульной разработки программ. // Вестник нижегородского университета им. Н.И. Лобачевского. – 2012. – №5(2). – С. 294-300.
9. Klint P., Storm T., Vinju J. RASCAL: A Domain Specific Language for Source Code Analysis and Manipulation // Proc. of Ninth IEEE International Working Conference on Source Code Analysis and Manipulation. – 2009. – P. 168–177.
10. Cordy J. R. The TXL source transformation language // Science of Computer Programming. – 2006. – N 61 (3). – P. 190–210.
11. Bravenboer M., Dam A., Olmos K. Eelco V. Program Transformation with Scoped Dynamic Rewrite Rules. // Technical Report UU-CS-2005-005, department of Information and Computing Sciences. – 2005.
12. Ицыксон В., Зозуля А. Автоматизированная трансформация программ при миграции на новые библиотеки. // Программная инженерия. – 2012. – № 6. – С. 8–14.