УДК 004.416.3+004.4'242

АВТОМАТИЗАЦИЯ РЕИНЖИНИРИНГА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ПРИ ПОРТИРОВАНИИ НА НОВЫЕ БИБЛИОТЕКИ С ПОМОЩЬЮ ЧАСТИЧНЫХ СПЕЦИФИКАЦИЙ

В. М. Ицыксон,
канд. техн. наук, доцент
Санкт-Петербургский государственный политехнический университет

Рассматривается подход к реинжинирингу программ, основанный на использовании частичных спецификаций библиотек. Описываются семантические примитивы для задания спецификаций библиотек, рассматриваются способы задания спецификации видимого поведения библиотек. Процесс реинжиниринга программы автоматизируется с помощью алгоритма, который проверяет совместимость двух библиотек, анализирует семантику исходной и целевой библиотек и производит преобразование программы путем выражения интерфейса старой библиотеки в терминах новой.

Ключевые слова *— библиотека, частичная спецификация, семантика программы, реинжиниринг программ, портирование.*

Введение

Одной из частых задач, встающих перед разработчиками программного обеспечения (ПО), является перенос существующих программ в окружение, использующее новые библиотеки. С точки зрения программной системы любой такой перенос является реинжинирингом, выражающимся в частичной переработке программы. При реинжиниринге модификации подвергаются те части программы, которые непосредственно взаимодействуют с библиотекой.

Рассмотрим подробнее ситуации, в которых разработчику требуется проводить реинжиниринг программ.

Задачи переноса программ в новое библиотечное окружение

Задача портирования в новую операционную систему (ОС) встает перед разработчиком, когда имеется приложение, функционирующее под управлением одной ОС (например, Windows), которое требуется запускать в другой ОС (например, Linux). Обычно API библиотек в разных ОС отличаются, а сами библиотеки имеют разное функциональное наполнение.

В качестве примера рассмотрим перенос программы, осуществляющей сетевой обмен с помощью сокетов, из ОС Linux в Windows. Отличие двух программ на языке C состоит в разном программном интерфейсе библиотек BSD-sockets и WinSock. Если не вдаваться в глубокую детализацию, то модификация программы заключается в замене заголовочных файлов; в добавлении функций WSAStartup и WSACleanup, инициализирующих и выгружающих библиотеку WinSock; в изменении некоторых функций API (closesocket вместо close и т. п.) и в изменении некоторых типов данных.

Задача адаптации приложения на новую аппаратную платформу возникает, когда программа, функционирующая на одной платформе (например, Intel x86), должна быть адаптирована для работы в другой платформе (например, на Apple iPhone). Адаптация заключается в интеграции программы с библиотеками, поддерживающими новую аппаратуру.

Задача перехода на новую версию библиотек появляется при выпуске разработчиками новых версий своих продуктов, которые реализуют новую функциональность или содержат исправления ошибок. Часто новые версии используют измененный программный интерфейс, который не полностью совместим с предыдущими версиями. Особенно часто такая ситуация проявляется при изменении старших (major) номеров версий библиотек.

Задача перехода на альтернативные библиотеки встает, когда для достижения поставленной

цели требуется частично модифицировать приложение, и функции, ранее реализованные с использованием одних подходов и библиотек, должны быть переписаны с использованием других. Примерами таких задач являются переход с использования мьютексов на использование семафоров, миграция с библиотеки GTK+ на QT или замена многопроцессного приложения многопоточным.

Переход на безопасные версии библиотек требуется из-за проблем некоторых стандартных библиотек языка С, когда их использование может быть причиной наличия уязвимостей в программах. Одним из подходов, решающих эту проблему, является переход на безопасные версии библиотек.

При миграции на другой язык программирования стоят две задачи: замена конструкций одного языка программирования на соответствующие конструкции другого и замена одной библиотеки на другую. Вторая задача является значительно более сложной, так как библиотечных функций намного больше, чем языковых конструкций, и их семантика обычно существенно сложнее.

Перечисленные ситуации показывают необходимость разработки подходов, позволяющих частично или полностью автоматизировать процесс портирования программы. Наиболее актуально это для языков программирования С и С++, компиляторы которых существуют практически для всех ОС, и с помощью которых реализован огромный объем программного кода.

Стандартные пути проведения портирования

Обычно перечисленные выше задачи решаются программистами вручную. При этом используется один из следующих подходов.

Ручное портирование приложения предполагает переход на новые библиотеки путем последовательной замены элементов старой библиотеки новыми. Этот подход является ресурсозатратным и требует длительной отладки и глубокого тестирования преобразованной программы.

При *портировании с использованием макроопределений* вызовы функций оформляются в виде макроопределений препроцессора (или макросов), настройка которых под конкретную библиотеку (макроподстановка) происходит с помощью условной компиляции на стадии препроцессирования. Существенное ограничение подхода связано с его применимостью только в случае, когда библиотеки отличаются лишь сигнатурами функций.

При *портировании с помощью создания промежуточного программного* слоя взаимодействие с новой библиотекой реализуется через функции-заглушки, сохраняющие синтаксис вызова старой библиотеки и содержащие вызовы функций новой. Такой подход наиболее прост в отладке, но накладывает ограничения на степень отличия библиотек друг от друга и требует повторного создания промежуточного слоя при новом портировании.

Все указанные подходы являются ручными и требуют проведения множества рутинных операций, в то время как задача переноса приложений может неоднократно повторяться для разных программ, при этом во всех случаях требуется проведение однотипного реинжиниринга, при котором операции модификации больше зависят от интерфейсов библиотек, чем от преобразуемой программы. Таким образом, задача автоматизации реинжиниринга программ при переносе в новые библиотеки является актуальной.

Постановка задачи

Насущность задачи трансформации программ при решении задач портирования приводит к следующей постановке задачи исследования. Необходимо разработать технологию реинжиниринга ПО, автоматизирующую процесс преобразования программы, использующей исходную библиотеку, в новую программу, использующую целевую библиотеку. Технология должна базироваться на формальных спецификациях обеих библиотек и обеспечивать автоматизированную трансформацию исходной программы, основанную на анализе таких спецификаций.

Полученные результаты

В рамках данной работы получены следующие основные результаты:

— определены и классифицированы способы взаимодействия программы с библиотечным окружением;

— предложен формальный аппарат для описания структуры и поведения библиотек;

— предложена методика автоматизированной трансформации программ при портировании в новое библиотечное окружение.

Существующие подходы к проблеме автоматизации реинжиниринга

Задача портирования приложений на основе семантических представлений библиотек в общем виде в настоящий момент не решена, и отсутствуют публикации на эту тему. Однако существуют смежные области, в которых также используются механизмы автоматизированной трансформации программ для решения других задач реинжиниринга. Рассмотрим ключевые группы имеющихся подходов.

Основные достижения в области трансформации приложений связаны с развитием *синтаксиче-*

ческих подходов, использующих синтаксические свойства программ: шаблонные преобразования и перезапись термов.

В *подходах, основанных на шаблонах*, по исходному коду программы строится модель, определяются шаблоны кода, используемые при поиске и трансформации. Далее осуществляется обнаружение участков кода по шаблону и применение трансформаций. Различаются подходы используемыми моделями, языком описания шаблонов и способом задания трансформаций. Примерами таких средств являются InjectJ [1] — система преобразования кода на языке Java, технология модификации программ на основе шаблонов [2] и система ReSharper [3]. Все эти подходы являются сугубо синтаксическими и ограничены необходимостью создания отдельных шаблонов для каждого преобразования.

Многие синтаксические подходы основаны на использовании *правил перезаписи* (rewriting rules), поддержанных мощными языками трансформаций. Абстрактная система перезаписи [4] включает множество объектов, над которыми выполняется перезапись, и множество отношений, которые задают возможные преобразования элементов из множества друг в друга.

Развитием подхода правил перезаписи является концепция *стратегий перезаписи* (rewriting strategies) [5]. Данная концепция легла в основу языка трансформации Stratego, реализованного в рамках системы Stratego/XT [6]. В качестве модели программы, над которой выполняется трансформация, используется система аннотированных термов — вариант абстрактных синтаксических деревьев, в которой элементы дерева могут содержать аннотации, дополняющие синтаксическую информацию о программе семантической, которая затем может использоваться в правилах перезаписи.

Еще одним представителем этой группы подходов является язык TXL, предназначенный для разработки различных систем трансформации программ [7]. В основе TXL лежит система перезаписи термов, работа с которой осуществляется при помощи специального функционального языка. Все правила перезаписи записываются в терминах этого языка, а синтаксис языка — в расширенной форме Бэкуса–Науэра. Данный подход получил практическое воплощение в компиляторе FreeTXL [8]. Похожие подходы используются в языке правил перезаписи ASF+SDF [9], реализованном в рамках платформы ASF+SDF Meta Environment, а также в системе DMS [10].

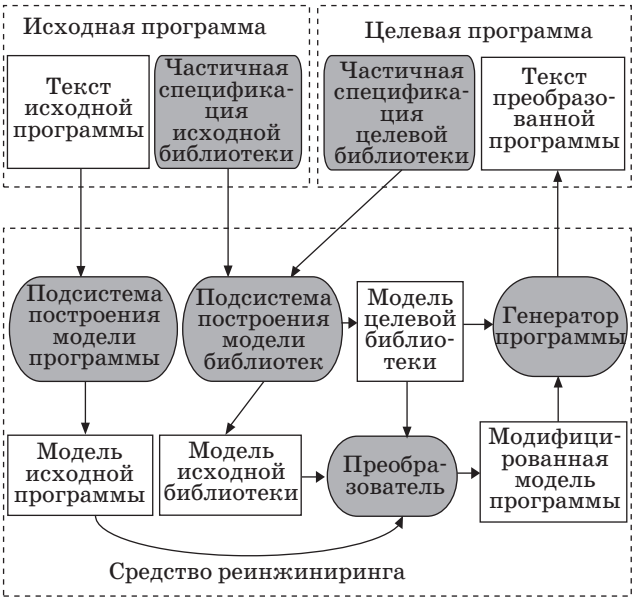
Основным недостатком перечисленных подходов является их синтаксическая ориентированность, ограничивающая их применение в основном только простыми трансформациями, когда семантика всех элементов программы известна заранее и при реинжиниринге одни синтаксические конструкции заменяются другими. Для проведения более сложных преобразований, когда алгоритм трансформаций программ управляется семантикой библиотек, требуются более сложные подходы.

Большинство *семантических подходов* помимо синтаксиса используют семантику конструкций языка программирования. Существует целый ряд методов, направленных на автоматизацию миграции приложений с одного языка программирования на другой. К наиболее характерным относятся подходы, предназначенные для переноса программ с устаревших языков или технологий программирования на более новые. В работах [11, 12] описывается автоматизация преобразования программ с языка C++ и Java в Java и C# соответственно. Существуют и другие подходы, применяемые для межязыкового портирования программ. Все они используют семантику языков и, иногда, конкретных библиотек для проведения миграции, но возможностей задания семантики целых библиотек и преобразования программ с учетом этой семантики они не предоставляют.

Предлагаемый подход

Предлагаемый в этой работе подход основан на использовании семантического описания двух библиотек. Семантика задается с помощью частичных спецификаций каждой библиотеки в отдельности. Анализируя семантические описания двух библиотек, можно до проведения реинжиниринга сделать вывод об их принципиальной совместимости. Например, приложение, работающее с файлами, можно трансформировать в приложение, использующее потоки ввода-вывода. Однако заменить библиотеку работы с файлами на библиотеку работы с семафорами невозможно ввиду их семантической несовместимости. Спецификация библиотеки создается программистом, проводящим портирование. Исходной информацией для создания семантических описаний может служить документация (описания API, исходные тексты и т. п.) и понимание принципов работы. Основные преимущества семантической спецификации — это формализованность представления и возможность повторного использования при последующих портированиях. Следует отметить, что, как и при создании других спецификаций, правильность задания конкретной спецификации библиотеки должна обеспечиваться программистом.

В случае если библиотеки семантически совместимы, то производится преобразование исходной



■ ***Рис. 1***. Общая схема реинжиниринга

программы с учетом спецификаций обеих библиотек. При этом преобразованию подвергаются все элементы исходной программы, каким-либо образом соприкасающиеся с интерфейсом библиотек.

Более подробно подход можно рассмотреть с помощью общей схемы проведения реинжиниринга (рис. 1). В соответствии с этой схемой решение задачи реинжиниринга происходит за пять этапов.

1. Создание частичных спецификаций всех библиотек, участвующих в реинжиниринге. Для описания спецификаций используется специальный язык PanLang, позволяющий задавать видимое поведение компонентов библиотеки.

2. Трансляция исходного текста программы и спецификаций библиотек во внутренние модельные представления.

3. Анализ совместимости исходной и целевой библиотек. Для этого проводится исследование возможности выражения семантики старой библиотеки с помощью примитивов новой.

4. Трансформация модели программы в новую модель, соответствующую целевой библиотеке. При трансформации элементы модели исходной библиотеки выражаются в терминах целевой.

5. Восстановление текста программы по модифицированной модели. Полученная новая программа использует для реализации функций новое библиотечное окружение.

Модели программной системы и библиотек

Рассмотрим предлагаемые в данной работе механизмы представления программных систем и способы задания спецификаций библиотек.

Модель программной системы

Модель программной системы используется для того, чтобы процедуры анализа и синтеза проводить не на основе избыточного и неудобного текстового представления, а базируясь на формальном графовом представлении. Выбор конкретной модели определяется предъявляемыми к ней требованиями. В случае формирования модели для проведения портирования основными требованиями к модели являются возможности:

— доступа ко всем синтаксическим элементам программы;

— восстановления полного исходного текста программы.

В зависимости от глубины процесса реинжиниринга в качестве модели программы можно использовать как структурные, так и поведенческие представления [13]. Чаще всего для этой цели применяются абстрактное синтаксическое дерево, граф потока управления, граф зависимостей по данным или абстрактный семантический граф [14]. В настоящей работе в качестве основы модели используется абстрактный семантический граф, являющийся наиболее гибким представлением, легко поддающимся расширению.

Частичные спецификации библиотек

Для полного описания какой-либо библиотеки необходимо специфицировать внешнее поведение и все внутренние аспекты реализации. Такого рода описания громоздки, зачастую их использование нецелесообразно. Для решения задачи автоматизации реинжиниринга не требуется задавать полные спецификации библиотек, достаточно описать лишь часть свойств, характеризующих их взаимодействие с программой и внешней средой. Причем в зависимости от решаемой задачи степень детализации описания может быть различной. Такого вида спецификации будем называть частичными.

Сама по себе библиотека по отношению к использующей ее программе может проявлять свое внешнее поведение следующими путями:

— через параметры функций API и возвращаемый результат;

— через глобальные переменные программы;

— через создаваемые объекты ОС (ресурсы).

Уровень выразительности спецификаций должен позволять не только описывать аспекты видимого поведения библиотек, но и задавать *явную семантику* элементов библиотеки. Это даст возможность определять семантическую совместимость библиотек и проводить автоматизированный реинжиниринг ПО.

Для задания явной семантики библиотеки в частичных спецификациях используются следующие механизмы:

— семантическая интерпретация значений переменных (используется для задания совместимости значений);

— семантическая интерпретация типов (используется для задания совместимости типов переменных);

— параметризируемые абстрактные действия функций (используются для сопоставления семантики отдельных библиотечных функций).

Суммируя перечисленные выше аспекты, определим элементы предлагаемого формализма частичных спецификаций.

Ресурсы — объекты окружения или ОС, имеющие собственный жизненный цикл и существующие отдельно от программы. Состояния ресурса могут меняться посредством вызовов API, по истечении тайм-аутов или по инициативе ОС. Чаще всего жизненный цикл ресурса задается детерминированным конечным автоматом. Примерами ресурсов могут быть файлы, потоки, сокеты, семафоры, мьютексы, динамическая память и т. п.

Семантические типы данных используются для задания семантической интерпретации конкретных значений определенных типов данных. В разных библиотеках различные значения переменных могут иметь одну и ту же семантическую нагрузку. Например, вызов функции shutdown библиотеки BSD-sockets в ОС Linux в качестве параметра how принимает одно из значений: SHUT_RD, SHUT_WR или SHUT_RDWR. В то же время в аналогичной функции в ОС Windows этот параметр может принимать значения SD_RECEIVE, SD_SEND или SD_BOTH. Введение семантической интерпретации позволит при реинжиниринге правильно сопоставить одни константы с другими.

Семантические действия, или просто *действия*, предназначены для задания семантического описания побочного эффекта, выполняемого в теле специфицируемой библиотечной функции. Действие характеризуется символическим именем и, при необходимости, параметрами. Например, функция exit имеет побочное действие — завершение программы, а функция CreateThread — создание нового потока выполнения. В некоторых случаях действия могут параметризироваться для описания более сложного поведения библиотечной функции. Например, наличие действия OPEN(Name) в теле функции означает, что данная функция осуществляет действие OPEN с параметром Name. Функции open и fopen из разных библиотек могут иметь в спецификации действие OPEN. Наличие одинаковых действий показывает, что поведение функции open можно выражать через поведение функции fopen. Конкретные имена действий выбираются разработчиком спецификаций и синтаксически задают абстрактную семантику поведения.

Описание поведения функций задается императивно с помощью укрупненного алгоритма функционирования. При этом степень детализации описания выбирается разработчиком в зависимости от поставленной задачи. Следует отметить, что алгоритм может содержать выполнение семантических действий для спецификации побочного эффекта.

Полная спецификация формализма, описывающего программные библиотеки, приведена в работе [15]. Выразительная мощность введенного формализма соответствует мощности системы взаимодействующих конечных автоматов. Это, с одной стороны, является достаточным для описания видимого поведения библиотек, а с другой стороны, позволяет сделать задачу совместимости библиотек алгоритмически разрешимой.

Частичные спецификации задаются разработчиком с помощью языка описания спецификаций PanLang [16], разработанного на кафедре компьютерных систем и программных технологий СПбГПУ. Язык имеет С-подобный синтаксис и обладает всеми необходимыми механизмами для формирования частичных спецификаций библиотек. Фрагмент частичной спецификации библиотеки работы с файлами приведен в листинге.

```
requires <stdio.h>, <stdlib.h>;
semantic type FILE_TYPE (FILE *);
semantic type FLAGS (char*) {
  READ:"r"; WRITE:"w"; ALL:"rw";
}
semantic type HW_RES (int) {
  ERR:[-inf;-1]; OK:[0;+inf];
}
resource FILE_RES (FILE_TYPE) {
  states OPEN, CLOSED;
  attribute FLAGS MODE;
}
action void READ(FILE);
function FILE fopen(FILE_NAME(char*), FLAGS mode) {
  f = new FILE_RES(OPEN);
  attr(f, MODE) = mode;
  return f;
}
function HW_RES fread(BUFF, SIZE, PORTON p=1, FILE f) {
  if (state(f) == $OPEN) {
    action READ(f);
    return $OK;
  }
  else {
    return $ERR;
  }
}
```

Фрагмент содержит секцию описания подключаемых файлов, определение типов, расши-

ряющих стандартные типы семантической интерпретацией значений. Определяется специальный ресурс, соответствующий дескриптору файлов, декларируются все действия, производимые библиотекой. Завершается спецификация поведенческими описаниями функций библиотеки, задающими укрупненное поведение и оперирующими введенными примитивами.

Особенность языка PanLang — отсутствие в поведенческих описаниях функций конструкций, с помощью которых можно организовать циклы или рекурсии. Такое ограничение переводит PanLang в класс автоматных языков. Описанная с помощью такого языка библиотека представляется конечным автоматом или системой конечных автоматов. При этом состояниями автоматов являются состояния ресурсов, входными символами — вызовы функций, а выходными символами — выполняемые семантические действия.

Подробное описание синтаксиса и семантики языка PanLang приведено в работе [16] и выходит за рамки данной статьи.

Методика реинжиниринга программного обеспечения

Перенос программы из одного библиотечного окружения в другое осуществляется на основе анализа двух частичных спецификаций. Главная решаемая при этом задача — обеспечение семантической эквивалентности. С точки зрения частичных спецификаций это означает, что трансформированная программа при работе с новой библиотекой должна повторять видимое поведение исходной программы при работе со старой библиотекой. Для этого проверяется семантическая совместимость двух библиотек и осуществляется трансформация программы.

Проверка совместимости библиотек

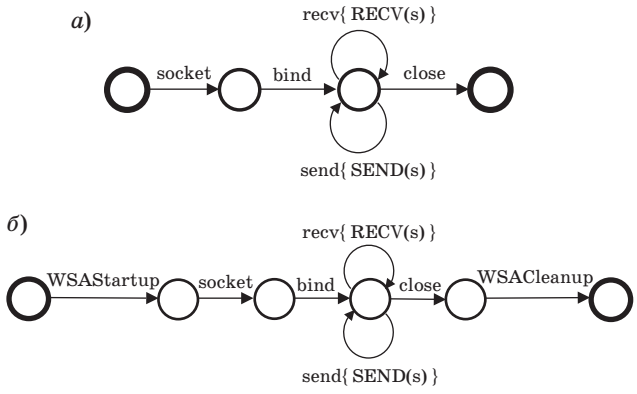
Проверка совместимости библиотек выполняется на основе анализа двух частичных спецификаций. Библиотеки считаются семантически совместимыми, если с помощью примитивов новой библиотеки можно выразить любые действия, выполняемые с помощью старой. Для этого необходимо убедиться в возможности задания любого поведения старой библиотеки в новой. Поведение библиотеки задается с помощью нескольких модельных примитивов, ключевыми из которых являются ресурсы и выполняемые действия. Поведенческая модель библиотеки представляется системой конечных автоматов, в которой входными символами являются вызовы функций API библиотеки, а выходными — выполняемые семантические действия [15]. Будем называть траекторией библиотеки конечную или бесконечную последовательность событий, которые могут быть сгенерированы функциями библиотеки. Под событиями здесь понимается выполнение семантического действия или изменение состояния ресурса.

Для проверки совместимости библиотек анализируются модельные представления двух спецификаций. Если средствами новой библиотеки можно создать все возможные траектории, реализуемые старой, то искомый реинжиниринг возможен, а спецификации совместимы.

Рассмотрим процедуру проверки совместимости на примере библиотек работы с сокетами BSD-sockets и WinSock2. Упрощенная поведенческая модель библиотеки BSD-sockets показана на рис. 2.

Если средствами новой библиотеки (например, WinSock2) можно повторить все траектории, порождаемые библиотекой BSD-sockets, то делается вывод о семантической совместимости библиотек. Если хоть одна из возможных траекторий не выразима в терминах новой библиотеки, то семантическая совместимость отсутствует. Модель на рис. 2, *а* может порождать бесконечное количество трасс, содержащих выполнение действий RECV и SEND (выполняемые действия на рисунках заключены в фигурные скобки).

Рассмотрим аналогичную упрощенную модель второй библиотеки WinSock2 (рис. 2, *б*)[1]. Сама модель является более сложной, так как содержит большее количество узлов и переходов, однако для нее множество порождаемых траекторий (также состоящее из выполнения семантических действий RECV и SEND) совпадает с множеством траекторий библиотеки BSD-sockets. То



■ ***Рис. 2.*** Фрагмент поведенческой модели библиотек BSD-sockets (*а*) и WinSock2 (*б*)

[1] В целях экономии места в приведенном примере поведенческая модель состоит из одного автомата, использующего комбинирование состояний инициализации библиотеки и UDP-сокета. В общем случае модель будет представлять собой систему взаимодействующих автоматов.

есть можно сделать вывод о совместимости этих библиотек.

Для решения задачи проверки совместимости окружений в общем виде используется предварительное преобразование поведенческих моделей библиотек в семантические автоматы, представляющие траектории поведения библиотек. При этом сама задача проверки совместимости сводится к задаче проверки изоморфизма этих семантических автоматов. Подробное описание семантических автоматов и алгоритмов проверки выходит за рамки данной работы.

Трансформация программы

Трансформация программы осуществляется на основе выполненной проверки совместимости библиотек. Побочным эффектом успешной проверки является генерация последовательности модификаций семантических автоматов, позволяющей получить новые семантические автоматы путем цепочки модификаций старых. Эти последовательности представляют собой специальные сценарии, построенные по такому же принципу, как и сценарии преобразования деревьев в работе [2].

Собственно трансформация программы проводится путем применения сценариев к модели исходной программы в тех узлах, которые соответствуют элементам старой библиотеки. При этом модификациям подвергаются подключаемые файлы, вызовы библиотечных функций. Семантические типы данных заменяются на эквивалентные, генерируются функции преобразования для интерпретации значений семантических типов. После преобразования модели восстанавливается код программы на языке С.

Если все преобразования сделаны корректно, то полученная программа эквивалентна исходной с точностью до семантики используемых библиотек. Корректность трансформированной программы определяется только корректностью задания частичных спецификаций библиотек. Если спецификации построены правильно, то разработанный подход гарантирует корректность новой программы, причем обеспечивается и синтаксическая корректность, и семантическая эквивалентность ее исходной программе.

Практическая реализация

Разработанные методика, модели и алгоритмы были реализованы в прототипе системы реинжиниринга. Система предназначена для портирования программ на языке С и используется для апробации разработанных подходов на реальных приложениях. Архитектура системы построена на основе схемы реинжиниринга, изображенной на рис. 1.

Построение модели (абстрактного семантического графа) программы осуществляется с помощью Clang, являющегося фронтэндом к системе LLVM [17]. Clang поддерживает современные стандарты языков С и С++, большинство расширений GNU. Сформированное дерево преобразуется во внутренний формат системы реинжиниринга для последующего добавления семантических свойств.

Подсистема построения модели библиотек использует грамматику языка PanLang для формирования внутреннего представления частичных спецификаций библиотек.

Созданный прототип использовался для проверки идей, заложенных в предлагаемый подход. Для этого были сформированы четыре тестовые задачи реинжиниринга:

— портирование Linux-приложения, работающего с сокетами, в ОС Windows;

— миграция приложения, использующего стандартную библиотеку ввода-вывода языка С, на использование системной библиотеки ввода-вывода;

— замена небезопасных функций работы со строками и участками памяти на безопасные версии;

— портирование многопоточного Windows-приложения, использующего библиотеку Windows Threads, в среду ОС Linux, использующую библиотеку POSIX Threads (pthreads).

Для всех библиотек (исходных и целевых), используемых в тестовых задачах, были созданы частичные спецификации на языке PanLang. Для каждой задачи были разработаны тестовые примеры на языке С, подлежащие трансформации. Во всех случаях проводились эксперименты по реинжинирингу программ в обоих направлениях — от исходной библиотеки к целевой и обратно.

На всех тестовых приложениях подход показал свою работоспособность, хотя выявил некоторые недоработки самого прототипа. К основным недостаткам прототипа следует отнести:

— не всегда полностью корректную вставку директив препроцессора по включению заголовочных файлов библиотек;

— возможность провести преобразования только на одном наборе флагов условной компиляции;

— отсутствие возможности сохранить комментарии в преобразованном программном коде.

Перечисленные недостатки относятся к программной реализации самого прототипа и не являются ограничениями разработанного подхода.

Заключение

В статье представлены основные результаты исследования в области портирования ПО в новое библиотечное окружение. Классифицированы основные механизмы взаимодействия программы с библиотеками, разработан формальный базис для задания частичных спецификаций библиотек. Собственно задание спецификации осуществляется с помощью специализированного декларативного языка PanLang. Процедура портирования формируется автоматизированно на основе анализа спецификаций двух библиотек. Подход реализован для программ на языке C в прототипе средства реинжиниринга.

Основными ограничениями текущей реализации являются работа только с простыми типами данных и невозможность проведения реинжиниринга в случае использования вызовов библиотечных функций по указателю.

Направления дальнейших исследований связаны с преодолением указанных ограничений, расширением разработанных подходов на объектно-ориентированные языки (в первую очередь на языки C++ и Java) и изучением возможности проведения межязыкового реинжиниринга.

Литература

1. **Inject/J project.** http://injectj.fzi.de/InjectJ/ (дата обращения: 15.12.2011).
2. **Itsykson V., Timofeyev D.** Source Code Modification Technology Based on Parametrized Code Patterns // 6th Central and Eastern European Software Engineering Conf. in Russia. Washington: IEEE Computer Society, 2010. P. 207– 213.
3. **JetBrains,** ReSharper. http://www.jetbrains.com/resharper/index.html (дата обращения: 10.12.2011).
4. **Ronald V. Book, Friedrich Otto.** String-rewriting Systems. — Springer, 1993. — 189 p.
5. **Marc Bezem, J. W. Klop, Roel de Vrijer.** Term Rewriting Systems. — Cambridge University Press, 2003. — 680 p.
6. **Program-Transformation.Org:** The Program Transformation Wiki. http://strategoxt.org/Transform/WebHome (дата обращения: 15.12.2011).
7. **J. R. Cordy.** The TXL Source Transformation Language // Science of Computer Programming. 2006. Vol. 61. N 3. P. 190–210.
8. **FreeTXL.** http://www.txl.ca/ (дата обращения: 10.12.2011).
9. **Compiling** language definitions: the ASF+SDF compiler — ACM, TOPLAS, 2002. http://dx.doi.org/10.1145/567097.567099 (дата обращения: 11.01.2012).
10. **The DMS** Software Reengineering Toolkit. http://www.semdesigns.com/products/DMS/DMSToolkit.html (дата обращения: 20.12.2011).
11. **X. Wang** et al. Reengineering Standalone C++ Legacy Systems into the J2EE Partition Distributed Environment: Proc. of the 28th Intern. Conf. on Software Engineering, ICSE '06. http://dx.doi.org/10.1145/1134285.1134359 (дата обращения: 20.12.2011).
12. **El-Ramly M., Eltayeb R., Alla H.** An experiment in automatic conversion of legacy Java programs to C#: Proc. of IEEE Intern. Conf. on Computer Systems and Applications. 2006. P. 1037–1045. http://dx.doi.org/10.1109/AICCSA.2006.205215 (дата обращения: 20.12.2011).
13. **Ицыксон В. М., Глухих М. И., Зозуля А. В., Власовских А. С.** Исследование средств построения моделей исходного кода программ на языках C и C++ // Научно-технические ведомости СПбГПУ. Информатика. Телекоммуникации. Управление. 2009. № 1 (72). С. 122–130.
14. **Datrix™** source code model and its interchange format: lessons learned and considerations for future work. 2001. http://dx.doi.org/10.1145/505894.505907 (дата обращения: 11.01.2012).
15. **Ицыксон В. М., Зозуля А. В.** Формализм для описания частичных спецификаций компонентов программного окружения // Научно-технические ведомости СПбГПУ. Информатика. Телекоммуникации. Управление. 2011. № 4. С. 81–90.
16. **Ицыксон В. М., Глухих М. И.** Язык спецификаций поведения программных компонентов // Научно-технические ведомости СПбГПУ. Информатика. Телекоммуникации. Управление. 2010. № 3. С. 63–71.
17. **Clang:** a C language family frontend for LLVM. http://clang.llvm.org/ (дата обращения: 14.01.2012).