Применение реинжиниринга при проектировании встраиваемых систем

Автор продолжает серию статей с описанием удобных методик и подходящих инструментальных средств, которые позволяют оптимизировать процесс проектирования программ и аппаратуры для современных встраиваемых устройств. Предлагается способ разработки новых систем с частичным использованием ранее написанного исходного кода (применение реинжиниринга). Рассматривается методика преобразования программного кода, которая подходит как для работы с приложениями высокого уровня, так и с файлами описания аппаратуры. Приводятся практические примеры работы с программным инструментарием системы реинжиниринга DMS, которая достаточно универсальна для поддержки нестандартных языков кодирования и нестандартных сред исполнения, что важно при проектировании встраиваемых устройств.

Анна СЕРГЕЕВА
annserge@rambler.ru

Введение

Когда разработчики встраиваемых систем приступают к созданию очередного нового проекта, то обычно начинают с внимательного изучения задания и анализа того, каким же должен быть конечный продукт, какими функциональными и аппаратными возможностями он должен обладать и с помощью каких средств это можно реализовать на деле. Помимо этого, в распоряжение разработчикам попадает ряд референсных дизайнов и/или образцовых проектов (ОП), которые изготовители микросхем прилагают к своим стартовым наборам. К тому же всегда есть возможность закачать со специализированных сетевых ресурсов различные примеры кода для тех или иных операционных систем (ОС), готовые функциональные модули и библиотеки и прочее. И попробовать применить их к конкретной разрабатываемой системе. Таким образом, по результатам серии проведенных экспериментов можно набрать множество фрагментов весьма разнородного кода, полезных в текущей разработке.

Также при разработке встраиваемых устройств часто бывает, что к реализации одной, скорее всего, специализированной ОС необходимо внести стек протоколов для какого-нибудь интерфейса от совершенно другого производителя или ОС. И в распоряжении разработчика изначально есть описание этого интерфейса, но только на стороннем языке, поэтому требуется его адаптация к виду, приемлемому для данного устройства.

Сегодня встроенная система, с одной стороны, это микропроцессор с его образцовым проектом, чей код написан на языке C, а также ПЛИС с описанием на языке VHDL, SystemC или Verilog. А с другой стороны, встроенная система — это зачастую веб-сервер, позволяющий менять рабочие настройки в удаленном режиме, плюс небольшая база данных, в которой хранится собранная информация по параметрам, датчикам и исполнительным механизмам системы. Все это означает, что важная часть кода системы реализуется и на HTML, SQL, PHP и на других прикладных языках.

Словом, в результате сбора исходных данных по проекту разработчики получают весьма большой объем разнородных файлов, код которых написан на самых разных языках. И здесь возникает такая проблема: все это многообразие нужно привести к общему виду, унифицировать код, частично перевести с одних языков на другие, адаптировать к особенностям конкретных специализированных ОС, из всего этого сделать базовое исполнение своего проекта и к тому же внести модификации под собственные потребности разрабатываемой системы.

Так каким же образом достаточно быстро разобраться со всей этой кучей раздобытых из разных источников примеров и сконвертировать многие исходные файлы кода в тот формат и на те языки, которые требуются в проекте?

Автор продолжает исследование и описание удобных методик и подходящих инструментальных средств, позволяющих оптимизировать процесс проектирования программ и аппаратуры для современных встраиваемых устройств [1, 2]. В данной статье приводится способ разработки новых систем с частичным использованием ранее написанного исходного кода. Этот подход называется реинжинирингом. Рассматривается методика преобразования программного кода, которая подходит как для работы с приложениями высокого уровня (различные действия в веб-интерфейсах, манипуляции с файлами, обращения к базам данных и т. д.), так и для работы с файлами описания аппаратуры (спецификации на низкоуровневых языках типа VHDL). Приводятся практические примеры работы с программным инструментарием системы реинжиниринга DMS, достаточно универсальной для поддержки нестандартных языков кодирования и нестандартных сред исполнения, что важно при проектировании встраиваемых устройств.

Пути разработки встраиваемых систем

Существует два варианта развития событий.

В первом случае вся архитектура системы и весь ее программный код полностью создается с нуля. Однако при этом следует принимать во внимание объем работы. Возникает необходимость самостоятельно сформулировать и формализовать полный перечень задач, поставленных перед разработчиками. Затем полностью создать весь программный код с нуля и обеспечить полное покрытие всех функциональных возможностей системы. Далее провести тщательное тестирование всего написанного программного кода, в полном объеме. И наконец, сформировать

и самостоятельно описать перечень всей сопроводительной документации.

Есть руководители проектов, которые отдают предпочтение именно этому подходу и успешно применяют его на деле. Но есть и другие руководители проектов, которые не находят этот вариант целесообразным и идут по другому пути.

В этом случае для реализации новых функциональных возможностей создаваемых систем дорабатывается уже имеющееся в эксплуатации прикладное и низкоуровневое программное обеспечение. Такой подход называется реинжинирингом систем.

Прежде чем начать разработку новой системы, необходимо выполнить тщательное документирование наследуемой системы или тех частей нескольких систем, которые будут унаследованы. На основе анализа имеющейся информации о наследуемой системе составляется ее модель. Причем функциональные возможности, реализованные в наследуемой системе, определяют часть требований к проектируемой (и надо сказать, в большинстве случаев это достаточно значительная часть) [3].

При таком подходе, разумеется, также выполняется довольно большой объем работ. Однако применение реинжиниринга позволяет перевести сам процесс разработки и поддержки готовых систем на новый качественный уровень и обеспечить возможность масштабирования таких систем. Здесь важно принимать во внимание следующие моменты:

- Проведение инжиниринга. Это довольно объемная работа, но ее выполнение, по сути, не так уж сложно.
- Затраты на обоснование требований к системе и их анализ. Возрастающий объем дополнительных знаний, собранных об исследуемой системе, приводит к необходимости их группирования в домены.
- Формируемые домены должны обладать достаточной гибкостью, позволяющей сделать процесс определения доменов и их диалектов быстрым. Это существенно, поскольку, как правило, код приложений бывает написан на нескольких разных языках.
- Поскольку компьютерам свойственна медленная обработка символьных вычислений, нужно ее распараллеливать. Используя специальные инструменты, можно провести качественный анализ ситуации и внести улучшения.
- Устаревшие системы слишком громоздки (порядка десятков тысяч файлов исходного кода). При обновлении системы можно добиться оптимизации объема используемого исходного кода.
- Для экономии задействованного рабочего пространства необходима тщательная проработка модели системы. А значит, нужно обеспечить устойчивую и достоверную технологию синтаксического анализа (парсинга), универсальную для произвольных языков кодирования.
- Среди других моментов: возможность учета различных версий программных продуктов, работа в условиях большого штата команды разработчиков, гарантия предоставления долгосрочных обязательств по поддержке поставляемых продуктов.

Цели реинжиниринга

При проведении реинжиниринга важно получить представление о структуре и наполнении существующей системы, построить ее модель, решить, какие фрагменты программного кода могут быть использованы в наследуемой системе, а также определить характер и объем наследуемых данных.

Задачи реинжиниринга

В процессе реинжиниринга необходимо определить архитектуру и логическую структуру существующей системы, спектр функциональных возможностей и целевую аудиторию системы, провести восстановление реляционной модели данных.

Этапы реинжиниринга

Процесс реинжиниринга, с учетом его применения для встраиваемых систем, можно разделить на несколько этапов:

- Сбор данных. В начале процесса следует определиться с тем объемом информации по уже существующим реализациям, который удалось получить в распоряжение до начала работ (исходные описания, тесты, базы данных и т. д.). При этом текущее состояние наследуемой системы фиксируется, и все изменения, которые могут быть в нее внесены после этого момента, в ходе реинжиниринга уже учитываться не будут.
- Определение системных архитектур. Здесь определяется перечень и структура всего базового оборудования и программной платформы, что необходимо для инсталляции и запуска наследуемой зафиксированной системы. Фактически так определяются архитектуры аппаратной и программной платформ, баз данных, телекоммуникаций.
- Автоматический реинжиниринг. На этом этапе на помощь приходят специализированные инструментальные средства визуального моделирования. Строятся модели, которые могут быть приняты в качестве исходных. Подвергаются автоматическому реинжинирингу и базы данных, и бизнес-логика (если есть соответствующие исходные коды на объектно-базированном или объектно-ориентированном языке), и описания разрабатываемой аппаратуры, и т. д.
- Редактирование диаграмм моделей. Полученные на предыдущем этапе, автоматически созданные модели читать и анализировать достаточно затруднительно, поэтому их необходимо отредактировать для большей наглядности и удобства восприятия.
- Построение функциональных моделей. Функциональные модели строятся на основе работающей наследуемой системы и проводимых с ней экспериментов. Здесь требуется достаточно точно учитывать варианты использования системы, допустимые действия и их последовательность. Все эти сведения помогут разработчикам более точно понимать функциональные возможности системы.
- Определение видов и характера взаимодействий системы. Необходимо описать, кто будет использовать систему, какие задачи она должна решать и какие функции выполнять для разных групп пользователей. Также нужно учесть, с какими программами или с каким внешним оборудованием осуществляет взаимодействие система. Определить, выполнение каких работ носит регулярный, периодический или условный характер.
- Детализация функциональных возможностей. На основе имеющихся моделей наследуемой системы и с учетом полученных данных о видах и характере взаимодействий системы выполняется детализация функциональных возможностей.

Модели, полученные в результате проведения реинжиниринга, служат хорошей основой для определения требований к новой проектируемой системе. Также они являются базой для построения функциональной и логической моделей новой системы.

Инструментальные средства автоматизации реинжиниринга

Среди существующих инструментальных средств, предназначенных для автоматизации проведения реинжиниринга на различных его этапах, широко востребованы средства автоматизации преобразования исходного кода как для прикладных, так и для низкоуровневых программ. С их помощью автоматически выполняется переход от устаревшей платформы, на которой была разработана система, к более усовершенствованной и подходящей для реализации новых функциональных возможностей в конкретной проектируемой системе.

Осуществляется перевод кода исходных файлов с одного языка на другой, а также переход с одной версии ОС на другую.

Здесь следует напомнить, что основной особенностью реализации встраиваемых систем является, как правило, применение нестандартных или интерпретируемых языков кодирования или же нестандартных сред исполнения. Ведь именно так реализовано огромное число широко востребованных программ.

Ввиду большого числа нюансов и деталей, характерных для нестандартных сред и языков, обычные универсальные средства автоматического перевода и анализа кода перестают здесь работать. Так что подобрать подходящий инструмент для получения до-

стоверного результата зачастую бывает весьма затруднительно.

Нередко возникают задачи перевода кода файлов аппаратного описания проектов с VHDL на Verilog или обратно. Но в любом случае в данной статье разбирается общий подход к переводу кода с одного языка на другой, который подходит и может быть применен к адаптации как прикладных, так и «железных» частей разрабатываемой системы.

Для того чтобы лучше разобраться в данном вопросе, нужно получить некоторое представление о принципах работы и организации систем автоматического преобразования программного кода.

Системы преобразования кода, имеющие промышленное применение

К системам преобразования кода, имеющим промышленное применение, предъявляются следующие требования:

- Для используемых на практике языков программирования и описания данных обеспечивается поддержка определений этих языков.
- Предоставляется поддержка правил переноса кода для таких языков.
- Поддерживается возможность достоверного применения этих правил к исходному коду.
- Осуществляется поддержка работы таких систем преобразования кода на востребованных потребителем аппаратных и программных платформах (в том числе на платформах, используемых для встраиваемых программных систем).

Также следует учитывать особенности реализации используемых программных компиляторов. Многие инструментальные средства компиляции (например, YACC) применяют синтаксические анализаторы (парсеры) с алгоритмами типа LL(1) или LALR(1), которые работают только с очень ограниченным числом языков. Для обхода таких ограничений большинство компиляторов и инструментов имеют, как правило, специально модифицированные парсеры. Это означает, что для широкого диапазона языков программирования сама по себе инфраструктура компилятора не является подходящей.

Таким образом, возникает необходимость получить полностью независимый от контекста механизм парсинга кода. На рынке современных программных средств представлен небольшой ряд подобных систем, среди них можно назвать такие, как REFINE (коммерческий продукт от компании Reasoning Systems [4]), Stratego/XT 0.9 (исследовательский инструмент от Stratego [5]), DMS (коммерческий продукт от компании Semantic Designs [6]).

Далее в статье основное внимание будет уделяться системе DMS Software Reengineering Toolkit, которая, по мнению автора, по своей специфике больше всего подходит для реинжиниринга аппаратуры и программ, разрабатываемых для встраиваемых систем, и ориентирована на работу, в первую очередь, именно с ними.

Основные отличия систем преобразования кода, имеющих промышленное применение, от инструментальных средств компиляции:

- конфигурируемость и высокая надежность используемой технологии синтаксического анализа (парсинга);
- интеграция этой технологии с шаблонными языками, используемыми для перезаписи исходного кода;
- возможность подробного восстановления кода исходных программ из структуры абстрактных синтаксических деревьев, полученных в результате преобразования.

Эти функциональные возможности используются для полномасштабного реинжиниринга систем (например, портирование кода), анализа качества программного кода и его улучшения (например, обнаружение и удаление клонов кода), обратного инжиниринга и других целей, в частности для осуществления долгосрочной поддержки разрабатываемого проекта.

Для практического применения таких систем преобразования в первую очередь должен быть определен синтаксис используемых языков. Поскольку конфигурировать эти инструменты достаточно удобно, гораздо легче производить их настройку, чем заниматься дополнительным построением внешних интерфейсов компиляторов. К тому же эти системы зачастую совместимы с предопределенными языковыми модулями большинства языков, таких как C, C++, Java, VHDL и т. д.

Механизм преобразования кода

Следует уточнить, что преобразование программного кода (так называемые правила перезаписи кода) используется для модификации программ непосредственно в терминах синтаксиса языков программирования. Такая перезапись, как правило, регламентируется в терминах абстрактной или конкретной выбранной грамматики языка и применяется ко всем формируемым абстрактным синтаксическим деревьям. (Все же другие программные преобразования могут быть реализованы уже с помощью процедурного кода или комбинации из нескольких поэтапных преобразований кода.)

Итак, типичное правило перезаписи кода имеет следующий общий вид:

```
LHS→RHS if <условие>
```

Здесь и LHS (left hand side, левая сторона правила), и RHS (right hand side, правая сторона правила) представляют шаблоны исходного языка, которые соответствуют произвольным подстрокам того или иного формализованного языка.

Условие, включаемое по оператору ***if***, является необязательной составляющей и зависит от переменных, использованных в шаблоне LHS.

Эти правила интерпретируются так: если конкретная часть исследуемой программы соответствует шаблону LHS, то заменить его на шаблон RHS, если соблюдается указанное условие.

Условие может содержать некоторые дополнительные ограничения или же вызывать дополнительную процедуру, по которой принимается решение о соблюдении условия.

Настоящие системы преобразования кода содержат ряд более сложных правил, базирующихся на этом примитивном примере. И именно таким образом в них реализуется возможность спецификации всех деталей описываемых шаблонов.

Приведем пример. Вот как в системе преобразования DMS выглядит упрощенный пример конвертации оператора присваивания в оператор автоинкремента на языке C (листинг 1). Практический смысл этого действия заключается в оптимизации кода за счет использования более быстрых одноместных операций. Правило записано с помощью синтаксиса внутреннего языка записи правил системы DMS.

```
default base domain C; //имя домена

rule use-auto-increment (v: lvalue):
        statement -> statement =
    "\v = \v +1" //синтаксис домена (LHS)
    rewrites to
    "\v++" //синтаксис домена (RHS)
    if no_side_effects(v); //условие правила
```

Листинг 1. Правило перезаписи кода в системе DMS

В данном примере определено правило (rule) с именем use-auto-increment и синтаксической переменной v синтаксического класса lvalue.

По этому правилу выполняется преобразование выражения statement, записанного на языке C.

Текст, заключенный в кавычки, соответствует допустимому исходному коду на языке C. Здесь используется регулярное выражение, в котором экранированный символ v (то есть сочетание \v) используется для указания переменной. Таким образом, правило становится универсальным и подходит для произвольных допустимых переменных исходного кода на языке C.

Первая приведенная строка, заключенная в кавычки, соответствует левой стороне правила, LHS, а вторая — правой стороне правила, RHS.

Выражение LHS действительно для любой допустимой переменной типа lvalue языка C. RHS же соответствует любому допустимому выражению, добавляющему единицу к переменной типа lvalue.

Для однозначного толкования правила (во избежание соответствия исходного и целевого шаблонов) в выражении LHS одна и та же синтаксическая переменная исполь-

зована дважды, и ее появление в шаблонной последовательности строго определено.

А в выражении RHS синтаксическая переменная использована для указания, что должна быть заменена часть кода, совпавшая с шаблоном в выражении LHS.

И наконец, условие в данном правиле — это зависимое от языка кодирования конкретное выражение проверки состояния. В данном примере выполняется проверка: не содержит ли фрагмент программы, совпавший по \v, каких-либо побочных эффектов (no side effects), что может сделать данное преобразование некорректным.

До начала применения каждого правила типовой механизм перезаписи кода сначала выполняет синтаксический анализ (парсинг) такого правила в соответствии с встроенным языком описания правил. А затем уже выполняет парсинг шаблона, заключенного в кавычки, в соответствии с правилами преобразуемого языка кодирования (в данном примере в качестве дефолтного домена (default domain) указан преобразуемый язык C). В результате такой обработки формируется дерево шаблонов.

В процессе преобразования механизм перезаписи кода сопоставляет шаблон LHS с отдельными частями программы и, в случае удовлетворения указанного условия, заменяет совпадающие блоки соответствующим шаблоном RHS.

Результат применения правила, приведенного в листинге 1, проиллюстрирован в листинге 2:

```
До:     (*Z)[a>>2]=(*Z)[a>>2]+1;
После:  (*Z)[a>>2]++;
```

Листинг 2. Результат преобразования кода

Обычно система преобразования кода содержит большое число правил, которые имеют большое число возможных мест применения в исследуемой программе.

То, каким именно образом система преобразования осуществляет выбор конкретных правил и точных мест их применения в программах, является достаточно сложным процессом, требующим отдельного описания, и в данной статье не рассматривается.

Теперь, получив некоторые общие знания о принципах работы систем преобразования программного кода, можно перейти к рассмотрению реализации одной из таких систем, перечислить ее функциональные возможности, указать преимущества и привести некоторые примеры практического применения.

DMS Software Reengineering Toolkit

Программный пакет DMS Software Reengineering Toolkit представляет собой набор инструментов для проведения реинжиниринга программных систем, настраиваемый под нужды конкретной задачи. (Имеются в виду такие виды работ, как извлечение документации, анализ, портирование, преобразование и модификация кода, а также смена интерфейса или другие крупные регулярно проводимые изменения.) Также предоставляется дополнительная возможность генерации доменно-зависимого программного кода.

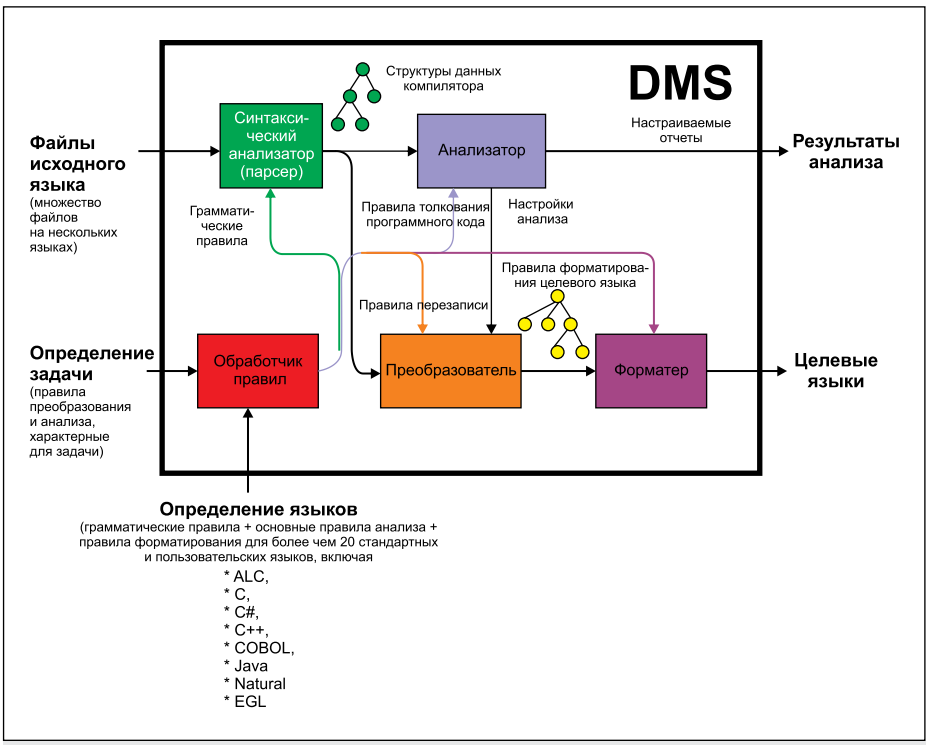

Рис. 1. Упрощенная модель системы преобразования кода DMS

Одно из важных преимуществ программного пакета DMS — полное отсутствие зависимости от конкретного языка кодирования. Благодаря этому DMS взаимодействует с популярными языками разработки аппаратуры (например, Verilog и C). Также возможна одновременная работа с использованием нескольких таких языков, а значит, DMS можно считать удобным средством для проведения совместной разработки аппаратуры и программ.

DMS поддерживает работу с программами разных уровней. Это могут быть языки программирования (такие как C и C++), языки разметки (HTML и XHTML), языки описания разрабатываемой аппаратуры (Verilog и VHDL), языки описания данных (SQL и PL) и различные доменно-зависимые языки [7].

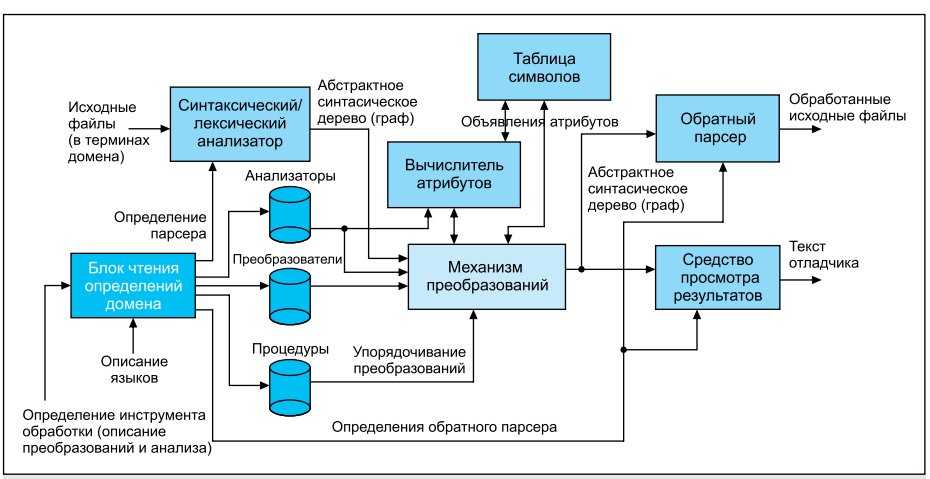

Рис. 2. Схема реализации системы преобразования кода DMS

Упрощенная модель системы DMS (рис. 1) может быть представлена как предельно обобщенный компилятор, который включает синтаксический анализатор (парсер), семантический анализатор, механизм преобразования программы (средство генерирования и оптимизации кода) и компоненты форматирования итогового выходного кода (формирующие код исходных файлов вместо двоичного кода).

Для стандартных компиляторов характерно, что каждая их компонента, как правило, зависит от исполняемых им функций или определяется контекстом перевода кода с конкретного исходного языка к виду целевого машинного кода. В отличие от таких компиляторов, в DMS каждая компонента имеет большое число настраиваемых параметров, что позволяет добиться значительной гибкости системы преобразования кода и подстроиться под требования конкретной решаемой задачи.

На рис. 2 приведен вариант схемы реализации системы DMS с набором настраиваемых инструментов.

По сути, система DMS поддерживает широкий спектр возможностей по выбору входного языка, виду проводимого анализа, типу выполняемого преобразования, а также настройке вида выходных данных. И такая гибкость имеет особое практическое значение.

Также, в отличие от стандартных компиляторов, DMS может выполнять одновременную обработку большого числа файлов, написанных на разных языках. Таким образом, обеспечивается возможность проведения анализа и/или последовательного изменения кода для крупных программных комплексов.

Промышленное применение

Возможности системы DMS позволяют ей решать многие задачи, стоящие перед разработчиками промышленных систем. Среди них анализ качества, реструктуризация, автоматическое портирование, генерирование удобочитаемого и хорошо оптимизированного программного кода. Проведенные промышленные испытания показали, что DMS способна работать с крупномасштабными программными системами со следующими характеристиками:

- объем исходного кода или спецификаций может доходить до нескольких миллионов строк;
- число исходных обрабатываемых файлов может исчисляться десятками тысяч;
- одновременно для описания исходного кода может быть задействовано несколько языков кодирования.

Для обеспечения достаточной вычислительной мощности системы DMS, удовлетворяющей таким высоким требованиям, нужно учитывать следующую особенность. Если DMS запускается на однопроцессорной аппаратной платформе, то она работает с одной скоростью, а если DMS запускается на аппаратной платформе с симметричной многопроцессорной обработкой (Symmetric MultiProcessing, SMP), то можно добиться более высокой производительности.

Например, при такой реализации процесс обработки атрибутов будет распараллеливаться автоматически. И соответственно, применение N-процессорной SMP-платформы зачастую может обеспечить значительное линейное ускорение работы системы.

Используемые платформы

Система DMS и разработана именно для запуска на таких платформах с симметричной многопроцессорной обработкой, базирующихся на архитектуре x86.

При разработке системы DMS принят во внимание целый ряд требований к реализации, учитывающих важные потребности современных потребителей таких систем:

- DMS запускается на всех разновидностях ОС семейства Windows.
- В качестве аппаратной платформы рекомендуются к применению однопроцессорные или мультипроцессорные SMP-системы от Intel или AMD.
- Используя возможности приложения Wine, DMS запускается на ОС семейств Linux, Solaris и MAC OS X.

Поддерживаемые языки

Для быстрого внедрения системы DMS ее разработчики реализовали поддержку внешних интерфейсов (так называемых доменов) для большинства востребованных языков кодирования. Это позволяет создавать настраиваемые пользовательские компиляторы, удобные инструменты анализа или инструменты достоверного преобразования кода. Перечень этих языков (приводится в порядке востребованности среди разработчиков встраиваемых систем) включает:

- C (диалекты Microsoft, ANSI, GNU, C99). Здесь реализованы поддержка препроцессорных директив, максимальные разрешения для имен и типов, анализ потоков данных и потоков управления, а также построение графов общесистемных вызовов.
- C++ (диалекты Microsoft, ANSI, GNU). Здесь реализованы поддержка препроцессорных директив и максимальные разрешения для имен и типов.
- C# (язык .NET от Microsoft). Поддерживаются версии 1.2, 2.0, 3.0 и 4.0.
- SystemC 2.1.
- Verilog 1995, 2001.
- SystemVerilog 3.1a.
- VHDL 1993.
- HTML 4.0, XHTML, включая диалект IE.
- XML.
- Java 1.1, 1.2, 1.3, 1.4, 1.5, 1.6. Здесь реализованы максимальные разрешения для имен и типов (включая файлы классов и JDK), анализ потоков данных и потоков управления, а также построение графов вызовов.
- SQL ANSI 2011.
- MATLAB M-файлы и Simulink.
- PHP3, PHP4 и PHP5.
- Python 2.6 и 3.1.
- PL/SQL (язык программирования баз данных Oracle Database).
- Delphi 6 (“Borland ObjectPascal”).
- Rational Rose UML (файлы.MDL).
- Visual Basic (VB.net, VB6, VBScript+ASP).
- Ada 83/95.
- COBOL 85 (“ANSI”), IBM VS COBOL II, IBM Enterprise COBOL. Здесь реализованы поддержка препроцессора, управление COPYLIB, функция построения отчетов, максимальные разрешения для имен и типов, а также анализ потоков данных и потоков управления.
- CFEngine v3. Реализована поддержка обеспечения политики безопасности и конфигурации системы.
- ECMAScript (ECMA-262, JavaScript [диалекты Microsoft и Netscape], ActionScript, ActionScript2, ActionScript3, ASP, JSP, HTML и XML).
- EGL и VAGen (IBM).
- FORTRAN 95/90/77.
- HLASM (IBM).
- JCL (IBM).
- JOVIAL (язык встраиваемых систем для военного применения), с полной поддержкой анализа имен и типов.
- Mathematica.
- IDL (Corba 2.3).
- Структурированный текст в формате стандарта IEC 61131-3 (языки управления промышленной автоматизации).
- Язык ассемблера для оборудования Motorola M6800/M6801/M6805/M6808/M6809/M6811/M6812.
- PARLANSE.
- Natural (язык программирования мейнфреймов) и Adabas (DDM-файлы).
- Pascal (ISO 7185).
- Perl 5.
- Pick Data Basic (универсальный диалект).
- PL/1 (IBM).
- Progress aka OpenEdge (4GL).

Функциональные возможности внешних интерфейсов поддерживаемых языков

В системе преобразования кода DMS для внешних интерфейсов (доменов) поддерживаемых языков предоставляются следующие функциональные возможности.

- Выполняется полный лексический анализ, включая чтение исходных файлов в кодировках ASCII (ISO8859-1) и UNICODE.
- Все грамматические правила указываются в явном виде и базируются непосредственно на технической документации и стандартах.
- Автоматическое построение полного абстрактного синтаксического дерева. Здесь учитываются все использованные комментарии и форматы именованных величин. Также обеспечиваются межпроцедурный и межфайловый анализ и преобразование кода (на одном или нескольких языках) в контексте одного рабочего пространства.

- Автоматическое конструирование системы трансформации исходного кода по принципу source-to-source, а также возможность определения настраиваемой грамматики и атрибутов для создания пользовательских анализаторов.
- Поддержка большого числа диалектов различных языков кодирования.
- Устойчивость системы и высокая достоверность результатов выполненного преобразования кода благодаря тщательному тестированию, реализованному во время разработки, и бэта-тестированию, проведенному большим числом пользователей во время практического применения системы.

Практика использования DMS для преобразования кода

Компания Semantic Designs, разработчик системы DMS, в 2009 году получила от компании Northrop Grumann, производителя бомбардировщиков B-2, ответственное задание по созданию средств модернизации управляющих программ и электронной начинки для B-2 с целью продления срока их службы.

Специалистами Semantic Designs был разработан один из важнейших компонентов модернизации: транслятор JOVIAL2C, который позволяет выполнять автоматизированную трансляцию старого кода управляющих программ, написанного на специализированном языке Jovial, в код на более привычном и распространенном языке C [8].

Созданный транслятор предоставляет возможность переписать заново программное обеспечение самолета и интегрировать все его системы — как старые, так и новые. Помимо этого, перевод систем управления на язык C поможет быстрее обновлять управляющие программы по мере необходимости.

Конечно, для большинства современных разработчиков встраиваемых систем задача обработки кода на таком достаточно устаревшем языке, как Jovial, не является самой востребованной. Но в данном случае у автора есть только одна задача — показать возможность трансляции кода, пусть даже и с такого архаичного языка. В качестве альтернативы, в частности, можно было бы представить обработку ассемблерного кода для ADSP2181 или его аналога. Тем не менее автор приводит данный пример использования системы преобразования кода DMS, поскольку он взят из реальной практики и довольно показателен: описываемый подход в разработке можно применить даже в столь нестандартной ситуации.

Итак, приведем практический пример, в котором с помощью средств системы DMS выполняется преобразование кода с устаревшего языка Jovial на современный универсальный C, что продиктовано требованиями модернизации конкретной программной платформы.

В листинге 3 приведен пример задания нескольких правил перевода кода с Jovial на C:

```
default source domain Jovial; //имя домена
default target domain C;

private rule refine_data_reference_dereference_NAME
      (n1:identifier@C,n2:identifier@C)
        :data_reference->expression
  = "\n1\:NAME @ \n2\:NAME" → "\n2->n1".

Private rule refine_for_loop_letter_2
      (lc:identifier@C,f1:expression@C,
      f2:expression@C,s:statement@C)
            //переменные шаблона
        :statement-> statement
  = "FOR \lc\:loop_control :
    \f1\:formula BY \f2\:formula; \s\:statement"
            //синтаксис исходного домена
    ->
    "{ int \lc = (\f1);
       for(;;\lc += (\f2)) { \s } }"
            //синтаксис целевого домена
    if is_letter_identifier(lc).
```

Листинг 3. Набор правил для преобразования кода с Jovial на C

А вот как выглядит результат преобразования (листинг 4):

```
Исходный код на языке Jovial:

FOR i: j*3 BY 2 ;
  x@mydata = x@mydata+I;

Результат перевода кода на язык C:

{ int i = j*3;
  for (;;i+=2)
    { mydata->x = mydata->x + i}
}
```

Листинг 4. Результат применения правил для преобразования кода с Jovial на C

Как правило, для полного перевода кода с языка Jovial выполняется множество подобных небольших преобразований с использованием порядка 2500 правил перевода кода.

Для большей наглядности приведем комплексный пример преобразования кода с Jovial на C.

Вот как выглядит исходный код на языке Jovial (листинг 5). Здесь используются вложенные таблицы с битовым смещением, определением типов, функций, строковых операций и комментариев.

Вот результат перевода кода на язык C (листинг 6). Это эквивалент кода из листинга 5 с применением написанного вручную макроса.

Из примера наглядно видно, что преобразование кода, помимо непосредственного перевода с языка на язык, может обеспечивать и значительное усовершенствование кода.

Окончание следует

```
START
   TABLE TFP'D'TWRDET (1:109,12:37);
   BEGIN
      % Main status boolean %
      ITEM TFP'G'TWRDET STATUS (V(YES),V(NO));
   END
   TYPE TFP'D'TWRDET'TABLE TABLE (7:23) W 3;
      BEGIN
         ITEM TFP'ITM S 3 POS(0,3); "cube axis"
      END

%begin proc%
PROC PROC'A(c1) S;
   BEGIN
      ITEM match'count U 6;
         %an item%
      ITEM c1 C 5; "parameter value"
      ITEM c2 C 7;
      IF c1 <= c2 AND c2 > c1;
         match'count = UBOUND(TFP'D'TWRDET,0) + UBOUND
(TFP'D'TWRDET'TABLE,0);
      "result off by 1 so adjust"
      match'count = match'count+1;
      BEGIN
         match'count = match'count/2;
         PROC'A = match'count; % return answer %
      END "cleanup and exit";
END "end proc"

TERM
```

Листинг 5. Исходный код на языке Jovial, до преобразования

```
#include "jovial.h"
static struct
   { /* Main status boolean */
      enum { V(yes$OF$tfp_g_twrdet$OF$tfp_d_twrdet),
         V(no$OF$tfp_g_twrdet$OF$tfp_d_twrdet) }
            tfp_g_twrdet _size_as_word;
         } tfp_d_twrdet[109][26];
typedef union
   { W(3);
   struct
      { POS(0, 3) S(3) tfp_itm:4 _align_to_bit; /* cube axis */
      };
   } tfp_d_twrdet_table[17];
static S proc_a(C(5) c1);
/* begin proc */
static S proc_a(C(5) c1)
{ __typeof__(proc_a(c1)) RESULT(proc_a);
   _main:
      { U(6) match_count;
         C(7) c2;
         if (CHARACTER_COMPARE(BYTE_CONVERT(C(7), c1, 7),
c2) <= 0
            && CHARACTER_COMPARE(c2, BYTE_CONVERT(C(7),
c1, 7)) > 0)
         match_count = UBOUND(tfp_d_twrdet, 2, 0) + 16;
         /* result off by 1 so adjust */
         match_count = (S(6))match_count + 1;
         { match_count = (S(6))match_count / 2;
            RESULT(proc_a) = (S(6))match_count; /* return answer */
         } /* cleanup and exit */
         ;
      }
   _return:
 return RESULT(proc_a);
} /* end proc */
```

Листинг 6. Полученный код на языке C, после преобразования

Литература

1. Сергеева А. Тестирование работоспособности промышленного компьютера // Компоненты и технологии. 2014. № 2.
2. Сергеева А. Одновременная разработка программ и аппаратуры для встраиваемых систем при помощи симулятора аппаратуры Vista Virtual Prototyping // Компоненты и технологии. 2014. № 3.
3. Chikofsky E., Cross J. Reverse Engineering and Design Recovery: A Taxonomy. — IEEE Software, 1990.
4. www.reasoning.com
5. www.strategoxt.org
6. www.semdesigns.com
7. Об инструментарии DMS Software Reengineering Toolkit — http://www.semdesigns.com/products/DMS/DMSToolkit.html
8. О проекте JOVIAL2C — http://www.tgdaily.com/technology/42426-us-upgrades-stealth-bomber-software