УДК 004

ИСПОЛЬЗОВАНИЕ RATIONAL RHAPSODY ДЛЯ ЭФФЕКТИВНОЙ РАЗРАБОТКИ И ТЕСТИРОВАНИЯ МОДЕЛИРУЮЩИХ СИСТЕМ НА БАЗЕ UML

Губский А.Е. Андрюхин А.И.
Донецкий Национальный Технический университет
кафедра прикладной математики и информатики
oleksandr.gubsky@gmail.com

Рассмотрены основные возможности среды визуального проектирования и генерации программ IBM Rational Rhapsody для систем реального масштаба времени и встраиваемых систем.

Общая постановка проблемы

Для соответствия рыночным требованиям и сохранения лидирующих позиций важно использовать самые современные инструменты разработки. За последнее десятилетие основой таких инструментов стал язык UML, признанный стандартом де-факто в создании сложных систем и программного обеспечения. Безусловно, самым эффективным инструментом, реализующим его возможности с учетом особенностей встраиваемых систем, является среда разработки IBM Rational Rhapsody.

Исследования

Использование Rational Rhapsody для эффективной разработки и тестирования моделирующих систем на базе UML

The IBM Rational Rhapsody Developer - это среда разработки на основе моделей, использующих языки UML и SysML. Ее основным назначением и применением является проектирование, отладка и тестирование программного обеспечения систем, работающих в реальном масштабе времени, встроенных систем.

IBM Rational Rhapsody Developer обеспечивает возможность ранней проверки алгоритма работы встроенных систем и программных решений для выявления недостатков на начальных стадиях жизненного цикла программного продукта. Используя средства быстрого создания прототипов, визуальной отладки и выполнения модели, с помощью Rational Rhapsody Developer можно быстрее получить финальные программные продукты, соответствующие заявленным требованиям.

Перечислим основные возможности системы Rhapsody:

- Визуальная среда разработки встроенного программного обеспечения с использованием UML и SysML.
- Имитационное моделирование позволяет воплотить в жизнь диаграммы для отладки на уровне проекта и проверки его правильности на ранних этапах.
- Генерация кода на C, C++, Java и Ada на основе диаграмм состояний для повышения эффективности разработки.
- Генерирование кода для многоядерных процессоров и визуализация многоядерного выполнения для упрощения разработки параллельных приложений.
- Генерирование кода и компоновка файлов для ведущих сред разработки встроенных программ и программ, работающих в реальном масштабе времени.
- Визуализация кода C#, полученного из Microsoft Visual Studio, и генерация кода C# с помощью Rational Rhapsody.
- Интеграция со средой разработки Eclipse позволяет создать единую среду программирования, моделирования и отладки программного обеспечения.
- Импорт существующего кода C, C++, Java и C# для визуализации и документирования.
- Гибкая среда разработки синхронизирует код и модель и позволяет выбрать удобный

способ работы.
- Отслеживание требований при проектировании, программировании и тестировании в интегрированной среде.
- Разработка программного обеспечения на хосте до получения целевого аппаратного обеспечения.
- Среда с реальным масштабом времени поддерживает выполнение 8-, 16-, 32- и 64-разрядных приложений.
- Автоматическое обеспечение согласованности архитектуры, проекта, кода и документации.
- Создание архитектуры приложений Data Distribution Service for Real-Time Systems (DDS) для управления сложными взаимосвязанных компонентов.
- Автоматизация документирования на всех этапах жизненного цикла продукта за счет интеграции с Rational Publishing Engine.
- Разработка приложений для автомобильной промышленности с использованием AUTOSAR — от выработки концепции до написания исходного кода.
- Использование профиля MARTE для разработки архитектуры многоядерных приложений.
- Возможности совместной работы благодаря использованию функций выделения различий и объединения на основе моделей, включая интеграцию с решением IBM Rational Team Concert на базе Jazz.
- Автоматизация тестирования на основе моделей с помощью дополнительного модуля Rational Rhapsody TestConductor.
- Предусмотрено три варианта поставки Rational Rhapsody Developer для различных языков программирования. Rational Rhapsody Developer for C++, C и Java предназначен для пользователей, которым необходима среда разработки на C++, C, Java и C#. Разработчики на Ada могут использовать вариант Rational Rhapsody Developer for Ada, поддерживающий только язык Ada, либо вариант Rational Rhapsody Developer, поддерживающий разработку на всех поддерживаемых языках.
- Расширяемые и настраиваемые возможности моделирования и генерации кода.
- Система Rhapsody функционирует в операционных системах Linux и Windows.

Столь широкая и разнообразная функциональность системы Rhapsody дает возможность создания программного обеспечения на более высоком качественном уровнею. Это позволяет создавать программы, затрачивая меньше времени и ресурсов на разработку, проектирование и тестирование.

Система Rhapsody предоставляет системным инженерам, разработчикам и отладчикам ПО общую среду разработки на основе визуального моделирования, в которой можно проанализировать требования, спроектировать систему и ПО, сгенерировать и разработать приложение, а также быстро, эффективно и своевременно протестировать текущие результаты на любом этапе процесса разработки: от анализа требований до готовой встраиваемой системы.

Model Driven Architecture (MDA) – создаваемая консорциумом OMG концепция модельно-ориентированного подхода к разработке программного обеспечения. Его суть состоит в построении абстрактной метамодели управления и обмена метаданными (моделями) и задании способов ее трансформации в поддерживаемые технологии программирования (Java, CORBA, XML и др.). Создание метамодели определяется технологией моделирования MOF (Meta Object Facility), являющейся частью концепции MDA.

Для того чтобы более детально разобраться в концепции MDA, для начала давайте взглянем на популярную методологию разработки программного обеспечения под названием RUP (Rational Unified Process). В соответствии с этой методологией, цикл разработки состоит из следующих основных шагов:
- Выработка и согласование требований
- Анализ
- Дизайн
- Реализация

– Тестирование
– Ввод в эксплуатацию

Большинство программ претерпевает изменения как во время разработки, так и в течение своей эксплуатации. Появляются новые требования, обнаруживаются ошибки реализации, становятся очевидными недостатки текущего дизайна. Классический процесс требует для внесения этих изменений повторить шаги 1-6. К сожалению, для небольших изменений часто есть соблазн их исправить на фразе реализации, не возвращаясь к фазам анализа и дизайна. Это может потенциально усложнить процесс поддержки развития и модификации программного продукта в будущем, так как текущая модель не соответствует текущему коду.

Модель, построенная на этапе дизайна (п.3), часто имеет большую ценность чем конкретная реализация и может пережить ее. Например, построение модели сложной системы может занять много человеко-месяцев кропотливого анализа предметной области, консультаций с экспертами, обработки тысяч страниц документации предметной области. Допустим, что после построения такой модели она была реализована в виде инсталлируемого продукта на языке C++. Через несколько лет с развитием интернет-технологий было принято еще реализовать этот продукт в виде веб-сервиса, на языке Java. Хорошо построенная модель позволит использовать большую часть работы по ее построению в новой реализации системы.

Тут мы впервые сталкиваемся с первым ограничением стандартного RUP процесса. Любая модель при ее построении часто несет на себе следы технологий под которые она разрабатывалась (в противном случае она будет слишком абстрактной для того чтобы быть использованной как документ на основании которого делается реализация). Например, в UML модели для C++ и для Java могут по-разному использоваться такие ОО концепции, как множественное наследование и интерфейсы (в Java нет множественного наследования реализации, а в C++ нет формального различия между классами и интерфейсами).

Как решение этой и других проблем, была предложена методология разработки ПО под названием MDA (Model Driven Architecture). В этой методологии различают два типа моделей: PIM — платформно независимая модель и PSM — платформно зависимая модель. MDA не специфицирует на каком языке описана PIM, но требует чтобы описание было на языке, который определен формально и пригоден к автоматической обработке. Для примера формального определения языков моделирования можно выбрать Meta Object Facility — специальный язык используемый OMG для определения других языков моделирования, в том числе UML. UML может быть использован для описания как PIM так и PSM. OCL может использоваться в дополнение к UML. При использовании UML в PSM-моделях скорее всего будет использоваться один из UML профилей, такие как, OMG CORBA Profile, Enterprise Application Integration Profile, UML/EJB Mapping, Common Warehouse Metamodel.

Основная идея MDA в том, что преобразование из PIM в PSM, а также же генерация кода из PSM, может производиться автоматически. Преобразования проводятся при помощи инструментов преобразования (transformation tools), которые в свою очередь используют правила преобразования. Эти правила будут написаны на языке, который будет описан стандартом QVT (Queries, Views, Transformations). Преобразования могут быть параметризованы, что позволит их подстраивать под нужды конкретных проектов.

Итак, на этапе анализа на основании требований вырабатывается платформно независимая модель системы (PIM). Она привязана к постановке задачи и предметной области и не зависит от таких деталей реализации, как, например, язык программирования или тип базы данных (реляционная, объектная, иерархическая и т.д.)

Далее, на этапе дизайна будет осуществлен выбор деталей реализации: платформ, языков, распределенной или централизованной архитектуры. На основании эти решений PIM будет преобразована в соответствующие платформо зависимые модели (PSM). Для этого преобразования, скорее всего, будут использоваться готовые инструменты преобразования и библиотеки правил преобразования. Из одной PIM может быть сгенерировано несколько PSM. Например, одна из PSM может основываться на CWM метамодели для реляционных баз данных и описывать модель данных. В тоже время другая модель может, используя UML/EJB Mapping, представить PSI в терминах

Enterprise Java Beans. Для стыковки разных PSM моделей в процессе PIM?PSM трансформации могут также быть сгенерированы так называемые «bridges» — связки между разными PSM моделями, сгенерированными из общей PIM модели.

Ну и наконец, из PSM при помощи других инструментов преобразований и других наборов правил будет сгенерирован код. Например, если Oracle был выбран как реляционная база данных, то будет использован набор правил, который из соответствующей PSM построит схему базы данных. Другой набор правил может сгенерировать Java код для EJB контейнера. Опять же, при использовании нескольких PSM моделей будут сгенерированы связки на уровне кода между ними, которые позволят сгенерированным Java Beans работать с сгенерированной схемой базы данных Oracle. Хотя преобразования будут делаться автоматически, выбор их параметров останется за дизайнером. Например, он может указать использовать ли определенные возможности платформы или нет, подсказать системе примерные объемы ожидаемых данных, и т.п.

В докладе многие из вышеуказанных достоинств **Rhapsody** рассматриваются при реализации итеративного и событийного методов моделирования.

Литература

[1] Rational Rhapsody - http://www-01.ibm.com/software/awdtools/rhapsody/

[2] Rhapsody - http://en.wikipedia.org/wiki/Rhapsody

[3] Основы UML и IBM Rational Rhapsody для разработчиков программного обеспечения http://www.swd.ru/index.php3?pid=158