Разработка систем реального времени с использованием UML и каркасов приложений

Дмитрий Рыжов
SWD Software Ltd.
email: d.ryzhov@swd.ru

Abstract (English)

Tool support for main concepts of Model driven development has brought a new era of UML based application development. Developers at last have received an opportunity for developing full applications based on models but not only generating their structure.

One of the key technology of these tools has became a behavior code generation based on UML 2.0 state and activity diagrams. In several tools a framework based approach to code generation is utilized. Framework implements all UML concepts and is used in the generated code.

Framework utilization has brought specialization of tools and their transformation to vertical solutions for application development in different areas. In this article we describe main concepts of Telelogic Rhapsody framework implementation. A key differentiator of this tool is a specialization on the development of real-time and embedded systems that determine particular features of the framework.

***Keywords**: Model Driven Development, UML concepts for real-time; behavior code generation; application frameworks.*

Abstract (Russian)

Поддержка основных концепций методологии разработки на основе моделей (MDD), реализованная в современных инструментах, стала началом новой эры использования языка UML при разработке систем и приложений. Разработчики программного обеспечения наконец получили возможность создавать полностью законченные приложения на основе моделей, а не только генерировать структуру их кода.

Одной из ключевых технологий данных инструментов является генерация поведенческого кода приложений на основе диаграмм поведения языка UML. Для этого в ряде инструментов успешно используется технология генерации кола на основе каркаса, предоставляющего реализацию концепций UML для разрабатываемого приложения.

Использование каркасов привело к специализации данных инструментов и их превращение в вертикальные решения для разработки приложений определенного типа. В данной статье рассмотрены основные концепции реализации каркаса приложений для инструмента разработки Telelogic Rhapsody. Особенностью данного инструмента является специализация на разработке встраиваемых систем и приложений реального времени, которая и определила особенности реализации каркаса.

***Keywords**: Методология разработки на основе модели, UML концепты реального времени; генерация поведенческого кода; каркасы приложений.*

1. Введение

Настоятельная необходимость применения унифицированного языка моделирования UML в качестве промышленного стандарта моделирования сложных систем способствовала развитию автоматизированных инструментов для упрощения процесса разработки на всех его стадиях: от анализа требований до функционального тестирования. Данные инструменты представляются особенно актуальными при разработке встраиваемых систем реального времени, поведенческие аспекты которых достаточно часто могут быть описаны посредством конечных автоматов языка UML. При этом, последние являются естественными

претендентами на автоматическую генерацию кода, тестирование, временной анализ и верификацию.

Существуют два основных подхода к генерации программного кода. Суть первого подхода заключается в генерации кода, предназначенного исключительно для компиляции, без необходимости использования этого кода разработчиками. Второй подход состоит в генерации кода, который был бы доступен для понимания, а по желанию и для последующей модификации разработчиками. Совершенно очевидно, что с точки зрения конечного пользователя, целесообразным является использование второго подхода. Однако автоматизированное создание читаемого программного кода представляет собой сложную задачу.

Одним из главных преимуществ объектно-ориентированного подхода является возможность использования проектирования основанного на абстракциях с возможностью повторного использования и легкой модификации. В рамках данного подхода при проектировании новых систем часто используют ранее разработанные каркасы приложений. **Каркас приложения** включает набор взаимодействующих классов, которые предоставляют необходимые сервисы при разработке систем определенного типа. Разработчики создают конкретные приложения на основе каркаса путем наследования классов каркаса и расширения их поведения. Использование каркаса является реализацией концепции объектно-ориентированного повторного использования существующего кода.

Среди основных преимуществ использования каркасов могут быть перечислены:

- Отсутствие необходимости создания новых приложений «с нуля», так как они повторно используют элементы каркаса.
- Каркасы *определяют* архитектуру целевых систем, обеспечивая набор предопределенных абстракций.
- Каркасы представляют собой открытые конструкции, так как могут наследоваться и переопределяться в конкретных приложениях.

В данном докладе описана стратегия генерации кода для приложений реального времени, реализованная в *Rhapsod*y, среде разработки на основе UML, разрабатываемой компанией Telelogic. Генерация кода в *Rhapsod*y осуществляется с использованием каркаса: в состав Rhapsody входит каркас OXF (Object Execution Framework), интерфейсы которого используются в сгенерированном коде, и который подключается при сборке приложения.

Например, с использованием каркаса код, сгенерированный для активного класса, включает в себя определенные для класса операции и их реализации, но функциональная возможность “активности” реализуется путем наследования от класса каркаса, реализующего данную функциональность. Это положение означает следующее:

- Каркас содержит ряд полезных абстракций для приложений реального времени, на которых основывается генерируемый код и которые придают конкретные значения концепциям UML (например, активным классам). Это, в свою очередь, облегчает понимание разработчиком сгенерированного кода.
- Значительная часть функциональности содержится в классах каркаса, а не создается каждый раз при генерации кода, что так же облегчает понимание программного кода.
- Благодаря механизму наследования, классы каркаса могут быть адаптированы под конкретные нужды разработчика и использованы при генерации кода.
- Каркас это библиотека, независимая от генератора кода. Классы каркаса могут быть использованы вне процесса генерирования кода.
- Каркас не ограничивает приложение от использования с сервисами каркаса сервисов операционной системы или других библиотек.

Ниже мы представим описание некоторых элементов каркаса OXF, имеющих отношение к вопросам реального времени. Нашим основным намерением является разъяснение подхода, применяемого в среде разработки Rhapsody в отношении значимых UML концептов реального времени, а также представить преимущества использования каркаса в качестве основы для генерирования кода. Мы будем использовать некоторые упрощения по сравнению с каркасом используемым в Rhapsody с целью облегчения понимания материала.

2. Многопоточные приложения

Многопоточные приложения моделируются в UML с помощью ***активных*** классов. Для обозначения активных классов используется стереотип active. На диаграммах UML активные классы отображаются двойной рамкой.

В каркасе *активный класс* представляет собой класс, который обладает потоком выполнения и

имеет функциональную возможность диспетчеризации событий и планирования таймаутов. Он представлен в каркасе как класс *OMActive*, от которого наследуется каждый активный класс в сгенерированном коде.

Активный класс *OMActive* содержит код, который управляет очередью событий. Он выполняет бесконечный цикл, выбирая события из очереди и рассылая их потребителям событий. Каждый пользовательский класс, который наследует от активного класса OMActive, по умолчанию приобретает этот поведенческий аспект.

С помощью механизма наследования возможна адаптация *OMActive* для реализации различных механизмов диспетчеризации событий. Например, можно определить класс *myActive*, который будет использовать две очереди событий вместо одной. Класс *myActive* наследуется от класса *OMActive* и переопределяет такие методы, как *execute()* и *queue()*. Rhapsody может быть настроена на использование класса *myActive* вместо *OMActive* при генерации кода. Это означает, что классы, помеченные как “активные”, будут автоматически наследовать от класса *myActive* вместо *OMActive*, а при сборке приложения будет подключаться адаптированный каркас.

Активный класс не обязательно является *потребителем* событий, но каждому потребителю необходимо иметь активный объект для управления своими входящими событиями. Назначение активных объектов осуществляется в среде Rhapsody с помощью установления ассоциаций между активными объектами и потребителями событий, называемыми *реактивными* объектами.

3. Генерация и обработка событий

За обработку событий в UML отвечают реактивные классы. ***Реактивным*** называется класс, способный реагировать на события, или, другими словами, являющийся потребителем событий. Такой класс представлен в каркасе классом *OMReactive*, от которого наследуется каждый реактивный класс в сгенерированном коде. Каждый реактивный класс имеет ассоциированный с ним активный класс.

Посылка события экземпляру реактивного класса осуществляется посредством вызова его операции *send()*, которая помещает событие в очередь в связанном с ним активном объекте при помощи вызова его операции *queue()*. Позднее, когда будут обработаны все предыдущие события из очереди, активный объект передает данное событие для обработки обратно в реактивный класс путем вызова его операции *handleEvent()*.

Процесс обработки событий, как правило, определяется диаграммой состояния для класса. В тоже же время, существует возможность определить произвольное поведение по обработке событий, переопределив операцию *handleEvent()* у реактивного класса.

Существует 3 типа пользовательских реактивных классов: *активные, пассивные и подчиненные*

- *Активный реактивный* класс сам управляет диспетчеризацией своих событий в своем собственном потоке (в этом случае пользовательский класс одновременно наследуется от *OMActive* и *OMReactive*).
- *Пассивный реактивный* класс передает свои события для диспетчеризации главному активному классу, который всегда создается каркасом. Диспетчеризация событий всех пассивных реактивных классов управляется этим активным классом.
- *Подчиненный реактивный* класс управляется назначенным пользователем активным классом (отличным от самого себя).

Назначение реактивному классу активного может осуществляться двумя способами:

- *Путем композиции*: реактивные классы, являющиеся частью активного реактивного класса управляются этим классом (если они сами не являются активными классами).
- *Путем явного назначения:* можно самостоятельно назначить реактивному классу активный класс, который будет выполнять диспетчеризацию его событий. Это осуществляется путем вызова метода *setEventManager()* для реактивного класса. В случае композиции, Rhapsody автоматически генерирует вызовы *setEventManager()* в конструкторах композитных классов.

4. Виды событий

Каждый класс может содержать операции и реагировать на события. **События** соответствуют неким внешним явлениям, которые влияют на поведение класса. Операции реализуют услуги или функциональные возможности, предоставляемые классом. В Rhapsody активные классы управляют диспетчеризацией событий реактивных классов, но не их операционными вызовами. Это означает, что выполнение операций всегда осуществляется в

потоке вызывающего объекта. Помимо прочего, существует еще ряд специальных событий, так называемых *событий вызова (triggered operation*s), которые также исполняются в потоке вызывающего объекта.

В Rhapsody различается три типа событий: сигналы, события времени и события вызова.

Сигналы – самые распространенные события, которые соответствуют асинхронным сигналам между объектами. В Rhapsody для обозначения сигналов используется термин *событие*. Они представлены в каркасе классом *OMEvent*, от которого наследуются все остальные события. События могут иметь параметры, реализуемые в виде атрибутов класса события. Кроме того, каждое событие ассоциируется со своим целевым (реактивным) объектом. Это позволяет активному объекту осуществлять диспетчеризацию события, направляя его соответствующему получателю когда до этого события дойдет очередь (напоминаем, что активный объект может управлять несколькими реактивными объектами). В связи с тем, что сигнальные события являются асинхронными, у них отсутствует возвращаемое значение.

События времени – события, сигнализирующие о том, что с некоторого момента прошло заданное количество времени. В Rhapsody для событий времени используется название *тайм-ауты*. Тайм-аут представляет собой событие особого рода и представлен в каркасе классом *OMTimeout*. Для создания тайм-аутов в каркасе используются активные объекты. Но они лишь создают тайм-ауты, а затем передают их менеджеру *тайм-аутов* (*OMTimeoutManager*)**.** Все активные классы используют единственный экземпляр менеджера тайм-аутов, который создается при инициализации каркаса и запускается в отдельном потоке. В любой момент времени менеджер тайм-аутов может содержать несколько тайм-аутов, события по которым должны быть сгенерированы при наступлении назначенного им времени. Менеджер тайм-аутов содержит *таймер*, который периодически уведомляет менеджера об истечении заданного промежутка времени. Каждый раз, когда менеджер тайм-аутов получает уведомление таймера, он проверяет тайм-ауты и помещает их в очереди событий в соответствующем активном объекте, откуда они рассылаются по адресатам (реактивным объектам), наравне с другими событиями в очереди. Объекты тайм-аутов сами по себе пассивны в том смысле, что они не содержат таймеров.

События вызова – такие (синхронные) операции, которые приводят к генерации одноименного события. Операции вызова могут выступать в качестве триггера для осуществления перехода в диаграмме состояний класса. Таким образом, их можно понимать как обычные операции, реализация которых обеспечивается диаграммой состояний. Для определения события вызова в Rhapsody используется тип *triggered operation*.

Когда некий объект вызывает такую операцию у реактивного объекта, производится создание одноименного события, которое сразу же передается реактивным объектом на обработку путем вызова своего метода *handleEvent()*, не помещая событие в очередь событий активного объекта. Выход из операции производится после завершения *handleEvent()***,** то есть после потребления события. Таким образом, события вызова являются событиями синхронного типа, диспетчеризация которых производится вызывающей стороной.

5. Реализация поведения

Rhapsody поддерживает конечные автоматы UML, разработанные на основе **диаграмм состояний** Harel и аналогичные им. Они включают в себя иерархическую декомпозицию состояния (ортогональные состояния и/или), события, несущие параметры, события времени, псевдосостояния (исходные состояния, история состояний, соединения, разветвления, переходы, выбор), завершающие переходы, действия на входе/выходе и прочие возможности. Кроме того, они включают в себя модель обработки событий в соответствии с определением в UML: каждый класс с диаграммой состояний является реактивным, следовательно, он имеет связанный с ним активный класс. Реактивный класс помещает события по мере их поступления в очередь активного класса, который осуществляет последовательную выборку событий из очереди и возвращает их обратно в реактивные классы для обработки в соответствии с диаграммой состояний.

Семантика конечных автоматов, или обработка событий в соответствии с диаграммой состояний объекта, реализуется посредством классов каркаса, таких как *OMEventConsumer*, *OMState*, *OMLeafState* и прочих. Описание этих классов не является предметом данной статьи.

Типы событий, поддерживаемых Rhapsody, были перечислены ранее. Также выше отмечено, что события времени реализуются в Rhapsody посредством *тайм-аутов* (*OMTimeout*), которые унаследованы от класса событий (*OMEvent*). Тайм-ауты могут быть использованы в качестве триггеров перехода из одного состояния в другое. Для этого на диаграмме состояний используется запись в виде *tm(n).* Эта запись означает, что переход из состояния осуществляется спустя n миллисекунд с момента входа в это состояние.

Для диаграмм состояний в языке UML определена семантика: *исполнение до завершения*. Это означает, что события потребляются одно за другим, из чего следует, что обработка последующего события не может быть начата до полной обработки предыдущего. В связи с этим, каждое событие может быть представлено как событие, трансформирующее диаграмму состояний из одной *стабильной конфигурации* в другую. При переходе в некоторое состояние по событию может оказаться, что выполняются условия, позволяющие произвести переход из этого состояния в другое и так далее. Исполнение до завершения означает, что такие переходы имеют высший приоритет и осуществляются вперед всякого события из очереди.

6. Совместное использование ресурсов

Одной из основных особенностей проектирования систем реального времени является наличие ***разделяемых ресурсов,*** которыми необходимо управлять в присутствии параллелизма. Каркас Rhapsody включает в себя абстракции для управления разделяемыми ресурсами, а именно:

- *OMOSMutex* является оберткой для мьютекса операционной системы и поддерживает операции *lock()* и *free()*. Мьютекс используется для управления ресурсами для эксклюзивного использования.
- *OMOSSemaphore* является оберткой для семафора операционной системы и поддерживает операции *wait()* и *signal()*. Семафоры используются для управления *ресурсами совместного использования.*

Весь класс или отдельные операции в пределах класса могут быть определены как *защищенные* путем присвоения стереотипа guarded. Класс, содержащий защищенные операции, называется *защищенным*. Защищенный класс может быть использован для моделирования эксклюзивного ресурса: в любой момент времени, может быть выполнена только одна копия одной защищенной операции (в пределах данного класса). Все защищенные классы наследуются от класса каркаса *OMProtected*. Он предоставляет им операции *lock()* и *free()* класса *OMOSMutex*. Генератор кода автоматически встраивает в каждую защищенную операцию пару lock-free. Метод реализации этих операций может быть изменен, например, на использование класса *OMOSSemaphore*. Защищенные классы, в таком случае, будут моделировать ресурсы совместного использования.

Как было упомянуто выше, в языке UML для потребления событий определяется семантика исполнения до завершения. Для сигналов и тайм-аутов данная семантика является по сути защищенной, так как у каждого реактивного класса существует единственный активный класс, поток которого осуществляет передачу события в данный класс. Для того, чтобы гарантировать реализацию принципа исполнения до завершения и в случае применения событий вызова, обработка событий защищается мьютексом. В результате чего, если в процессе обработки события из другого потока будет вызвана операция объекта для генерации синхронного события вызова, тот поток будет заблокирован на мьютексе и будет вынужден дождаться окончания обработки предыдущего события.

Необходимо помнить, что в UML действия, которые могут вызываться при входе, выходе и переходе из состояния, должны затрачивать “нулевое” время на исполнение. Так как обработка события является атомарным действием, ненулевое время исполнения может надолго блокировать другие потоки при попытке послать событие вызова. Вся активность, требующая ненулевого времени исполнения, должна выполняться объектом при нахождении внутри состояния.

7. Отладка, тестирование и временной анализ модели

Помимо *функциональных* требований, к системам реального времени предъявляются ограничения по времени реакции, обязательные для исполнения. Процесс проверки системы с точки зрения данного аспекта сводится к *временному анализу*. Существует два принципиальных подхода к осуществлению такого анализа:

- *Эмпирический*, который характеризуется вводом тестовых данных в систему и измерением соответствующих времен реакции и
- *Теоретический*, характеризующийся применением метода математического анализа, позволяющего рассчитать общую рабочую характеристику при условии наличия достаточной временной информации о времени исполнения отдельных задач.

Rhapsody предоставляет возможность отладки с использованием модели путем анимации динамических диаграмм состояний и диаграмм последовательности. Пользователь может осуществить пошаговую отладку отдельных частей приложения и визуально наблюдать результаты на анимируемых диаграммах состояний и диаграммах последовательности. Такие возможности поддерживаются другими элементами каркаса, которые не обсуждаются в данной публикации.

Пошаговая отладка приложения является хорошим способом тестирования функциональности системы. Но наиболее важным для приложений реального времени представляется то, что Rhapsody, помимо всего прочего, упрощает проведение эмпирического временного анализа. Для этого создается драйвер, который моделирует ввод в систему внешних событий. Он может быть реализован в виде скрипта или диаграммы состояний, генерирующей события. Драйвер затем активируется, и система отрабатывает запрограммированное поведение в соответствии со временем, требующемся для выполнения операций. Во время исполнения приложения в режиме трассировки в специальном файле сохраняется история вызовов с временными отметками. После этого разработчик может просмотреть результаты трассировки в графическом виде и оценить результаты на предмет их соответствия временным ограничениям.

Такая оценка параметров может быть выполнена как на машине разработчика, так и на целевом устройстве. Последний вариант предпочтительнее в связи с возможностью измерения времени реакции реальной целевой системы.

8. Состав каркаса

Каркас приложения в Rhapsody состоит из четырех основных частей.

- *Execution Framework* – обеспечивает основу для исполнения UML моделей, принципы которого были рассмотрены в этой статье.
- *OS Abstraction Level* - является оберткой сервисов операционной системы, используемых каркасом. Наличие данного слоя позволяет легко портировать каркас на различные операционные системы. Данный слой предоставляет остальным классам каркаса интерфейсы для работы с потоками, объектами синхронизации, очередями событий ОС и таймерами.
- *Сервисные классы* - предоставляют приложению и остальным классам каркаса сервисные функции, такие как различные реализации контейнеров, управление памятью и др.
- *Animation Framework* - подключается к приложению в режиме отладки и обеспечивают взаимодействие приложения с Rhapsody для управления исполнением и отображения результатов исполнения на анимированных диаграммах, а также для трассировки.

До сих пор речь шла об основной версии каркаса, для работы которой необходима операционная система. Однако Rhapsody позволяет генерировать код и для приложений, работающих без операционной системы. В этом случае используются две другие версии каркаса:

- *Interrupt Driven Framework* – версия каркаса для приложений с асинхронной обработкой событий, но без использования сервисов операционной системы.
- Continous Framework – версия каркаса для приложений, использующих только синхронные взаимодействия, не требующих диспетчеризации событий.

9. Заключение

Элементы каркаса и различные конструкции, которые поддерживаются в среде Rhapsody, обеспечивают концепциям UML конкретную интерпретацию, относящуюся к вопросам реального времени. Значительным преимуществом подхода с использованием каркаса является относительная открытость подобной интерпретации, благодаря чему разработчики могут изменять поведение генерируемого кода, определенное по умолчанию, и настраивать его под свои нужды.