Объектно-ориентированное расширение технологии RTST

А. Иванов, Дм.Кознов, А.Лебедев, Т. Мурашова, В.Парфенов, А.Терехов,
Санкт-Петербургский Государственный Университет,
ГП Терком.

Аннотация

Технология RTST (Real-Time Software Technology) является визуальным средством для создания встроенных систем реального времени. Она основывается на новой объектно-ориентированной методологии и поддерживается интегрированным набором инструментальных средств. Технология позволяет специфицировать ПО систем реального времени при помощи высокоуровневых графических нотаций и обеспечивает поддержку полного жизненного цикла ПО для таких систем: от анализа предметной области и выработки требований к системе до автоматической генерации ПО с последующим его сопровождением. При помощи интегрированных, на основе общего репозитория, моделей и соответствующих им типов диаграмм, технология позволяет в процессе разработки смотреть на систему с разных точек зрения и автоматически поддерживает согласованность и непротиворечивость разрабатываемого ПО.

Технология ориентирована на использование в больших коллективах, в многопользовательском режиме работы и предполагает согласованную работу аналитиков, проектировщиков и разработчиков системы.

В данной статье приводится краткое введение в технологию RTST с обзором методологии, процесса проектирования и набора инструментальных средств, их поддерживающих.

1 Ведение

Технология программирования встроенных систем реального времени RTST успешно использовалась в течение нескольких лет в различных проектах и показала свою эффективность. Необходимость такой технологии была вызвана насущными практическими проблемами, сколько-нибудь серьезного анализа существующих технологий произведено не было. В частности, из всего богатства концепции объектно-ориентированной парадигмы использовалась только одна – инкапсуляция. Особый упор в RTST был сделан на этап реализации ПО, при этом этапы анализа и проектирования были поддержаны существенно слабее. Другими словами, если удалось разбить создаваемую систему на объекты и формализовать их связи, средств RTST вполне достаточно, чтобы получить эффективную реализацию. Но практика показала, что искомое разбиение само по себе является весьма нетривиальной задачей, многие его недостатки обнаруживаются только на поздних этапах работы. Стала очевидной необходимость развития технологии «вверх».

Еще одним поводом для нашей работы стало знакомство с широким кругом современных объектно-ориентированных систем и концепций, ставшее возможным благодаря сотрудничеству с университетом города Гамбурга и технического университета Гамбург-Харбург.

В результате изучения и критического анализа существующих объектно-ориентированных технологий в основу объектно-ориентированного расширения RTST методологии были положены два следующих подхода:

- Объектно-ориентированный метод разработки ПО под названием Унифицированный Метод [1]. Этот метод является одним из наиболее распространенных на сегодняшний момент объектно-ориентированных методов. Он создан в Rational Software Corporation под руководством Г. Буча, Д. Рембо и И. Джакобсона. Существующая в настоящий момент версия 1.0 Унифицированного языка моделирования (Unified Modeling Language - UML) является стандартом де-факто в области объектно-ориентированных методологий и нотаций. Унифицированный Метод появился в результате слияния метода Буча [2] и OMT [3]. Впоследствии в нем нашли свое отражение определенные концепции OOSE [4].
- SDL-методология [5], уже хорошо апробированная в прежней версии RTST. Эта методология предложена международным комитетом CCITT для создания

телекоммуникационных систем. Основной ее составляющей является язык SDL, созданнный для спецификации поведения и структуры разрабатываемых систем [5,6]. Было выпущено несколько версий этого языка. В общую методологию вошли другие нотации среди которых хочется отметить MSC-нотацию [7], созданнаю для спецификации сценариев взаимодействия объектов, а также функциональную нотацию [8], которая рекомендуется для описания функциональных требований к системе. MSC и функциональная нотация реализуются только в новой версии RTST.

2 Архитектура встроенных систем реального времени

RTST предназначена для создания встроенных систем реального времени. В общем виде архитектуру таких систем можно определить следующим образом.

Система погружена во внешнюю среду, частью которой может быть человек (оператор, пользователь) и обладает следующими свойствами:

1. Управляется вычислительной системой.
2. Представляет собой взаимодействующую совокупность программных и аппаратных средств.
3. Взаимодействует с внешней средой посредством обмена дискретными сигналами, удовлетворяя определенным ограничениям на время приема и обработки входного сигнала и выдачи соответствующих выходных сигналов.

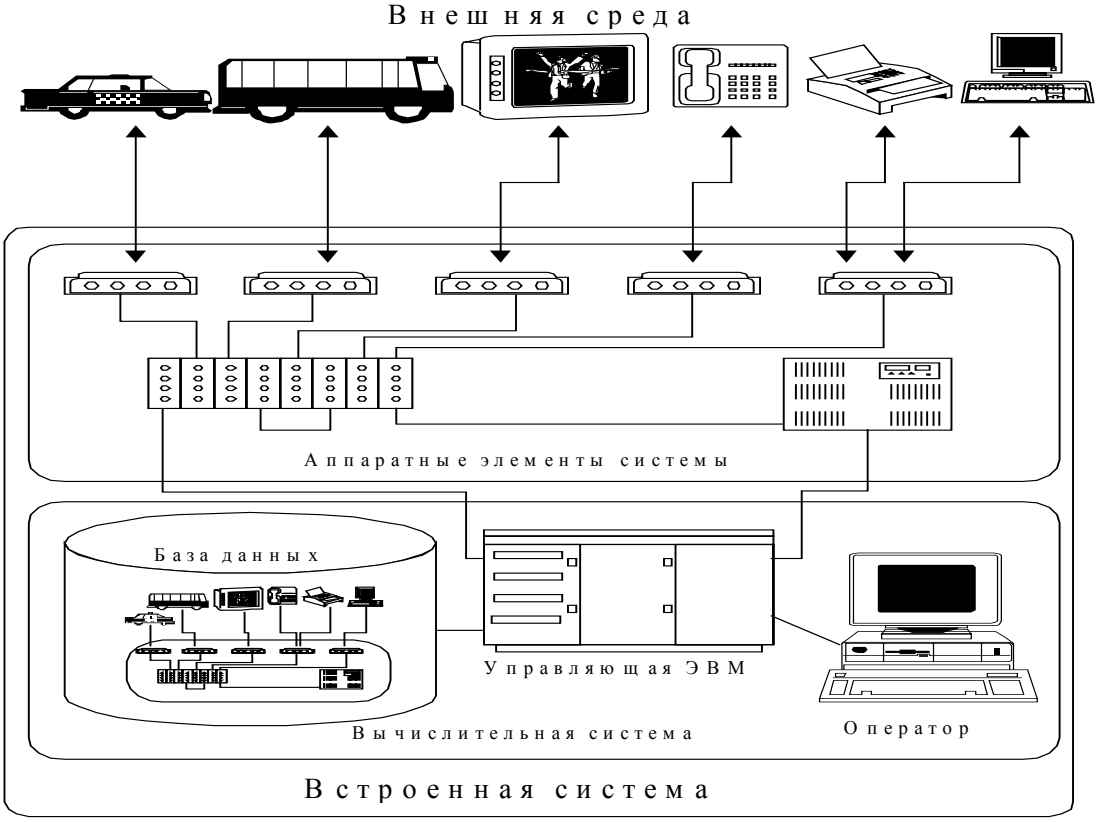

Рис 1. Встроенная система реального времени

Внешняя среда существует независимо от системы, однако система знает правила ее поведения, умеет адекватно реагировать на информацию, поступающую из нее, выдавая информацию, возможно, оказывающую влияние на состояние и поведение внешней среды. Как правило, внешней средой является фрагмент реального мира, “обозреваемого” системой, а ее структурная сложность зависит от назначения системы.

Подобно внешней среде, аппаратные средства являются частью реального мира, “обозреваемого” программным обеспечением встроенной системы, однако, при этом они являются частью самой системы и участвуют в реализации ее функций, работая согласованно с программным обеспечением. Они обеспечивают прием информации, ее первичную обработку, перевод в цифровую форму и передачу программному обеспечению, а также обратные преобразования для передачи информации во внешнюю среду. “Интеллектуальный уровень» аппаратных средств может быть различным.

Вычислительная система осуществляет управление всей встроенной системой. Она обеспечивает работу программного обеспечения, хранение информации о состоянии системы и внешней среды и поддерживает интерфейс с оператором.

Программное обеспечение вычислительной системы разбивается на функциональное и системное. Функциональное ПО реализует решение возложенных на систему задач. Системное ПО осуществляет управление ресурсами вычислительной системы и обеспечивает доступ к ним со стороны функционального ПО.

Данное определение системы реального времени делает акцент на ее структуру. Таким образом, фиксируется класс архитектур систем, на разработку которых ориентируется RTST.

В работе [4] дается определение и предлагается классификация систем реального времени с точки зрения их функционалистики. При этом сама система рассматривается как черный ящик. В описании применения подхода OOSE к созданию систем реального времени основной упор делается на описании их функционалистики (различные способы синхронизации, проблема разбиения системы на параллельно работающие компоненты и т.д.). При этом не делается акцента на то, что система может состоять из качественно различных компонент. Эти различные компоненты вынесены за пределы системы, в ее окружение, и обсуждается лишь проблема построения различных типов интерфейсов системы с ее внешним миром. Однако реальные задачи часто бывают смешанными – надо сделать саму систему реального времени и реализовать часть ее окружения. При этом оказывается, что если два этих процесса (разработку самой системы и ее окружения) проводить независимо друг от друга, поддерживая только общие интерфейсы, то не удается эффективно реализовать всю систему в целом.

В определении системы реального времени, приведенном в работе [9], аппаратные элементы явно выделены и включены в систему. Однако, как и в предыдущем определении, никак не отражена неоднородность окружения системы и отсутствует информация о возможной качественной неоднородности самой системы.

3 Процесс разработки системы

RTST методология поддерживает процесс разработки системы, состоящий из следующих фаз [3].

- Анализ.
  На этой стадии предметная область описывается в виде набора объектов и строится описание функциональных требований к системе. Система рассматривается с точки зрения некоторой “идеальной” модели окружения, без к внимания деталям реализации.

- Проектирование.
  Эта фаза служит для адаптации и уточнения структуры объектов в зависимости от выбранного окружения. Конкретизация “идеальной” модели происходит в соответствии с особенностями целевой платформы и спецификой предъявляемых к ней требований.

- Реализация. На этой стадии происходит собственно реализация системы на основе результатов предыдущих стадий.

- Тестирование. На этой фазе происходит тестирование и верификация созданной системы.

На рис. 2 показана взаимосвязь этих фаз.

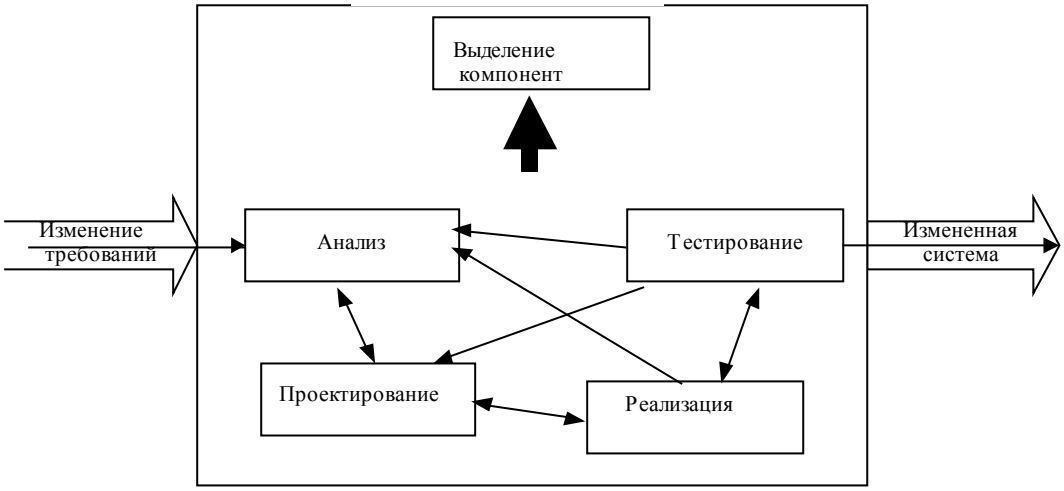

Рис. 2. Процесс разработки системы

Дадим несколько определений.

Под словом «моделирование» мы понимаем построение последовательности представлений о системе (моделей), позволяющих смотреть на систему с разных точек зрения. Эти представления путем постепенной детализации позволяют перейти от начального описания системы к ее реализации.

Язык моделирования RTST – это средство построения спецификаций для разрабатываемой системы. Язык делится на части, называемые моделями. Разные модели служат для построения различных видов спецификаций системы – спецификаций, выполненных с разных точек зрения, взглядов на систему.

Использование моделей осуществляется с помощью нотаций. Нотации включают в себя как текстовые, так и графические представления информации. Одной и той же модели может соответствовать несколько нотаций – т.е. семантика этих нотаций одинакова (они представляют одну и ту же модель), но средства отображения одних и тех же сущностей модели - разные .

Термин *модель* может использоваться и в другом значении – как результат применения языка моделирования RTST (или его части) к спецификации системы. Можно говорить о построении модели системы в целом (например, модель системы, построенная на фазе анализа) или о создании той или иной специальной модели для системы (структурной, функциональной и т.д.). В дальнейшем из контекста будет понятно, в каком значении используется термин *модель*.

На всех этапах создания системы в RTST используются следующие модели (эта классификация заимствована из [3]):

- Функциональная модель, которая описывает, что система должна делать.
- Структурная модель, определяющая структуру системы и внешнего мира.
- Динамическая модель, которая описывает динамику поведения объектов, выделенных в структурной модели для обеспечения функций, которые описанны в функциональной модели.

В RTST предлагается следующее деление этих моделей на подмодели и их связь с конкретными нотациями:

- Функциональная модель, имеет две нотации - нотацию случаев использования системы и функциональную нотацию. Первая заимствована из OOSE-методологии [4], для второй за основу взята функциональная нотация из работы [8].
- Структурная модель:
  - Модель классов и, соответственно, нотация для этой модели. Основана на модели классов UML [1].

- Модель объектов и объектная нотация. Основана на модели объектов UML [1].

- Динамическая модель
  - Сценарная модель, позволяющая описывать взаимодействие объектов системы между собой и с внешним миром. Использует нотацию для описания сценариев и нотацию для спецификации схем сценариев. Для первой взята за основу MSC, для второй – MSC-overview [7].
  - Поведенческая модель, которая дает возможность специфицировать деятельность объектов, необходимую для реализации всевозможных взаимодействий, описанных с помощью модели сценариев. Представлена расширенной конечно-автоматной нотацией, основанной на SDL [6] и модифицированной STD-нотацией из OMT [3].

Большое значение имеют “прозрачность” переходов между этапами разработки системы и возможность проследить соответствие между сущностями разных моделей системы. В RTST они обеспечиваются за счет использования общего репозитория, в котором хранятся все модели системы.

4 Функциональная модель

Результаты анализа требований к системе формализуются с помощью функциональной модели, которая определяет, какие функции должны выполняться системой, вне зависимости от ее структуры. Наполнение (реализация) функций будет происходить в терминах объектов, обладающих определенными свойствами и поведением.

Создание функциональной модели, как правило, начинается со спецификации случаев использования этой системы различными внешними агентами. Случаи использования предназначены для того, чтобы составить представление о системе с точки зрения предоставления услуг ею различным “действующим лицам”. Примерами таких “действующих лиц”, или внешних агентов, могли бы быть: пользователь системы, оператор, система, внешняя по отношению к данной.

Рассмотрим пример.

Рис. 3. Пример диаграммы случаев использования

На этой диаграмме системой является Мобильная радиотелефонная сеть, абонент и оператор являются внешними агентами, а сервис коротких сообщений, оплата услуг и т.д. являются случаями использования системы.

Каждый из случаев использования системы может быть рассмотрен как функция системы и подвергнут дальнейшей декомпозиции в терминах функций. В результате возникает дерево функциональной декомпозиции.

На рис. 4 приводится пример функциональной декомпозиции для разговорного сервиса.

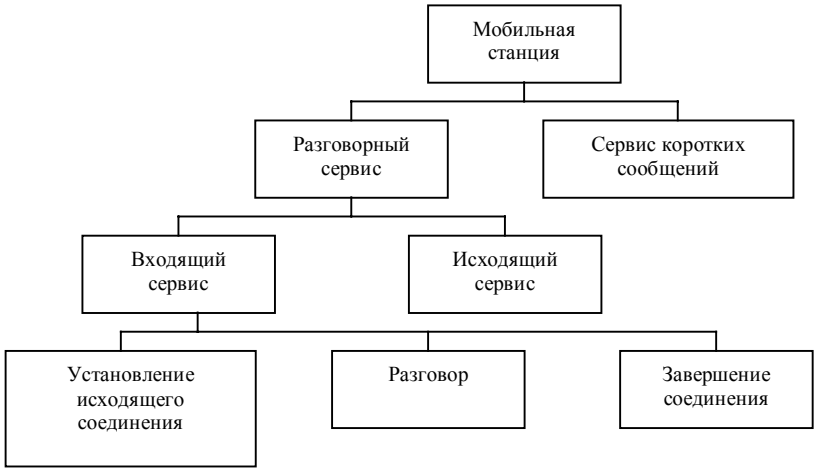

Рис. 4. Фрагмент функциональной декомпозиции системы

Каждая функция может быть разбита на несколько более мелких. Такое разбиение обычно соответствует либо структурной декомпозиции функциональных требований (на фазе анализа), либо выделению подзадач, решение которых требуется для осуществления основной цели (на фазе проектирования).

Функциональная модель предоставляет средства для специфиции требований к системе. Дальнейшая детализация этих требований, а также их проекция на функционалистику системы и на ее структуру, осуществляется с помощью диаграмм объектов и сценариев. На диаграмме объектов изображаются главные действующие лица (объекты), участвующие в исполнении этой функции, и связи между ними. Само исполнение описывается с помощью сценариев, в которых участвуют описанные выше действующие лица.

5 Структурная модель

В RTST-методологии вводятся следующие понятия – класс, экземпляр класса, объект-роль.

Согласно объектно-ориентированной парадигме реальный мир (система и окружение) представляется как множество взаимодействующих объектов. Для классификации объектов вводится понятие «класс» (тип объекта). Считается, что все объекты, принадлежащие одному классу, при одинаковых условиях ведут себя одинаково.

В RTST вместо термина «объект» используется термин «экземпляр класса», при этом имеется понятие «объект-роль», которое обозначает те роли, которые экземпляры классов могут играть в системе. Один и тот же экземпляр класса может играть разные роли в системе в разные моменты своего существования или даже одновременно. Связи между этими понятиями изображены на рис. 5.

Приведем пример. Абонентов у телефонной станции может быть 10 000, а ролей у этих абонентов только две - вызываемый и вызывающий абонент. Таким образом, класс Абонент имеет 10 000 экземпляров и всего два объекта-роли.

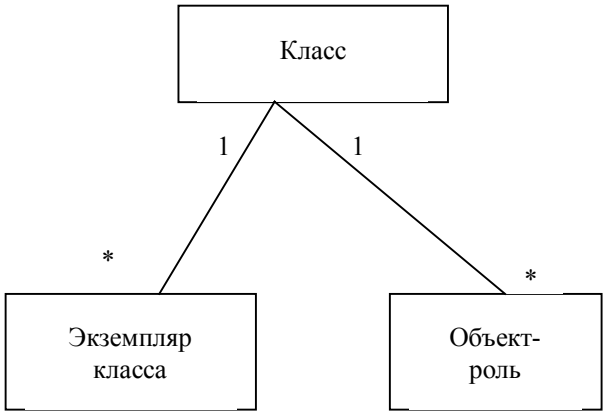

Рис. 5. Взаимосвязь между понятиями класс, объект-роль и экземпляр класса

Модель классов и модель объектов для разрабатываемой системы могут строится в произвольном порядке. Если предметная область достаточно знакома, то разработчики могут сразу рисовать классы для нее. Если же нет, то возможна, например, следующая стратегия - сначала появляются объекты системы, для них строятся сценарии, и на основе этой информации делается вывод о функционалистике объектов, после чего можно решить, какие объекты к каким классам можно отнести (более подробную информацию о стратегиях построения модели классов можно найти в работе [2]).

Считается, что структурная модель разрабатываться после функциональной, поскольку последняя специфицирует требования к системе, а первая - структуру системы. Однако, на практике возможны всякие сочетания. Важно лишь, чтобы в итоге все модели были корректно связаны друг с другом через общие сущности.

5.1 Модель классов

Модель классов RTST основывается на известной UML нотации. В ней имеются два основных вида структурных сущностей – классы и категории.

Классы являются главными кирпичиками, из которых строится система. Они имеют атрибуты, операции, отношения с другими классами и поведение. Под отношениями прежде всего понимаются связи, существующие для обмена информацией между различными компонентами системы, как некоторый возможный способ общения двух сущностей.

Модель классов RTST поддерживает наследование, которое позволяет более эффективно использовать уже существующие компоненты и конструировать новые за счет выделения общих свойств различных классов. Если один класс наследуется другим, то последний наследует все атрибуты, операции и ассоциации предка. Не наследуется реализация поведения класса, описанная при помощи конечно-автоматной модели (т.е. с использованием SDL и STD диаграмм). Допускается множественное наследование.

Классы могут быть связаны друг с другом отношениями (ассоциациями), отражающими зависимости между их экземплярами. При соединении классов ассоциацией для каждого из них может быть определена его роль в этой ассоциации с указанием возможного количества экземпляров класса, участвующего в этой ассоциации в данной роли (не следует путать роли ассоциации и объекты-роли).

Отношение агрегирования (включения) является ассоциацией специального вида. Наличие такой ассоциации между двумя классами говорит о том, что агрегат владеет объектом или объектами агрегируемого класса.

В ассоциации могут описываться сообщения, которыми экземпляры вовлеченных в нее классов могут обмениваться друг с другом. Для сообщений имеется возможность описывать параметры.

Спецификация сообщений в ассоциациях отличает RTST от UML нотации и от SDL. В UM сообщения (события, в диаграммах состояний) не связаны с описанием классов. В SDL сообщения могут описываться или в классе (process), или в его роли (gate).

Категория является средством группировки классов. Они могут содержать в себе другие классы и/или категории.

На шаге конструирования модели классов существует возможность описать классы, чьи экземпляры не являются собственно компонентами системы - классы внешних агентов системы. Они необходимы для моделирования разнородного окружения системы.

Модель классов поддерживается специальным программным средством - редактором классов.

5.2 Модель объектов

Диаграмму объектов можно интерпретировать как моментальный снимок структуры системы в некоторый момент ее существования.

Модель объектов не является средством для задания конфигурации системы (определения количества экземпляров классов на момент старта системы и связи конкретных экземпляров между собой). Ее задачей является спецификация различных ролей, которые могут играть экземпляры классов. Именно для этих ролей-объектов в модели сценариев описаваются протоколы взаимодействия, которые переходят в описания поведения всех классов, чьи объекты-роли участвовали в этих протоколах.

Следует отметить, что один объект-роль, соответствующий данному классу, может принадлежать только к одной диаграмме объектов, поскольку объект-роль, принадлежащий к тому же классу, но представленный на другой диаграме, является другой ролью данного класса в системе.

Использование модели объектов осуществляется с помощью редактора объектов.

6 Динамическая модель

Динамическая модель описывает поведение системы - взаимодействие между различными ее компонентами и самой системы с окружением. Другими словами, динамическая модель описывает процессы, протекающие в системе в терминах событий, действий и состояний.

6.1 Сценарии RTST

Взаимодействия объектов в системе удобно представлять в виде сценариев. Каждой функции из функциональной модели RTST можно сопоставить диаграмму объектов, которая будет отображать типичную “конфигурацию” объектов, задействованных в осуществлении данной функции, а также набор сценариев, которые ее реализуют. В этих сценариях принимают участие объекты-роли, определенные на диаграмме объектов для данной функции. Одна диаграмма объектов может соответствовать нескольким функциям.

Сценарий представляет собой упорядоченную во времени последовательность событий, которыми, как правило, являются посылки и приемы сообщений объектами. На начальных шагах спецификации системы сценарии можно создавать не только с помощью специальной графической нотации, но и в текстовом виде.

Построение сценариев для функций начинается с определения прямых “веток”, т.е. идеального исполнения функции. При этом из рассмотрения исключаются граничные и ошибочные ситуации, а также частные случаи. Затем рассматриваются всевозможные специальные случаи – встречные ситуации, поведение функции при сбоях и ошибках и т.д.

Построение поведения системы (поведения ее классов) на основе сценариев, а не напрямую, позволяет в более наглядном виде представлять общие процессы, протекающие в системе, и, отталкиваясь от них, конструировать внутреннее поведение участников этих процессов.

Ниже представлен пример - сценарий взаимодействия банкомата и пользователя, который снимает деньги со счета.

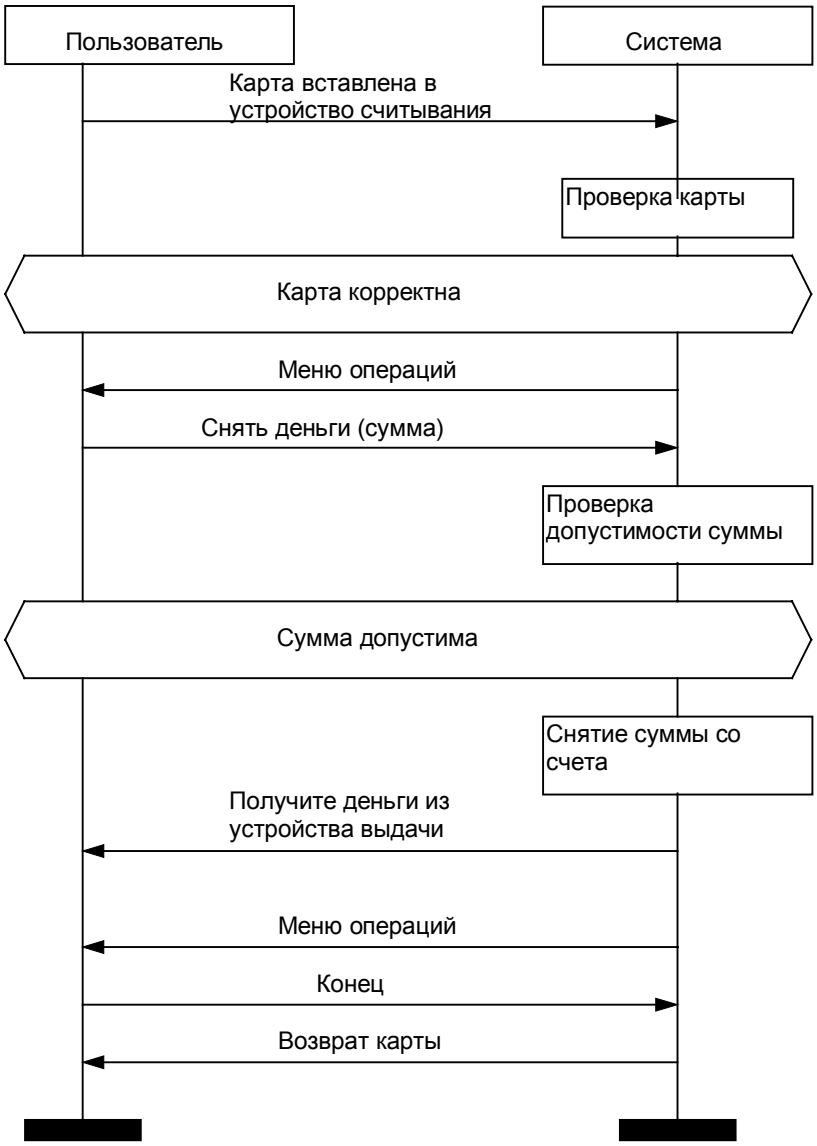

Рис. 6. Сценарий взаимодействия пользовате с банкоматом для протокола снятия денег со счета при корректной карте

Модель сценариев в RTST поддерживается специальным редактором, который позволяет создавать сами сценарии, и редактором схем сценариев. Последний основан на нотации MSC-overview, определенной в [7] для навигации по MSC-диаграммам.

6.2 Поведенческая модель

Поведенческая модель системы является спецификацией поведения составляющих ее классов. В дальнейшем будем говорить о поведенческой модели отдельного класса. Построение такой модели начинается с анализа всех сценариев, в которых участвуют объекты-роли данного класса. Поведенческая модель класса считается завершенной после того, как она покрывает все

сценарии, в которых участвуют его объекты-роли, и в ней обрабатываются все события, которые могут оказать влияние на экземпляры этого класса.

Поведенческая модель поддерживается в RTST двумя нотациями - первая является модификацией диаграмм состояний OMT [3], вторая основывается на расширенном конечном автомате SDL [6]. Эти нотации отличаются способом графического представления поведенческой модели. В наборе программных средств RTST им соответствуют два редактора: SDL-редактор и STD-редактор.

SDL-нотация позволяет с большей степенью графической подробности изображать поведение класса, в то время как с помощью STD-нотации можно создавать более компактные спецификации.

Поведенческую модель часто бывает нецелесообразно строить для каждого класса системы. В [2] приводятся критерии (для различных предметных областей) того, когда имеет смысл строить конечно-автоматную спецификацию поведения класса. В RTST основным критерием того, что класс нуждается в такой спецификации, является его принадлежность к подсистеме, которая ориентирована на работу в реальном времени. Конечно, далеко не все классы таких подсистем достаточно сложны для того, чтобы иметь конечно-автоматную спецификацию своего поведения, но мы здесь не будем подробно останавливаться на этом вопросе.

6.3 Переход от модели сценариев к поведенческой модели

RTST позволяет разработчикам автоматически получать заготовки поведенческой модели класса по сценариям, в которых участвуют его объекты-роли.

Отрезок поведения объекта-роли от одного приема сообщения до другого переходит в одно состояние и переход из этого состояния в следующее. Так, на рис. 7 отрезку AB соответствует состояние A и переход их него по приему сообщения 1. Когда Объект 1 получает сообщение 1, то он выходит из этого состояния, посылает сообщение 2 и сообщение 3 и переходит в состояние B, из которого выходит, получив сообщение 4.

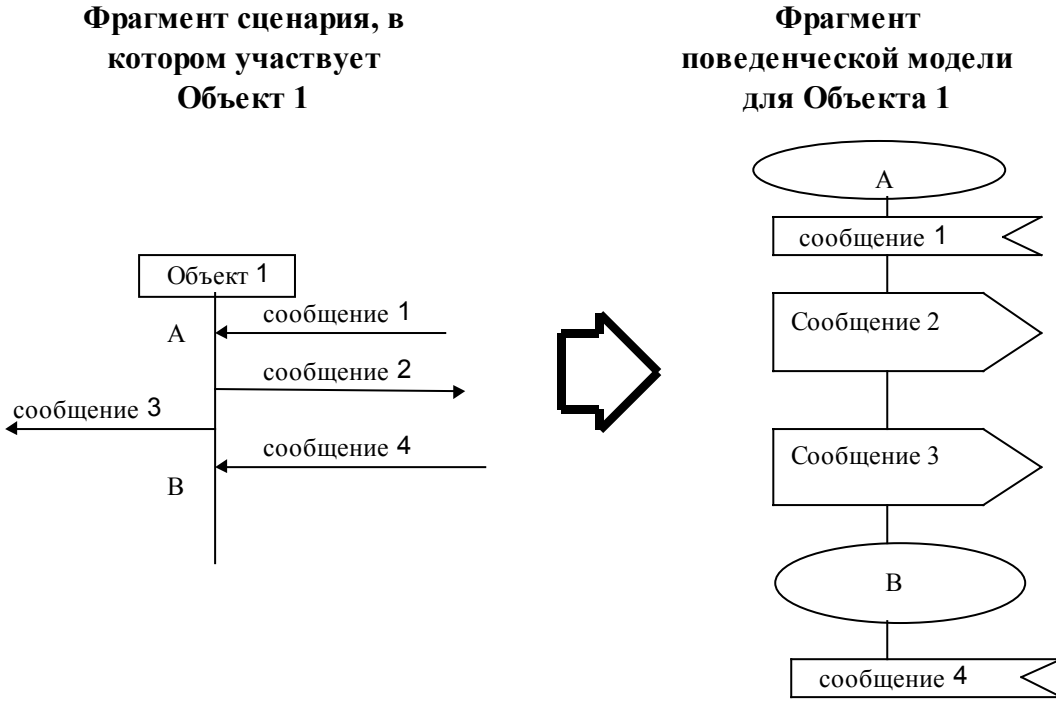

Рис. 7. Пример перехода от сценарной к поведенческой модели

Таким образом, для Объекта 1 по данному фрагменту сценария получается линейная цепочка состояний и переходов. После автоматического получения таких цепочек для всех объектов данного класса по всем сценариям, в которых они участвуют, разработчики

производят модификацию полученных заготовок (уже вручную) – сливают состояния, расширяют и дополняют переходы, создают новые состояния и т.д. Сценарии не являются средством полной спецификации взаимодействий между объектами, поэтому после получения по ним заготовок для поведения класса, дальнейшее соответствие между сценариями и поведенческими моделями не поддерживается.

7 Репозиторий RTST

Целостность и непротиворечивость создаваемой с помощью RTST системы обеспечивает комплекс программных средств, который называется репозиторием или единой базой данных системы. В процессе разработки системы мы дополняем, уточняем и детализируем единый проект, который полностью включает в себя всю существующую на данный момент информацию о системе и ее окружении. При этом проблема поддержания соответствия и непротиворечивости информации о системе, представленной на диаграммах разных типов, сводится к проблеме поддержания целостности информации в базе данных. Таким образом, каждый редактор диаграмм является некоторым интерфейсом репозитория, отбражая, в соответствии с определенной нотацией, информацию из базы, и позволяя пользователю редактировать эту информацию. В результате отпадает необходимость в специальных функциях преобразования данных при переходе от одной модели к другой. Данный подход, который авторы RTST сочли оптимальным, содержит в себе, однако, и некоторые проблемы.

Во-первых, опыт использования предыдущей версии RTST показал, что требование полной корректности хранящейся в базе данных информации хотя и является выполнимым, но накладывает на пользователя подчас очень жесткие ограничения, снижающие эффективность его работы (например, невозможность сохранить незавершенные диаграммы, скорее всего, привела бы к их дроблению, необоснованному с точки зрения структуры информации на этих диаграммах). Было принято решение, введя дублирование части данных, хранящихся в репозитории, в некоторых ситуациях позволять пользователю нарушать целостность базы данных. При визуализации содержимого репозитория все видимые компоненты, которые содержат в себе данные, нарушающие целостность, выделяются специальным цветом.

Вторая проблема заключается в том, что каждая диаграмма несет в себе, помимо информации о системе, сведения о графическом представлении этой информации, включающие в себя, например, размеры и расположение отдельных элементов данной диаграммы. В RTST все эти сведения хранятся непосредственно в диаграмме, которая хранится отдельно от репозитория. Кроме информации о графическом представлении сущностей репозитория в диаграмме сохраняются и ссылки на сами эти сущности в репозиторий. При активизации диаграммы производится проверка соответствия хранящегося в ней графического представления и информации в базе данных, и, в случае обнаружения рассогласований, диаграмма, в полуавтоматическом режиме, приводится в соответствие с текущей версией системы. Такая схема позволяет создать более эффективную реализацию графических редакторов, а также производить временное отчуждение диаграмм, например, внедряя их в какие-либо документы как ActiveX объекты.

Особое место, с точки зрения репозитория RTST, занимает модель сценариев. Дело в том, что она предназначена для изображения типовых протоколов между объектами, а не для полной спецификации взаимодействия между ними. Таким образом, мы не можем требовать точного соответствия описания поведения объектов класса и спецификации их взаимодействия с другими объектами вследствие неполноты последней, хотя, как показано в одном из предыдущих разделов, имеется определенное соответствие между сущностями этих двух моделей. Попытка поддерживать частичную непротиворечивость этих представлений системы, потребовала бы существенной работы пользователя по указанию этих соответствий и наложила бы дополнительные ограничения на его работу, не предоставив ему при этом достаточного контроля (невозможно контролировать что-либо по неполной спецификации). Отсутствие соответствия между сценарной и поведенческой моделями означает, что репозиторий не содержит сущностей, общих для этих двух моделей (хотя они используют одни и те же сущности модели классов), поэтому для их связи необходим отдельный процесс генерации прототипа поведенческой модели по сценарной.

8 Генерация исполняемого кода по моделям RTST

Язык моделирования RTST предоставляет средства высокого уровня для спецификации создаваемой системы. Однако конечной целью проекта является получение реализации системы на некотором наборе языков программирования. В RTST предусмотрена генерация кода на таких языках по модели классов и поведенческой модели. При этом предполагается, что дополнять и модифицировать полученный код «вручную» нельзя, однако модели имеют возможность тесной интеграции с языками программирования (например, в поведенческую модель можно вставлять тексты на языке реализации системы, которые при генерации они будут вставлены без изменения в соответствующие места конечного кода).

Генераторы представляют собой конверторы в различные языки программирования. Результаты конвертирования можно компилировать, с помощью соответствующего транслятора, в коды целевой платформы.

Для исполнения этих кодов необходима библиотека динамической поддержки, в роли которой может выступать некоторая специальная операционная система.

В RTST предусмотрена возможность генерации кода для виртуальной машины в целях отладки полученных спецификаций. Работа пользователя с этой виртуальной машиной должна осуществляться с помощью специального отладчика, интегрированного со средой RTST .

9 Заключение

Сформулируем в заключение основные особенности RTST.

- Ориентирована на создание встроенных систем реального времени, т.е. программно-аппаратные комплексы, где система реального времени является одной из подзадач.
- Позволяет упростить процесс общения и взаимодействия разработчиков, создающих систему, предоставляя в их распоряжение высокоуровневые графические средства спецификации, что позволяет существенно облегчить создание системы на этапах анализа и проектирования.
- По высокоуровневым спецификациям технология позволяет автоматически получать реализацию системы.
- Интегрирует инструментальные средства на основе общего репозитория и позволяет автоматически выявлять неверные, с точки зрения методологии RTST, шаги разработчика. Таким образом, непротиворечивость и целостность создаваемой системы поддерживается на протяжении всего цикла ее разработки.
- Существование единого на всех стадиях разработки набора моделей и соответствующих им нотаций позволяет обеспечить итеративность процесса разработки.
- Предоставляется возможность отладки и тестирования создаваемой системы в тех терминах, в которых она была описана.

10 Список литературы

[1] http:// www.rational.com

[2] G. Booch. Object-Oriented Analysis And Design With Application, second edition. The Benjamin/Cummings Publishing Company, Inc. 1994.

[3] J. Rambaugh, M. Blaha, W.Premerlani, F.Eddy, W.Lorensen, Object-Oriented Modeling and Design, Prentice-Hall. 1991.

[4] I.Jacobson. Object-Oriented Software Engineering. ASM press. 1992.

[5] Braek, Th. Haugen. Engineering Real Time Systems. Prentice Hall International (UK) Ltd. 1993.

[6] Revised Recommendation Z.100 CCITT Specification and Description Language (SDL). 1992.

[7] New Recommendation Z.120 CCITT Message Sequence Chart (MSC),1992.

[8] ITU-T Recommendation Z.100 - Appendices I and II. SDL Metodology Guidence. SDL Bibligraphy. 1993.

[9] B.Selic, G.Gullekson, P.T. Ward. Real-Time Object-Oriented Modeling. John Wiley & Sons. Inc. 1994.