

**ОБЪЕКТНО-ОРИЕНТИРОВАННАЯ МЕТОДОЛОГИЯ
РАЗРАБОТКИ ИНФОРМАЦИОННЫХ СИСТЕМ
И ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ
СИСТЕМ РЕАЛЬНОГО ВРЕМЕНИ**

Возрастающая сложность современного программного обеспечения привела к созданию специальной научной дисциплины — компьютерной инженерии (Software Engineering), основной задачей которой является создание эффективных методов разработки сложных программных систем.

Объектно-ориентированные методологии разработки программного обеспечения (первое направление) стали интенсивно развиваться с конца 80-х годов. В 1997 г. OMG (Object Management Group) приняла UML [1], появившийся в результате слияния ряда известных методологий, в качестве стандарта языка объектно-ориентированного моделирования. К настоящему моменту уже несколько компаний предложили CASE-средства, реализующие UML. Еще одним объектно-ориентированным подходом является методология ROOM [2], созданная для разработки систем реального времени. Одновременно в течение последних 20 лет международным комитетом ITU развиваются стандарты для разработки телекоммуникационных систем (второе направление): SDL [3], MSC [4] и т.д. Существует большое количество фирм (преимущественно в Европе), выпускающих программные продукты, которые реализуют эти стандарты. Кроме того, с 70-х годов развиваются структурные методологии разработки программного обеспечения (третье направление): SADT [5], IDEF-стандарты [6], метод Йордана [7] и т.д. В настоящее время эти методологии прочно закрепились в области разработки информационных систем. Они являются эффективным средством анализа систем в целом и успешно применяются, в частности для проектирования реляционных баз данных, автоматической генерации форм, отчетов и т.п.

Три этих направления существуют достаточно независимо и обособленно: разные методологические основы, разные фирмы-производители и т.д. Однако в последнее время наблюдается некоторое сближение этих подходов.

В SDL-мире прочно укрепилось мнение, что объектно-ориентированные методологии разработки программного обеспечения могут быть средством анализа системы, а ее дальнейшую спецификацию предполагается проводить с помощью SDL. В рамках международного проекта INSYDE [8] была разработана формальная спецификация правил трансляции OMT [9] в SDL. Многие фирмы в той или иной степени реализовали эту идею в своих SDL-продуктах.

Известные средства разработки баз данных, образовавшиеся на основе структурного подхода к созданию программного обеспечения и к настоящему времени выделяющиеся в самостоятельное направление, предоставляют мосты в CASE-средства, реализующие UML. Основой для такой интеграции является тот факт, что диаграммы классов UML есть расширение диаграмм сущность — связь, используемых для спецификации структуры базы данных.

В данной работе мы описываем объектно-ориентированную методологию Real и одноименное CASE-средство, ее реализующее. Методология Real основывается, главным образом, на UML, SDL, ROOM и отражает перечисленные интеграционные тенденции. Помимо стандартных для объектно-ориентированного подхода черт в Real добавлены дополнительные возможности, направленные на две специальные области программного обеспечения: для информационных систем и для систем реального времени. Такой подход позволяет во многих случаях избежать дополнительной настройки CASE-средства, а если она и понадобится, то будет намного более простой и дешевой, чем, например, настройка на эти области CASE-средств, реализующих UML.

Естественно, что Real не претендует на то, чтобы покрыть все возможности программных продуктов соответствующих областей. В то же время, учитывая современный уровень развития локальных и глобальных информационных сетей и возрастающую сложность программного обеспечения, в информационных системах все большую популярность приобретает технология клиент-сервер, т.е. многие информационные системы приобретают ярко выраженный событийно-ориентированный аспект, который глубоко проработан в методологиях разработки программного обеспечения систем реального времени. С другой стороны, большие распределенные системы реального времени нуждаются, как

правило, в хранении, доступе и передаче огромного количества информации (например, тарификационной и аутентификационной), а не только управляющих сигналов и данных трафика. Таким образом, методология Real подходит для разработки программного обеспечения обеих областей, но наиболее эффективна для их пересечения.

Естественно, что Real (методология и CASE-средство)¹ является "наследником" RTST [12, 13] и развивается на протяжении последних 10 лет на кафедре системного программирования математико-механического факультета С.-Петербургского государственного университета.

В настоящей работе излагаются основы методологии Real (процесс и модели²), затем описываются соответствующие инструментальные средства. Далее обсуждаются специфические возможности Real (методологии и CASE-средства), предназначенные для разработки информационных систем и систем реального времени.

1. Основы методологии Real

Не останавливаясь в общем на процессе разработки программного обеспечения (об этом см., например, в [16]), перечислим, какие модели используются в Real для описания разрабатываемой системы:

- Модель требований к системе:
 - Описательная модель — в текстовом виде описывает некоторые требования к системе.
 - Модель случаев использования — описывает требования, предъявляемые к системе ее окружением, т.е. отвечает на вопрос "что и для кого должна делать система?".
 - Функциональная модель — описывает разбиение случаев использования и функций на подфункции. Дает ответ на вопрос "как должны реализовываться функции системы в терминах своих подфункций?".
- Динамическая модель:
 - Модель объектов — описывает роли объектов системы и отвечает на вопрос "какие объекты взаимодействуют при выполнении функций системы?".

¹ Предыдущая версия описывается в работах [10, 11].

² Более формальное определение моделей Real можно найти в [14].

Модель взаимодействий — описывает сценарии взаимодействия объектов системы между собой и с пользователями, т.е. дает ответ на вопрос “как объекты взаимодействуют друг с другом для выполнения функций системы?”.

Поведенческая модель — описывает алгоритмы поведения объектов системы, т.е. отвечает на вопрос “как должен вести себя каждый объект для реализации функций системы?”.

- Статическая модель:

Модель классов — описывает внутреннюю структуру системы, структуры данных, используемые в ней, т.е. отвечает на вопрос “как должна выглядеть система изнутри?”.

В Real большой упор был сделан на связность моделей, на контроль целостности информации о проекте, представленной внутри как одной модели, так и в нескольких.

Модель требований. Работа над системой в Real начинается с построения *описательной модели*, в которую прежде всего входят первичные требования заказчика. Среди них могут быть как функциональные требования, так и любые другие (эффективность, стоимость и т.п.). Описательная модель хранится в Real в виде обычного текста и формально не связана с остальными моделями. Эта модель может быть использована и для окончательной спецификации нефункциональных требований.

В качестве примера рассмотрим список требований к системе документооборота:

1. Управлять движением документов по заданному маршруту от одного пользователя к другому.
2. Описывать маршруты документов, описывать и регистрировать типы документов, отношения и взаимодействия между ними.
3. Все документы должны регистрироваться, храниться.
4. Создавать новые документы зарегистрированного типа как системой, так и пользователем.
5. Обеспечивать целостность данных при пересылке по сети.
6. Поддерживать систему разделения прав пользователей.
7. В качестве документов могут быть использованы любые виды документов, поддерживаемых пакетом MS Office 97.
8. Система должна работать в ОС Windows, через Internet, в качестве среды разработки использовать MS Visual Basic, MS Visual C++, HTML, MS Access.

На основе требований заказчика формулируется полный список функциональных требований к системе, которые оформляются в терминах модели случаев использования и модели функций. Окончательное техническое задание на систему может быть сгенерировано по модели требований Real в том виде, который нужен заказчику (ГОСТ, какой-либо международный или внутрикорпоративный стандарт и т.п.).

Модель случаев использования в Real предназначена для описания стыка системы с окружением. В ее терминах описываются все пользователи системы, а также все ее функции (случаи использования), различимые с точки зрения этих пользователей. В дальнейшем с использованием могут быть связаны классы. Для случаев использования, в свою очередь, можно создавать диаграммы этого же типа, т.е. подвергать случаи использования дальнейшей декомпозиции в рамках той же модели.

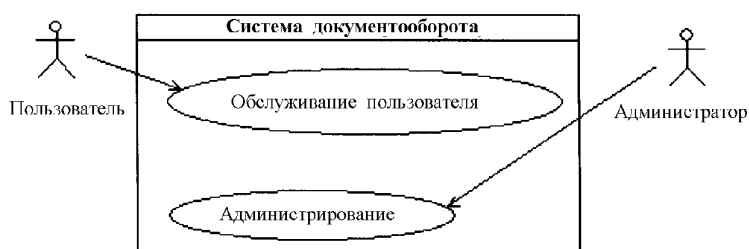


Рис. 1. Пример диаграммы случаев использования.

На рис.1 представлена диаграмма случаев использования для системы документооборота. На ней изображены два возможных пользователя системы: “Пользователь” и “Администратор”. Для системы выделяют два случая использования: “Обслуживание пользователя” и “Администрирование”.

Функциональная модель предназначена для дальнейшей декомпозиции функций системы. Она состоит из набора деревьев функций, корнями которых являются случаи использования. Дерево может содержать узлы двух видов: собственно функции и использование описанных ранее функций. Кроме того, функция может иметь свойство групповой, это означает, что ее “дети” фактически находятся вместо нее на том же месте. Связь родительского узла с дочерними может иметь метку, описывающую характер в связи (см. [14, 15]).

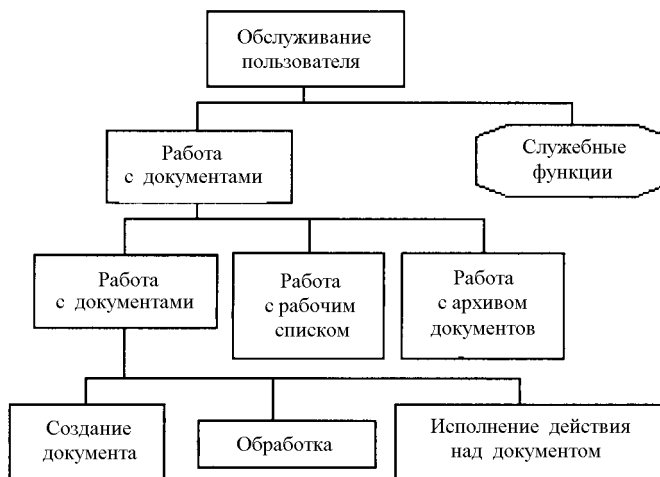


Рис. 2. Пример диаграммы функций.

На рис. 2 показано, как случай использования “Обслуживание пользователя” разбивается на подфункции.

Отметим, что модель случаев использования в Real является подмножеством одноименной модели UML. То, что в UML делается с помощью не вошедшей в Real части модели случаев использования, в Real предлагается делать с помощью модели функций, которая является вариацией функциональной модели из структурных методологий разработки программного обеспечения. Модель функций Real основана на модели функций, предложенной в [15], однако оттуда были убраны некоторые детали (в Real не предполагается так широко использовать модель функций, поскольку нам не хотелось бы подталкивать разработчика к алгоритмическому методу разработки системы) и добавлены использование функций, группы функций, а также связь с моделью случаев использования.

Динамическая модель описывает поведение системы — взаимодействие между различными ее компонентами, взаимодействие системы с ее окружением и поведение самих компонент.

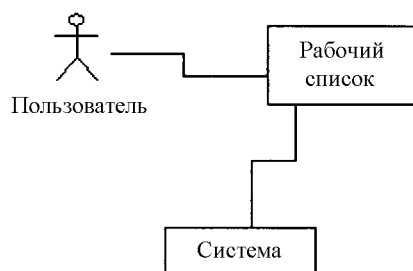


Рис. 3. Пример диаграммы объектов.

На начальных этапах разработки можно придерживаться одной из двух стратегий. Первая: сначала специфицировать классы системы, а затем объекты и сценарии взаимодействия. Она будет использоваться с большей вероятностью, если разработчикам хорошо знакома предметная область. Возможна и другая стратегия — в том случае, если на этапе анализа приходится изучать незнакомую предметную область [16].

Основное назначение *модели объектов* — описание различных ролей, которые могут играть экземпляры классов системы. Каждой функции из функциональной модели Real можно сопоставить диаграмму объектов, назначение которой — описать типичную ”конфигурацию” объектов, задействованных в осуществлении данной функции, а также описать связи между ними. При использовании объектно-ориентированного подхода выполнение функций системы реализуется как совместная деятельность нескольких объектов.

Основными ее элементами являются объекты-роли и отношения между ними. Связь модели объектов и модели сценариев — такая же, как в UML. По диаграмме объектов предусмотрена автоматическая генерация диаграмм классов. На рис. 3 изображена диаграмма объектов для функции ”Обслуживание пользователя”.

Динамику взаимодействия объектов для реализации функции (*модель взаимодействия*) удобно представлять в виде сценариев. В этих сценариях принимают участие объекты-роли, определенные на диаграмме объектов для данной функции или ее надфункций. Сценарий представляет собой упорядоченную во времени последовательность событий, которыми, как правило, являются посылки и приемы сообщений объектами. На рис. 4 изображен один из возможных сценариев функции ”Работа с документами”.

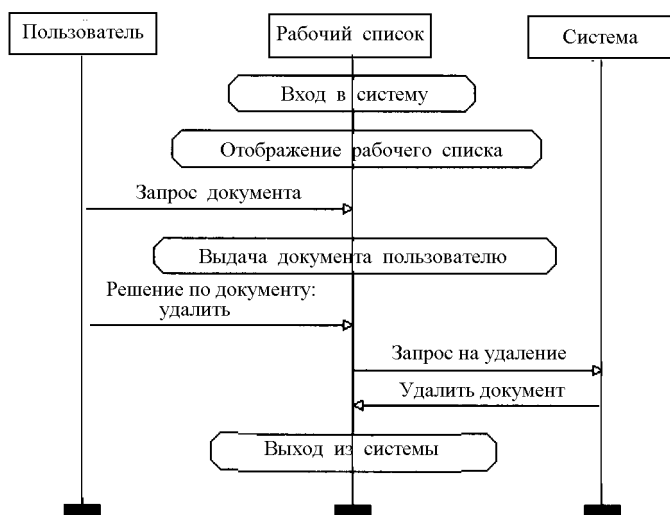


Рис. 4. Пример сценария взаимодействия.

Построение сценариев для функции начинается с определения "прямых веток", т.е. идеального исполнения функции. При этом из рассмотрения исключаются граничные, ошибочные ситуации, частные случаи и т.п., для них впоследствии тоже строятся сценарии либо они специфицируются другими средствами.

Поведенческая модель описывает поведение составляющих систему классов с помощью расширенного конечного автомата и представлена в Real двумя нотациями: в стиле STD [2] и SDL [3]. Фактически, поведенческая модель определяет процессы, протекающие в системе в терминах состояний³, событий⁴ и действий⁵. В дальнейшем будем говорить о поведенческой модели отдельного класса. Построение такой модели можно начать с анализа всех сценариев, в которых участвуют объекты-роли данного класса. Проектирование поведения системы (поведения ее классов) на основе сценариев, а не напрямую, позволяет в

³ Состояние — стабильное положение объекта, когда он готов к принятию запросов на взаимодействие со стороны других объектов. С состоянием может быть связана некоторая деятельность объекта (например, входная и выходная). Состояние может быть сложным, т.е. содержать подсостояния.

⁴ Событие — получение сообщения или окончание срока действия таймера.

⁵ Действие — посылка сообщения, установка таймера, блок кода на целевом языке.

более наглядном виде представлять общие процессы, протекающие в программном обеспечении, и, отталкиваясь от них, конструировать внутреннее поведение участников этих процессов.

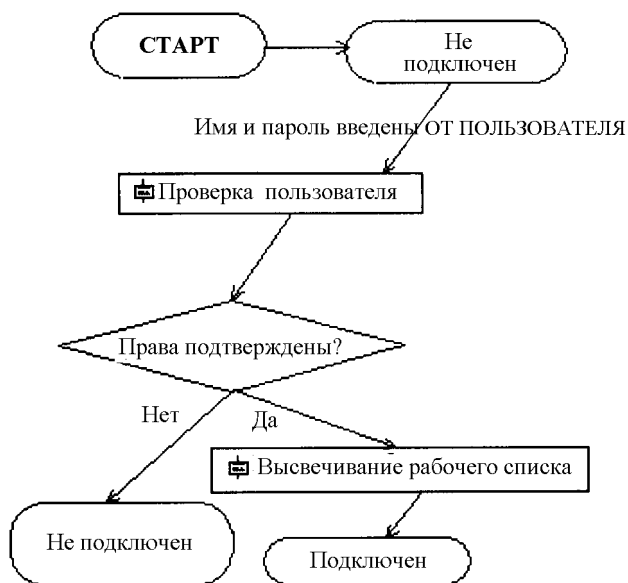


Рис. 5. Пример STD-диаграммы.

На рис. 5 приведена STD-диаграмма, описывающая поведение класса “Рабочий список” системы документооборота.

Для модели сценариев мы использовали вариант MSC-нотации [4, 17]. Более того, Real позволяет связать эту модель с моделью классов иначе, чем в UML — сообщения в сценариях (а также в поведенческой модели) в Real можно выбирать из интерфейсов, описанных в модели классов. Отметим также, что в SDL-методологии никакой формальной связи между SDL и MSC не предполагается, и в известном продукте SDT [19], например, эта связь исключительно динамическая.

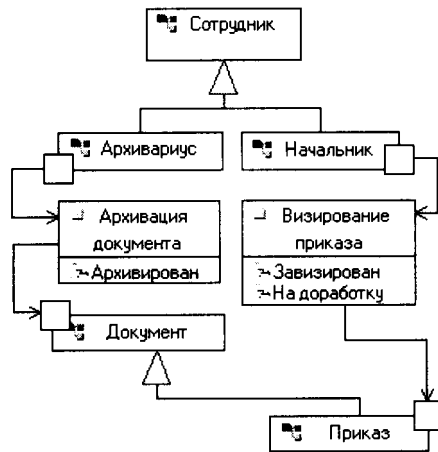


Рис. 6. Пример диаграммы классов.

Статическая модель. После того, как созданы основные сценарии системы, можно переходить к спецификации их участников — объектов, т.е. к построению *модели классов*. Эта модель классов строится на протяжении всего процесса разработки программного обеспечения (подробнее см. [16]).

В Real в модели классов могут быть следующие виды сущностей:

- класс — описание группы однородных объектов;
- шаблон — параметризованный класс с возможностью получения из него обычного класса подстановкой значений параметров;
- интерфейс — описание правил взаимодействия классов;
- представление — аналог конструкции VIEW языка SQL.

На рис. 6 приводится диаграмма классов. На ней изображены классы “Начальник” и “Архивариус”, а также их общий предок — класс “Сотрудник”. Кроме того, там же изображен класс “Документ” и один из его “потомков” — класс “Приказ”. При этом экземпляры класса “Архивариус” могут взаимодействовать с экземплярами класса “Документ” по интерфейсу “Архивация документа”, а экземпляры класса “Начальник” могут взаимодействовать с экземплярами класса “Приказ” по интерфейсу “Визирование приказа”.

Модель классов Real реализует достаточно полное подмножество модели классов UML. Кроме того, в ней есть интерфейсы и порты из

ROOM, при этом последние существенно расширены. Модель классов Real содержит также средства моделирования схемы баз данных.

2. Инструментальные средства

CASE-средство Real представляет собой комплект взаимосвязанных редакторов, объединенных в интегрированную среду разработки:

- текстовый редактор;
- редактор случаев использования;
- редактор функций;
- редактор классов;
- редактор объектов;
- редактор STD-диаграмм;
- редактор SDL-диаграмм;
- редактор сценариев взаимодействия;
- редактор скриптов;
- открытая библиотека доступа к репозиторию.

Все данные проекта хранятся в единой базе данных — репозитории, который может быть размещен в любой реляционной СУБД. Кроме этого, существует библиотека OLE-объектов для доступа к нему как из встроенного в Real редактора скриптов, так и из произвольной среды, поддерживающей OLE-автоматизацию. В Real в качестве языка скриптов используется VBScript — подмножество Visual Basic.

3. Разработка информационных систем

CASE-средство Real поддерживает разработку традиционных уровней информационных систем: схемы базы данных, бизнес-логику, пользовательского интерфейса (экранных форм). Для создания схемы базы данных предлагается использовать модель классов, которая содержит некоторое расширение для баз данных. В этой модели есть несколько элементов, предназначенных непосредственно для проектирования баз данных. Во-первых, это специальный тип атрибута — индекс — с возможностью тиражирования индексов по ассоциациям (foreign keys). Во-вторых, некоторый аналог конструкции VIEW языка SQL, предназначенный для визуального редактирования сложных запросов. Отметим, что в UML, SDL, ROOM такие возможности отсутствуют.

Если схема базы данных изначально не выходит за рамки возможностей, предоставляемых реляционными СУБД, то дальнейшая генерация самой базы данных и обеспечение доступа к ней достаточно прозрачны. Если же в схеме данных использовались элементы объектно-ориентированного подхода (прежде всего наследование), то требуется либо объектно-ориентированная целевая СУБД, либо специальные механизмы работы с данными. Вместе с тем бывает удобно даже с реляционной базой данных общаться в терминах модели классов. В Real с этой целью можно сгенерировать библиотеку классов, берущих на себя большую часть общения с базой данных без использования SQL-запросов.

В стандартную поставку Real входит скрипт, генерирующий схему базы данных на SQL DDL, а также объектно-ориентированную библиотеку доступа к базе данных в виде набора C++-классов и OLE-объектов.

Средства создания экранных форм в Real в настоящий момент находятся в стадии разработки, но основные идеи ясны уже сейчас.

Пользовательский интерфейс в информационных системах, в основном, состоит из различных экранных форм, большая часть которых достаточно однообразна: это либо редактирование полей одной записи базы данных и/или полей записей, с ней связанных, либо редактирование списка однородных элементов и т.п. Генерацию форм можно производить автоматически по модели классов, учитывая связи между классами и их свойства.

Отметим, что во многих CASE-средствах существуют встроенные возможности работы с экранными формами, например, в Designer/2000, CASE/4/0. Однако все большую популярность приобретает тенденция интеграции CASE-средств с известными средами быстрой разработки программного обеспечения — Visual Basic, PowerBuilder, Delphi, поскольку нет необходимости дублировать соответствующие подсистемы этих сред (в данном случае редакторы экранных форм). Именно этот подход реализован в CASE-средстве ERwin (о генерации экранных форм в ERwin можно прочитать в [18]). И Real идет по этому же пути: в настоящий момент ведутся работы по интеграции с Visual Basic, в дальнейшем планируется интегрироваться также с PowerBuilder и Delphi. Отличия Real от ERwin здесь в следующем: мы предлагаем сохранять формы как отдельные элементы модели классов, в то время как ERwin их только генерирует. При этом если изменится структура базы данных, разработчик будет вынужден либо регенерировать формы заново, потеряв все изменения, сделанные вне CASE-средства, либо

”вручную” исправлять эти формы, чтобы они соответствовали изменившейся схеме данных. В нашем подходе хранение полной информации о формах в репозитории позволяет при повторной их генерации (например, после изменения схемы базы данных) учесть и не потерять все изменения, которые были сделаны в Visual Basic.

Для спецификации бизнес-логики предлагается использовать поведенческую модель, которую в дальнейшем предполагается расширить для описания бизнес-процессов.

4. Разработка программного обеспечения систем реального времени

Основное назначение Real применительно к системам реального времени — проектирование сложной управляющей логики с последующей возможностью автоматической генерации кода. Отметим, что методология Real не ориентирована специальным образом на разработку оборудования и программного обеспечения, непосредственно с ним контактирующего (драйверов устройств и т.п.), а также сетевых протоколов нижнего уровня. Однако мы считаем, что для этих задач она подходит примерно в той же степени, как и UML.

Как показывает практика, прямые ветки сложных алгоритмов достаточно удобно и наглядно определять с помощью сценариев. На начальных этапах разработки системы нужно четко определить логику всех взаимодействий. При этом правила поведения системы в ошибочных ситуациях в большинстве случаев можно доработать позднее. По сценариям можно сгенерировать STD- или SDL-диаграммы и продолжить создание спецификаций уже в этом стиле, учитывая все допустимые варианты поведения системы.

В терминах Real основной структурной единицей системы реального времени является объект. Объекты взаимодействуют друг с другом через интерфейсы. Под взаимодействием понимается посылка сообщений, вызов методов и обращение к атрибутам интерфейса. Поскольку в последнее время все большее число систем реального времени становятся распределенными и сетевыми, понятие интерфейса приобретает особую важность. Раньше ситуация была иной, примером чего служат ранние версии языка SDL, в которых интерфейсы отсутствовали. Интерфейсы Real сильно отличаются от портов (gate) SDL (составом, способом связи с классами) и интерфейсов UML (составом, способом свя-

зи с классами, способом изображения), а также интерфейсов в ROOM (составом и способом изображения).

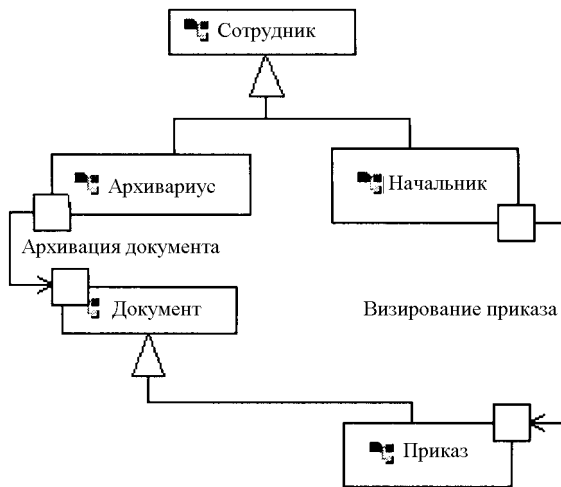


Рис. 7. Пример диаграммы классов с сокращенной нотацией интерфейсов.

В системах реального времени важную роль играют абстракции точек входа и выхода у различных компонент программного обеспечения. Поэтому в модель классов Real был добавлен из ROOM специальный элемент — порт.

CASE-средство Real предусматривает несколько способов изображения портов и интерфейсов. Пример, изображенный на рис. 6, в Real может быть изображен в сокращенной форме (рис. 7). Кроме того, возможен вариант, когда порты не изображаются вообще, а интерфейсы изображаются как ассоциации: это удобно, когда объекты данного класса через данный порт единственным образом могут быть связаны с объектами другого класса.

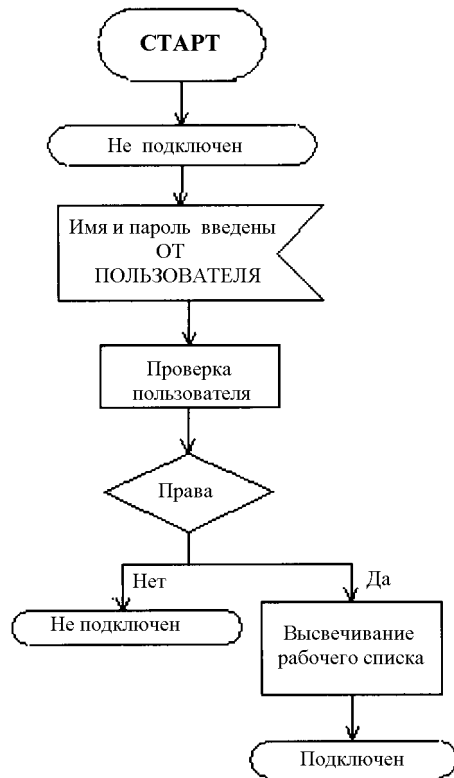


Рис. 8. Пример SDL диаграммы.

Компонента программного обеспечения, определенная в виде класса с портами и интерфейсами, может иметь конечно-автоматное поведение, описываемое в терминах поведенческой модели Real. Поведенческая модель, в свою очередь, представляется двумя альтернативными нотациями: первая основана на варианте STD, представленном в ROOM, вторая — на расширенном конечном автомате SDL. С помощью STD-нотации удобно определять поведение компонент системы на ранних этапах разработки: множество мелких деталей можно временно убрать из поля зрения. В то же время на SDL-диаграммах можно наглядно изобразить мельчайшие подробности алгоритмов.

Эта возможность становится полезной на поздних этапах проектирования. При этом информацию, изображенную на STD-диаграммах, можно "загрузить" на SDL-диаграммы, таким образом результаты ранних этапов при переходе к более формальной спецификации не будут потеряны. В рамках Real, STD и SDL нотации предназначены для описания единой поведенческой модели, так что всегда возможно и обратное — загрузка на STD-диаграмму результатов работы с SDL-редактором. На рис. 8 показана SDL-диаграмма, изображающая тот же фрагмент поведенческой модели, что и SDL-диаграмма на рис. 5. Такой подход отличается от предлагаемого в INSYDE [8]: там STD-диаграммы просто транслируются в SDL, обратная трансляция не предусматривается и целостность двух представлений одной и той же информации не поддерживается. На поведенческую модель Real сильно повлияла технология SDL/PLUS 20]: мы тоже не используем типы данных и выражения SDL, но применяем более гибкую стратегию связи с языками реализации [14] вместо заранее фиксированного языка [20].

5. Заключение

Основной характерной чертой Real является интеграция средств создания информационных систем и систем реального времени. Кратко перечислим моменты, которым было уделено основное внимание:

- средства создания компонентного программного обеспечения;
- проработка конечно-автоматного подхода к спецификации поведения системы;
- средства проектирования баз данных (объектных и реляционных);
- связность моделей и специальные средства для поддержания их непротиворечивости.

Отметим также, что инструментальное средство Real, реализующее одноименную методологию, в ближайшее время выйдет в свет, и все, о чем рассказывается в данной статье, в нем реализовано.

Указатель литературы

1. UML semantics, version 1.1. <http://www.rational.com/uml>. New York 1997. 162 p.
2. Selic B., Gullekson G., Ward P.T. Real-time object-oriented modeling. John Wiley & Sons. Inc. 1994. 525 p.

3. **ITU Recommendation Z.100: Specification and Description Language (SDL).** 1993. 204 p.
4. **ITU Recommendation Z.120 Message Sequence Chart (MSC).** 1996. 95 p.
5. **Marca D.A., McGowan C.L.** SADT: Structured analysis and design techniques New York: McGraw-Hill, 1988.
6. **Вендров А.М.** CASE-технологии: Современные методы и средства проектирования информационных систем. М.: Финансы и статистика, 1998. 176 с.
7. **Yourdon E.** Modern structured analysis. Prentice-Hall, New Jenersy. 1989. 672 p.
8. **Wasowski M., Witaszek D., Verschaeve K.** et al. Methodology (the complete OMT*). Report 1.4, December 1995. 102 p.
9. **Rambaugh J., Blaha M., Premerlani W.** et al. Object-oriented modeling and design. Prentice-Hall. New Jenersy. 1991. 500 p.
10. **Долгов П., Иванов А., Кознов Дм. и др.** Объектно-ориентированное расширение технологии RTST // Записки семинара кафедры системного программирования "CASE-средства RTST++". Вып. 1. — СПб: Издательство С.-Петербургского университета, 1998. С. 17–36.
11. **Иванов А., Кознов Дм., Мурашева Т.** Поведенческая модель RTST++ // Записки семинара кафедры системного программирования "CASE-средства RTST++". Вып. 1. — СПб: Издательство С.-Петербургского университета, 1998. С. 37–49.
12. **Парфенов В.В., Терехов А.Н.** RTST — технология программирования встроенных систем реального времени // Системная информатика. Вып. 5: Архитектурные, формальные и программные модели. Новосибирск, 1997. С. 228–256.
13. **Терехов А.Н.** RTST — технология программирования встроенных систем реального времени // Записки семинара кафедры системного программирования "CASE-средства RTST++". Вып. 1. — СПб: Издательство С.-Петербургского университета, 1998. С. 3–17.
14. **Романовский К., Ивановский Б., Кознов Дм., Долгов П.** Обзор нотаций методологии Real // http://www.tepcom.ru/produkts/real/Report_Notations_A.asp.
15. **ITU SDL methodology guidelines and bibliography.** Appendices i to recommendation Z.100. 1993. 170 p.
16. **Booch G.** Object-oriented analysis and design with application, second edition. The Benjamin / Cummings Publishing Company, Inc. 1994. 589 p.
17. **Andersson M., Bergstrand J.** Formalizing use cases with message sequence charts: Master thesis. Department of Communication Systems at Lund Institute of Technology. EFD LTH. May 1995. 82 p.
18. **Тоцев А.** ERwin и автоматическая генерация кода клиентских приложений // Компьютер Пресс, 1998. № 10.
19. **SDT 3.1.** Technical Presentation. Telelogic, 1997. 54 p.
20. **Бардзинь Я.М., Калкинъш А.А., Стродс Ю.Ф., Сыцко В.А.** Язык спецификаций SDL/PLUS и его применения. Рига, 1988,. 313 с.