Лекция № 7. Визуальное моделирование систем реального времени.

В этой лекции делается обзор использования визуального моделирования при разработке систем реального времени. Далее определяются конечные автоматы, реактивные системы и подробно рассматривается использование диаграмм конечных автоматов (диаграмм состояний и переходов) для моделирования телекоммуникационных систем и аппаратных систем. Делается обзор инструментальных средств.

Обзор области. Визуальное моделирование широко используется при моделировании систем реального времени, так же как при разработке любого ПО, однако при этом имеется ряд особенностей:

- визуальные модели активно используются при структурной декомпозиции систем;
- с их помощью проектируются и визуализируются сценарии работы систем;
- наконец, визуальные модели активно используются при моделировании поведения компонент.

В первом случае визуальные модели позволяют:

- строго описывать компоненты системы и их интерфейсы, а также связи между компонентами (как правило, в виде каналов, по которым компоненты обмениваются сообщениями); в UML 2.0. для этих целей используются диаграммы компонент;
- выполнить иерархическую декомпозицию сложной системы на множество вложенных уровней; эти уровни (здесь используются различные термины, например, блоки, подсистемы) также как и компоненты, имеют хорошо определенные интерфейсы, связываются друг с другом каналами, а в «листьях» этой декомпозиции содержатся компоненты; отличие компонент от подсистем в том, что у первых есть поведение (то есть исполняемый код), а последние – «пустая» обертка, содержащая в себе другие подсистемы и или компоненты; кроме того, в качестве средств декомпозиции визуальные формализмы часто поддерживают парадигму многоуровневых сетевых протоколов, закрепленную эталонной моделью ISO/OSI; в UML 2.0 для этих целей используются диаграммы композитных структур.

При моделировании и визуализации сценариев работы систем используются различные варианты диаграмм сценариев (например, диаграммы последовательности UML 2.0), позволяющие в наглядной форме показать порядок обмена компонент сообщениями. Такие диаграммы бывают крайне полезны при изучении телекоммуникационных стандартов (то есть при анализе и на ранних стадиях проектирования), в проектировании – при моделировании отдельных критических участков взаимодействия компонент, в тестировании – при анализе трасс работы системы.

Кроме того, известны интересные подходы для использования таких диаграмм для моделирования производительности работы систем реального времени (performance modeling).

При моделировании поведения систем реального времени традиционно используются конечные автоматы и их визуальная нотация – диаграммы состояний и переходов (state transition diagrams, statecharts, в UML 2.0 эти диаграммы носят названия диаграмм конечных автоматов). Конечные автоматы активно используются при проектировании, в частности, телекоммуникационных и аппаратных систем. Кроме того, этот формализм часто используется при тестировании – с его помощью описываются поведенческие требования к подсистемам, а также моделируются тестовые обходы системы.

Мы остановимся на конечных автоматах и рассмотрим две интересные практические области их использования – при проектировании событийно-ориентированной логики телекоммуникационных систем, а также для проектирования интегральных схем в контексте использования языка VHDL. И в том и в другом случае по спецификациям генерируется исполняемый код, и эта возможность действительно широко используется в промышленности, поддерживается зрелыми средствами разработки.

Средства структурной декомпозиции систем реального времени были тщательно проработаны в рамках языка SDL [8]. Именно там такие концепции как блок, канал, были впервые строго определены как абстракции моделирования, снабжены исполняемой семантикой и графической нотацией. В методологии ROOM [9] было формализовано понятие интерфейса для компонент систем реального времени, а также введено понятие порта как абстракции точки подключения. Аналогичные концепции (точка подключения, линия подключения, виртуальный канал и пр.) были предложены в [11, 12], в рамках метода создания телекоммуникационных систем. Расширение концепции интерфейса и порта для удобств визуального моделирования компонентных систем (введение в интерфейс методов, переменных и сообщений) предложено в [10].

Интересные средства моделирования множественных соединений и поддержка уровневой декомпозиции компонент имеются в ROOM [9]. Такие концепции языка SDL как сложные каналы и блоки с подблоками, а также точки соединения (connects) и порты (gates) для типов блоков также позволяют создавать многоуровневые структуры, однако таких удобных средств, как в ROOM, в SDL для этих целей нет. В UML 2.0. в диаграммах компонент и композитных структур реализованы идеи ROOM.

Самым зрелым на сегодняшний момент языком сценариев является стандарт язык MSC [7]. Он позволяет создавать не только прямые ветки сценариев, но и полноценные «ветвящиеся» алгоритмы в виде иерархии диаграмм. Кроме того, в этом стандарте есть интересные средства для декларативного задания поведения – так называемые недетерминизмы. Диаграммы последовательности UML 2.0 в существенной степени базируются на MSC. Эти диаграммы используются также при моделировании производительности (performance modeling) систем реального времени и информационных систем [15].

Диаграммы состояний и переходов как строгий и формальный язык моделирования был введен Харелов в 1987 году [5], улучшен и расширен в [6]. Однако еще до этого времени, в рамках языка SDL, фактически, эти же диаграммы уже активно развивались и использовались. На настоящий момент полноценный вариант этих диаграмм реализован в UML 2.0. Основной способ использования этих диаграмм в индустрии – проектирование сложной событийно-ориентированной логики компонент с последующей генерацией исполняемого кода. Используются эти диаграммы и в тестировании – для описания сложных поведенческих требований к подсистемам [14], а также для моделирования тестовых обходов системы [13].

О конечный автоматах. Множество процессов в технике и бизнесе хорошо моделируются с помощью конечных автоматов. Говоря неформально, конечный автомат – это устройство или процесс, реакция которого на внешние воздействие зависит не только от тог, какого это воздействие, но также и от истории его работы. Использование конечных автоматов при разработке ПО и аппаратуры в виде диаграмм повышает концептуальный уровень разработки и, что немаловажно– дает возможность автоматически генерировать программный код.

Формально определим конечный автомат (автомат Мили) как систему S = {A, Q, V, δ, χ}, в которой:

$A = \{a_1, \ldots, a_n\}$ – входной алфавит;
$V = \{v_1, \ldots, v_m\}$ – выходной алфавит;
$Q = \{q_1, \ldots q_k\}$ – алфавит состояний;
δ: Q x A -> Q – функция перехода;
χ: Q x A - > V – функция вывода.

Обычно выделяют одно состояние в качестве начального. Работа автомата представляется как переработка входного потока в выходной. Из входного потока читается очередной символ и в соответствии с текущим состоянием происходит запись определенного символа в выходной поток. А автомат переходит в следующее состояние.

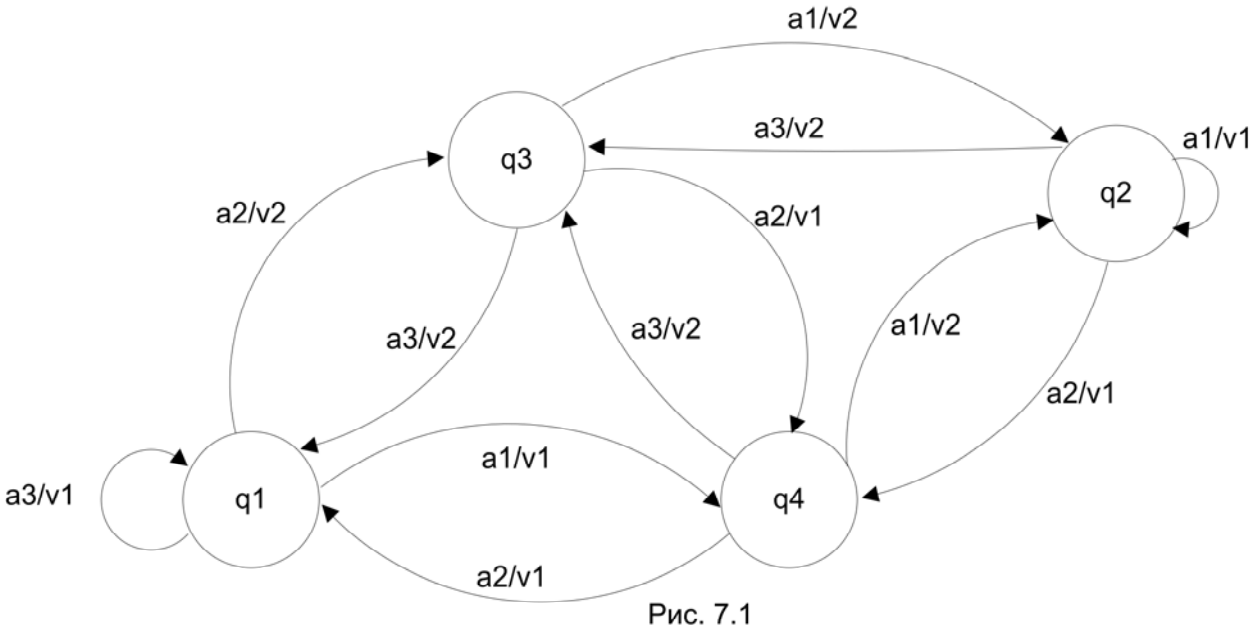

Рис. 7.1

На рис. 7.1 изображен автомат с алфавитом состояний Q = {q1,q2,q3,q4}, входным алфавитом A = {a1,a2,a3}, выходным алфавитом V={v1,v2}.

Автоматом Мура будем называть автомат, в котором функция χ не зависят от входного символа.

Легко показать, что автомат Мура и автомат Мили задают в точности оно и то же множество алгоритмов.

Реактивные системы. Приведем теперь общие признаки систем, которые удобно моделировать с помощью конечных автоматов:

- наличие постоянного взаимодействия с окружением, причем это взаимодействие может носить асинхронный, непредсказуемый характер;
- прерываемость, т.е. система должна быть готова обрабатывать запросы наивысшего приоритета в то время, когда она занята какой-либо другой работой;
- реакция на запросы имеет строгие временные ограничения;
- наличие различных сценариев работы системы, которые зависят от значения каких-либо данных и от прошлого (истории) системы;
- организация системы в виде параллельно работающих компонент.

Такие системы получили название реактивных (reactive). Самым распространенным случаем реактивных систем являются встроенные устройства – то есть электронно-механические системы, работающие автономно (то есть независимо от человека) и активно взаимодействующие с внешней средой. Примером таких систем являются различные бортовые компьютеры (самолета, автомобиля и пр.), мобильные телефоны и пр. Эти устройства удобно представлять как системы, ожидающие и обрабатывающие сигналы, принадлежащие некоторому конечному множеству. При этом естественно считать, что в каждый момент времени они пребывают в каком-то фиксированном состоянии, и, в соответствии с этим, определяется степень их готовности к обработке того или иного внешнего сигнала. Кроме того, часто они не только делают какую-либо внутреннюю работу, но и выдают определенные сигналы вовне.

Мы рассмотрим два вида таких систем, при разработке которых активно используются конечные автоматы: интегральные микросхемы и встроенных систем телекоммуникационные протоколы.

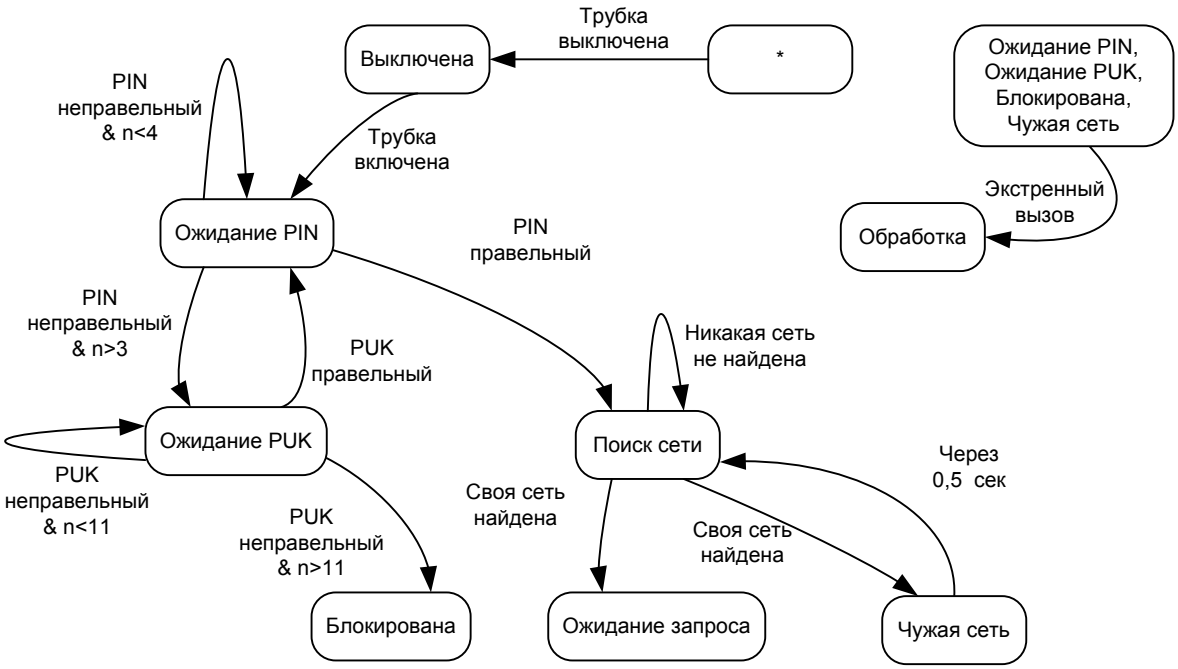
Выключена
Трубка выключена
*
Ожидание PIN, Ожидание PUK, Блокирована, Чужая сеть
PIN неправельный & n<4
Трубка включена
Экстренный вызов
Обработка
Ожидание PIN
PIN правельный
PIN неправельный & n>3
PUK правельный
Никакая сеть не найдена
Ожидание PUK
Поиск сети
Через 0,5 сек
PUK неправельный & n<11
Своя сеть найдена
PUK неправельный & n>11
Своя сеть найдена
Блокирована
Ожидание запроса
Чужая сеть

Рис. 7.2

Моделирование аппаратуры с помощью конечных автоматов. Конечные автоматы являются адекватным понятийным представлением последовательного устройства. Большинство аппаратных устройств, начиная с триггера и заканчивая процессором, может быть выражена конечным автоматом или набором объединенных конечных автоматов. Типичная логическая схема устройства может быть представлена в виде черного ящика, имеющего входы и выходы. Поведение данного устройства определяется логикой, реализуемой внутри этого черного ящика.

Введение в язык VHDL.

Пример. Данный пример конечного автомата был взят из области проектирования FPGA-модулей[1]. FPGA предоставляет набор логических вентилей и регистров, позволяя разработчику описывать способ их соединения друг с другом, задавая тем самым логику его работы. В проекте, который послужил основой для данного примера, используется FPGA из семейства Virtex 2, производимого фирмой XILINX[2]. Взятый в качестве примера FPGA-модуль расположен на одной плате с DSP-процессором[3] TigerShark и предназначен для проведения несложного тестирования работы процессора.

Процессор TigerShark предоставляет несколько интерфейсов для взаимодействия с другими аппаратными компонентами. Имеется подробная спецификация интерфейсов данного процессора. В рассматриваемом проекте процессор тестируется на соответствие лишь небольшому подмножеству этой спецификации.

В данном примере есть четыре взаимодействующих процесса:

Процесс reset_p. Он посылает процессору сигнал reset («сброс»). Согласно спецификации через 128 тактов процессор должен сбросить этот сигнал. Процесс reset_p проверяет, был ли этот сигнал вовремя сброшен DSP-процессором.

Процесс read. Читает данные из некоторого регистра процессора TigerShark.

Процесс write. Записывает данные в некоторый регистр процессора TigerShark.

Процесс dispatcher. Организует работу трех вышеперечисленных процессов.

Опишем работу диспетчера более подробно. Сначала он находится в состоянии idle (начальное состояние). Затем после выполнения некоторых начальных действий переходит в состояние reset, затем начинает проверять реакцию TigerShark на сигнал «сброс», взаимодействуя при этом с процессом reset_p.

После окончания тестирования процессора на сигнал reset, диспетчер достигает состояния read_SYSCON и переходит к тестированию операций чтения и записи. Сначала он читает данные из регистра SYSCON и убеждается, что они именно такие, какими должны быть изначально (т. е. он должен прочесть некоторое значение по умолчанию). При этом диспетчер взаимодействует с процессом read. Далее, побывав в состоянии write_block, диспетчер начинает взаимодействовать с процессом write, а именно записывает в некоторые регистры процессора тестовые данные длиной 8 байт. Диспетчер

---

[1] FPGA – Field Programmable Gate Array

[2] www.xilinx.com

[3] DSP – Digital Signal Processor

проходит состояние read_comp_block и опять начинает взаимодействовать с read. Данные из регистров, в который происходила запись, считываются, и происходит сравнение данных, которые были прочитаны, с ранее записанными в регистры. Тем самым осуществляется проверка правильности операций чтения и записи в регистры процессора TigerShark. Отметим, что существует несколько способов чтения/записи значений регистров, в данном примере проверяется только способ, принятый по умолчанию.

Завершив проверку чтения/записи диспетчер достигает состояния changemode. Теперь для окончания тестирования ему осталось изменить состояние DSP-процессора. TigerShark может находиться в двух состояниях: master и slave. Сразу после включения питания процессор находится в состоянии slave. Это означает, что во время тестирования он не сможет выполнять никаких других программ. По завершении тестирования его нужно перевести в состояние master, чтобы он мог начать автономную работу. Диспетчер осуществляет переключение состояния процессора и попадает снова в свое начальное состояние idle.

Ниже приведен VHDL-код, определяющий логику работы FPGA-модуля, и STD-диаграмма, полученная в результате возвратного проектирования конечного автомата, описанного в процессе dispatcher.

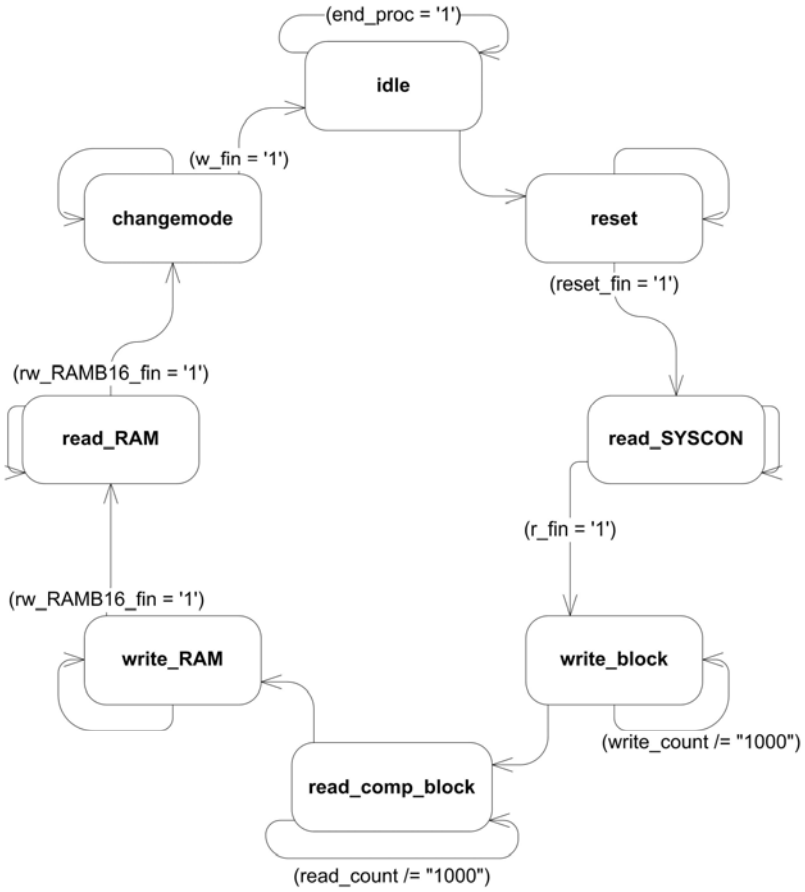

Рис. 7.3.

Основная литература

1. Ю.Г.Карпов. Теория автоматов. – СПб.: Питер, 2003. – 206 с.
2. Е.А.Суворова, Ю.Е.Шейлин. Проектирование цифровых схем на VHDL. – СПб.: «БХВ-Петербург», 2003. – 556 с.
3. Д.В. Кознов. Языки визуального моделирования: проектирование и визуализация программного обеспечения. – Учебное пособие. Изд-во СПбГУ, 2004. – 143с.
4. Г.Буч, А.Якобсон, Дж. Рамбо. UML. – Изд. 2-ое. Питер, 2006. – 735 с.

Дополнительная литература

5. Harel D., Statecharts: a visual formalism for complex systems. – Sci. Computer Program., vol.8, 1987. – P. 213-274.
6. D.Harel, M.Politi. Modeling Reactive Systems with Statecharts: state machine approach. – McGraw-Hill. 1998. – 258 p.
7. ITU Recommendation Z.120. Formal description techniques –Message Sequence Chart. – 1999. – 98 p.
8. ITU Recommendation Z.100: Specification and Description Language. – 08/2002. – 206 p.
9. B.Selic, G.Gullekson, P.T. Ward. Real-Time Object-Oriented Modeling. – John Wiley & Sons. Inc. 1994. – 525 p.
10. Кознов Дм. В. Проблемы разработки компонентного программного обеспечения. //Объектно-ориентированное визуальное моделирование / Под ред. Проф. Терехова А.Н. – СПб: Издательство С.-Петербургского университета, 1999. – С.86-100.
11. 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.
12. В.В.Парфенов "Проектирование и реализация программного обеспечения встроенных систем с использованием объектно-базированного подхода". Автореферат на соикания степени кандидата ф.-м. наук. СПб: Издательство С.-Петербургского университета, 1995. // или http://www.math.spbu.ru/.
13. В. В. Кулямин, А. К. Петренко, А. С. Косачев, И. Б. Бурдонов. Подход UniTesK к разработке тестов. – Программирование, 29(6), 2003. – С. 25-43.
14. П.Д. Дробинцев. Интегрированная технология обеспечения качества программных продуктов с помощью верификации и тестирования. Диссертация на соискание ученой степени кандидата технических наук. – СПб, СПГПУ, 2006. – 238 с.
15. E.Dimitrov, A.Schmietendorf, R.Dumke. UML-Based Performance Engineering Possibilities and Techniques. – Software, January/February 2002 (Vol. 19, No. 1). – P. 74-83.