УДК 004.4'244

МЕТОДИКА СИНТЕЗА ТЕСТОВ АППАРАТУРЫ ПО СПЕЦИФИКАЦИЯМ НА ЯЗЫКЕ UML

А. В. Березкин,
магистр техники и технологии, аспирант
А. С. Филиппов,
канд. техн. наук, доцент
Санкт-Петербургский государственный политехнический университет

Язык UML рассматривается как язык описания спецификаций аппаратуры, из которых могут быть получены ее поведенческие тесты. Предлагается использовать данный язык в начале и в середине маршрута проектирования цифровых устройств, когда определяется их структура и функциональность на уровне последовательности управляющих воздействий. Эти спецификации являются документами, по которым создается RTL-описание устройств, а разработанная методика служит для проверки соответствия RTL-описаний UML-спецификациям. Данная проверка осуществляется путем генерации тестов устройств на основании UML-спецификаций.

Ключевые слова *— UML, моделирование аппаратуры, верификация, тестирование.*

Введение

Для задания спецификации на встраиваемые системы удобно использовать языки описания высокого уровня. Наиболее часто используется язык UML (Unified Modeling Language 2.0) [1], специально предназначенный для решения подобных задач и поддержанный инструментальными средствами. Существующие работы (см. далее) представляют методики, позволяющие выполнять формальную верификацию спецификаций на языке UML. Разработанные по данным спецификациям системы тоже верифицируются, однако этот процесс требует введения дополнительных информационных сущностей в процесс разработки. Иными словами, на данный момент нет средств, позволяющих выполнять верификацию систем на основе непосредственно их спецификаций.

Одним из важнейших методов верификации является тестирование — проверка систем в ходе их нормальной работы в рамках заранее подготовленных сценариев, в соответствии с которыми и разрабатывают тесты.

Актуально синтезировать функциональные тесты непосредственно на основе спецификации системы. Целью данной работы является разработка методики синтеза таких тестов. Проблема рассматривается, в основном, для аппаратуры, однако принципы, заложенные в методику, могут быть применены и к программному обеспечению.

Верификация в маршруте проектирования вычислительных систем

Синтез функциональных тестов выполняется исходя из спецификации системы, которая создается в ходе ее проектирования. Следовательно, этот процесс является частью всего маршрута проектирования вычислительных систем. Определим точки маршрута, в которых следует синтезировать и использовать такие тесты. Рассмотрим для этого упрощенную схему маршрута проектирования цифровых систем (рис. 1).

В ходе проектирования системы разработчики создают различные информационные сущности (артефакты), описывающие систему или отдельные ее свойства. На начальных этапах проектирования эти сущности абстрактны (модели, неформальные описания), на более поздних — конкретны (код, образы памяти, используемые компоненты). Создаваемые артефакты анализируются различными методиками верификации. К примеру, техническое задание может подвергаться экспертизе; спецификация системы — проверке моделей; программный код — статическому анализу. В любом случае верификация предполагает проверку соответствия друг другу двух артефактов, даже если один из них не является прямым продуктом процесса разработки конкретной системы (например, в случае статического анализа программный код проверяется

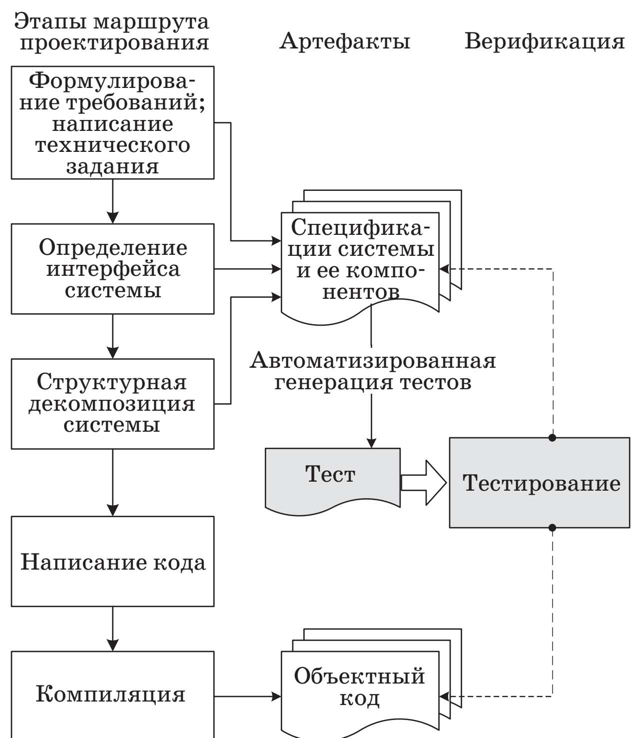
■ *Рис. 1. Этапы и артефакты маршрута проектирования*

с помощью такого «внешнего» артефакта, как набор обобщенных правил написания корректного кода).

Существенно отметить, что при современных технологиях проектирования цифровых устройств и систем для описания аппаратуры используются языки высокого уровня HDL. Эти формализованные описания представляют собой «аппаратные коды», которые могут быть использованы для автоматизированной компиляции схем аппаратуры. Таким образом, проектирование аппаратного обеспечения систем во многом стало походить на проектирование программного обеспечения.

Тестирование проверяет соответствие разработанной системы (т. е. программного и аппаратного кода) набору требований к работе системы. Если данный набор требований выражен достаточно строго в виде спецификации системы, он может быть использован для генерации тестов.

Предлагаемый подход представлен на рис. 2. На первых этапах проектирования создается набор строгих спецификаций на некотором универсальном языке. Далее на основании этих спецификаций разработчики пишут исходный код компонентов системы. После компиляции системы появляется возможность ее тестирования,



■ *Рис. 2. Включение автоматизированного тестирования в маршрут проектирования*

и для этого предлагается использовать тесты, сгенерированные из спецификаций, созданных на начальных этапах разработки. Таким образом, предлагаемый подход позволяет установить, насколько разработанная система соответствует требованиям спецификаций, сформулированным ранее. Полнота сгенерированных тестов целиком зависит от того, насколько полная спецификация представлена генератору тестов.

Для того чтобы предложенный подход было удобно применять, язык описания спецификаций системы должен удовлетворять некоторым требованиям. Этот язык должен быть распространенным и доступным, универсальным, способным представлять как абстрактные сущности, так и конкретные. Немаловажна также инструментальная поддержка языка. Всеми перечисленными особенностями обладает UML, что и служит причиной его выбора.

Другая причина выбора языка — огромное количество исследований, посвященных языку и его применениям. В настоящий момент UML рассматривается разработчиками и исследователями не только как язык моделирования программного обеспечения, но и как универсальный язык проектирования любых систем и процессов с применением объектно-ориентированного подхода (см., например, работы [2–5]).

В настоящий момент известно также большое количество программных средств для работы с UML. Обширные списки таких средств можно легко найти на открытых интернет-ресурсах, например на странице [6]. Их анализ показывает, что генерация кода из UML-описаний — хорошо проработанная задача. Как видно, имеется целый ряд средств, которые могут генерировать код на таких языках, как C/C++, C#, Java. Однако весьма затруднительно найти средство, генерирующее аппаратный код из UML-описания. В то же время имеется множество литературы, посвященной методикам генерации аппаратного кода из UML-описания. В качестве примеров можно привести работы [4, 5]. Спецификации, рассматриваемые в данных работах, носят как структурный, так и поведенческий характер. В них предлагается устройства и другие элементы аппаратуры (в том числе их входы и выходы) описывать с помощью классов, а их соединения — с помощью стереотипов и ассоциаций. Предлагается также использовать диаграмму состояний для задания конечных автоматов.

Рассмотренные источники представляют существенный интерес для нашей темы — генерации аппаратных тестов, поскольку тест сам является определенным образом организованным алгоритмом, а при практическом подходе — программным или аппаратным кодом. В то же время генерация теста является отдельной задачей, хотя и связанной с генерацией кода. Например, в работе [7] рассматривается методика генерации простых тестов объектно-ориентированного дизайна программной системы. Но основная масса работ, посвященных генерации тестов из UML-описания, носит теоретический, а не практический характер, как, например, работы [8, 9], в которых описываются алгоритмы генерации тестовых примеров (test case) из диаграмм классов, состояний, коммуникации и деятельности. Работ, в которых анализируется практический подход, гораздо меньше, что открывает большие перспективы для исследований.

Учитывая все вышесказанное, можно отметить, что наибольший интерес представляет задача генерации аппаратных тестов из UML, которая и рассматривается в статье.

Использование UML для проектирования аппаратных систем

Генерация аппаратных тестов из спецификации системы — одна из разновидностей задач, связанных с UML. Следовательно, не все элементы языка UML целесообразно использовать для решения этой задачи. Рассмотрим диаграммы UML, которые отражают структуру и функциональность системы, достаточную для генерации кода и, в частности, тестов. В источниках, упомянутых в предыдущем разделе, используются следующие шесть диаграмм UML: диаграмма классов, диаграмма состояний, диаграмма деятельности, диаграмма коммуникации, диаграмма последовательности и диаграмма синхронизации. Поэтому можно предположить, что и для нашей задачи понадобятся диаграммы из этого набора.

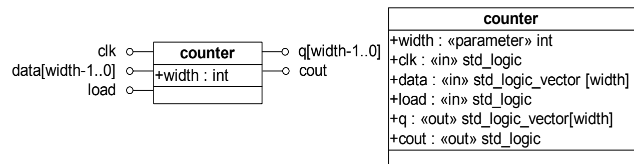
Для выделения набора диаграмм, необходимых для генерации аппаратных тестов, определим, какие данные требуются для создания таких тестов.

- Информация об интерфейсе системы (входы, выходы, параметры), поскольку тест будет воздействовать на систему через ее интерфейсную часть. Для задания такой информации лучше всего подходит диаграмма классов, поскольку она создана для описания статических объектов.
- Информация о последовательности тестовых воздействий на устройство, т. е. об алгоритме тестирования. Алгоритмы описываются в UML с помощью диаграмм деятельности.
- Оценка реакции системы на внешние воздействия во время тестирования. Для этой цели необходимо описать правильное поведение устройства с точностью до внутренних сигналов. Это можно сделать с помощью диаграммы последовательности, которая позволяет устанавливать в произ-

вольной форме причинно-следственные связи между различными событиями.

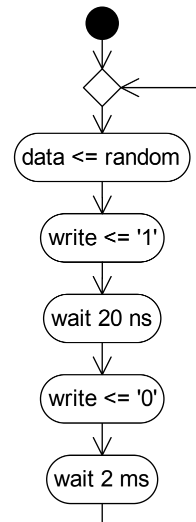
Таким образом, в нашем случае достаточно применения трех видов диаграмм: диаграммы классов, диаграммы деятельности и диаграммы последовательности. Рассмотрим кратко методы использования этих диаграмм в описании аппаратных систем.

Два варианта использования диаграммы классов для описания интерфейса простого устройства (счетчика с параллельной загрузкой) показаны на рис. 3.



■ *Рис. 3.* *Использование диаграммы классов UML для описания интерфейса счетчика*

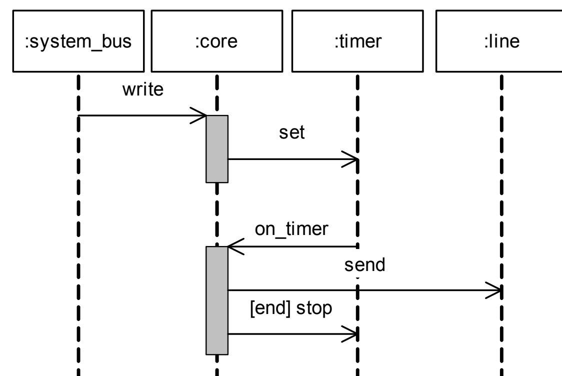
На примере слева для отображения входов и выходов счетчика используются объекты типа «интерфейс», а параметр устройства моделируется с помощью атрибута объекта. На примере справа входы, выходы, а также параметры устройства моделируются с помощью атрибутов с разными нестандартными для UML типами и стереотипами. Предлагается использовать следующие стереотипы: «parameter» для параметров устройства, «in» для входов устройства и «out» для его выходов. Для атрибутов со стереотипами «in» и «out» возможно также использование нестандартных типов — std_logic и std_logic_vector. Данные типы полностью соответствуют одноименным типам из библиотеки IEEE std_logic_1164 для VHDL [10], включаемой во все средства проектирования на VHDL.



■ *Рис. 4.* *Использование диаграммы деятельности для описания воздействий на устройство*

Первый способ более универсальный: его можно использовать для формирования структурных спецификаций устройства, соединяя входы и выходы устройств линиями ассоциации. Второй способ проще для генерации кода: в самом деле, все интерфейсные характеристики устройства находятся внутри одного объекта.

Пример использования диаграммы деятельности для описания воздействий, которые подаются на некоторое устройство в процессе его нормальной работы, показан на рис. 4. Диаграмма деятельности позволяет синтезировать генерирующую часть теста, которая является сценарием тестирования.



■ *Рис. 5.* *Использование диаграммы последовательности для описания поведения устройства*

Пример использования диаграммы последовательности для описания работы некоторого абстрактного передатчика показан на рис. 5. Диаграмма последовательности используется для отображения внутреннего поведения устройства, для пояснения взаимосвязи между возникающими в нем управляющими сигналами.

Прямоугольники в верхней части диаграммы отображают компоненты устройства. Вертикальные пунктирные линии — «линии жизни». Стрелками обозначаются сообщения, которыми обмениваются компоненты устройства. Сообщение трактуется как переход одного управляющего сигнала из неактивного состояния в активное (например, из «0» в «1»). Вертикально вытянутые прямоугольники — спецификации исполнения (execution specification). В нотации UML 2.0 данными элементами обозначаются части линии жизни объекта, представляющие целостный период его работы. В нашем случае они трактуются как элементы, объединяющие сообщения в причинно-следственные связи. Такая трактовка почти не отступает от стандартной трактовки спецификации исполнения, однако не предусматривает задания в явном виде некоторых ограничений. Пример из рассматриваемого рисунка: после сиг-

нала write, направляемого в core, указанный узел отправляет сигнал set таймеру.

Диаграмма взаимодействия может дополняться разнообразными условиями, которые могут по-разному трактоваться (в зависимости от реализации). В рассмотренном рисунке условие сообщения stop записано в квадратных скобках, что является отступлением от нотации UML 2.0 (но допустимо в UML 1.x). Для обеспечения совместимости такой записи с UML 2.0 условия можно задавать как часть имени сообщения. Также могут добавляться временные требования, ограничения, счетчики и т. д.

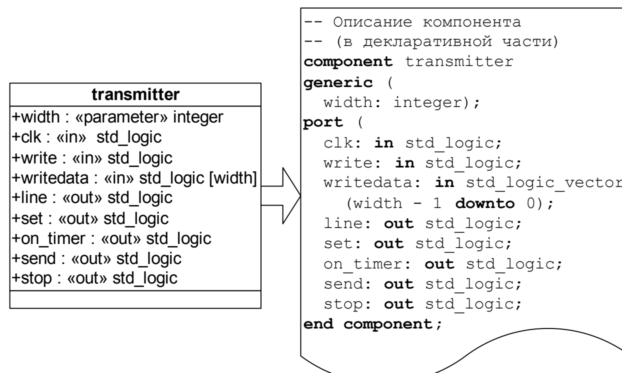
Использование диаграммы взаимодействия дает возможность генерировать наблюдающую часть теста — собственно верификатор, который проверяет последовательность управляющих сигналов, сравнивая ее с эталонной. Иными словами, представленный подход позволяет тестировать поток управления цифрового устройства. Для тестирования других аспектов системы (потока данных, быстродействия, пропускной способности и т. д.) необходимо использовать другие специализированные тесты.

Генерация тестов аппаратуры из диаграмм UML

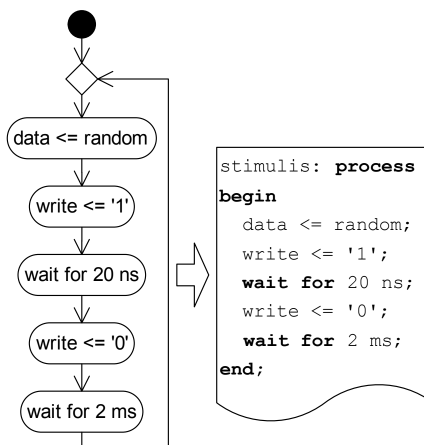
Один из наиболее распространенных языков для создания тестов аппаратуры — VHDL Testbench. Тестирование с помощью данного языка поддерживается многими средствами проектирования и моделирования аппаратуры. В дальнейшем изложении мы будем опираться на этот стандарт как на целевой.

Как говорилось ранее, для генерации тестов используется три вида диаграмм UML: диаграмма классов для извлечения интерфейсной информации, диаграмма деятельности для извлечения последовательности тестовых воздействий и диаграмма последовательности для извлечения поведенческой информации. Рассмотрим использование этих видов диаграмм в задаче генерации тестов.

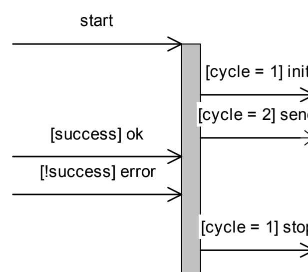
Применение диаграммы классов в генерировании интерфейсной части testbench (описание интерфейса тестируемого устройства) демонстрирует рис. 6. Тестируемое устройство — передатчик из рис. 5. Необходимо отметить, что, поскольку в testbench тестируемое устройство рассматривается как цельный блок, его внутренние сигналы, фигурирующие в диаграмме последовательности, недоступны. Поэтому, чтобы сделать их наблюдаемыми, необходимо вынести их на интерфейсный уровень, что должно найти отражение как на диаграмме классов, так и в коде устройства. Например, на рассматриваемом рисунке



■ *Рис. 6.* *Генерация интерфейсной части testbench из диаграммы классов UML*



■ *Рис. 7.* *Пример использования диаграммы деятельности для синтеза генерирующей части теста*




```
if ((state = idle) and (start = '1')) then
  cycle := 0;
  state <= after_cluster_0;
elsif ((state = after_cluster_0)
    and ((ok = '1') or (error = '1'))) then
  cycle := 0;
  state <= after_cluster_2;
elsif ((state = after_cluster_2)
    and (stop = '1')) then
  state <= idle;
else
  cycle <= cycle + 1;
end if;
```

■ *Рис. 8.* *Генерация верификатора из одной спецификации исполнения*

сигналы set, on_timer и др. являются внутренними, вынесенными на интерфейсный (внешний) уровень.

Рассмотрим использование диаграмм деятельности для синтеза генерирующей части теста. Поскольку диаграммы деятельности представляют собой схемы алгоритмов, их трансляция в код не представляет сложностей (рис. 7).

Рассмотрим использование диаграмм последовательности для генерации наблюдающей (верифицирующей) части теста. Генератор этой части теста оперирует со спецификациями исполнения, которые объединяют сообщения в причинно-следственные цепочки. Каждая такая цепочка транслируется в конечный автомат. «Входящие» в спецификацию исполнения сообщения управляют конечным автоматом, т. е. приводят к изменению его состояния. Входящие сообщения считаются причинами. «Исходящие» сообщения считаются следствиями, их наличие проверяется конечным автоматом, когда он находится в определенных состояниях. Поясним сказанное примером.

На рис. 8 показана часть некоторой диаграммы последовательности — спецификация исполнения, расположенная на линии жизни некоторого гипотетического управляющего устройства. Ниже приведен соответствующий ему код, управляющий состоянием верификатора.

Как видно из рисунка, вся спецификация исполнения делится на «кластеры», каждый из которых — это группа смежных входящих сообщений. Каждому кластеру ставится в соответствие одно состояние управляющего конечного автомата. Переход от одного состояния (кластера) к другому происходит под воздействием входящих сообщений за исключением самого последнего сообщения, которое используется для остановки верификатора независимо от того, какое это сообщение.

Верификатор может использовать ряд внутренних счетчиков-переменных. В рассматриваемом примере используется счетчик cycle, значение которого равно количеству тактов после последнего входящего сообщения. Также используется объект success, который может быть сигналом, счетчиком, внутренней переменной.

Каждое исходящее сообщение при генерации порождает одно проверяющее утверждение (assert ... report ... severity). Например, второе исходящее сообщение ([cycle = 2] send) порождает утверждение вида

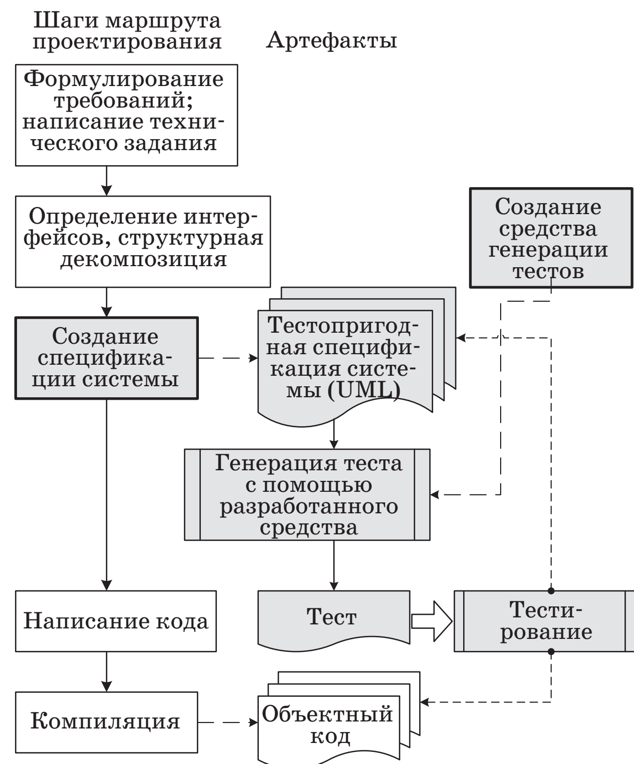
```
if ((state = after_cluster_0) and (cycle = 2)) then
    assert (send = '1')
    report ("send error")
    severity error;
end if;
```

Таким образом, верификатор проверяет, что в заданное время при заданных условиях сообщение, которое должно появиться согласно спецификации, действительно появляется в устройстве.

Применение методики

Применение методики включает в себя два этапа, на каждом из которых данная методика применяется полностью, но с разной целью. Первый раз — для создания спецификации системы, которая может быть использована для генерации теста («тестопригодная» спецификация); второй раз — для создания средства преобразования этой спецификации в аппаратный тест. Применение методики на первом этапе полностью вписывается в маршрут проектирования цифровой системы и соответствует схеме, представленной на рис. 2. Реализация второго этапа выполняется независимо, один раз. Первый этап выполняется на основании материала, изложенного в разд. «Использование UML для проектирования аппаратных систем»; второй этап — с использованием материала из разд. «Генерация тестов аппаратуры из диаграмм UML».

Полная схема применения методики (оба этапа) представлена на рис. 9. Используются следующие обозначения:



■ *Рис. 9.* *Схема применения методики*

• этапы применения методики подсвечены серым и обведены утолщенной рамкой;

• процессы, которые выполняются автоматизированно благодаря применению методики, также подсвечены и выделены курсивом;

• остальные подсвеченные элементы — это артефакты, получаемые в результате применения методики.

Выбор технических средств для создания генератора тестов

Для создания генератора тестов подходит любое средство проектирования UML, которое поддерживает генерацию кода на произвольном языке. На данный момент имеется большое количество таких средств, как платных, так и свободных. Многие средства поддерживают импорт и экспорт моделей через формат XMI, который является разновидностью XML для обмена метамоделями [11], и это позволяет отделить процесс разработки UML-диаграммы от ее преобразования в тест. В ряде работ (например, в [5]) предлагается использовать XSLT для преобразования XMI-описания в произвольный код.

Заключение

В данной статье была предложена методика создания синтеза поведенческих функциональных тестов на основе высокоуровневой спецификации UML. Главный акцент сделан на генерации тестов для аппаратуры, однако принципы, представленные в работе, могут быть применены и к программному обеспечению. Применение методики позволит создавать тесты систем, проверяющие их соответствие спецификациям, созданным ранее.

Литература

1. **UML 2.0** // Object Management Group. http://www.omg.org/spec/UML/2.0/ (дата обращения: 6.04.2010).
2. **Шопырин Д. Г., Шалыто А. А.** Объектно-ориентированный подход к автоматному программированию // Информационно-управляющие системы. 2003. № 5. С. 29–39.
3. **Леонтьев А. Е.** Применение UML при проектировании встраиваемых систем цифровой обработки сигналов // Информационно-управляющие системы. 2004. № 2. С. 38–44.
4. **Coyle F. P., Thornton M. A.** From UML to HDL: a Model Driven Architectural Approach to Hardware-Software Co-Design // Information Systems: New Generations Conf., 4–6 April 2005. P. 88–93.
5. **Rieder M. et. al.** Synthesized UML, a Practical Approach to Map UML to VHDL // Rapid Integration of Software Engineering: Lecture Notes in Computer Science. — Berlin, Germany: Springer, 2006. Vol. 3943. P. 203–217.
6. **Unified** Modeling Language: Tools // DMOZ: open directory project. http://www.dmoz.org/Computers/Programming/Methodologies/Modeling_Languages/Unified_Modeling_Language/Tools/ (дата обращения: 6.04.2010).
7. **Pires W., Brunet J., Ramalho F.** UML-based Design Test Generation // Proc. of the 2008 ACM Symp. on Applied Computing, Fortaleza, Ceara, Brazil, 16–20 Mar. 2008 / Association for Computer Machinery (ACM). — N. Y., USA, 2008. P. 735–740.
8. **Chen M., Mishra P.** Coverage-driven Automatic Test Generation for UML Activity Diagrams // Proc. of the 18th ACM Great Lakes Symp. on VLSI, Orlando, Florida, USA, 4–6 May 2008 / Association for Computer Machinery (ACM). N. Y., USA, 2008. P. 139–142.
9. **Gnesi S., Latella D., Massink M.** Formal Test-case Generation for UML Statecharts // Proc. of the Ninth IEEE Intern. Conf. on Engineering Complex Computer Systems Navigating Complexity in the e-Engineering Age, 14–16 April 2004 / IEEE Computer Society. Washington, D. C., USA, 2004. P. 75–84.
10. **std**_logic_1164 multi-value logic system. http://standards.ieee.org/downloads/1076/1076.2-1996/std_logic_1164.vhdl (дата обращения: 6.04.2010).
11. **XML** Metadata Interchange. http://www.omg.org/technology/documents/formal/xmi.htm (дата обращения: 6.04.2010).