ИСПОЛЬЗОВАНИЕ МЕТОДА ПОКРЫТИЙ ПРИ ВЕРИФИКАЦИИ МОДЕЛЕЙ IDEF-0

А. Ю. Подъячев

Санкт-Петербургский институт информатики и автоматизации РАН

СПИИРАН, 14-я линия ВО, д. 39, Санкт-Петербург, 199178

<gedo@yandex.ru>

УДК 004.05

Подъячев А. Ю. **Использование метода покрытий при верификации моделей IDEF-0** // Труды СПИИРАН. Вып. 5. — СПб.: Наука, 2007.

Аннотация. *Разработка программного обеспечения с использованием моделирования зачастую сталкивается с проблемами ресурсоемкости проверки моделей комплексных систем. Данная статья рассматривает метрические показатели симуляционного тестирования и их применение в контексте непосредственной верификации моделей.* — Библ. 9 назв.

UDC 004.05

Podyachev A. Y. **Coverage methods in IDEF-0 models verification** // SPIIRAS Proceedings. Issue 5. — SPb.: Nauka, 2007.

Abstract. *Today's model driven development of complex software designs requires reliable verification methods. This paper presents metrics of simulation testing and they application in models verification context.* — Bibl. 9 items.

1. Введение

В настоящее время быстрая разработка сложного программного обеспечения требует все более гибких методов проверки, в том числе и формальных моделей на предварительных этапах разработки. Формально верификация выявляет корректность дизайна в соответствии с заданным поведением. При этом проверяется, удовлетворяет ли именованный граф переходов состояний спецификации этого поведения. Как правило, поведение выражается в терминах логических отношений или конечных автоматов.

Одним из логических методов является определение бессодержательного удовлетворения спецификации, где частями спецификации являются случаи гипотетических ошибок безотносительно спецификации.

Например, спецификация «каждый запрос разрешен» полностью соответствует дизайну, в котором не отсылается не одного запроса. Похожий метод заключается в проверке истинности спецификации (спецификация истинна, если она удовлетворяет всем изменениям схемы). Такие критерии как очевидность, пробелы или истинность спецификации выявляют, как правило, сразу несколько проблем. Наиболее сложной задачей остается проверка полноты и завершенности спецификации. Несомненно, спецификации пишутся вручную, и их завершенность зависит полностью от компетентности человека, который ее пишет. Причины необходимости проверки полноты очевидны: ошибочное поведение поможет избежать затрат в дальнейшем при ручной проверке, или исключить их вовсе, если это поведение не покрыто спецификацией. Фактически, поведение, не покрытое спецификацией, может не попасть во внимание архитектора, который часто является единственным, кто планирует архитектуру будущего продукта.

Симуляционное тестирование позволяет акцентировать внимание на задаче создании максимально гибкого процесса верификации. Каждый входной вектор данных для графа влияет на разницу его прохождения. Граф будет адекватен, если его прохождение требует всех представленных входных данных. В случае сложного графа проверка всех путей со всеми различными данными является весьма ресурсоемкой задачей. Симуляционное тестирование, как правило, позволяет проверить архитектуру графа в соответствии с несколькими входными векторами одновременно, что весьма упрощает задачу. Вектора должны выбираться так чтобы проверка была наиболее гибкой, но и в этом случае оставшиеся ошибки дизайна могут влиять на сценарий верификации. Т.к. Симуляционное тестирование является эвристикой, которая замещает огромную задачу проверки всех входных векторов, то очень важно измерить полноту проверяемости входных последовательностей. Такую задачу можно решить, применив методы определения метрик покрытия. Метрики покрытия используются для того чтобы контролировать процедуру процесса проверки, определяющего необходимость большего или меньшего количества входных наборов данных. Измерение метрик должно быть частью проектирования и инициироваться входными данными. Например, метрики покрытия кода, при архитектуре, описанной на любом языке проектирования, или метрике числа строк кода выполняемых в течение симуляции. Далее будет рассмотрено несколько примеров метрик и способов их взаимодействия. Практика показывает, что, несмотря на степень точности оценивания по какой либо метрике, одна метрика не может абсолютно точно оценить качество проекта в целом. Для полноты верификации необходимо системное использование наборов метрик.

2. Симуляционное тестирование

Симуляционное тестирование подразумевает построение тестовой спецификации одновременно с построением программного продукта. Тестовая спецификация содержит условия и методы проверки реализуемой модели. Выполнение спецификации завершается в соответствии с выбранной конечной последовательностью входных данных в виде теста. Так, совокупность реализации имеет последовательность l входных данных, а тестом является конечная последовательность $t=i_0,i_1,\ldots,i_n\in(2^l)$ входных наборов. Реализация системной архитектуры может быть представлена различными формализмами. Например, формализм языка описания системы. Типичная программа в таком формализме определяет входные и выходные переменные различных модулей системы, и, используя управляющие и определяющие выражения, взаимодействия между модулями и окружением. Рассмотрим простой управляющий граф. Каждое выражение формализма отвечает контрольному состоянию и соответствует узлу графа. Определяющие выражения имеют одного наследника и контролирующие выражения, такие как if или while, имеют несколько наследников, соответственно ветвлению логики выполнения программы. Если дизайн описан именно таким графом, то все взаимодействия можно описать обходом этого графа. Формально, если граф G с набором состояний V, и входным тестовым набором $t=i_0,i_1,\ldots,i_n\in(2^l)^{n+1}$, то выполнение графа G на наборе t это последовательность переходов:

$$\langle i_0,I_0^0,\ldots,I_{m_0}^0,O_0\rangle,\langle i_1,I_0^1,\ldots,I_{m_1}^1,O_1\rangle,\ldots,\langle i_n,I_0^n,\ldots,I_{m_n}^n,O_n\rangle\in\left(2^I*V^+*2^O\right)^{n+1},$$

Труды СПИИРАН. 2007. Вып. 5. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2007. Issue 5. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

таких что l_0^0 является начальным состоянием графа G, для всех 0≤i≤n, состояние l_{mi}^i отвечает выражению чтения/записи, O_i является новым обозначением выходных значений переменных, и l_0^{i+1} соответствует управляющему потоку графа для положения l_{mi}^i до чтения i_{i+1}. Следовательно, позиции l_1^{i+1}, l_{mi+1}^{i+1} отвечают контрольным потокам графа от l_0^{i+1} до момента чтения следующего входного параметра. Часто входные и выходные переменные игнорируются, и основное внимание уделяется взаимодействию элементов.

3. Метрики покрытия при симуляционной проверке

Как правило, каждая метрика связана с особенностями представления архитектуры или цели верификации. Все метрики зависят от набора входных параметров или тестов $t\in(2^I)$ в соответствии с проектом подлежащем симуляционной оценке.

Метрика синтаксического покрытия помогает определить специфический критерий для описания синтаксической части проекта и измерений. Хорошее покрытие согласно данной метрике рассматривается как предусловия для использования более сложных и ресурсоемких метрик покрытия.

Метрика покрытия кода соответственно покрывает программный код или граф этого проекта. Измерение покрытия кода требует сравнительно небольших усилий и довольно легко интерпретируется. Это делает ее наиболее популярной метрикой в применении. Наиболее широко распространенными являются метрики покрытия операторов и ответвлений логики. Объект считается покрытым, если он посещается или вызывается во время выполнения входной последовательности тестов. Полностью формальное определение проекта подразумевает использование языков программирования, тогда как полуформальное определение проекта возможно и в правилах соответствия графическому представлению проекта. Для того чтобы входной последовательностью теста $t\in(2^I)$ при проходе по графу G и тестовом наборе t, запланированном, как последовательность перемещений, соответствовал набор состояний $l_0,\ldots,l_m$, мы предполагаем, что выражение T покрыто набором t, если 0≤j≤m так, чтобы контрольное состояние l_j удовлетворяло T. Также ветка кода (l, l') между двумя контрольными отметками покрыта набором t, если 0≤j≤m-1 так, что l_j= l и l_{j+1}= l'. Более сложные метрики позволяют измерить путь выражений отмечая пройденные переходы через узлы модели. Например, покрытие выражений проверяет является ли выражение Boolean верным при нескольких вариантах использования переменных данного типа (верно ли выражение a_1==a_2 как при a_1=a_2=0, так и при a_1=a_2=1).

Метрики, основанные на покрытии циклов, зависят от количества циклов и описывают сложность дизайна системы. Как правило, такое покрытие достаточно просто измерить и интерпретировать. Но в отличие от покрытия кода данные метрики трудно использовать для того чтобы сгенерировать новые тесты которые прямо симулируют еще не покрытые участки модели. Наиболее широко используемыми метриками здесь являются покрытие условий и выходов из цикла. Условие покрывается, если оно будет использовано, по крайней мере, один раз во время обработки входной последовательности. Подобным образом покрываются и выходные переменные, если их значение было изме-

нено по отношению к инициализации. Формально, для цепи S и входной последовательности tЄ(2^I), такой чтобы выполнение S по t соответствовало последовательности состояний <i_0,c_0,o_0>,<i_1,c_1,o_1>,…,<i_n,c_n,o_n>, можно сказать, что условие lЄL покрыто набором t, если j≥0, так что lЄc_0, но при этом l не принадлежит c_j. Также выходные значения oЄu покрыты набором t, если они удовлетворяют условию u<j_1<j_2 так что oЄc_0, но при этом o∉c_{j_1} и также O∉c_{j_2}. Проверка покрытия выходного значения требует, чтобы переменная изменилась как минимум два раза.

Метрики покрытия семантики. Метрики семантики оценивают часть функциональности проекта описанного при помощи входных данных языкового средства программирования. Такие метрики требуют непосредственного участия человека и гораздо более сложные, чем метрики синтаксиса. Рассмотрим некоторые метрики:

Покрытие формальных утверждений. Покрытие утверждений часто называется функциональным покрытием, пользователь предоставляет список утверждений относительно переменных проекта. Утверждения описывают некоторые условия, которые могут быть удовлетворены в процессе выполнения или изменения состояний проекта в процессе его выполнения. Они могут быть пропозиционными (срез текущего состояния системы) или временными (описывающими поведение в соответствии с несколькими циклами выполнения). Тест t покрывает утверждение a, если выполнение проекта по t удовлетворяет a. Метрика покрытия утверждений проверяет какие из утверждений покрываются при заданном наборе входных данных.

Покрытие изменений. При покрытии изменений пользователь вносит небольшое изменение в проект и проверяет степень отклонения текущего поведение в сторону некорректной работы. Покрытие теста t измеряется в процентах изменений дизайна по отношению к количеству изменений, приведшему к падению при t, т.е. процентах от изменений которые набор t обнаружил. Список интересующих изменений проекта может быть создан вручную или автоматически согласно некоторым требованиям. Например, локальным изменением может быть подмена одного единственного значения выходной переменной в цикле. Budd и Angluin [2] определяли окружение Ф(P) программы P как набор всех возможных видоизменений полученных из P посредством предопределенных локальных изменений. При покрытии изменений основной целью является покрытие всего окружения проекта, т.е. найти набор таких данных входного теста, что для каждого изменения дизайна существует, по крайней мере, один тест, который не завершится успешно. При проверке модели метрика покрытия изменений является наиболее ресурсоемкой.

4. Метрики покрытия при проверке моделей

Для того чтобы метрики можно было успешно применить в оценке качества моделей их нужно адаптировать к конкретным условиям. При синтаксическом покрытии проект должен быть представлен в виде программного кода или граф-схемы алгоритма в соответствии с методом измерения данной метрики. В процессе проверки мы посещаем все доступные части проекта, и метрики, используемые при симуляции, не могут быть применены напрямую для проверки модели. Поэтому необходимо адаптировать эти метрики замещением вопроса «посещена ли часть дизайна при проверке во время симуляции» вопросом «иг-

рает ли данная часть большую роль в успехе процесса проверки». Здесь под «ролью» подразумевается, что эта часть является неотъемлемой при покрытии или не пустым покрытием спецификации.

Покрытие кода. Предположим, что G будет граф-схемой проекта и φ является спецификацией удовлетворяющей G. Выражение т модели G покрыто спецификацией φ, если исключение т из G приводит к бессодержательному покрытию φ в измененном графе. Таким образом, при покрытии ветки <l,l'> графа, G ее выключение приводит к пустому удовлетворению спецификации φ. В терминах покрытия мы проверяем то, что исключение поведения, при котором переменные имеют частично покрытые утверждения для частных выражений, приводят бессодержательному удовлетворению φ. Например, если выражение a_1==a_2, и оно соответствует блоку операторов B, то мы проверяем, будет ли измененный дизайн, посредством исключения B и при a_1=0 и a_2=0, а также и при a_1=1 и a_2=1, удовлетворять спецификации.

Покрытие циклов. Вернемся к тому, что метрики покрытия условий и выходных значений проверяют, изменяется ли определенные условия и переменные в процессе обработки входных параметров. Мы замещаем этот вопрос вопросом будет ли отключение изменений приводить к бессодержательному соответствию спецификации. Так, условие lЄλ покрыто, если пустая спецификация удовлетворяется в цикле, полученном при замене переменной l на ее начальное значение. Подобным образом выходное значение oЄO покрыто, если пустая спецификация удовлетворена цепью полученной при допущении только единичного изменения значения O. Так, если начальное значение o=0 то нужный цикл будет получен при фиксации значения, как только оно будет изменено на 1.

Из числа метрик покрытия семантики, покрытие изменений уже адаптировано к условиям проверки модели. Как описано выше адаптированные метрики становятся более точными при проверке эффекта от изменений в графе не только соответствий спецификации, но и проверке соответствию пустой спецификации. Далее рассматриваются другие метрики покрытия семантики.

При покрытии графа состояний конечного автомата (СКА) мы берем абстрактную диаграмму F и проверяем влияние изменений и пропусков участков последовательности СКА на результат проверки модели по спецификации φ. При покрытии состояний, для состояния ω в F нужно проверить влияние пропуска ω или изменение выходных значений в ω для непустого соответствия спецификации. Очевидно, что измененная СКА, полученная из F посредством пропуска ω, имеет сокращенный набор поведения. Таким пропуском можно проверить только бессодержательное покрытие. С другой стороны для измененного графа СКА, полученного подменой выходного значения o Є O в ω, можно также изменить спецификацию и проверить ложность и пустое покрытие.

При покрытии путей схемы мы проверяем влияние пропуска или изменения конечного пути удовлетворяющего спецификации проекта. Путь π длины c в F является последовательностью конечных состояний $\omega_1,\ldots,\omega_c$ так что для всех $1\le i\le c-1$ мы имеем $\theta_{next}(\omega_i,\omega_{i+1})$. Путь π покрыт φ если измененная схема, полученная при пропуске всех поведений, которые содержат π, будет бессодержательно удовлетворять φ. С другой стороны можно представить модификации, которые замещают путь π на π' в графе. Таким образом, получается измененный граф, который может ложно или бессодержательно удовлетворять φ. В таком случае необходимо проверить оба условия покрытия данного участка графа. Все возможные изменения СКА могут быть представлены последова-

тельно для каждого набора элементов или подмножества наборов и будут соответственно отображать покрытие структуры, узлов или всего дерева.

5. Расчет покрытия

Для отображения результатов проверки модели достаточно хорошо подходят оба метода - и пустое, и ложное покрытие. Предположим, F является диаграммой состояний конечных автоматов, φ - спецификация удовлетворяющая F, и F' измененная диаграмма СКА. Если F' не удовлетворяет φ то можно однозначно сказать, что φ покрыто ошибочно. Если F' удовлетворяет φ, то это может быть бессодержательное покрытие φ. Таким образом, мы проверяем, удовлетворяет ли измененная схема спецификации, только если спецификация также подверглась изменению.

Основной алгоритм вычисляет степень ошибочного покрытия изменений, которые сделаны подменой значений переменных в наборах выходных значений состояния ω в диаграмме конечных автоматов. Идея заключается в поиске очевидного пути для измененной диаграммы СКА F' и его прохождения для опровержения спецификации. Пространство состояний продукта $2^X * S$, где X является набором переменных состояния, S – область состояний для прохода автомата, и переходы внутри продукта определены взаимодействием F и системы автоматического выполнения. Для того чтобы вычислить набор покрытых состояний необходимо добавить |X| новых переменных. Они нужны для передачи состояния ω, в котором замещается переменная q[7] . Вследствие этого можно построить дополненную схему алгоритма вместе с пространством состояний $2^X * 2^X * S$, где первый компонент состояния <ω,u,s> является рассматриваемым состоянием, а два других используются как при обычной автоматизации переходов внутри продукта. Значение первой компоненты берется неопределенным при инициализации и остается неизменным. Копия расширенной схемы с первым компонентом ω поможет проверить, содержит ли изменение F прежний путь, при замещении q в состоянии ω (в этом случае замещение q в ω нарушает спецификацию). Таким образом, когда расширенная схема в состоянии <ω,ω,s>, то набор последующих состояний содержит все наборы <ω,u,t> так что u – наследник ω и tЄδ(s,σ'), где σ' является простой меткой ω в F' ω,q. Данная ситуация описывает структурное покрытие, при котором выходное значение состояния ω подменяется при всех проходах. Также мы можем определить расширенную схему, в которой данное значение будет подменяться только один раз (проверка покрытия узла), или заданное количество раз (проверка определенного дерева состояний). С помощью символьного алгоритма находится набор P всех сочетаний <ω,u,s> , при которых существует оптимальный путь в графе.

В свою очередь, при проверке покрытия кода существует необходимость проверки: будет ли спецификация бессодержательно удовлетворена при пропуске частей кода. Согласно этому наиболее просто отследить изменения относительно программного кода. Путь k является количеством элементов подлежащих проверке, например количество строк кода. Введем новую переменную m, которая будет изменяться в интервале значений k. Индекс i для этого значения будет показывать изменение в элементе li, который мы хотим пропустить. Программный код выполняется с использованием отладчика, так что l_i в коде замещается выражением «if (m<>i) then li else skip». Измеряемый код отобра-

Труды СПИИРАН. 2007. Вып. 5. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2007. Issue 5. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

жает все изменения схемы. Производимый граф состояний конечного автомата будет также содержать все измененные состояния. Теперь можно применить упомянутый выше символьный алгоритм для определения всех изменений ведущих к бессодержательному покрытию спецификации.

При покрытии графа конечного автомата покрытие состояний и переходов может быть вычислено при помощи метрик основанных на подсчете изменений. Например, для покрытия путей графа мы исключим все последовательности, которые содержат конечный путь $\pi=\omega_1,\ldots, \omega_c$. Пусть M_π будет триггером, который отсеивает пути, которые содержат π в качестве подпути. Т.е. M_π является набором СКА, принимающим на вход пути ρ не содержащие π. Т.к. M_π наблюдает только за значениями контрольных переменных описывающих состояния, то набор входных переменных M_π является набором контрольных значений X от F, и M_π не имеет выходных значений. Для текущего пути π, измененный граф F' является результатом СКА $F * M_\pi$, который содержит только перечень из F не имеющих π в качестве подпути. Тогда π покрыто спецификацией φ, если F'_π бессодержательно удовлетворяет этой спецификации. Для набора путей $\{\pi_1,\ldots, \pi_k\}$ можно вычислить набор покрытых путей методом описанным ранее. Пространство состояний расширенной схемы состоит из пяти составляющих <m,x,u,r,s>, где m описывает набор путей, x – логическое выражение, и u,r,s-компоненты множества $F * M_{\pi m}$. Наследником <m,x,u,r,s> является набор <m,x,u',r',s'>, такой что <u',r',s'> будет преемником <u,r,s>, между заданным нами путем СКА $F'_{\pi m}$ и идеальным путем автомата $A\varphi[\psi\leftarrow x]$, где ψ подформула определяемая логикой x.

Подобным способом определяются изменения путей, которые замещают конечные пути π на π' той же длины, но направляющие систему другое выполнение. Если измененный путь имеет длину 1, то замещение перенаправляет один переход. Для пути π замещенного измененным путем π' нужно использовать селектор $M_{\pi,\pi'}$. В схеме F'_π от F по $M_{\pi,\pi'}$ все π замещаются π'. Нужно помнить, что для измененных путей, в отличие от просто пропущенных, можно определить как ложное, так и бессодержательное покрытие. Вычисление ложности такой схемы идентично определению данной метрики для простого графа, где область состояний расширенной схемы описывается четырьмя наборами <m,u,r,s>, где m - описывает измененный путь, и остальные параметры являются компонентами F'_π. В качестве примера рассмотрим простой граф, полученный в результате работы транслятора IDEF-0 в UML. Далее представлены исходная и полученная модель.

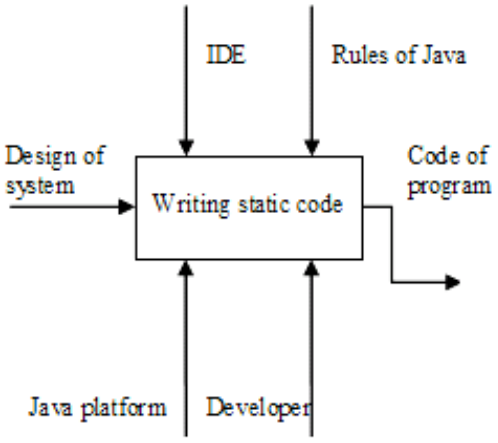

Рис 1. Исходная диаграмма IDEF-0.

Труды СПИИРАН. 2007. Вып. 5. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2007. Issue 5. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

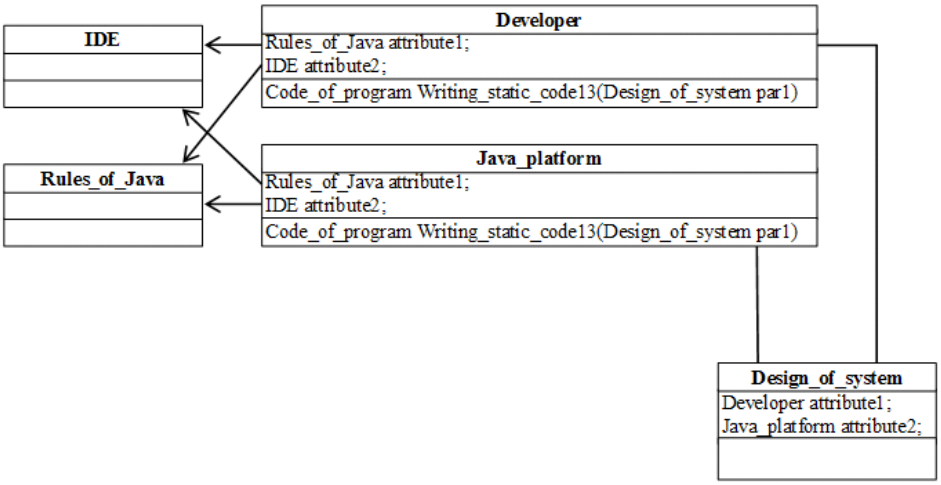

Рис 2. Полученная диаграмма классов.

Данная диаграмма классов должна удовлетворять спецификации в качестве которой выступает исходная IDEF-0 модель. Имеем пути графа:

Design_of_system->Java_Platform->IDE;

Design_of_system->Java_Platform->Rules_of_Java;

Design_of_system->Developer->IDE;

Design_of_system->Developer-> Rules_of_Java;

Для каждого состояния в наборе определены соответствующие параметры, полученные из исходной диаграммы. Имеется набор состояний, при исключении которых часть спецификации удовлетворяется бессодержательно. Но при исключении любого из вышеуказанных путей мы получаем лишь частное бессодержательное удовлетворение спецификации. Таким образом, необходимо исключить два, минимально пересекающихся, пути, чтобы полностью бессодержательно удовлетворить спецификацию. Исключив первый и третий путь, получаем бессодержательное удовлетворение спецификации, что и демонстрирует, то, что данный граф покрыт данной спецификацией.

6. Заключение

Приведенные методы проверки и определения метрик моделей позволят достаточно гибко оценивать качество моделей безотносительно нотации, в которой модель была описана. В контексте разрабатываемого метода проверки трансляции формальных моделей приведенная методика позволит провести сравнительный анализ одних и тех же показателей для модели в исходной нотации до трансляции и полученной результирующей модели другой нотации. Естественно открытым остается вопрос адаптации метрик применимых только в одной нотации, например метрика покрытия кода в нотации UML и ее отображение в терминах нотации IDEF-0. Также приведенные критерии проверки соответствия спецификации будут использованы для создания систематизированного подхода при комплексной проверке моделей. При помощи математического аппарата, описывающего логические взаимосвязи показателей для параллельно оцениваемых моделей, можно выработать набор рекомендаций по дальнейшей декомпозиции, упрощению или возможности усложнения моделей.

ЛИТЕРАТУРА

1. *Подъячев А. Ю., Афанасьев С. В.* Тестирование трансляции формальных моделей // Труды СПИИРАН. СПб.: Наука, 2006. Вып. 3, т. 2. С.156–161.
2. *Budd T. A., Angluin D.* Two notions of correctness and their relation to testing // Acta Informatica. 1992, issue 18. P. 31–45.
3. *Armon R., Fix L., Flaisher A., Grumberg O., Piterman N., Vardi M. Y.* Enhanced vacuity detection for linear temporal logic in Computer Aided Verification // Proc. 15th International Conference. Springer-Verlag, 2003.
4. *Beatty D., Bryant R.* Formally verifying a microprocessor using a simulation methodology // Proc. 31st Design Automation Conference. IEEE Computer Society, 1994. P. 596–602.
5. *Bening L., Foster H.* Principles of verifiable RTL design – a functional coding style supporting verification processes. Kluwer Academic Publishers, 2000.
6. *Clarke E. M., Grumberg O., Peled D.* Model Checking. MIT Press, 1999.
7. *Chockler H., Kupferman O., Yardi M. Y.* Coverage Metrics for Formal Verification, Jerusalem, 2003.
8. *Peled D.* Software Reliability Methods. Springer-Verlag, 2001.
9. *Born M., Schieferdecker I., Li M.* UML Framework for Automated Generation of Component-Based Test Systems. Software Engineering Applied to Networking and Parallel // Distributed Computing, Reims, France, 2000.